Hands-On Pattern Recognition

Challenges in Machine Learning, Volume 1

Hands-On Pattern Recognition
Challenges in Machine Learning, Volume 1

Isabelle Guyon, Gavin Cawley,
Gideon Dror, and Amir Saffari, editors

Nicola Talbot, production editor

Microtome Publishing
Brookline, Massachusetts
www.mtome.com

Hands-On Pattern Recognition
Challenges in Machine Learning, Volume 1

Isabelle Guyon, Gavin Cawley,
Gideon Dror, and Amir Saffari, editors

Nicola Talbot, production editor

ISBN-13: 978-0-9719777-1-6

Series Foreword

During recent years, a team of motivated researchers led by Isabelle Guyon has done an admirable job conceiving and organizing performance evaluations in machine learning and inference, in the form of competitions or challenges. This book opens the series Challenges in Machine Learning. It contains papers by the top ranking challenge participants, providing instructive analyses of the results. It also includes tutorials and theoretical papers on topics addressed by the challenges.

Designing good challenges is far from trivial. The team benefitted from Isabelle's experience as a member of technical staff at Bell Laboratories, where she was part of a group that held world records in pattern recognition tasks, while at the same time employing theoreticians proving theorems about statistical learning. This group, which I fondly remember from the time I spent there as a student, always put great emphasis on benchmarking, but at the same time it avoided the trap of searching only the vicinity of local optima by tweaking existing methods — quite the contrary; in the 1990s, the group came up with Support Vector Machines, a development to which Isabelle made significant contributions.

While these methods are now part of our standard toolkit, Isabelle has moved on to design benchmarks for tasks that are harder to evaluate. This is not only a great service to the community, but it will also enable scientific progress on problems that are arguably more difficult than classical pattern recognition. In particular, the benchmarks include the fascinating problem of causal inference. Finding causal directions from observations is not only a profound issue for the philosophy of science, but it can also develop into an important area for practical inference applications. According to Hans Reichenbach, all statistical associations arise from causal mechanisms. However, machine learning has so far focused on the statistical 'surface' of things. Penetrating this surface would help us detect regularities that are more robust to issues that make our life difficult today, including nonstationarity and covariate shifts. It may also move us closer to the long term goal of building intelligent systems that learn about the structure of the world in an autonomous way.

Bernhard Schölkopf
Max Planck Institute, Tübingen, Germany

Foreword

Machine learning is about building machines that learn. Building machines is engineering. The idea is to *create an artefact*. The hope is that these artefacts are *useful* (typically to others). The machines have to solve some end-user problem. The present book grapples with a number of key issues central to this task — how to represent the data, how to select suitable models, and how to evaluate performance.

Engineers design many types of machine — flying machines, communication machines etc. The question of how to evaluate performance arises in many areas. The question of representing data also arises (although it means something different). If one compares the state-of-the-art in performance measurement and prediction in mature engineering disciplines, Machine Learning looks primitive in comparison. Communications engineers [4] can design systems and predict their performance in messy real world situations very well. The same is true in aeronautics [2]. Why is Machine Learning lagging behind? And what can be done about it?

One thing that more mature engineering disciplines seem to have in common is a wide variety of "ways of knowing" or acceptable research practices. It is well accepted in aeronautical engineering that it is useful to have design rules of thumb [2]. In fact many scholars have argued that the traditional view of engineering as applied science is back-the-front [5]. There are a range of different categorisations of "useful knowledge" different to the tired "pure versus applied" (see for example Mokyr's [3] distinction between propositional and prescriptive knowledge). How do philosophical reflections on the nature of engineering knowledge affect the development of machine learning, and how is it relevant to the present book? Simply, different ways of knowing require different means of inquiry. The benchmark competitions summarised in this book *are* a different way of knowing. They are analogous to the principled (but not scientifically *derived*) empirical studies in many branches of engineering that complement more well-honed scientific knowledge.

But this is a starting point — a beginning rather than an end. There are in fact many profound scientific questions to be answered regarding performance evaluation. For example, a satisfactory theory of cross-validation still eludes the community. And the plethora of different performance measures need to brought into better order. Self-bounding learning algorithms [6] (that not only estimate an object of interest but also estimate how well it is estimated) deserve further study. There are many more questions to be answered.

Much machine learning research is driven by the interests of the researcher. It is often technique-oriented rather than problem driven. End users often neither understand nor care about the distinctions between variants of different learning algorithms. A problem-oriented perspective is rare (an exception is [1]). However, end-users *do* care about performance, how to represent their data and how to choose models. These topics are the focus of this book, which marks a great first step in building a richer understanding of the engineering of machines that learn.

Robert C. Williamson
Canberra, Australia.

[1] Vic Barnett, *Comparative Statistical Inference*, (3rd Edition) John Wiley and Sons, Chichester 1999.

[2] Walter G. Vincenti, *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*, The Johns Hopkins University Press, Baltimore 1990.

[3] Joel Mokyr, *The Gifts of Athena: Historical Origins of the Knowledge Economy*, Princeton University Press, Princeton, 2002.

[4] John G. Proakis and Masoud Salehi, *Communication Systems Engineering*, Pearson Education, 2003.

[5] Marc J. de Vries, "The Nature of Technological Knowledge: Extending Empirically Informed Studies into What Engineers Know", *Techné*, **6**:3, 1–21, 2003.

[6] Yoav Freund, "Self bounding learning algorithms", *COLT '98: Proceedings of the eleventh annual conference on Computational learning theory, ACM Press*, 247–258, 1998.

Preface

Recently organized competitions have been instrumental in **pushing the state-of-the-art in machine learning, establishing benchmarks to fairly evaluate methods,** and **identifying techniques, which really work.**

This book harvests three years of effort of hundreds of researchers who have participated to three competitions we organized around five datasets from various application domains. Three aspects were explored:

- **Data representation.**
- **Model selection.**
- **Performance prediction.**

With the proper data representation, learning becomes almost trivial. For the defenders of fully automated data processing, the search for better data representations is just part of learning. At the other end of the spectrum, domain specialists engineer data representations, which are tailored to particular applications. The results of the "Agnostic Learning vs. Prior Knowledge" challenge are discussed in the book, including longer versions of the best papers from the IJCNN 2007 workshop on "Data Representation Discovery" where the best competitors presented their results.

Given a family of models with adjustable parameters, Machine Learning provides us with means of "learning from examples" and obtaining a good predictive model. The problem becomes more arduous when the family of models possesses so-called hyper-parameters or when it consists of heterogenous entities (e.g. linear models, neural networks, classification and regression trees, kernel methods, etc.) Both practical and theoretical considerations may yield to split the problem into multiple levels of inference. Typically, at the lower level, the parameters of individual models are optimized and at the second level the best model is selected, e.g. via cross-validation. This problem is often referred to as model selection. The results of the "Model Selection Game" are included in this book as well as the best papers of the NIPS 2006 "Multi-level Inference" workshop.

In most real world situations, it is not sufficient to provide a good predictor, it is important to assess accurately how well this predictor will perform on new unseen data. Before deploying a model in the field, one must know whether it will meet the specifications or whether one should invest more time and resources to collect additional data and/or develop more sophisticated models. The performance prediction challenge asked participants to provide prediction results on new unseen test data AND to predict how good these predictions were going to be on a test set for which they did not know the labels ahead of time. Therefore, participants had to design both a good predictive model and a good performance estimator. The results of the "Performance Prediction Challenge" and the best papers of the "WCCI 2006 workshop of model selection" will be included in the book.

A selection of the special topic of JMLR on model selection, including longer contributions of the best challenge participants, are also reprinted in the book.

Isabelle Guyon, Gavin Cawley, Gideon Dror, Amir Saffari, Editors. January 2011.

Table of Contents

Part I

Introduction

Chapter 1

Challenges in Data Representation, Model Selection, and Performance Prediction

Isabelle Guyon ISABELLE@CLOPINET.COM
ClopiNet, Berkeley, CA 94708, USA

Amir Saffari AMIR@YMER.ORG
Graz University of Technology, Austria

Gideon Dror GIDEON@MTA.AC.IL
Academic College of Tel-Aviv-Yaffo, Israel

Gavin Cawley GCC@CMP.UEA.AC.UK
University of East Anglia, UK

Abstract

We organized a series of challenge for the conferences IJCNN/WCCI 2006, NIPS 2006 and IJCNN 2007 to explore various aspects of machine learning, ranging from the choice of data representation to the selection of the best model. The class of problems addressed are classification problems encountered in pattern recognition (classification of images, speech recognition), medical diagnosis, marketing (customer categorization), text categorization (filtering of spam). All three challenges used the same five datasets, formatted in two data representations: raw data and preprocessed data in a feature-based representation. Post-challenge submissions can still be made at: http://www.agnostic.inf.ethz.ch/. Several chapters in this volume are contributed by top ranking challenge participants who describe their methods in details.

Keywords: Supervised learning; Classification; Competition; Performance prediction; Model selection; Agnostic Learning; Prior Knowledge; Domain Knowledge; Boosting; Ensemble methods; Kernel methods; Support Vector Machines; SVM; LSSVM; Data Grid models.

1.1. Introduction

Challenges have proved to be a great stimulus for research in machine learning, pattern recognition, and robotics. Robotics contests seem to be particularly popular, with hundreds of events every year, the most visible ones probably being the DARPA Grand Challenges of autonomous ground vehicle navigation and RoboCup, featuring several challenges for robots including playing soccer or rescuing people. In data mining and machine learning, several conferences have regularly organized challenges over the past 10 years, including the well established Text Recognition Conference (e.g., TREC) and the Knowledge Discovery in Databases cup (KDD cup). More specialized pattern recognition and bioinformatics conference have also held their own contests, e.g. CASP for protein structure prediction, DREAM for reverse engineering biological networks, ICDAR for document analysis, and the PASCAL Visual Object Challenge (VOC) for object recognition. The European network of excellence PASCAL2 has actively sponsored a number of challenges around hot themes in machine learning, which have punctuated workshop at NIPS and other conferences. These contests are oriented towards scientific research and the main reward for the winners is to disseminate the product of their research and obtain recognition. In that respect, they play a different role than challenges like the Netflix prize, which offer large monetary rewards for solving a task of value to the Industry (movie referral in than particular case), but are narrower scope. Attracting hundreds of participants and the attention of a broad audience of specialists as well as sometimes the general public, these events have been important in several respects: (1) pushing the state-of-the art, (2) identifying techniques which really work, (3) attracting new researchers, (4) raising the standards of research, (5) giving the opportunity to non-established researchers to make themselves rapidly known.

In 2003, we organized a challenge on the theme of feature selection (Guyon et al., 2005) whose results were discussed at NIPS 2003. A book was published collecting tutorial papers and the best papers from the challenge participants (Guyon et al., 2006a). We have continued organizing challenges regularly every year, exploring various aspects of machine learning: model selection, causal discovery, and active learning (see `http://clopinet.com/challenges`). The present paper summarizes the results of three challenges: The IJCNN/WCCI 2006 "performance prediction challenge" (Guyon et al., 2006b), the NIPS 2006 "model selection game", and the IJCNN 2007 "agnostic learning *vs.* prior knowledge challenge" (Guyon et al., 2007, 2008).

1.2. Motivations for this series of challenges

Predictive modeling for classification and regression is a central problem addressed in statistics, data mining and machine learning. Examples include pattern recognition (handwriting recognition, speech recognition, object classification, text classification), medical diagnosis and prognosis, spam filtering, etc. In such problems, the goal is to predict an outcome (a category or a continuous variable), given patterns represented as vectors of features, graphs, texts, etc. The standard approach to tackle such problems is to construct a predictive model from examples of pairs pattern/outcome. After training, the model should be capable of making predictions of outcome give new examples, not used for training (generalization).

Machine learning researchers have been devoting much effort in the past few decades to inventing and improving learning algorithms. In proportion,less effort has been dedicated to problems of **data representation**, **model selection**, and **performance prediction**. This paper that summarizes the results of challenges we organized around these topics. The questions addressed are the following:

- **Data representation:** The difficulty of learning from examples can be alleviated with a proper data representation. However, finding such representations rely on expert domain knowledge. Conversely, adding more training data may yield better performance without requiring such knowledge. How should human resources be rather exploited: in collecting more data or in incorporating domain knowledge in the design of the data representation?

- **Model selection:** There are many learning machine architectures and algorithms to choose from. Given a certain amount of available training data, what strategy should be adopted to deliver the best predictive model, including choosing the model family and the model architecture, and tuning all hyper-parameters and parameters?

- **Performance prediction:** It is one thing to deliver the best model, but it is a different thing to know how well it will perform on new, previously unseen, data. The former problem is that of model selection. The latter is that of performance prediction. A good estimator of performance prediction is obviously a good model selection criterion. However, there may exist simpler model selection criteria allowing only to rank models according to predictive power without predicting their performance. The problem of performance prediction is to make best possible use of the training data to both train the model and predict its prediction accuracy on future test data.

1.3. Datasets

In all challenges we used the same five datasets, however, the data were formatted differently and scrambled to prevent the participants to use results of previous challenges as a head start and give an even chance to new competitors. The tasks are five two-class classification problems spanning a variety of domains (marketing, handwriting recognition (HWR), drug discovery, text classification, and ecology) and a variety of difficulties, with sufficiently many examples to obtain statistically significant results. The input variables are continuous or binary, sparse or dense. Some raw data representations are not feature based. In some problems, the class proportions are very imbalanced. A detailed report on the data preparation is available (Guyon, 2005). The main data characteristics are summarized in Table 1.1. Non-feature based representations are supplied for HIVA (molecular structure) and NOVA (emails) and were used in the "data representation" competition.

Table 1.1: Datasets of the three challenges

Dataset	Domain	Number of examples (train/valid/test)	Percent pos. class	Number of features	
				Raw data	Preproc.
ADA	Marketing	4147 / 415 / 41471	28.4	14	48
GINA	HWR	3153 / 315 / 31532	49.2	784	970
HIVA	Drug discovery	3845 / 384 / 38449	3.5	Molecules	1617
NOVA	Text classif.	1754 / 175 / 17537	28.5	Text	16969
SYLVA	Ecology	13086 / 1309 / 130857	6.2	108	216

1.4. Design of the challenges

1.4.1. General evaluation procedure

The design of the challenges was informed by experience gained from another challenge we organized previously on feature selection (Guyon et al., 2005). In particular, we used a system of on-line submission, which provided the competitors with immediate feed-back on a small subset of the data called the validation set. The organizers provided initial submissions to bootstrap the challenge. A toolkit including some of the methods performing best in previous challenges was also provided (the so-called Challenge Learning Object Package CLOP (Saffari and Guyon, 2006), see Appendix). At the end of a development period, the validation set labels were revealed. The final ranking was performed on a large separate test set. The test set labels will remain hidden to permit meaningful comparison with post-challenge submissions.

Performance was measured in balanced error rate (BER), which is the average of the error rate on the positive class and the error rate on the negative class. As is known, for i.i.d. errors corresponding to Bernouilli trials with a probability of error p, the standard deviation of the error rate E computed on a test set of size m is $\sqrt{p(1-p)/m}$. This result can be adapted to the balanced error rate. Let us call m_+ the number of examples of the positive class, m_- the number of examples of the negative class, p_+ the probability of error on examples of the positive class, p_- the probability of error on examples of the negative class, and E_+ and E_- the corresponding empirical estimates. Both processes generating errors on the positive or negative class are Bernouilli processes. By definition, the balanced error rate is $BER = (1/2)(E_+ + E_-)$, and its variance is $\text{var}(BER) = (1/4)(\text{var}(E_+) + \text{var}(E_-))$. The standard deviation of the BER using m_+ and m_- examples is therefore

$$\sigma = \frac{1}{2}\sqrt{\frac{p_+(1-p_+)}{m_+} + \frac{p_-(1-p_-)}{m_-}}. \tag{1.1}$$

For sufficiently large test sets, we may substitute p_+ by E_+ and p_- by E_- to compute σ.

To rank the participants we adopted the following scheme: The entries were first ranked for each individual dataset. Then a global ranking was obtained based on the average rank over all five datasets. Different ranking scores incorporating the BER were used in the different challenges.

1.4.2. Performance prediction challenge

We ran first the competition on **performance prediction**, which was easiest technically to organize. In that competition, in addition to providing predicted labels on test data, the participants had to also provide an **estimate of their performance on the test set**. For the performance prediction challenge, the ranking score balanced the classification accuracy and performance prediction accuracy. Denoting as BER the balanced error rate actually computed from predictions made on test examples, and BER_{guess} the challenger's own performance prediction, we defined our ranking score as:

$$S = BER + \delta_{BER}(1 - e^{-\delta_{BER}/\sigma}) \ , \tag{1.2}$$

where $\delta_{BER} = |BER_{\text{guess}} - BER|$ measures in absolute value the difference between the computed BER and the predicted BER. The multiplicative factor $(1 - e^{-\delta_{BER}/\sigma})$ accounts for our uncertainly of the exact BER, since we can only estimate it on a finite test set of size m. If the BER error bar σ is small compared to the error of the challenger δ_{BER}, then this factor is just one. The ranking score becomes simply $BER + \delta_{BER}$. But if σ is large relative to the error made by the challenger, we have $S \simeq BER$.

The challenge started September 30th, 2005 and ended March 1st, 2006 (duration: 21 weeks). Therefore, the competitors had several months to build classifiers with provided (labeled) training data. We estimated that 145 entrants participated. We received 4228 "development entries" (entries not counting towards the final ranking). A total of 28 participants competed for the final ranking by providing valid challenge entries (results on training, validation, and test sets for all five tasks). We received 117 submissions qualifying for the final ranking (a maximum of 5 entries per participant was allowed). The participation doubled in number of participants and entry volume compared to the feature selection challenge. The results of the challenge were discussed at the IJCNN/WCCI 2006 conference. The website of the performance prediction challenge including details on the results is available at: http://www.modelselect.inf.ethz.ch/. The submissions of the website are closed because the same datasets were used in the follow-up ALvsPK challenge, whose website remains open.

1.4.3. ALvsPK challenge on data representation

The **agnostic learning *vs.* prior knowledge challenge** (AlvsPK) had two parallel tracks: AL and PK. For the "agnostic learning" (AL) track we supplied data preprocessed to provide a simple feature-based representation, suitable for use with any off-the-shelf machine learning or data mining package. The pre-processing used was identical to that used in the previous challenge on performance prediction, but with a new split of the data. The participants had no knowledge of the identity of the features in the agnostic track. The raw data representations were supplied in the "prior knowledge" (PK) track. They were not necessarily in the form of data tables. For instance, in the drug discovery problem the raw data consists of a representation of the three dimensional structure of the drug molecules; in the text processing problem, the raw data are messages posted to USENET newsgroups. The participants had full knowledge of the meaning of the representation of the data in the PK track. Therefore, PK competitors had the opportunity to use domain knowledge to build better predictors and beat last year's AL results or make new "agnostic" entries. Note that the training/test splits used are the same in both tracks, but the example ordering is different in each data subset to hinder matching patterns in the two representations and/or submitting results with the representation prescribed for the other track.

The Balanced Error Rate (BER) was used for scoring the participants and otherwise the modalities of evaluation were similar as those of the previous challenge on performance prediction. The challenge started on October 1st, 2006 and ended on August 1st, 2007 (duration: 10 months). Two milestone rankings of the participants were made using the test set, without revealing either the test labels or the test performance: on December 1st, for the "model selection game", and on March 1st, to allow us to publish intermediate results (Guyon et al., 2007). To be eligible for the final ranking, submissions had to include results on all the tasks of the challenge in either track, on the test data. However, recognizing that domain knowledge is task specific, prizes were given for each task individually in the "prior knowledge" track. For each group, only the last five entries in either track counted towards the final ranking. The results of the ALvsPK challenge were discussed at the IJCNN 2007 conference. Details can be found on the website or the challenge http://www.agnostic.inf.ethz.ch/.

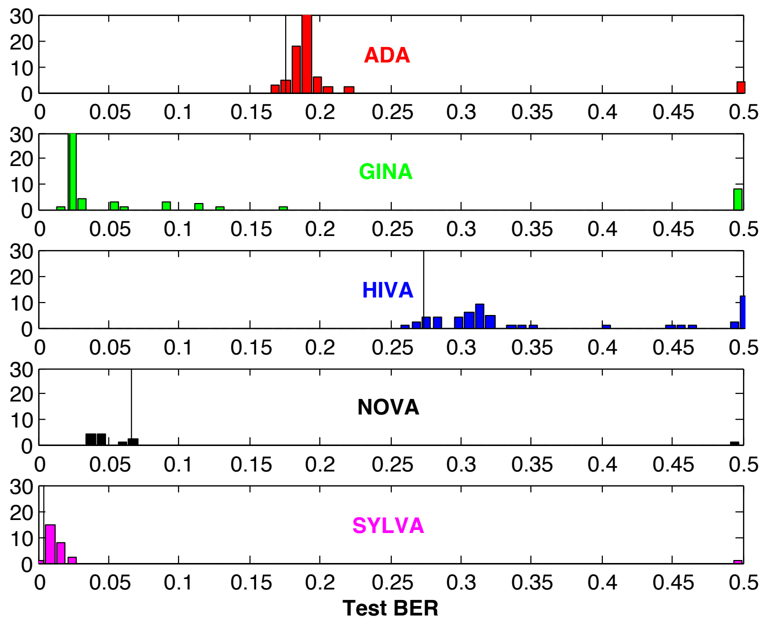
1.4.4. Model selection game

In previous challenges, we noted that different teams using similar classification methods (even sometimes the same software package) obtained very different results. We conjectured that this variance may be due to differences in model selection strategies. To stimulate research in model selection and verify our conjecture, we organized a **model selection game**, within the ALvsPK challenge. Using the data of the AL track, the competitors were asked to return results with the constraint of using only models from the toolkit that the organizers provided: the Challenge Learning Object Package CLOP (Saffari and Guyon, 2006), see Appendix). The package was available for downloading from the web site of the challenge, and the latest version is available from http://clopinet.com/CLOP, see Appendix for a brief description. That toolkit includes classical methods and some of the algorithms that worked best in the previously organized challenges. Hence comparisons between model selection methods were facilitated. The results of the model selection game were discussed ant the NIPS 2006 conference. Details can be found on the website or the challenge http://www.agnostic.inf.ethz.ch/.

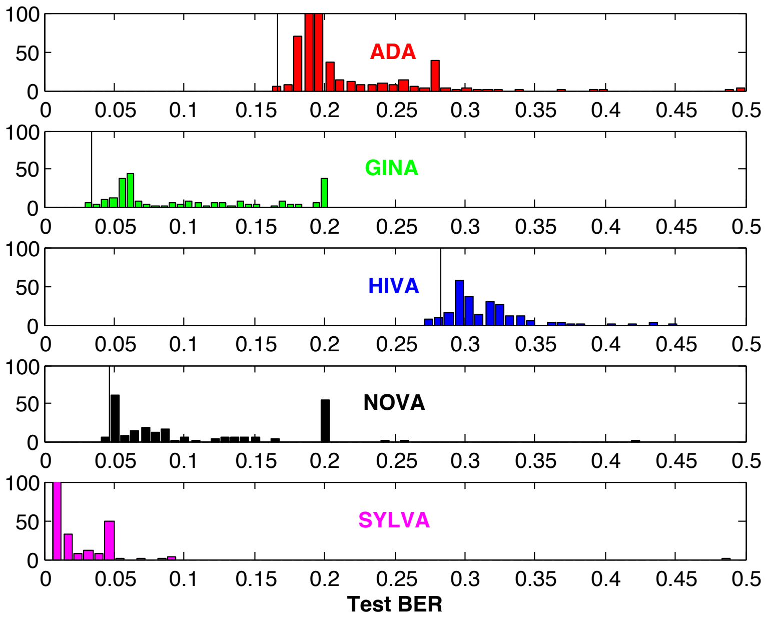
1.5. Results

1.5.1. General observations

In Figure 1.1, we show the distribution of performance on the test set of the entries who qualified for ranking in the ALvsPK challenge. Graphs similar to those of the AL track were obtained in the challenge on performance prediction, which uses the same datasets. We see that the datasets vary in difficulty and that there are noticeable differences between the two tracks.



(*a*)



(*b*)

Figure 1.1: Distribution of test set Balanced Error Rate (BER). (a) Prior knowledge (PK) track. (b) Agnostic learning (AL) track. The thin vertical line indicates the best ranked entry (only the 5 last of each participant are ranked).

HIVA (drug discovery) seems to be the most difficult dataset: the average BER and the spread are high. ADA (marketing) is the second hardest. The distribution is very skewed and has a heavy tail, indicating that a small group of methods "solved" the problem, which was not obvious to others. NOVA (text classification) and GINA (digit recognition) come next. Both datasets have classes containing multiple clusters. Hence, the problems are highly non-linear. This may property of the data explain the very long distribution tails. Finally, SYLVA (ecology) is the easiest dataset, due to the large amount of training data.

We surveyed the participants to get more details about the methods employed. The survey reveals that the preprocessing methods used in the challenge on performance prediction and in the AL track of the ALvsPK challenge were very elementary. Tree classifiers most often use no preprocessing at all. The most common preprocessing is feature standardization (subtract the mean and divide by the standard deviation for each feature). A few entries used PCA or ICA to extract features. Most entries used no feature selection. Some entries resampled the training data to balance the two classes. There does not seem to be a correlation between the type of preprocessing used and how well people did in the challenge.

As classification methods, a variety of algorithms were used in top ranking entries, including ensembles of decision trees, kernel methods/SVMs, Bayesian Neural Networks, ensembles of linear methods, and Naïve Bayes. It is interesting to note that single or ensembles thereof did generally better than mixed models.

1.5.2. Results of the Performance Prediction Challenge

The winner by average rank for the **performance prediction challenge** is Roman Lutz (Lutz, 2006). The best average score was obtained by Gavin Cawley and Nicola Talbot (Cawley and Talbot, 2007), who obtained also the best guessed BER. Radford Neal obtained the best AUC (data not shown). The full result tables are found on the web-site of the challenge (`http://www.modelselect.inf.ethz.ch/`).

The top ranking entries have made errors on their performance prediction of the same order of magnitude as the error bar of the performance computed on test examples. This is an important achievement considering that the training set is ten times smaller than the test set and the validation set 100 times smaller.

We examined how the various methods did with respect to optimizing the test BER and the δ_{BER}. In Figure 1.2 each point represents one of the 117 final entries, for each dataset. The best ranking entry according to the challenge score is indicated by an arrow.

The symbols code for the methods used:

- X: Mixed or unknown method.
- TREE: Ensembles of trees (like Random Forests, RF).
- NN/BNN: Neural networks or Bayesian neural nets.
- NB: Naïve Bayes.
- LD/SVM/KLS/GP: Methods linear in their parameters, including kernel methods and linear discriminant (LD), Support Vector Machines (SVM), Kernel Least-Squares (KLS) and LS-SVM, Gaussian Processes (GP).

Figure 1.2 reveals that Naïve Bayes did very well on two datasets (ADA and SYLVA) but poorly on others. Similarly, kernel methods did very well on most datasets (they rank first for HIVA, GINA, and NOVA), but they did poorly on ADA. This failure on ADA make them rank only fifth in the overall ranking. They are the most frequently used type of method, but their performance shows a lot of variance. On the contrary, ensembles of decision trees are not so popular, but they perform consistently well on all datasets in this challenge, even though they are never the top entry for any of the datasets. The challenge winner used an ensemble of decision trees. Similarly, Bayesian neural networks did well on all datasets, even though they were not best for any. They end up ranking third in the overall scoring.

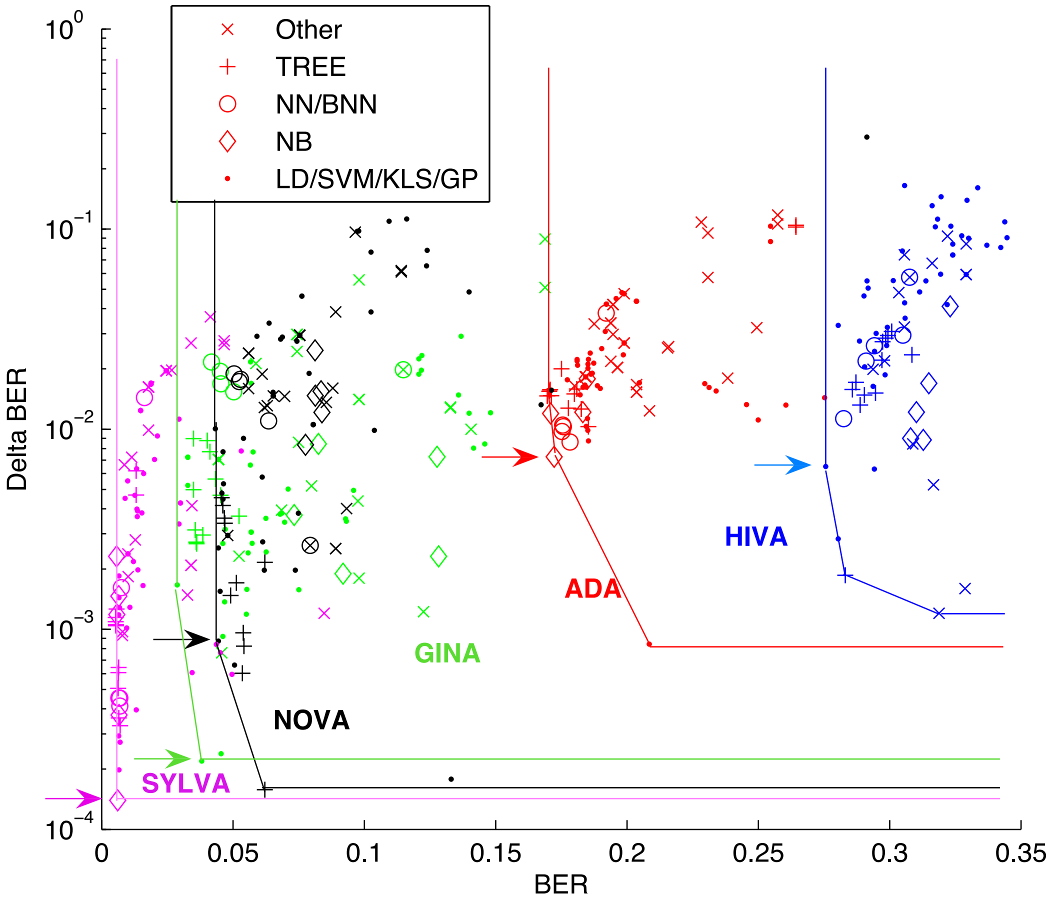


Figure 1.2: Methods employed. We show a scatter plot of the methods employed in the IJCNN06 challenge, coarsely grouped into five categories. The arrows indicate the winners of the performance prediction challenge. The lines indicate the Pareto front (see text for details).

This analysis also tells us something about the datasets: SYLVA has a large number of training examples, so all methods essentially perform well, even the simple naïve Bayes. We know by design that GINA and NOVA are very non-linear. The top ranking participants used highly non-linear methods and naïve Bayes failed. HIVA seems to also be in this category, not too surprisingly: chemical molecules with *very* different structures can be active or inactive drugs. ADA has a particularity that makes kernel methods fail and which should be further investigated. We conjecture that is could be because the variables are mixed categorical/binary/continuous.

Even though it is interesting to see how well methods performed as a function of the classification techniques, the most interesting thing is to analyze the methods of prediction of the BER and the methods of model selection, since this was the theme of the challenge.

We can roughly categorize the methods used as follows:

- **Nested cross-validation loops.** The entry who obtained the best average guess error (Cawley and Talbot, 2007), with average guess error: 0.0034, used a rather sophisticated cross-validation scheme. The hyperparameters were adjusted with a "virtual leave-one-out" cross-validation (VLOO). For regularized least-square kernel classifiers (LS-SVMs, kernel ridge regression, RLSC, Gaussian processes), it is possible to compute the leave-one-out error without training several classifiers (each time leaving one example out). Only one training with the entire training set and some inexpensive additional calculations are required. Gavin Cawley explored various loss-functions for the VLOO method using LS-SVMs for classification. He selected the best loss function with an outer loop of cross-validation (drawing 100 random 90%training-10%validation splits; we call this 100CV). After selecting his model, he re-estimated performance by 100CV using fresh data splits. The entrant who obtained the second best average guess error (Reunanen, 2007a), average guess error: 0.0048) performed a similar type of cross-validation called "cross-indexing", which uses nested cross-validation loops. The BER guess was obtained by the cross-validation performance in the outer loop. Nested cross-validation loops can be expensive computationally, however, in the case of the use of VLOO, the computational increase is minimal. We note however that VLOO is not available for all methods. Approximate formulas are available for SVMs (as used by Olivier Chapelle, average guess error: 0.0137) and neural networks.

- **Plain cross-validation.** Many participants used plain cross-validation, with a preference for 10-fold cross-validation. They chose their hyperparameters on the basis of the smallest cross-validated BER. The same cross-validated BER (CV BER) was used to guess the test BER, hoping that the bias introduced by using only 90% of the training data would be compensated by the bias introduced by selecting the model having smallest CV BER. This strategy seems to have been reasonable since the challenge winner (Lutz, 2006), average guess error: 0.0059) used it. He won because of his good BER performance. The second best entrant (Cawley and Talbot, 2007) had better BER guesses (average guess error: 0.0034). A few entrants took care of balancing the fraction of positive and negative examples in the data splits to reflect the proportions found in the entire training set – stratified cross-validation (Dahinden, 2010, this volume).

- **Other methods.** A few entrants performed model selection or estimated the performance on future data using training data, without reserving validation data or performing cross-validation, which is possible for regularized and Bayesian methods not prone to overfitting. This was used as a model selection strategy for the naïve Bayes classifier (Boullé, 2007a). Radford Neal for his Bayesian Neural Network predicts the error rate using training data only, but this was not one of the best performing methods (average guess error: 0.0122). Other methods include the use of performance bounds for SVM model selection like the Radius Margin bound (Olivier Chapelle). Bagging methods (including Random Forests) use bootstrap resampling. The final model makes decisions according to a vote of the models trained on the various bootstrap samples. The error rate of the model can be estimated using the "out-of-bag" samples. This method was used by Nicolai Meinshausen (average guess error: 0.0098). The least reliable method was to use the validation set of the challenge to predict the test set performance. The best ranking participants did not use this method.

1.5.3. Results of the Model Selection Game

Using models from the provided toolkit (CLOP, see Appendix), the best model selection game participants (Escalante et al., 2009; Reunanen, 2007b) closely matched the performances of the best entrants in the AL track using their own methods and considerably outperformed the baseline performances provided by the organizers using CLOP models. This validates their model selection techniques, which use efficient search algorithms and cross-validation to evaluate models.

All model selection methods rely on two basic elements: (1) a scoring function to evaluate the models, and (2) a search algorithm to explore the space of all possible models. In the two last challenges we organized (Guyon et al., 2005; Guyon et al., 2006b), the most successful scoring functions were based on cross-validation; participants relying on the training set error (eventually corrected by some complexity penalty terms) or on the validation set error overfitted the training data. Conversely, cross-validation users could afford searching the model space quite intensively without apparently incurring overfitting problems. Hence, the winners singled themselves out by effectively searching model space. This may be achieved either by brute force grid search using a computer cluster, or by some more refined search methods using a variety of algorithmic advances or simple heuristics. We briefly describe a few.

The winner of the game, Juha Reunanen, proposed a new variant of cross-validation called cross-indexing, which increases the accuracy of performance prediction in nested cross-validation loops (Reunanen, 2007b). Closely matching the performances of the winner, Hugo Jair Escalante used a search technique biologically inspired called "particule swarm model selection" (Escalante et al., 2009). In this method, each candidate model is represented as a particle in the solution space; and by using a population of particles, as well as a fitness function, it emulate the behavior of biological societies (swarm), which objective is to obtain common goals for the entire population. Examples of this behavior on biological populations are bird flocking and fish schooling. Also noteworthy is the method of Gavin Cawley (Cawley and Talbot, 2007a) who won the "performance prediction challenge" and whose results have not been outperformed in the game. He proposed the use of a Bayesian regularization at the second level of inference, adding a regularization term to the model selection criterion corresponding to a prior over the hyper-parameter values, where the additional regularization parameters are integrated out analytically. Finally, new promising methods of multi-level optimization (Bennett et al., 2006; Kunapuli et al., 2010, this volume) were proposed at the NIPS workshop where the results of the game were discussed, but must be optimized before they can be applied to sizeable datasets like the ones of the challenge.

The problem of selecting an optimum K in K-fold cross-validation has not been addressed. K=10 seems to be the default value everyone uses.

1.5.4. Results of the ALvsPK challenge

In the Agnostic Learning *vs.* Prior Knowledge challenge (ALvsPK), the final ranking of submissions was also based on the balanced error rate (BER) on the test set (the average of the error rate for the positive class and the error rate for the negative class). The Area Under the *ROC* Curve (AUC) was also computed, but not used for scoring. People scoring well with the BER generally perform also well with the AUC, but the opposite is not necessarily true since it is difficult to learn the correct bias when the two classes are unbalanced with respect of number of examples. The top ranking participants did well with respect to both metrics, but ranked in a slightly different order. To obtain the overall ranking we averaged the ranks of participants in each track after normalizing by the number of entries. The number of submissions was unlimited, but only the five last "complete" submissions for each entrant in either

Table 1.2: PK better than AL comparison results

	ADA	GINA	HIVA	NOVA	SYLVA
Min PK BER	0.170	**0.019**	**0.264**	**0.037**	**0.004**
Min AL BER	**0.166**	0.033	0.271	0.046	0.006
Median PK BER	**0.189**	**0.025**	0.310	**0.047**	**0.008**
Median AL BER	0.195	0.066	**0.306**	0.081	0.015
Pval ranksum test	**$5\ 10^{-8}$**	**$3\ 10^{-18}$**	0.25	**$8\ 10^{-6}$**	**10^{-18}**
Jorge Sueiras				–	
Juha Reunanen (Reunanen, 2007a)		+			+
Marc Boullé (Boullé, 2007b)	+	+		–	–
Roman Lutz (Lutz, 2006)					+
Vladimir Nikulin (Nikulin, 2007)	–	+			+
Vojtech Franc	+	+			
CWW		–	–		
Reference (gcc) (Cawley and Talbot, 2007b)	+	+	–		
Pvalue sign test	0.31	0.19	0.25	0.25	0.31

track were included in the final ranking. For the first few weeks of the challenge, the top of the rankings were largely dominated by agnostic track (AL) submissions. However, the learning curves for the agnostic learning and prior knowledge tracks eventually crossed for all datasets, except for ADA. After approximately 150 days the PK performance asymptote was reached. The asymptotic performances are reported at the top of Table 1.2. In contrast, in the IJCNN-06 performance prediction challenge, using the same data as the AL track, the competitors attained almost their best performance within about 60 days and kept improving only slightly afterward.

Figure 1.1, shows the distribution of the test BER for all entries. There were approximately 60% more submissions for the AL track than in the PK track. This indicates that the "prior knowledge" track was harder to enter. However, **the participants who did enter the PK track performed significantly better on average than those who entered the AL track**, on all datasets except for HIVA. To quantify this observation we ran a Wilcoxon rank sum test on the difference between the median values of the two tracks (Table 1.2). We also performed paired comparisons for entrants who entered both tracks, using their last 5 submissions. In Table 1.2, a "+" indicates that the entrant performed best in the PK track and a "–" indicates the opposite. We see that **the entrants who entered both tracks did not always succeed in obtaining better results in the PK track**. The *p*-values of the sign test do not reveal a significant dominance of PK over AL or vice versa in that respect (all are between 0.25 and 0.5). However, for HIVA and NOVA the participants who entered both tracks failed to get better results in the PK track. We conclude that, while on average PK seems to win over AL, success is uneven and depends both on the domain and on the individuals' expertise.

Agnostic learning methods

The winner of the "agnostic learning" track is Roman Lutz, who also won the Performance Prediction Challenge (IJCNN06) (Lutz, 2006), using boosting techniques. Gavin Cawley from the organization team made a reference entry (not counting towards the competition) using LSSVMs, which slightly outperforms that of Lutz. The improvements he made can partly be attributed to the introduction of an ARD kernel, which automatically down-weighs the least relevant features and to a Bayesian regularization at the second level of inference (Cawley and

Talbot, 2007b,a). The second best entrant is the Intel group, also using boosting methods (Tuv et al., 2009). The next best ranking entrants include Juha Reunanen and Hugo Jair Escalante, who have both been using CLOP models provided by the organizers and have proposed innovative search strategies for model selection: Escalante is using a biologically inspired particle swarm technique (Escalante et al., 2007, 2009) and Reunanen a cross-indexing method to make cross-validation more computationally efficient (Reunanen, 2007a,b). Other top ranking participants in the AL track include Vladimir Nikulin (Nikulin, 2007) and Jörg Wichard (Wichard, 2007) who both experimented with several ensemble methods, Erinija Pranckeviciene (Pranckeviciene et al., 2007; Pranckeviciene and Somorjai, 2010, this volume) who performed a study of linear programming SVM methods, and Marc Boullé who introduced a new data grid method (Boullé, 2007a, 2010, this volume). Mehreen Saeed (Saeed, 2010, this volume) achieved the best result on NOVA with a hybrid approach using mixture models and neural networks. Eugene Tuv In the following sections, we look into more details at the methods employed in the "prior knowledge" track to outperform the results of the "agnostic track".

Agnostic learning *vs.* prior knowledge: analysis per dataset

ADA: THE MARKETING APPLICATION

The task of ADA is to discover high revenue people from census data, presented in the form of a two-class classification problem. The raw data from the census bureau is known as the Adult database in the UCI machine-learning repository (Kohavi and Becker, 1994). The 14 original attributes (features) represent age, workclass, education, marital status, occupation, native country, etc. and include continuous, binary and categorical features. The PK track had access to the original features and their descriptions. The AL track had access to a preprocessed numeric representation of the features, with a simple disjunctive coding of categorical variables, but the identity of the features was not revealed. We expected that the participants of the AL *vs.* PK challenge could gain in performance by optimizing the coding of the input features. Strategies adopted by the participants included using a thermometer code for ordinal variables (Gavin Cawley) and optimally grouping values for categorical variables (Marc Boullé). Boullé also optimally discretized continuous variables, which make them suitable for a naïve Bayes classifier (Boullé, 2007a). However, the advantage of using prior knowledge for ADA was marginal. The overall winner on ADA is in the agnostic track (Roman Lutz), and the entrants who entered both tracks and performed better using prior knowledge do not have results statistically significantly better. We conclude that optimally coding the variables may not be so crucial and that good performance can be obtained with a simple coding and a state-of-the-art classifier.

GINA: THE HANDWRITING RECOGNITION APPLICATION

The task of GINA is handwritten digit recognition, the raw data is known as the MNIST dataset (LeCun and Cortes, 1998). For the "agnostic learning" track we chose the problem of separating two-digit odd numbers from two-digit even numbers. Only the unit digit is informative for this task, therefore at least 1/2 of the features are distracters. Additionally, the pixels that are almost always blank were removed and the pixel order was randomized to hide the meaning of the features. For the "prior knowledge" track, only the informative digit was provided in the original pixel map representation. In the PK track the identities of the digits (0 to 9) were provided for training, in addition to the binary target values (odd *vs.* even number). Since the prior knowledge track data consists of pixel maps, we expected the participants in perform image pre-processing steps such as noise filtering, smoothing, de-skewing, and feature extraction (points, loops, corners) and/or use kernels or architectures exploiting geometric invariance by small translation, rotation, and other affine transformations, which have proved to work well on this dataset (LeCun and Cortes, 1998). Yet, the participants in the PK track adopted very simple strategies, not involving a lot of domain knowledge. Some just relied on the performance boost obtained by the removal of the distracter features (Vladimir Nikulin, Marc Boullé, Juha Reunanen). Others exploited the knowledge of the individual class labels and created multi-class of hierarchical classifiers (Vojtech Franc, Gavin Cawley). Only the reference entries of Gavin Cawley (which obtained the best BER of 0.0192) included domain knowledge by using RBF kernels with tunable receptive fields to smooth the pixel maps (Cawley and Talbot, 2007a). In the future, it would be interesting to assess the methods of Simard *et al* (Simard et al., 2003) on this data to see whether further improvements are obtained by exploiting geometrical invariances. The agnostic track data was significantly harder to analyze because of the hidden class heterogeneity and the presence of feature distracters. The best GINA final entry was therefore on the PK track and all four ranked entrants who entered both tracks obtained better results in the PK track. Further, the differences in performance are all statistically significant.

HIVA: THE DRUG DISCOVERY APPLICATION

The task of HIVA is to predict which compounds are active against the AIDS HIV infection. The original data from the NCI (Collins, 1999) has 3 classes (active, moderately active, and inactive). We brought it back to a two-class classification problem (active & moderately active *vs.* inactive), but we provided the original labels for the "prior knowledge" track. The compounds are represented by their 3d molecular structure for the "prior knowledge" track (in SD format). For the "agnostic track" we represented the input data as a vector of 2000 sparse binary variables. The variables represent properties of the molecule inferred from its structure by the ChemTK software package (version 4.1.1, Sage Informatics LLC). The problem is therefore to relate structure to activity (a QSAR — quantitative structure-activity relationship problem) to screen new compounds before actually testing them (a HTS — high-throughput screening problem). Note that in such applications the BER is not the best metric to assess performance since the real goal is to identify correctly the compounds most likely to be effective (belonging to the positive class). We resorted to using the BER to make comparisons easier across datasets. The raw data was not supplied in a convenient feature representation, which made it impossible to enter the PK track using agnostic learning methods, using off-the-shelf machine learning packages. The winner in HIVA (Chloé-Agathe Azencott of the Pierre Baldi Laboratory at UCI) is a specialist in this kind of dataset, on which she is working towards her PhD (Azencott et al., 2007; Azencott and Baldi, 2010, this volume). She devised her own set of low level features, yielding a "molecular fingerprint" representation, which outperformed the ChemTK features used on the agnostic track. Her winning entry has a test BER of 0.2693, which is significantly better than the test BER of the best ranked AL entry of 0.2827 (standard error 0.0068). The results on HIVA are quite interesting because most agnostic learning entrants did not even attempt to enter the prior knowledge track and the entrants that did submit models for both tracks failed to obtain better results in the PK track. One of them working in an institute of pharmacology reported that too much domain knowledge is sometimes detrimental; experts in his institute advised against using molecular fingerprints, which ended up as the winning technique.

NOVA: THE TEXT CLASSIFICATION APPLICATION

The data of NOVA come from the 20-Newsgroup dataset (Mitchell, 1999). Each text to classify represents a message that was posted to one or several USENET newsgroups. The raw data is provided in the form of text files for the "prior knowledge" track. The preprocessed data for the "agnostic learning" track is a sparse binary representation using a bag-of-words with a vocabulary of approximately 17000 words (the features are simply frequencies of words in text). The original task is a 20-class classification problem but we grouped the classes into two categories (politics and religion *vs.* others) to make it a two-class problem. The original class labels were available for training in the PK track but not in the AL track. As the raw data consist of texts of variable length it was not possible to enter the PK track for NOVA without performing a significant pre-processing. All PK entrants in the NOVA track used a bag-of-words representation, similar to the one provided in the agnostic track. Standard tricks were used, including stemming. Gavin Cawley used the additional idea of correcting the emails with an automated spell checker. No entrant who entered both tracks outperformed their AL entry with their PK entry in their last ranked entries, including the winner! This is interesting because the best PK entries made throughout the challenge significantly outperform the best AL entries (BER difference of 0.0089 for an standard error of 0.0018), see also Figure 1.1. Hence in this case, **the PK entrants overfitted and were unable to select among their PK entries those, which would perform best on test data**. This is not so surprising because the validation set on NOVA is quite small (175 examples). Even though the bag-of-words representation is known to be state-of-the-art for this kind of applications, it would be interesting to compare it with more sophisticated representations. To our knowledge, the best results on the 20 Newsgroup data were obtained by the method of distributional clustering by Ron Bekkerman (Bekkerman et al., 2003).

SYLVA: THE ECOLOGY APPLICATION

The task of SYLVA is to classify forest cover types. The forest cover type for 30×30 meter cells was obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data (Blackard and Dean, 1998). We converted this into a two-class classification problem (classifying Ponderosa pine *vs.* everything else). The input vector for the "agnostic learning" track consists of 216 input variables. Each pattern is composed of 4 records: 2 true records matching the target and 2 records picked at random. Thus 1/2 of the features are distracters. The "prior knowledge" track data is identical to the "agnostic learning" track data, except that the distracters are removed and the meaning of the features is revealed. For that track, the identifiers in the original forest cover dataset are revealed for the training set. As the raw data was already in a feature vector representation, this task was essentially testing the ability of the participants in the AL track to perform well in the presence of distracter features. The PK track winner (Roman Lutz) in his Doubleboost algorithm exploited the fact that each pattern was made of two records of the same pattern to train a classifier with twice as many training examples. Specifically, a new dataset was constructed by putting the second half of the data (variables 55 to 108) below the first half (variables 1 to 54). The new dataset is of dimension $2n$ times 54 (instead of n times 108). This new dataset is used for fitting the base learner (tree) of his boosting algorithm. The output of the base learner is averaged over the two records belonging to the same pattern. This strategy can be related to the neural network architectures using "shared weights", whereby at training time, the weights trained on parts of the pattern having similar properties are constrained to be identical (LeCun and Cortes, 1998). This reduced the number of free parameters of the classifier.

1.6. Conclusions

This paper presented the results of three competitions organized around the same datasets. The challenge series was very successful in attracting a large number of participants who delivered many interesting ways of approaching performance predictions.

We observed that rather unsophisticated methods (e.g. simple 10-fold cross validation) did well to predict performance. Nested cross-validation loops are advisable to gain extra prediction accuracy. They come at little extra computational expense, if the inner loop uses virtual leave-one-out. Successful model selection techniques include cross-validation and regularization methods. Ensemble and Bayesian methods provide an efficient alternative to model selection by constructing committees of classifiers. A number of sophisticated methods with appealing theoretical motivations were proposed for the model selection special session, but the authors did not compete in the challenge. We believe there is still a gap to be filled between theory and practice in this game of performance prediction and model selection.

The Agnostic Learning *vs.* Prior Knowledge challenge (ALvsPK) compared the "agnostic learning" (AL) approach putting all the effort on the classifier and the "prior knowledge" (PK) approach capitalizing on human domain knowledge. For the first few months of the challenge, the participants of the AL track led over the PK track, showing that the development of good AL classifiers is considerably faster. As of March 1st 2007, PK was leading over AL on four out of five datasets. We extended the challenge five more months, but few significant improvements were made during that time period. On datasets not requiring real expert domain knowledge (ADA, GINA, SYLVA), the participants entering both track obtained better results in the PK track, using a special-purpose coding of the inputs and/or the outputs, exploiting the knowledge of which features were uninformative, and using "shared weights" for redundant features. On the datasets requiring most real expert domain knowledge (HIVA and NOVA), several entrants failed to capitalize on prior knowledge. For both HIVA and NOVA, the winning data representation consisted of a high-dimensional vector of low level features ("molecular fingerprints" and "bag-of-words"). From the analysis of this challenge, we conclude that agnostic learning methods are very powerful. They quickly yield (in 40 to 60 days) a level of performance close to the best achievable performance. General-purpose techniques for exploiting prior knowledge in the encoding of inputs or outputs or the design of the learning machine architecture (*e.g.* via shared weights) may provide an additional performance boost, but exploiting real domain knowledge is both difficult and time consuming. This fact seems to be a recurrent theme in machine learning publications and further confirmation is provided by the results of our challenge.

We incorporated the best identified methods in our challenge toolkit, CLOP `http://clopinet.com/CLOP`. The challenge web site remains open for post-challenge submissions at `http://www.agnostic.inf.ethz.ch/`, where supplementary analyzes and complete result tables are also made available.

Acknowledgments

We are very thankful to the institutions that originally provided the data. The organization of this challenge was a team effort to which many have participated. We are particularly grateful to Olivier Guyon (MisterP.net) our webmaster. Prof. Joachim Buhmann (ETH Zurich) who provided computer resources and all the advisors, beta-testers and sponsors are gratefully acknowledged (see `http://www.agnostic.inf.ethz.ch/credits.php` for a full list). The Challenge Learning Object Package (CLOP) is based on code to which many people have contributed Weston et al. (2005); Saffari and Guyon (2006). This project has been supported by the Pascal network of excellence funded by the European Commission and the National Science Foundation under Grants N0. ECCS-0424142 and N0. ECCS-0736687. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Amir Saffari acknowledges the support of the Austrian FFG project Outlier (820923) under the FIT-IT program.

References

C.-A. Azencott and P. Baldi. Virtual high-throughput screening with two-dimensional kernels. In I. Guyon, et al., editor, *Hands on Pattern Recognition*. Microtome, 2010, this volume.

C. A. Azencott, A. Ksikes, S. J. Swamidass, J. H. Chen, L. Ralaivola, and P. Baldi. One- to four-dimensional kernels for virtual screening and the prediction of physical, chemical, and biological properties. *J. Chem. Inf. Model.*, 2007. Available at http://pubs3.acs.org/acs/journals/doilookup?in_doi=10.1021/ci600397p.

R. Bekkerman, R. El-Yaniv, N. Tishby, and Y. Winter. Distributional word clusters vs words for text categorization. *J. Machine Learning Research*, 3, 2003. Available at citeseer.ist.psu.edu/article/bekkerman02distributional.html. Code available at http://www.cs.technion.ac.il/~ronb/.

K. Bennett, J. Hu, X. Ji, G. Kunapuli, and J.-S. Pang. Model selection via bilevel optimization. In *Proc. IJCNN06*, pages 3588–3505, Vancouver, Canada, July 2006. INNS/IEEE. Available at http://clopinet.com/isabelle/Projects/modelselect/Papers/Bennett_paper_IJCNN06.pdf.

J. A. Blackard and D. J. Dean. Forest cover type, 1998. Available at http://kdd.ics.uci.edu/databases/covertype/covertype.html.

M. Boullé. Compression-based averaging of Selective Naïve Bayes classifiers. In I. Guyon and A. Saffari, editors, *JMLR, Special topic on model selection*, volume 8, pages 1659–1685, Jul 2007a. URL http://www.jmlr.org/papers/volume8/boulle07a/boulle07a.pdf.

M. Boullé. Report on preliminary experiments with data grid models in the agnostic learning vs. prior knowledge challenge. In *Proc. IJCNN07*, Orlando, Florida, Aug 2007b. INNS/IEEE.

M. Boullé. Data grid models for preparation and modeling in supervised learning. In I. Guyon, et al., editor, *Hands on Pattern Recognition*. Microtome, 2010, this volume.

G. Cawley and N. Talbot. Preventing over-fitting during model selection via Bayesian regularisation of the hyper-parameters. In I. Guyon and A. Saffari, editors, *JMLR, Special topic on model selection*, volume 8, pages 841–861, Apr 2007a. URL http://www.jmlr.org/papers/volume8/cawley07a/cawley07a.pdf.

G. C. Cawley and N. L. C. Talbot. Agnostic learning versus prior knowledge in the design of kernel machines. In *Proc. IJCNN07*, Orlando, Florida, Aug 2007b. INNS/IEEE.

G. C. Cawley and N. L. C. Talbot. Preventing over-fitting during model selection using Bayesian regularisation. In I. Guyon and A. Saffari, editors, *JMLR, Special topic on model selection*, volume 8, pages 841–861, April 2007. URL http://jmlr.csail.mit.edu/papers/volume8/cawley07a/cawley07a.pdf.

J. M. Collins, Associate Director. The DTP AIDS antiviral screen program, 1999. Available at http://dtp.nci.nih.gov/docs/aids/aids_data.html.

C. Dahinden. An improved Random Forests approach with application to the performance prediction challenge datasets. In I. Guyon, et al., editor, *Hands on Pattern Recognition*. Microtome, 2010, this volume.

H. J. Escalante, M. Montes, and L. E. Sucar. PSMS for neural networks: Results on the IJCNN 2007 agnostic *vs.* prior knowledge challenge. In *Proc. IJCNN07*, Orlando, Florida, Aug 2007. INNS/IEEE.

H. J. Escalante, M. Montes, and L. E. Sucar. Particle swarm model selection. In I. Guyon and A. Saffari, editors, *JMLR, Special topic on model selection*, volume 10, pages 405–440, Feb 2009. URL http://www.jmlr.org/papers/volume10/escalante09a/escalante09a.pdf.

I. Guyon. Datasets for the agnostic learning *vs.* prior knowledge competition. Technical report, Clopinet, 2005. Available at http://clopinet.com/isabelle/Projects/agnostic/Dataset.pdf.

I. Guyon, S. Gunn, A. Ben-Hur, and G. Dror. Result analysis of the nips 2003 feature selection challenge. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 545–552. MIT Press, Cambridge, MA, 2005. Available at http://books.nips.cc/papers/files/nips17/NIPS2004_0194.pdf.

I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, Editors. *Feature Extraction, Foundations and Applications*. Studies in Fuzziness and Soft Computing. Physica-Verlag, Springer, 2006a.

I. Guyon, A. Saffari, G. Dror, and J. Buhmann. Performance prediction challenge. In *IEEE/INNS conference IJCNN 2006*, Vancouver, July 16-21 2006b.

I. Guyon, A. Saffari, G. Dror, and G. Cawley. Agnostic *vs.* prior knowledge challenge. In *Proc. IJCNN07*, Orlando, Florida, Aug 2007. INNS/IEEE.

I. Guyon, A. Saffari, G. Dror, and G. Cawley. Analysis of the IJCNN 2007 agnostic learning vs. prior knowledge challenge. *Neural Networks*, 21(2-3):544–550, 2008.

R. Kohavi and B. Becker. The Adult database, 1994. Available at ftp://ftp.ics.uci.edu/pub/machine-learning-databases/adult/.

G. Kunapuli, J.-S. Pang, and K. Bennett. Bilevel cross-validation-based model selection. In I. Guyon, et al., editor, *Hands on Pattern Recognition*. Microtome, 2010, this volume.

Y. LeCun and C. Cortes. The MNIST database of handwritten digits, 1998. Available at http://yann.lecun.com/exdb/mnist/.

R. W. Lutz. Logitboost with trees applied to the WCCI 2006 performance prediction challenge datasets. In *Proc. IJCNN06*, pages 2966–2969, Vancouver, Canada, July 2006. INNS/IEEE.

T. Mitchell. The 20 Newsgroup dataset, 1999. Available at http://kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html.

V. Nikulin. Non-voting classification with random sets and boosting. In *Proc. IJCNN07 Data Representation Discovery workshop*, Orlando, Florida, Aug 2007.

E. Pranckeviciene and R. Somorjai. Liknon feature selection: Behind the scenes. In I. Guyon, et al., editor, *Hands on Pattern Recognition*. Microtome, 2010, this volume.

E. Pranckeviciene, R. Somorjai, and M. N. Tran. Feature/model selection by the linear programming SVM combined with state-of-art classifiers: What can we learn about the data. In *Proc. IJCNN07*, Orlando, Florida, Aug 2007. INNS/IEEE.

J. Reunanen. Model selection and assessment using cross-indexing. In *Proc. IJCNN07*, Orlando, Florida, Aug 2007a. INNS/IEEE.

J. Reunanen. Resubstitution error is useful for guiding feature selection. *Unpublished techreport*, 2007b.

M. Saeed. Hybrid learning using mixture models and artificial neural networks. In I. Guyon, et al., editor, *Hands on Pattern Recognition*. Microtome, 2010, this volume.

A. Saffari and I. Guyon. Quick start guide for CLOP. Technical report, Graz University of Technology and Clopinet, May 2006. Available at http://ymer.org/research/files/clop/QuickStartV1.0.pdf.

P. Simard, D. Steinkraus, and J. Platt. Best practice for convolutional neural networks applied to visual document analysis. In *International Conference on Document Analysis and Recogntion (ICDAR)*, pages 958–962, Los Alamitos, 2003. IEEE Computer Society.

E. Tuv, A. Borisov, G. Runger, and K. Torkkola. Feature selection with ensembles, artificial variables, and redundancy elimination. In I. Guyon and A. Saffari, editors, *JMLR, Special topic on model selection*, volume 10, pages 1341–1366, Jul 2009. URL http://www.jmlr.org/papers/volume10/tuv09a/tuv09a.pdf.

J. Weston, A. Elisseeff, G. Bakir, and F. Sinz. The Spider machine learning toolbox. 2005. Available at http://www.kyb.tuebingen.mpg.de/bs/people/spider/.

J. Wichard. Agnostic learning with ensembles of classifiers. In *Proc. IJCNN07*, Orlando, Florida, Aug 2007. INNS/IEEE.

Appendix: The Challenge Learning Object Package (CLOP)

The CLOP package can be downloaded from the web-site http://clopinet.com/CLOP. The Spider package on top of which CLOP is built, uses Matlab® objects (The MathWorks, http://www.mathworks.com/). Two simple abstractions are used:

- **data**: Data objects include two members `X` and `Y`, `X` being the input matrix (patterns in lines and features in columns), `Y` being the target matrix (i.e. one column of ± 1 for binary classification problems).
- **algorithms**: Algorithm objects representing learning machines (e.g. neural networks, kernel methods, decision trees) or preprocessors (for feature construction, data normalization or feature selection). They are constructed from a set of hyper-parameters and have at least two methods: train and test. The train method adjusts the parameters of the model. The test method processes data using a trained model.

For example, you can construct a data object `D`:

```
> D = data(X, Y);
```

The resulting object has 2 members: `D.X` and `D.Y`. Models are derived from the class algorithm. They are constructed using a set of hyperparameters provided as a cell array of strings, for instance:

```
> hyperparam = {'h1=val1', 'h2=val2'};
> model0 = algorithm(hyperparam);
```

In this way, hyperparameters can be provided in any order or omitted. Omitted hyperparameters take default values.

To find out about the default values and allowed hyperparameter range, one can use the "default" method:

```
> default(algorithm)
```

The constructed model `model0` can then be trained and tested:

```
> [Dout, model1] = train(model0, Dtrain);
> Dout = test(model1, Dtest);
```

`model1` is a model object identical to `model0`, except that its parameters (some data members) have been updated by training. Matlab uses the convention that the object of a method is passed as first argument as a means to identify which overloaded method to call. Hence, the "correct" `train` method for the class of `model0` will be called. Since Matlab passes all arguments by value, `model0` remains unchanged. By calling the trained and untrained model with the same name, the new model can overwrite the old one. Repeatedly calling the method "train" on the same model may have different effects depending on the model.

To save the model is very simple since Matlab objects know how to save themselves:

```
> save('filename', 'modelname');
```

This feature is very convenient to make results reproducible, particularly in the context of a challenge.

The Spider (with some CLOP extensions) provides ways of building more complex "compound" models from the basic algorithms with two abstractions:

- **chain**: A chain is a learning object (having a train and test method) constructed from an array of learning objects. Each array member takes the output of the previous member and feeds its outputs to the next member.
- **ensemble**: An ensemble is also a learning object constructed from an array of learning objects. The trained learning machine performs a weighted sum of the predictions of the array members.

A typical model chains modules for preprocessing, feature selection, classification, and postprocessing.

Until December 1st 2006, the challenge participants had the opportunity to participate in a model selection game using CLOP. For the purpose of the game, a valid *model* was defined as a combination of learning objects from a predefined list (type `whoisclop` at the MATLAB prompt to get a the full list of allowed CLOP learning objects; to check that a particular object is a valid CLOP object, type `isclop(object)`).

A typical model may include some of the following modules: preprocessing, feature selection, classification, and postprocessing. Table 3 shows a list of the modules provided.

Table 1.3: CLOP modules provided for the model selection game.

Object name	**Description**
Preprocessing	
`standardize`	Subtract feature mean and divide by stdev.
`normalize`	Divide patterns by their Euclidean norm.
`shift_n_scale`	Offset and scale all values.
`pc_extract`	Construct features from principal components.
`subsample`	Take a subsample of training patterns.
Feature selection	
`s2n`	Signal-to-noise ratio filter method.
`relief`	Relief filter method.
`gs`	Gram-Schmidt orthogonalization forward selection.
`rffs`	Random Forest feature selection.
`svcrfe`	SVC-based recursive feature elimination.
Classifier	
`kridge`	Kernel ridge regression
`naive`	naive Bayes classifier
`gentleboost`	Regularized boosting
`neural`	Two layer neural network
`rf`	Random Forest (ensemble of trees)
`svc`	Support vector classifier
Postprocessing	
`bias`	Post-fitting of the bias value.

Journal of Machine Learning Research 11(Jan):61–87, 2010 Submitted 11/09; Published 1/10

Chapter 2

Model Selection: Beyond the Bayesian/Frequentist Divide

Isabelle Guyon GUYON@CLOPINET.COM
ClopiNet
955 Creston Road
Berkeley, CA 94708, USA

Amir Saffari SAFFARI@ICG.TUGRAZ.AT
Institute for Computer Graphics and Vision
Graz University of Technology
Inffeldgasse 16
A-8010 Graz, Austria

Gideon Dror GIDEON@MTA.AC.IL
The Academic College of Tel-Aviv-Yaffo
2 Rabeinu Yerucham St., Jaffa
Tel-Aviv 61083, Israel

Gavin Cawley GCC@CMP.UEA.AC.UK
School of Computing Sciences
University of East Anglia
Norwich, NR4 7TJ, U.K.

Editor: Lawrence Saul

Abstract

The principle of parsimony also known as "Ockham's razor" has inspired many theories of model selection. Yet such theories, all making arguments in favor of parsimony, are based on very different premises and have developed distinct methodologies to derive algorithms. We have organized challenges and edited a special issue of JMLR and several conference proceedings around the theme of model selection. In this editorial, we revisit the problem of avoiding overfitting in light of the latest results. We note the remarkable convergence of theories as different as Bayesian theory, Minimum Description Length, bias/variance tradeoff, Structural Risk Minimization, and regularization, in some approaches. We also present new and interesting examples of the complementarity of theories leading to hybrid algorithms, neither frequentist, nor Bayesian, or perhaps both frequentist and Bayesian!

Keywords: model selection, ensemble methods, multilevel inference, multilevel optimization, performance prediction, bias-variance tradeoff, Bayesian priors, structural risk minimization, guaranteed risk minimization, over-fitting, regularization, minimum description length

2.1. Introduction

The problem of learning is often decomposed into the tasks of fitting parameters to some training data, and then selecting the best model using heuristic or principled methods, collectively referred to as *model selection* methods. Model selection methods range from simple yet powerful cross-validation based methods to the optimization of cost functions penalized for model complexity, derived from performance bounds or Bayesian priors.

This paper is not intended as a general review of the state-of-the-art in model selection nor a tutorial; instead it is a synthesis of the collection of papers that we have assembled. It also provides a unifying perspective on Bayesian and frequentist methodologies used in various model selection methods. We highlight a new trend in research on model selection that blends these approaches.

The reader is expected to have some basic knowledge of familiar learning machines (linear models, neural networks, tree classifiers and kernel methods) and elementary notions of learning theory (bias/variance tradeoff, model capacity or complexity, performance bounds). Novice readers are directed to the companion paper (Guyon, 2009), which reviews basic learning machines, common model selection techniques, and provides elements of learning theory.

When we started organizing workshops and competitions around the problem of model selection (of which this collection of papers is the product), both theoreticians and practitioners welcomed us with some scepticism; model selection being often viewed as somewhat "old hat". Some think that the problem is solved, others that it is not a problem at all! For Bayesian theoreticians, the problem of model selection is circumvented by averaging all models over the posterior distribution. For risk minimization theoreticians (called "frequentists" by the Bayesians) the problem is solved by minimizing performance bounds. For practitioners, the problem is solved using cross-validation. However, looking more closely, most theoretically grounded methods of solving or circumventing model selection have at least one hyper-parameter left somewhere, which ends up being optimized by cross-validation. Cross-validation seems to be the universally accepted ultimate remedy. But it has its dark sides: (a) there is no consensus on how to choose the fraction of examples reserved training and for validation; (b) the overall learning problem may be prone to over-fitting the cross-validation error (Cawley and Talbot, 2009). Therefore, from our point of view, the problem of optimally dividing the learning problem into multiple levels of inference and optimally allocating training data to these various levels remains unsolved, motivating our efforts. From the novel contributions we have gathered, we are pleased to see that researchers are going beyond the usual Bayesian/frequentist divide to provide new creative solutions to those problems: we see the emergence of multi-level optimization methods, which are both Bayesian and frequentist. How can that be? Read on!

After explaining in Section 2.2 our notational conventions, we briefly review a range of different Bayesian and frequentist approaches to model selection in Section 2.3, which we then unify in Section 2.4 under the framework of multi-level optimization. Section 2.5 then presents the advances made by the authors of papers that we have edited. In Section 2.6, we open a discussion on advanced topics and open problems. To facilitate reading, a glossary is appended; throughout the paper, words found in the glossary are indicated in boldface.

2.2. Notations and Conventions

In its broadest sense, *model selection* designates an ensemble of techniques used to select a model, that best explains some data or phenomena, or best predicts future data, observations or the consequences of actions. This broad definition encompasses both scientific and statistical modeling. In this paper, we address only the problem of statistical modeling and are mostly concerned with **supervised learning** from independently and identically distributed (*i.i.d.*) data. Extensions to **unsupervised learning** and non *i.i.d.* cases will be discussed in Section 2.6.

The goal of supervised learning is to predict a target variable $y \in \mathscr{Y}$, which may be continuous (regression) or categorical or binary (classification). The predictions are made using observations $\boldsymbol{x}$ from a domain $\mathscr{X}$, often a vectorial space of dimension n, the number of features. The data pairs $\{\boldsymbol{x}, y\}$ are independently and identically distributed according to an unknown (but fixed) distribution $P(\boldsymbol{x}, y)$. A number m of pairs drawn from that distribution are given, forming the training data $D = \{(\boldsymbol{x}_k, y_k), k = 1, ...m\}$. We will denote by $X = [x_{ki}]$, $k = 1, ...m, i = 1, ...n$, the matrix of dimensions (m, n) whose rows are the training patterns and whose columns are the features. Finally, we denote by $\boldsymbol{y}$ the column vector of dimensions $(m, 1)$ containing the target values y_k.

There are several formulations of the supervised learning problem:

- **Function approximation** (induction) methods seek a function f (called *model* or *learning machine*) belonging to a model class $\mathscr{F}$, which minimizes a specified risk functional (or maximizes a certain utility). The goal is to minimize an *expected risk* $R[f] = \int \mathscr{L}(f(\boldsymbol{x}), y)\, dP(\boldsymbol{x}, y)$, also called *generalization error*, where $\mathscr{L}(f(\boldsymbol{x}), y)$ is a loss function (often a negative log likelihood) measuring the discrepancy between $f(\boldsymbol{x})$ and y. Since $P(\boldsymbol{x}, y)$ is unknown, only estimates of $R[f]$ can be computed, which we call *evaluation functions* or *estimators*. Function approximation methods differ in the choice of evaluation function and optimization algorithm and include **risk minimization**, **PAC leaning**, **maximum likelihood optimization**, and **MAP learning**.

- **Bayesian and ensemble methods** make predictions according to model averages that are convex combinations of models $f \in \mathscr{F}$, that is, which belong to the convex closure of the model class $\mathscr{F}^*$. Such methods differ in the type of model averaging performed. **Bayesian learning** methods approximate $E_f(y|\boldsymbol{x}) = \int_{f \in \mathscr{F}} f(\boldsymbol{x})\, dP(f)$, an expectation taken over a class of models $\mathscr{F}$, using an unknown probability distribution $P(f)$ over the models. Starting from a "prior", our knowledge of this distribution is refined into a "posterior" when we see some data. **Bagging** ensemble methods approximate $E_D(f(\boldsymbol{x}, D))$, where $f(\boldsymbol{x}, D)$ is a function from the model class $\mathscr{F}$, trained with m examples and $E_D(\cdot)$ is the mathematical expectation over all training sets of size m. The key point in these methods is to generate a diverse set of functions, each providing a different perspective over the problem at hand, the ensemble thus forming a consensus view.

- **Transduction** methods make direct predictions of y given $\boldsymbol{x}$ and X, bypassing the modeling step. We do not address such methods in this paper.

The desired properties of the chosen predictor include: good generalization performance, fast training/prediction, and ease of interpretation of the predictions. Even though all of these aspects are important in practice, we will essentially focus on the first aspect: obtaining the best possible generalization performance. Some of the other aspects of model selection will be discussed in Section 2.6.

The parametrization of f differentiates the problem of *model selection* from the general machine learning problem. Instead of parameterizing f with one set of parameters, the model selection framework distinguishes between *parameters* and *hyper-parameters*. We adopt the simplified notation $f(\boldsymbol{x}; \boldsymbol{\alpha}, \boldsymbol{\theta})$ for a model of parameters $\boldsymbol{\alpha}$ and hyper-parameters $\boldsymbol{\theta}$. It should be understood that different models may be parameterized differently. Hence by $f(\boldsymbol{x}; \boldsymbol{\alpha\theta})$ we really mean $f(\boldsymbol{x}; \boldsymbol{\alpha}(\boldsymbol{\theta}), \boldsymbol{\theta})$ or $f_{\boldsymbol{\theta}}(\boldsymbol{x}; \boldsymbol{\alpha})$. For instance, for a linear model $f(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}$, $\boldsymbol{\alpha} = \boldsymbol{w}$; for a kernel method $f(\boldsymbol{x}, \boldsymbol{\alpha}) = \sum_k \alpha_k K(\boldsymbol{x}, \boldsymbol{x}_k)$, $\boldsymbol{\alpha} = [\alpha_k]$. The hyper-parameters may include indicators of presence or absence of features, choice of preprocessing methods,, choice of algorithm or model sub-class (e.g., linear models, neural networks, kernel methods, etc.), algorithm or model sub-class parameters (e.g., number of layers and units per layer in a neural network, maximum degree of a polynomial, bandwidth of a kernel), choice of post-processing, etc. We also refer to the parameters of the prior $P(f)$ in **Bayesian/MAP learning** and the parameters of the **regularizer** $\Omega[f]$ in **risk minimization** as hyper-parameters even if the resulting predictor is not an explicit function of those parameters, because they are used in the process of learning.

In what follows, we relate the problem of model selection to that of *hyper-parameter selection*, taken in is broadest sense and encompassing all the cases mentioned above.

We refer to the adjustment of the model parameters $\boldsymbol{\alpha}$ as the *first level of inference*. When data are split in several subsets for the purpose of training and evaluating models, we call m_{tr} the number of *training examples* used to adjust $\boldsymbol{\alpha}$. If the hyper-parameters $\boldsymbol{\theta}$ are adjusted from a subset of data of size m_{va}, we call the examples used to adjust them at this *second level of inference* the *"validation sample"*. Finally we call m_{te} the number of test examples used to evaluate the final model. The corresponding empirical estimates of the *expected risk* $R[f]$, denoted $R_{tr}[f]$, $R_{va}[f]$, and $R_{te}[f]$, will be called respectively *training error*, *validation error*, and *test error*.

2.3. The Many Faces of Model Selection

In this section, we track *model selection* from various angles to finally reduce it to the unified view of *multilevel inference*.

2.3.1. Is Model Selection "Really" a Problem?

It is legitimate to first question whether the distinction between parameters and hyper-parameters is relevant. Splitting the learning problem into two levels of inference may be convenient for conducting experiments. For example, combinations of preprocessing, feature selection, and post-processing are easily performed by fixing $\boldsymbol{\theta}$ and training $\boldsymbol{\alpha}$ with off-the-shelf programs. But, the distinction between parameters and hyper-parameters is more fundamental. For instance, in the model class of kernel methods $f(\boldsymbol{x}) = \sum_k \boldsymbol{\alpha}_k K(\boldsymbol{x}, \boldsymbol{x}_k; \boldsymbol{\theta})$, why couldn't we treat both $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$ as regular parameters?

One common argument is that, for fixed values of $\boldsymbol{\theta}$, the problem of learning $\boldsymbol{\alpha}$ can be formulated as a convex optimization problem, with a single unique solution, for which powerful mathematical programming packages are available, while the overall optimization of $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$ in non-convex. Another compelling argument is that, splitting the learning problem into several levels might also benefit to the performance of the learning machine by "alleviating" (but not eliminating) the problem of **over-fitting**. Consider for example the Gaussian redial basis function kernel $K(\boldsymbol{x}, \boldsymbol{x}_k; \boldsymbol{\theta}) = \exp(-\|\boldsymbol{x} - \boldsymbol{x}_k\|^2/\theta^2)$. The function $f(\boldsymbol{x}) = \sum_{k=1}^{m} \alpha_k K(\boldsymbol{x}, \boldsymbol{x}_k; \boldsymbol{\theta})$ is a universal approximator if θ is let to vary and if the sum runs over the training examples. If both $\boldsymbol{\alpha}$ and θ are optimized simultaneously, solutions with a small value of θ^2 might be picked, having zero training error but possibly very poor generalization performance. The model class $\mathscr{F}$ to which f belongs has infinite capacity $C(\mathscr{F})$. In contrast, for a fixed value of the hyper-parameter θ^o, the model $f(\boldsymbol{x}) = \sum_{k=1}^{m} \boldsymbol{\alpha}_k K(\boldsymbol{x}, \boldsymbol{x}_k; \theta^o)$ is linear in its parameters $\boldsymbol{\alpha}_k$ and has a finite capacity, bounded by m. In addition, the capacity of $f(\boldsymbol{x}) = \sum_{k=1}^{m} \boldsymbol{\alpha}_k^o K(\boldsymbol{x}, \boldsymbol{x}_k^o; \theta)$ of parameter θ for fixed values $\boldsymbol{\alpha}_k^o$ and x_k^o is very low (to see that, note that very few examples can be learned without error by just varying the kernel width, given fixed vectors x_k^o and fixed parameters $\boldsymbol{\alpha}_k^o$). Hence, *using multiple levels of inference may reduce over-fitting, while still searching for solutions in a model class of universal approximators*.

This last idea has been taken one step further in the method of *structural risk minimization* (Vapnik, 1979), by introducing new hyper-parameters in learning problems, which initially did not have any. Consider for instance the class of linear models $f(\boldsymbol{x}) = \sum_{i=1}^{n} w_i x_i$. It is possible to introduce hyper-parameters by imposing a structure in parameter space. A classical example is the structure $\|\boldsymbol{w}\|^2 \leq A$, where $\|\boldsymbol{w}\|$ denotes the Euclidean norm and A is a positive hyper-parameter. For increasing values of A the space of parameters is organized in nested subsets. Vapnik (1998) proves for Support Vector Machines (SVM) and Bartlett (1997) for neural networks that tighter performance bounds are obtained by increasing A. The newly introduced parameter allows us to monitor the **bias/variance tradeoff**. Using a Lagrange multiplier, the problem may be replaced by that of minimizing a regularized risk functional $R_{reg} = R_{tr} + \gamma \|\boldsymbol{w}\|^2$, $\gamma > 0$, where the training loss function is the so-called "hinge loss" (see *e.g.,* Hastie et al., 2000). The same regularizer $\|\boldsymbol{w}\|^2$ is used in ridge regression (Hoerl, 1962), "weight decay" neural networks (Werbos, 1988), regularized radial-basis function networks (Poggio and Girosi, 1990), Gaussian processes (MacKay, 1992), together with the square loss function. Instead of the Euclidean norm or 2-norm, the 1-norm regularizer $\|\boldsymbol{w}\|_1 = \sum_i |w_i|$ is used in LASSO (Tibshirani, 1994) and 1-norm versions of SVMs (see *e.g.,* Zhu et al., 2003), logistic regression (Friedman et al., 2009), and Boosting (Rosset et al., 2004). Weston et al. (2003) have proposed a 0-norm regularizer $\|\boldsymbol{w}\|_0 = \sum_i \mathbf{1}(w_i)$, where $\mathbf{1}(x) = 1$, if $x \neq 0$ and 0 otherwise.

Interestingly, each method stems from a different theoretical justification (some are Bayesian, some are frequentist and some a a little bit of both like PAC-Bayesian bounds, see, for example, Seeger, 2003, for a review), showing a beautiful example of theory convergence (Guyon, 2009). Either way, for a fixed value of the hyper-parameter A or γ the complexity of the learning problem is lower than that of the original problem. We can optimize A or γ at a second level of inference, for instance by cross-validation.

2.3.2. Bayesian Model Selection

In the Bayesian framework, there is no model selection *per se*, since learning does not involve searching for an optimum function, but averaging over a posterior distribution. For example, if the model class $\mathscr{F}$ consists of models $f(\boldsymbol{x}; \boldsymbol{\alpha}, \boldsymbol{\theta})$, the Bayesian assumption is that the parameters $\boldsymbol{\alpha}$ and hyper-parameters $\boldsymbol{\theta}$ of the model used to generate the data are drawn from a prior $P(\boldsymbol{\alpha}, \boldsymbol{\theta})$. After observing some data D the predictions should be made according to:

$$E_{\boldsymbol{\alpha},\boldsymbol{\theta}}(y|\boldsymbol{x}, D) = \int\int f(\boldsymbol{x}; \boldsymbol{\alpha}, \boldsymbol{\theta})\, P(\boldsymbol{\alpha}, \boldsymbol{\theta}|D)\, d\boldsymbol{\alpha}\, d\boldsymbol{\theta}\ .$$

Hence there is no selection of a single model, but a summation over models in the model class $\mathscr{F}$, weighed by $P(\boldsymbol{\alpha}, \boldsymbol{\theta}|D)$. The problem is to integrate over $P(\boldsymbol{\alpha}, \boldsymbol{\theta}|D)$.[1] A two-level decomposition can be made by factorizing $P(\boldsymbol{\alpha}, \boldsymbol{\theta}|D)$ as $P(\boldsymbol{\alpha}, \boldsymbol{\theta}|D) = P(\boldsymbol{\alpha}|\boldsymbol{\theta}, D)P(\boldsymbol{\theta}|D)$:

$$E_{\boldsymbol{\alpha},\boldsymbol{\theta}}(y|\boldsymbol{x}, D) = \int \left(\int f(\boldsymbol{x}; \boldsymbol{\alpha}, \boldsymbol{\theta}) P(\boldsymbol{\alpha}|\boldsymbol{\theta}, D)\, d\boldsymbol{\alpha} \right) P(\boldsymbol{\theta}|D)\, d\boldsymbol{\theta}\ . \tag{2.1}$$

Bayesian model selection decomposes the prior $P(\boldsymbol{\alpha}, \boldsymbol{\theta})$ into parameter prior $P(\boldsymbol{\alpha}|\boldsymbol{\theta})$ and a "hyper-prior" $P(\boldsymbol{\theta})$. In **MAP learning**, the type-II likelihood (also called the "evidence") $P(D|\boldsymbol{\theta}) = \sum_{\boldsymbol{\alpha}} P(D|\boldsymbol{\alpha}, \boldsymbol{\theta})P(\boldsymbol{\alpha}|\boldsymbol{\theta})$ is maximized with respect to the hyper-parameters $\boldsymbol{\theta}$ (therefore assuming a flat prior for $\boldsymbol{\theta}$), while the "regular" parameters $\boldsymbol{\alpha}$ are obtained by maximizing the posterior $\boldsymbol{\alpha}^* = \text{argmax}_{\boldsymbol{\alpha}} P(\boldsymbol{\alpha}|\boldsymbol{\theta}, D) = \text{argmax}_{\boldsymbol{\alpha}} P(D|\boldsymbol{\alpha}, \boldsymbol{\theta})P(\boldsymbol{\alpha}|\boldsymbol{\theta})$.[2]

2.3.3. Frequentist Model Selection

While Bayesians view probabilities as being realized in the idea of "prior" and "posterior" knowledge of distributions, frequentists define probability in terms of *frequencies of occurrence of events*. In this section, the "frequentist" approach is equated with risk minimization.

1. The calculation of the integral in closed form may be impossible to carry out; in this case, variational approximations are made or numerical simulations are performed, sampling from $P(\boldsymbol{\alpha}, \boldsymbol{\theta}|D)$, and replacing the integral by the summation over a finite number of models.
2. In some Bayesian formulations of multi-layer Perceptrons, the evidence framework maximizes over $\boldsymbol{\theta}$ but marginalises over the weights, rather than maximizing, so in this case the MAP can apply to the parameters or the hyper-parameters or both.

There are obvious ties between the problem of *model selection* and that of *performance prediction*. *Performance prediction* is the problem of estimating the *expected risk* or *generalization error* $R[f]$. *Model selection* is the problem of adjusting the capacity or complexity of the models to the available amount of training data to avoid either **under-fitting** or **over-fitting**. Solving the *performance prediction* problem would also solve the *model selection* problem, but model selection is an easier problem. If we find an ordering index $r[f]$ such that for all pairs of functions $r[f_1] < r[f_2] \Rightarrow R[f_1] < R[f_2]$, then the index allows us to correctly carry out *model selection*. Theoretical performance bounds providing a *guaranteed risk* have been proposed as ranking indices (Vapnik, 1998). Arguably, the tightness of the bound is of secondary importance in obtaining a good ranking index. Bounds of the form $r[f] = R_{tr}[f] + \varepsilon(C/m_{tr})$, where C characterizes the capacity or complexity of the model class, penalizes complex models, but the penalty vanishes as $m_{tr} \to \infty$. Some learning algorithms, for example, SVMs (Boser et al., 1992) or boosting (Freund and Schapire, 1996), optimize a *guaranteed risk* rather than the *empirical risk* $R_{tr}[f]$, and therefore provide some guarantee of good *generalization*. Algorithms derived in this way have an embedded model selection mechanism. Other closely related penalty-based methods include Bayesian **MAP learning** and **regularization**.

Many models (and particularly *compound models* including feature selection, preprocessing, learning machine, and post-processing) are not associated with known performance bounds. Common practice among frequentists is to split available training data into m_{tr} training examples to adjust parameters and m_{va} validation examples to adjust hyper-parameters. In an effort to reduce variance, the validation error $R_{va}[f]$ may be averaged over many data splits, leading to a cross-validation (CV) estimator $R_{CV}[f]$. The most widely used CV method is *K-fold cross-validation*. It consists in partitioning training data into $K \simeq (m_{tr} + m_{va})/m_{va}$ disjoint subsets of roughly equal sizes (up to rounding errors), each corresponding to one validation set (the complement being used as training set). In stratified cross-validation, the class proportions of the full data sets are respected in all subsets. The variance of the results may be reduced by performing Q times K-fold cross-validation and averaging the results of the Q runs. Another popular method consists in holding out a single example at a time for validation purposes. The resulting cross-validation error is referred to as "leave-one-out" error $R_{LOO}[f]$. Some preliminary study design is necessary to determine the sufficient amount of test data to obtain a good estimate of the generalization error (Langford, 2005), the sufficient amount of training data to attain desired generalization performances, and an adequate split of the training data between training and validation set. See Guyon (2009) for a discussion of these issues.

2.4. Multi-level Inference: A Unifying View of Model Selection

What is common among the various views of model selection is the idea of multiple levels of inference, each level corresponding to one set of parameters or hyper-parameters. Consider a two-level case for a model class $f(\boldsymbol{x};\boldsymbol{\alpha},\boldsymbol{\theta})$ parameterized by one set of parameters $\boldsymbol{\alpha}$ and one set of hyper-parameters $\boldsymbol{\theta}$. From the frequentist (risk minimization) point of view, instead of jointly optimizing a risk functional with respect to all parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$, one creates a hierarchy of optimization problems:[3]

$$f^{**} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} R_2[f^*, D] \;, \quad \text{such that} \quad f^* = \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} R_1[f, D] \tag{2.2}$$

where R_1 and R_2 are first and second level risk functionals.

3. It would be more correct if the argmin was assigned to parameters not functions, since the search domain is over parameters, and write $\boldsymbol{\theta}^{**} = \operatorname{argmin}_{\boldsymbol{\theta}} R_2[f^*, D]$, such that $\boldsymbol{\alpha}^* = \operatorname{argmin}_{\boldsymbol{\alpha}} R_1[f, D], f^* = f(\boldsymbol{x}, \boldsymbol{\alpha}^*)$, but we adopt a shorthand to emphasize the similarities between the frequentist and Bayesian approaches.

From the Bayesian point of view, the goal is to estimate the integral of Equation (2.1). There are striking similarities between the two approaches. To make the similarity more obvious, we can rewrite Equation (2.1) to make it look more like Equation (2.2), using the notation f^{**} for $E_{\boldsymbol{\alpha},\boldsymbol{\theta}}(y|\boldsymbol{x},D)$:

$$f^{**} = \int f^{*}\, e^{-R_2}\, d\boldsymbol{\theta}\ , \ \text{ such that } \ f^{*} = \int f\, e^{-R_1}\, d\boldsymbol{\alpha} \tag{2.3}$$

where $R_1 = -\ln P(\boldsymbol{\alpha}|\boldsymbol{\theta},D)$ and $R_2 = -\ln P(\boldsymbol{\theta}|D)$. Note that in Bayesian multi-level inference f^* and f^{**} do not belong to $\mathscr{F}$ but to $\mathscr{F}^*$, the closure of $\mathscr{F}$ under convex combinations.

More generally, we define a *multi-level inference problem* as a learning problem organized into a hierarchy of learning problems. Formally, consider a machine learning toolkit which includes a choice of learning machines $\mathscr{A}[\mathscr{B},R]$, where $\mathscr{B}$ is a model space of functions $f(\boldsymbol{x};\boldsymbol{\theta})$, of parameters $\boldsymbol{\theta}$ and R is an evaluation function (e.g., a risk functional or a negative log posterior). We think of $\mathscr{A}[\mathscr{B},R]$ not as a procedure, but as an "object", in the sense of object oriented programming, equipped with a method "train", which processes data according to a training algorithm:[4]

$$f^{**} = \text{train}(\mathscr{A}[\mathscr{B},R_2],D); \tag{2.4}$$

This framework embodies the second level of inference of both Equations (2.2) and (2.3). The solution f^{**} belongs to $\mathscr{B}^*$, the convex closure of $\mathscr{B}$. To implement the first level of inference, we will consider that $\mathscr{B}$ is itself a learning machine and not just a model space. Its model space $\mathscr{F}$ includes functions $f(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\alpha})$ of variable parameters $\boldsymbol{\alpha}$ ($\boldsymbol{\theta}$ is fixed), which are adjusted by the "train" method of $\mathscr{B}$:

$$f^{*} = \text{train}(\mathscr{B}[\mathscr{F},R_1],D); \tag{2.5}$$

The solution f^* belongs to $\mathscr{F}^*$, the convex closure of $setF$. The method "train" of $\mathscr{A}$ should call the method "train" of $\mathscr{B}$ as a subroutine, because of the nested nature of the learning problems of Equations (2.2) and (2.3). Notice that it is possible that different subsets of the data D are used at the different levels of inference.

We easily see two obvious extensions:

(i) *Multi-level inference:* Equations (2.4) and (2.5) are formally equivalent, so this formalism can be extended to more than two levels of inference.

(ii) *Ensemble methods:* The method "train" returns either a single model or a linear combination of models, so the formalism can include all ensemble methods.

We propose in the next section a new classification of *multi-level inference* methods, orthogonal to the classical Bayesian versus frequentist divide, referring to the way in which data are processed rather than the means by which they are processed.

2.5. Advances in Multi-level Inference

We dedicate this section to reviewing the methods proposed in the collection of papers that we have edited. We categorize multi-level inference modules, each implementing one level of inference, into *filter*, *wrapper*, and *embedded methods*, borrowing from the conventional classification of feature selection methods (Kohavi and John, 1997; Blum and Langley, 1997; Guyon et al., 2006a). *Filters* are methods for narrowing down the model space, without training

4. We adopt a Matlab-style notation: the first argument is the object of which the function is a method; the function "train" is overloaded, there is one for each algorithm. The notations are inspired and adapted from the conventions of the Spider package and the CLOP packages (Saffari and Guyon, 2006).

the learning machine. Such methods include preprocessing, feature construction, kernel design, architecture design, choice of prior or regularizers, choice of a noise model, and filter methods for feature selection. They constitute the highest level of inference[5]. *Wrapper methods* consider the learning machine as a black-box capable of learning from examples and making predictions once trained. They operate with a search algorithm in hyper-parameter space (for example grid search or stochastic search) and an evaluation function assessing the trained learning machine performances (for example the cross-validation error or the Bayesian evidence). They are the *middle-ware* of multi-level inference. *Embedded* methods are similar to wrappers, but they exploit the knowledge of the learning machine algorithm to make the search more efficient and eventually jointly optimize parameters and hyper-parameters, using multi-level optimization algorithms. They are usually used at the lowest level of inference.

2.5.1. Filters

Filter methods include a broad class of techniques aiming to reduce the model space $\mathscr{F}$ prior to training the learning machine. Such techniques may use "prior knowledge" or "domain knowledge", data from prior studies or from R&R (repeatability and reproducibility) studies, and even the training data themselves. But they do not produce the final model used to make predictions. Several examples of filter methods are found in the collection of papers we have edited:

Preprocessing and feature construction. An important part of machine learning is to find a good data representation, but choosing an appropriate data representation is very domain dependent. In benchmark experiments, it has often been found that generating a large number of low-level features yields better result than hand-crafting a few features incorporating a lot of expert knowledge (Guyon et al., 2007). The feature set can then be pruned by feature selection. In the challenges we have organized (Clopinet, 2004-2009) the data were generally already preprocessed to facilitate the work of the participants. However, additional normalizations, space dimensionality reduction and discretization were often performed by the participants. Of all space dimensionality reduction methods *Principal Component Analysis* (PCA) remains the most widely used. Several top-ranking participants to challenges we organized used PCA, including Neal and Zhang (2006), winners of the NIPS 2003 feature selection challenge, and Lutz (2006), winner of the WCCI 2006 performance prediction challenge. *Clustering* is also a popular preprocessing method of dimensionality reduction, championed by Saeed (2009) who used a Bernoulli mixture model as an input to an artificial neural network. In his paper on data grid models Boullé (2009) proposes a new method of *data discretization*. It can be used directly as part of a learning machine based on data grids (stepwise constant predictors) or as a preprocessing to other learning machines, such as the Naïve Bayes classifier. Of particular interest in this paper is the use of *data dependent priors*.

Designing kernels and model architectures. Special purpose neural network architectures implementing the idea of "weight sharing" such as Time Delay Neural Networks (Waibel, 1988) or two-dimensional convolutional networks (LeCun et al., 1989) have proved to be very effective in speech and image processing. More recently a wide variety of special purpose kernels have been proposed to incorporate domain knowledge in kernel learning algorithms. Examples include kernels invariant under various transforms (Simard et al., 1993; Pozdnoukhov and Bengio, 2006), string matching kernels (Watkins, 2000), and

5. Preprocessing is often thought of as a "low-level" operation. However, with respect to model selection, the selection of preprocessing happens generally in the "outer loop" of selection, hence it is at the highest level.

other sequence and tree kernels (Vishwanathan and Smola, 2003). Along these lines, in our collection of papers, Chloé Agathe Azencott and Pierre Baldi have proposed two-dimensional kernels for high-thoughput screening (Azencott and Baldi, 2009). Design effort has also be put into general purpose kernels. For instance, in the paper of Adankon and Cheriet (2009) , the SVM regularization hyper-parameter C (box-constraint) is incorporated in the kernel function. This facilitates the task of multi-level inference algorithms.

Defining regularizers or priors. Designing priors $P(f)$ or **regularizers** $\Omega[f]$ or structuring parameter space into parameters and several levels of hyper-parameters can also be thought of as a filter method. Most priors commonly used do not embed domain knowledge, they just enforce Ockham's razor by favoring simple (smooth) functions or eliminating irrelevant features. Priors are also often chosen out of convenience to facilitate the closed-form calculation of Bayesian integrals (for instance the use of so-called "conjugate priors", see *e.g.,* Neal and Zhang, 2006). The 2-norm regularizer $\Omega[f] = \|f\|^2_{\mathscr{H}}$ for kernel ridge regression, Support Vector Machines (SVM) and Least-Square Support Vector Machines (LSSVM) have been applied with success by many top-ranking participants of the challenges we organized. Gavin Cawley was co-winner of the WCCI 2006 performance prediction challenge using LSSVMs (Cawley, 2006). Another very successful regularizer is the **Automatic Relevance Determination** (ARD) prior. This regularizer was used in the winning entry of Radford Neal in the NIPS 2003 feature selection challenge (Neal and Zhang, 2006). Gavin Cawley also made top ranking reference entries in the IJCNN 2007 ALvsPK challenge (Cawley and Talbot, 2007b) using a similar ARD prior. For linear models, the 1-norm regularizer $\|\boldsymbol{w}\|$ is also popular (see *e.g.,* Pranckeviciene and Somorjai, 2009), but this has not been quite as successful in challenges as the 2-norm regularizer or the ARD prior.

Noise modeling. While the prior (or the regularizer) embeds our prior or domain knowledge of the model class, the likelihood (or the loss function) embeds our prior knowledge of the noise model on the predicted variable y. In regression, the square loss corresponds to Gaussian noise model, but other choices are possible. For instance, recently, Gavin Cawley and Nicola Talbot implemented Poisson regression for kernel machines (Cawley et al., 2007). For classification, the many loss functions proposed do not necessarily correspond to a noise model, they are often just bounding the 0/1 loss and are used for computational convenience. In the Bayesian framework, an sigmoidal function is often used (like the logistic or probit functions) to map the output of a discriminant function $f(\boldsymbol{x}_k)$ to probabilities p_k. Assuming target values $y_k \in \{0,1\}$, the likelihood $\Pi_k p_k^{y_k}(1-p_k)^{1-y_k}$ corresponds to the cross-entropy cost function $\sum_k y_k \ln p_k + (1-y_k)\ln(1-p_k)$. A clever piece-wise S-shaped function, flat on the asymptotes, was used in Chu et al. (2006) to implement sparsity for a Bayesian SVM algorithm. Noise modeling is not limited to noise models for the target y, it also concerns modeling noise on the input variables $\boldsymbol{x}$. Many authors have incorporated noise models on $\boldsymbol{x}$ as part of the kernel design, for example, by enforcing invariance (Simard et al., 1993; Pozdnoukhov and Bengio, 2006). A simple but effective means of using a noise model is to generate additional training data by distorting given training examples. Additional "unsupervised" data is often useful to fit a noise model on the input variables $\boldsymbol{x}$. Repeatability and reproducibility (*R&R*) studies may also provide data to fit a noise model.

Feature selection filters. Feature selection, as a filter method, allows us to reduce the dimensionality of the feature space, to ease the computations performed by learning machines. This is often a necessary step for computationally expensive algorithms such as neural networks. Radford Neal for instance, used filters based on univariate statistical tests to prune the feature space before applying his Bayesian neural network algorithm (Neal and Zhang, 2006). Univariate filters were also widely used in the KDD cup 2009, which involved classification tasks on a very large database, to cut down computations (Guyon et al., 2009b). Feature selection filters are not limited to univariate filters. Markov blanket methods, for instance, provide powerful feature selection filters (Aliferis et al., 2003). A review of filters for feature selection can be found in Guyon et al. (2006a, Chapter 3).

2.5.2. Wrappers

Wrapper methods consider learning machines as *black boxes* capable of internally adjusting their parameters $\boldsymbol{\alpha}$ given some data D and some hyper-parameter values $\boldsymbol{\theta}$. No knowledge either of the architecture, of the learning machines, or of their learning algorithm should be required to use a wrapper. Wrappers are applicable to selecting a classifier from amongst a finite set of learning machines ($\boldsymbol{\theta}$ is then a discrete index), or an infinite set (for continuous values of $\boldsymbol{\theta}$). Wrappers can also be used to build ensembles of learning machines, including Bayesian ensembles. Wrappers use a *search algorithm* or a *sampling algorithm* to explore hyper-parameter space and an *evaluation function* (a risk functional $R_D[f(\boldsymbol{\theta})]$, a posterior probability $P(f(\boldsymbol{\theta})|D)$, or any model selection index $r[f(\boldsymbol{\theta})]$) to assess the performance of the sample of trained learning machines , and, either select one single best machine or create an ensemble of machine voting to make predictions.

Search and sampling algorithms. Because the learning machines in the wrapper setting are "black boxes", we cannot sample directly from the posterior distribution $P(f(\boldsymbol{\theta})|D)$ (or according to $\exp -R_D[f(\boldsymbol{\theta})]$ or $\exp -r[f(\boldsymbol{\theta})]$). We can only compute the evaluation function for given values of $\boldsymbol{\theta}$ for which we run the learning algorithm of $f(\boldsymbol{\theta})$, which internally adjusts its parameters $\boldsymbol{\alpha}$. A **search strategy** defines which hyper-parameter values will be considered and in which order (in case a halting criterion ends the search prematurely). Gavin Cawley, in his challenge winning entries, used the Nelder-Mead simplex algorithm (Cawley and Talbot, 2007a). **Monte-Carlo Markov Chain MCMC methods** are used in Bayesian modeling to sample the posterior probability and have given good results in challenges (Neal and Zhang, 2006). The resulting ensemble is a simple average of the sampled functions $F(\boldsymbol{x}) = (1/s)\sum_{i=1}^{s} f(\boldsymbol{x}|\boldsymbol{\theta}_k)$. Wrappers for *feature selection* use all sort of techniques, but sequential *forward selection* or *backward elimination* methods are most popular (Guyon et al., 2006a, Chapter 4). Other stochastic search methods include biologically inspired methods such as *genetic algorithms* and *particle swarm optimization*. Good results have been obtained with this last method in challenges (H. J. Escalante, 2009), showing that extensive search does not necessarily yield over-fit solutions, if some regularization mechanism is used. The authors of that paper rely for that purpose on weight decay and early stopping. Frequentist ensemble methods, including Random Forests (Breiman, 2001) and Logitboost (Friedman et al., 2000) also gave good results in challenges (Lutz, 2006; Tuv et al., 2009; Dahinden, 2009).

Evaluation functions. For Bayesian approaches, the standard evaluation function is the "evidence", that is the marginal likelihood (also called type-II likelihood) (Neal and Zhang, 2006), or, in other words, the likelihood at the second level of inference. For frequentist approaches, the most frequently used evaluation function is the cross-validation estimator. Specifically, K-fold cross-validation is most often used (H. J. Escalante, 2009; Dahinden, 2009; Lutz, 2006; Reunanen, 2007). The values $K = 10$ or $K = 5$ are typically used by practitioners regardless of the difficulty of the problem (error rate, number of examples, number of variables). Computational considerations motivate this choice, but the authors report a relative insensitivity of the result in that range of values of K. The leave-one-out (LOO) estimator is also used, but due to its high variance, it should rather be avoided, except for computational reasons (see in Section 2.5.3 cases in which the LOO error is inexpensive to compute). These estimators may be poor predictors of the actual learning machine performances, but they are decent model selection indices, provided that the same data splits are used to compute the evaluation function for all models. For **bagging** methods (like Random Forests, Breiman, 2001), the bootstrap estimator is a natural choice: the "out-of-bag" samples, which are those samples not used for training, are used to predict performance. Using empirical estimators at the second level on inference poses the problem of possibly over-fitting them. Some authors advocate using evaluation functions based on prediction risk bounds: Koo and Kil (2008) and Claeskens et al. (2008) derive in this way information criteria for regression models (respectively called "modulus of continuity information criterion" or MCIC and "kernel regression information criterion" or KRIC) and Claeskens et al. (2008) and Pranckeviciene and Somorjai (2009) propose information criteria for classification problems (respectively called "support vector machine information criterion" SVMIC and "transvariation intensity"). The effectiveness of these new criteria is compared empirically in the papers to the classical "Akaike information criterion" or AIC (Akaike, 1973) and the "Bayesian information criterion" or BIC (Schwarz, 1978).

2.5.3. Embedded Methods

Embedded methods are similar to wrappers. They need an evaluation function and a search strategy to explore hyper-parameter space. But, unlike wrapper methods, they exploit specific features of the learning machine architecture and/or learning algorithm to perform multi-level inference. It is easy to appreciate that knowledge of the nature and structure of a learning machine can allow us to search hyper-parameter space in a more efficient way. For instance, the function $f(\boldsymbol{x};\boldsymbol{\alpha},\boldsymbol{\theta})$ may be differentiable with respect to hyper-parameters $\boldsymbol{\theta}$ and it may be possible to use *gradient descent* to optimize an evaluation function $r[f]$. Embedded methods have been attracting substantial attention within the machine learning community in the past few years because of the mathematical elegance of some of the new proposed methods.

Bayesian embedded methods. In the Bayesian framework, the embedded search, sampling or summation over parameters and hyper-parameters is handled in an elegant and consistent way by defining priors both for parameters and hyper-parameters, and computing the posterior, perhaps in two steps, as indicated in Equation (2.3). Of course, it is more easily said than done and the art is to find methods to carry out this integration, particularly when it is analytically intractable. Variational methods are often used to tackle that problem. Variational methods convert a complex problem into a simpler problem, but the simplification introduces additional "variational" parameters, which must then be optimized, hence introducing another level of inference. Typically, the posterior is bounded from above by a family of functions parameterized by given variational parameters. Optimizing the variational parameters yields the best approximation of the posterior (see *e.g.*, Seeger, 2008). Bayesian pragmatists optimize the evidence (also called type-II likelihood or marginal likelihood) at the second level of inference, but non-purists sometimes have a last recourse to cross-validation. The contributions of Boullé (2007, 2009) stand out in that respect because they propose model selection methods for classification and regression, which have no last recourse to cross-validation, yet performed well in recent benchmarks (Guyon et al., 2008a, 2009b). Such methods have been recently extended to the less studied problem of rank regression (Hue and Boullé, 2007). The methods used are Bayesian in spirit, but make use of original data-dependent priors.

Regularized functionals. In the frequentist framework, the choice of a prior is replaced by the choice of a regularized functional. Those are two-part evaluation functions including the empirical risk (or the negative log-likelihood) and a regularizer (or a prior). For kernel methods, a 2-norm regularizer is often used, yielding the classical penalized functional $R_{reg}[f] = R_{emp}[f] + \gamma \|f\|^2_{\mathscr{F}}$. Pranckeviciene and Somorjai (2009) explore the possibilities offered by a 1-norm regularizer. Such approaches provide an embedded method of feature selection, since the constraints thus imposed on the weight vector drive some weights to exactly zero. We emphasized in the introduction that, in some cases, decomposing the inference problem into multiple levels allows us to conveniently regain the convexity of the optimization problem involved in learning. Ye et al. (2008) propose a multiple kernel learning (MKL) method, in which the optimal kernel matrix is obtained as a linear combination of pre-specified kernel matrices, which can be brought back to a convex program. Few approaches are fully embedded and a wrapper is often used at the last level of inference. For instance, in kernel methods, the kernel parameters may be optimized by gradient descent on the regularized functional, but then the regularization parameter is selected by cross-validation. One approach is to use a bound on the generalization error at the second level of inference. For instance, Guermeur (2007) proposes such a bound for the multi-class SVM, which can be used to choose the values of the "soft margin parameter" C and the kernel parameters. Cross-validation may be preferred by practitioners because it has performed consistently well in benchmarks (Guyon et al., 2006b). This motivated Kunapuli et al. (2009) to integrate the search for optimal parameters and hyper-parameters into a multi-level optimization program, using a regularized functional at the lower level, and cross-validation at the upper level. Another way of integrating a second level of inference performed by cross-validation and the optimization of a regularized functional at the first level of inference is to use a closed-form expression of the leave-one-out error (or a bound) and optimize it by gradient descent or another classical optimization algorithm. Such *virtual leave-one-out* estimators, requiring training a single classifier on all the data (see *e.g.,* Cawley and Talbot, 2007a; Debruyne et al., 2–8, in the collection of papers we have assembled).

2.6. Advanced Topics and Open Problems

We have left aside many important aspects of model selection, which, space permitting, would deserve a longer treatment. We briefly discuss them in this section.

2.6.1. Ensemble Methods

In Section 2.4, we have made an argument in favor of unifying *model selection* and *ensemble methods*, stemming either from a Bayesian or frequentist perspective, in the common framework of *multi-level optimization*. In Sections 2.5.1, 2.5.2 and 2.5.3, we have given examples of model selection and ensemble methods following *filter*, *wrapper* or *embedded* strategies. While this categorization has the advantage of erasing the dogmatic origins of algorithms, it blurs some of the important differences between model selection and ensemble methods. Ensemble methods can be thought of as a way of circumventing model selection by voting among models rather than choosing a single model. Recent challenges results have proved their effectiveness (Guyon et al., 2009b). Arguably, model selection algorithms will remain important in applications where model simplicity and data understanding prevail, but ever increasing computer power has brought ensemble methods to the forefront of multi-level inference techniques. For that reason, we would like to single out those papers of our collection that have proposed or applied ensemble methods:

Lutz (2006) used boosted shallow decision trees for his winning entries in two consecutive challenges. Boosted decision trees have often ended up among the top ranking methods in other challenges (Guyon et al., 2006a, 2009b). The particular implementation of Lutz of the Logitboost algorithm (Friedman et al., 2000) use a "shrinkage" regularization hyper-parameter, which seems to be key to attain good performance, and is adjusted by cross-validation as well as the total number of *base learners*. Dahinden (2009) successfully applied the Random Forest (RF) algorithm (Breiman, 2001) in the performance prediction challenge (Guyon et al., 2006b). She demonstrated that with minor adaptations (adjustment of the bias value for improved handling of unbalanced classes), the RF algorithm can be applied without requiring user intervention. RF continues to be a popular and successful method in challenges (Guyon et al., 2009b). The top ranking models use very large ensembles of hundreds of trees. One of the unique features of RF algorithms is that they subsample both the training examples and the features to build base learners. Using random subsets of features seems to be a winning strategy, which was applied by others to ensembles of trees using both boosting and bagging (Tuv et al., 2009) and to other base learners (Nikulin, 2009). Boullé (2007) also adopts the idea of creating ensembles using base learners constructed with different subsets of features. Their base learner is the naïve Bayes classifier and, instead of using random subsets, they select subsets with a forward-backward method, using a maximum A Posteriori (MAP) evaluation function (hence not requiring cross-validation). The base learners are then combined with an weighting scheme based on an information theoretic criterion, instead on weighting the models with the posterior probability as in Bayesian model averaging. This basically boils down to using the logarithm of the posterior probabilities instead of the posterior probabilities themselves for weighting the models. The weights have an interpretation in terms of model compressibility. The authors show that this strategy outperforms Bayesian model averaging on several benchmark data sets. This can be understood by the observation that when the posterior distribution is sharply peaked around the posterior mode, averaging is almost the same as selecting the MAP model. Robustness is introduced by performing a more balanced weighting of the base learners. In contrast with the methods we just mentioned, which choose identical base learners (trees of naïve Bayes), other successful challenge participants have built heterogeneous ensembles of learning machines (including, for example, linear models, kernel methods, trees, naïve Bayes, and neural networks), using cross-validation to evaluate their candidates for inclusion in the ensemble (Wichard, 2007; IBM team, 2009). While Wichard (2007) evaluates classifiers independently, IBM team (2009) uses a forward selection method, adding a new candidate in the ensemble based on the new performance of the ensemble.

2.6.2. PAC Bayes Approaches

Unifying Bayesian and frequentist model selection procedures under the umbrella of *multi-level inference* may shed new light on correspondences between methods and have a practical impact on the design of toolboxes incorporating model selection algorithms. But there are yet more synergies to be exploited between the Bayesian and the frequentist framework. In this section, we would like to capture the spirit of the PAC Bayes approach and outline possible fruitful directions of research.

The PAC learning framework (Probably Approximately Correct), introduced by Valiant (1984) and later recognized to closely resemble the approach of the Russian school popularized in the US by Vapnik (1979), has become the beacon of frequentist learning theoretic approaches. It quantifies the generalization performance (the *Correct* aspect) of a learning machine via performance bounds (the *Approximate* aspect) holding in probability (the *Probable* aspect):

$$Prob\Big[(R[f]-R_{emp}[f]) \leq \varepsilon(\delta)\Big] \geq (1-\delta)\ ,$$

In this equation, the confidence interval $\varepsilon(\delta)$ (*Approximate* aspect) bounds, with probability $(1-\delta)$ (*Probable* aspect),the difference between the *expected risk* or generalization error $R[f]$ and the *empirical risk*[6] $R_{emp}[f]$ (*Correct* aspect). Recently, many bounds have been proposed to quantify the generalization performance of algorithms (see *e.g.,* Langford, 2005, for a review). The idea of deriving new algorithms, which optimize a bound $\varepsilon(\delta)$ (*guaranteed risk* optimization) has been popularized by the success of SVMs (Boser et al., 1992) and boosting (Freund and Schapire, 1996).

The PAC framework is rooted in the frequentist philosophy of defining probability in terms of *frequencies of occurrence of events* and bounding differences between mathematical expectations and frequencies of events, which vanish with increasingly large sample sizes (law of large numbers). Yet, since the pioneering work of Haussler et al. (1994), many authors have proposed so-called PAC-Bayes bounds. Such bounds assess the performance of existing Bayesian algorithms (see *e.g.,* Seeger, 2003), or are used to derive new Bayesian algorithms optimizing a guaranteed risk functional (see Germain et al. 2009 and references therein).

This is an important paradigm shift, which bridges the gap between the frequentist *structural risk minimization* approach to model selection (Vapnik, 1998) and the *Bayesian prior* approach. It erases the need for assuming that *the model used to fit the data comes from a concept space of functions that generated the data*. Instead, priors may be used to provide a "structure" on a chosen model space (called *hypothesis space* to distinguish it from the *concept space*), which does not necessarily coincide with the *concept space*, of which we often know nothing. Reciprocally, we can interpret structures imposed on a hypothesis space as our prior belief that certain models are going to perform better than others (see, for instance, the examples at the end of Section 2.3.1).

This opens the door to also regularizing the second level of inference by using performance bounds on the cross-validation error, as was done for instance in Cawley and Talbot (2007a) and Guyon (2009).

2.6.3. Open Problems

- **Domain knowledge:** From the earliest embodiments of Okcham's razor using the number of free parameters to modern techniques of regularization and bi-level optimization, model selection has come a long way. The problem of finding the right structure remains, the rights prior or the right regularizer. Hence know-how and domain knowledge are still required. But in a recent challenge we organized called "agnostic learning *vs.* prior knowledge" (Guyon et al., 2008b) it appeared that the relatively small incremental improvements gained with prior knowledge came at the expense of important human effort. In many domains, collecting more data is less costly than hiring a domain expert. Hence there is pressure towards improving machine learning toolboxes and, in particular equipping them with model selection tools. For the competitions we organized (Clopinet, 2004-2009), we made a toolbox available with state-of-the-art models (Saffari and Guyon, 2006), which we progressively augmented with the best performing methods.

6. at the first level of inference, this would be the training error $R_{tr}[f]$; at the second level of inference this may be the validation error $R_{va}[f]$

The Particle Swarm Optimization (PSO) model selection method can find the best models in the toolbox and reproduce the results of the challenges (H. J. Escalante, 2009). Much remains to be done to incorporate filter and wrapper model selection algorithms in machine learning toolboxes.

- **Unsupervised learning:** Multi-level optimization and model selection are also central problems for **unsupervised learning**. When no target variable is available as "teaching signal" one can still define regularized risk functionals and multi-level optimization problems (Smola et al., 2001). Hyper-parameters (e.g., "number of clusters") can be adjusted by optimizing a second level objective such as model stability (Ben-Hur et al., 2002), which is an erzatz of cross-validation. The primary difficulty with model selection for unsupervised learning is to validate the selected model. To this day, there is no consensus on how to benchmark methods, hence it is very difficult to quantify progress in this field. This is why we have so far shied away from evaluating unsupervised learning algorithms, but this remains on our agenda.

- **Semi-supervised learning:** Very little has been done for model selection in **semi-supervised learning** problems, in which only some training instances come with target values. Semi-supervised tasks can be challenging for traditional model selection methods, such as cross-validation, because the number of labeled data is often very small. Schuurmans and Southey (2001) used the unlabeled data to test the consistency of a model, by defining a metric over the hypothesis space. Similarly, Madani et al. (2005) introduced the co-validation method, which uses the disagreement of various models on the predictions over the unlabeled data as a model selection tool. In some cases there is no performance gain by using the unlabeled data for training (Singh et al., 2008). Deciding whether all or part of the unlabeled data should be used for training (*data selection*) may also be considered a model selection problem.

- **Non *i.i.d.* data:** The problem of non *i.i.d.* data raises a number of other questions because if there are significant differences between the distribution of the training and the test data, the cross-validation estimator may be worthless. For instance, in causal discovery problems, training data come from a "natural" distribution while test data come from a different "manipulated" distribution (resulting from some manipulations of the system by an external agent, like clamping a given variable to given values). Several causal graphs may be consistent with the "natural distribution" (not just with the training data, with the true unknown distribution), but yield very different predictions of manipulated data. Rather selecting a single model, it make more sense to select a model class. We have started a program of data exchange and benchmarks to evaluate solutions to such problems (Guyon et al., 2008a, 2009a).

- **Computational considerations:** The selection of the model best suited to a given application is a multi-dimensional problem in which prediction performance is only one of the dimensions. Speed of model building and processing efficiency of deployed models are also important considerations. Model selection algorithms (or ensemble methods) which often require many models to be trained (e.g., wrapper methods with extensive search strategies and using cross-validation to validate models) may be unable to build solutions in a timely manner. At the expense of some acceptable loss in prediction performance, methods using *greedy search strategies* (like forward selection methods) and *single-pass evaluation functions* (requiring the training of only a single model to evaluate a given hyper-parameter choice), may considerably cut the training time. Greedy search methods include forward selection and backward elimination methods. Single-pass evaluation functions include penalized training error functionals (regularized functionals, MAP estimates) and virtual-leave-one-out estimators. The latter allows users to compute the leave-one-out-error at almost no additional computational expense than training a single predictor on all the training data (see *e.g.,* Guyon et al., 2006a, Chapter 2, for a review). Other tricks-of-the-trade include following *regularization paths* to sample the hyper-parameter space more effectively (Rosset and Zhu, 2006; Hastie et al., 2004). For some models, the evaluation function is piecewise linear between a few discontinuous changes occurring for a few finite hyper-parameter values. The whole path can be reconstructed from only the values of the evaluation function at those given points. Finally, Reunanen (2007) proposed clever ways of organizing nested cross-validation evaluations in multiple level of inference model selection using cross-indexing. The author also explored the idea of spending more time to refine the evaluation of the most promising models. Further work needs to be put into model selection methods, which simultaneously address multiple objectives, including optimizing prediction performance and computational cost.

2.7. Conclusion

In the past twenty years, much effort has been expended towards finding the best regularized functionals. The many embodiments of Ockham's razor in machine learning have converged towards similar regularizers. Yet, the problem of model selection remains because we need to optimize the regularization parameter(s) and often we need to select among various preprocessings, learning machines, and post-processings. In the proceedings of three of the challenges we organized around the problem of model selection, we have collected a large number of papers, which testify to the vivid activity of the field. Several researchers do not hesitate to propose heretic approaches transcending the usual "frequentist" or Bayesian dogma. We have seen the idea of using the Bayesian machinery to design regularizers with "data-dependent priors" emerge (Boullé, 2007, 2009), much like a few years ago data-dependent performance bounds (Bartlett, 1997; Vapnik, 1998) and PAC-Bayes bounds (Haussler et al., 1994; Seeger, 2003) revolutionized the "frequentist" camp, up to then very fond of uniform convergence bounds and the VC-dimension (Vapnik and Chervonenkis, 1971). We have also seen the introduction of regularization of cross-validation estimators using Bayesian priors (Cawley and Talbot, 2007a). Ensemble methods may be thought of as a way of circumventing model selection. Rather, we think of model selection and ensemble methods as two options to perform multi-level inference, which can be formalized in a unified way.

Within this general framework, we have categorized approaches into *filter*, *wrapper* and *embedded* methods. These methods complement each other and we hope that in a not too distant future, they will be integrated into a consistent methodology: Filters first can prune model space; Wrappers can perform an outer level model selection to select pre/post processings and feature subsets; Embedded methods can perform an inner level hyper-parameter selection integrated within a bi-level optimization program. We conclude that we are moving towards a unified framework for model selection and there is a beneficial synergy between methods, both from a theoretical and from a practical perspective.

Acknowledgments

This project was supported by the National Science Foundation under Grant N0. ECS-0424142. Amir Saffari was supported by the Austrian Joint Research Project Cognitive Vision under projects S9103-N04 and S9104-N04. Any opinions, findings, and conclusions or recommenda-

tions expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Appendix A. Glossary

Automatic Relevance Determination (ARD) prior. The ARD prior was invented for neural networks (MacKay, 1992): all network input variables and all neuron outputs (internal features) are weighed by a scaling factor κ_i, before being independently weighted by the network connections. A hyper-prior must be chosen to favor small values of the κ_i, which makes the influence of irrelevant variables or features naturally fade away. For kernel methods, ARD falls under the same framework as the $\|f\|^2_{\mathscr{H}}$ regularizer, for a special class of kernels using variable (feature) scaling factors. For instance, the ARD prior is implemented by defining the Gaussian kernel (for positive hyper-parameters κ_i):

$$K(\boldsymbol{x}_h, \boldsymbol{x}_k) = \exp\left\{ -\sum_{j=1}^{n} \kappa_i (x_{h,j} - x_{k,j})^2 \right\}$$

instead of the regular Gaussian kernel $K(\boldsymbol{x}_h, \boldsymbol{x}_k) = \exp\left\{-\kappa \|\boldsymbol{x}_h - \boldsymbol{x}_k\|^2\right\}$.

Base learner. In an ensemble method, the individual learning machines that are part of the ensemble.

Bagging. Bagging stands for bootstrap aggregating. Bagging is a parallel ensemble method (all **base learners** are built independently from training data subsets). Several data subsets of size m are drawn independently with replacement from the training set of m examples. On average each subset thus built contains approximately 2/3 of the training examples. The ensemble predictions are made by averaging the predictions of the baser learners. The ensemble approximates $E_D(f(\boldsymbol{x}, D))$, where $f(\boldsymbol{x}, D)$ is a function from the model class $\mathscr{F}$, trained with m examples and $E_D(.)$ is the mathematical expectation over all training sets of size m. The rationale comes from the **bias/variance decomposition** of the **generalization error**. The "out-of-bag" samples (samples not used for learning for each data subset drawn for training) may be used to create a bootstrap prediction of performance.

Bayesian learning. Under the Bayesian framework, it is assumed that the data were generated from a double random process: (1) a model is first drawn according to a **prior** distribution in a **concept space**; (2) data are produced using the model. In the particular case of supervised learning, as for **maximum likelihood learning**, a three-part data generative model is assumed: $P(\boldsymbol{x})$, $f \in \mathscr{F}$, and a zero-mean noise model. But, it is also assumed that the function f was drawn according to a prior distribution $P(f)$. This allows us to compute the probability of an output y given an input $\boldsymbol{x}$, $P(y|\boldsymbol{x}) = \int_{f \in \mathscr{F}} P(y|\boldsymbol{x}, f) dP(f)$, or its mathematical expectation $E(y|\boldsymbol{x}) = \int_{f \in \mathscr{F}} f(\boldsymbol{x}) dP(f)$, averaging out the noise. After training data D are observed, the prior $P(f)$ is replaced by the **posterior** $P(f|D)$. The mathematical expectation of y given $\boldsymbol{x}$ is estimated as: $E(y|\boldsymbol{x}, D) = \int_{f \in \mathscr{F}} f(\boldsymbol{x}) dP(f|D)$. Hence, learning consists of calculating the posterior distribution $P(f|D)$ and integrating over it. The predictions are made according to $E(y|\boldsymbol{x}, D)$, a function not necessarily belonging to $\mathscr{F}$. In the case of classification, $E(y|\boldsymbol{x}, D)$ does not take values in $\mathscr{Y}$ (although thresholding the output just takes care of the problem). If we want a model in $\mathscr{F}$, we can use the Gibbs algorithm, which picks one sample in $\mathscr{F}$ according to the posterior distribution $P(f|D)$, or use the **MAP learning** approach. In Bayesian learning, analytically integrating over the posterior distribution is often impossible and the integral may be approximated by finite sum of models, weighted by positive coefficients (see **variational methods**) or by sampling models from the posterior distribution (see **Weighted majority algorithm** and **Monte-Carlo Markov Chain** or MCMC). The resulting estimators of $\hat{E}(y|\boldsymbol{x},D)$ are convex combinations of functions in $\mathscr{F}$ and, in that sense, Bayesian learning is similar to **ensemble methods**.

Bias/variance decomposition. In the case of a least-square loss, the bias/variance decomposition is given by $E_D[(f(\boldsymbol{x};D) - E[y|\boldsymbol{x}])^2] = (E_D[f(\boldsymbol{x};D)] - E(y|\boldsymbol{x}))^2 + E_D[(f(\boldsymbol{x};D) - E_D[f(\boldsymbol{x};D)])^2]$. The second term (the “variance” of the estimator $f(\boldsymbol{x},D)$) vanishes if $f(\boldsymbol{x};D)$ equals $E_D[f(\boldsymbol{x};D)$. The motivates the idea of using an approximation of $E_D[f(\boldsymbol{x};D)$ as a predictor. In **bagging** the approximation is obtained by averaging over functions trained from m examples drawn at random with replacement from the training set D (bootstrap method). The method works best if $\mathscr{F}$ is not biased (i.e., contains $E(y|\boldsymbol{x})$). Most models with low bias have a high variance and vice versa, hence the well-known bias/variance tradeoff.

Concept space. A space of data generative models from which the data are drawn. Not to be confused with **model space** or **hypothesis space**.

Empirical risk. An estimator of the **expected risk** that is the average of the loss over a finite number of examples drawn according to $P(\boldsymbol{x},y)$: $R_{emp} = (1/m)\sum_{k=1}^{m} \mathscr{L}(f(\boldsymbol{x}_k),y_k)$.

Ensemble methods. Methods of building predictors using multiple **base learners**, which vote to make predictions. Predictions of y are made using a convex combination of functions $f_j \in \mathscr{F}$: $F(\boldsymbol{x}) = \sum_j p_j f_j(\boldsymbol{x})$, where p_j are positive coefficients. The two most prominent ensemble methods are bagging (Breiman, 1996) and boosting (Freund and Schapire, 1996). Bagging is a parallel ensemble method (all trees are built independently from training data subsets), while boosting is a serial ensemble method (trees complementing each other are progressively added to decrease the residual error). Random Forests (RF) (Breiman, 2001) are a variant of bagging methods in which both features and examples are subsampled. Boosting methods come in various flavors including Adaboost, Gentleboost, and Logitboost. The original algorithm builds successive models (called “weak learners”) by resampling data in a way that emphasizes examples that have proved hardest to learn. Newer versions use a weighting scheme instead of resampling (Friedman, 2000).

Expected risk. The mathematical expectation of a risk functional over the unknown probability distribution $P(\boldsymbol{x},y)$: $R[f] = \int \mathscr{L}(f(\boldsymbol{x}),y)\, dP(\boldsymbol{x},y)$. Also called **generalization error**.

Generalization error. See **expected risk**.

Greedy search strategy. A search strategy, which does not revisit partial decisions already made, is called “greedy”. Examples include forward selection and backward elimination in feature selection.

Guaranteed risk. A bound on the **expected risk**. See **PAC learning** and **Structural Risk Minimization** (SRM).

Hypothesis space. A space of models, which are fit to data, not necessarily identical to the **concept space** (which is often unknown).

Loss function. A function $\mathscr{L}(f(\boldsymbol{x}),y)$, which measures the discrepancy between target values y and model predictions $f(\boldsymbol{x})$. Examples include the square loss $(y-f(\boldsymbol{x}))^2$ for regression of the 0/1 loss $\mathbf{1}[f(\boldsymbol{x})\neq y]$ for classification).

MAP learning. Maximum a posteriori (MAP) learning shares the same framework as Bayesian learning, but it is further assumed that the posterior $P(f|D)$ is concentrated and that $E(y|\boldsymbol{x},D)$ can be approximated by $f^*(\boldsymbol{x})$, with $f^* = \text{argmax}_f P(f|D) = \text{argmax}_f P(D|f)P(f) = \text{argmin}_f -\ln P(D|f) - \ln P(f)$. If we assume a uniform prior, we are brought back to maximum likelihood learning. If both $P(D|f)$ and $P(f)$ are exponentially distributed ($P(y|\boldsymbol{x},f) = \exp -\mathscr{L}(f(\boldsymbol{x}),y)$ and $P(f) = \exp -\Omega[f]$), then MAP learning is equivalent to the minimization of a regularized risk functional.

Maximum likelihood learning. It is assumed that the data were generated by an input distribution $P(\boldsymbol{x})$, a function f from a **model space** $\mathscr{F}$ coinciding with the **concept space**, and a zero-mean **noise model**. For regression, for instance, if Gaussian noise $\varepsilon \sim \mathscr{N}(0,\sigma^2)$ is assumed, y is distributed according to $P(y|\boldsymbol{x},f) = \mathscr{N}(f(\boldsymbol{x}),\sigma^2)$. In the simplest case, $P(\boldsymbol{x})$ and the noise model are not subject to training (the values of $\boldsymbol{x}$ are fixed and the noise model is known). Learning then consists in searching for the function f^*, which maximizes the likelihood $P(D|f)$, or equivalently (since $P(\boldsymbol{x})$ is not subject to training) $f^* = \text{argmax}_f P(\boldsymbol{y}|X,f) = \text{argmin}_f -\ln P(\boldsymbol{y}|X,f)$. With the *i.i.d.* assumption, $f^* = \text{argmax}_f\ \Pi_{k=1}^m P(y_k|\boldsymbol{x}_k,f) = \text{argmin}_f \sum_{k=1}^m -\ln P(y_k|\boldsymbol{x}_k,f)$. For distributions belonging to the exponential family $P(y|\boldsymbol{x},f) = \exp\{-\mathscr{L}(f(\boldsymbol{x}),y)\}$, the maximum likelihood method is equivalent to the method of minimizing the **empirical risk**. In the case of Gaussian noise, this corresponds to the method of least squares.

Model space. A space of predictive models, which are fit to data. Synonym of **hypothesis space**. For Bayesian models, also generally coincides with the **concept space**, but not for frequentists.

Monte-Carlo Markov Chain (MCMC) method. To approximate Bayesian integrals one can sample from the posterior distribution $P(f|D)$ following a Monte-Carlo Markov chain (MCMC), then make predictions according to $\hat{E}(y|\boldsymbol{x},D) = \sum_j f_j(\boldsymbol{x})$. In a MCMC, at each step new candidate models $f_j \in \mathscr{F}$ are considered, in a local neighborhood of the model selected at the previous step. The new model is accepted if it provides a better fit to the data according to the posterior distribution or, if not, a random decision is made to accept it, following the Gibbs distribution (better models having a greater chance of acceptance).

Over-fitting avoidance. Model selection is traditionally associated with the so-called problem of **over-fitting** avoidance. Over-fitting means fitting the training examples well (i.e., obtaining large model likelihood or low empirical risk values, computed from training data), but generalizing poorly on new test examples. Over-fitting is usually blamed on too large a large number of free parameters to be estimated, relative to the available number of training examples. The most basic model selection strategy is therefore to restrict the number of free parameters according to "strict necessity". This heuristic strategy is usually traced back in history to the principle known as Ockham's razor "Plurilitas non est ponenda sin necessitate" (William of Ockham, 14th century). In other words, of two theories providing similarly good predictions, the simplest one should be preferred, that is, shave off unnecessary parameters. Most modern model selection strategies claim some affiliation with Ockham's razor, but the number of free parameters is replaced by a measure of **capacity** or **complexity** of the model class, $C[\mathscr{F}]$. Intuitively, model classes with large $C[\mathscr{F}]$ may include the correct model, but it is hard to find. In this case, even models with a low training error may have a large generalization error (high "variance"; over-fitting problem). Conversely, model classes with small $C[\mathscr{F}]$ may yield "biased" models, that is, with both high training and generalization error (under-fitting). See **bias/variance decomposition.**.

PAC learning. The "probably approximately correct" (PAC) learning procedures, seek a function minimizing a **guaranteed risk** $R_{gua}[f] = R_{emp}[f] + \varepsilon(C, \delta)$ such that with (high) probability $(1-\delta)$, $R[f] \leq R_{gua}[f]$. C is a measure of capacity or complexity.

Regularizers and regularization. The regularization method consists of replacing the minimization of the **empirical risk** $R_{emp}[f]$ by that of $R_{reg}[f] = R_{emp} + \Omega[f]$. A regularizer $\Omega[f]$ is a functional penalizing "complex" functions. If both $R_{tr}[f]$ and $\Omega[f]$ are convex, there is a unique minimum of $R_{reg}[f]$ with respect to f. In **MAP learning**, $-\ln P(f)$ can be thought of as a **regularizer**. One particularly successful regularizer is the 2-norm regularizer $\|f\|^2_{\mathscr{H}}$ for model functions $f(\boldsymbol{x}) = \sum_{k=1}^m \alpha_k K(\boldsymbol{x}, \boldsymbol{x}_k)$ belonging to a Reproducing Kernel Hilbert Space $\mathscr{H}$ (kernel methods). In the particular case of the linear model $f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x}$, we have $\|f\|^2_{\mathscr{H}} = \|\boldsymbol{w}\|^2$, a commonly used regularized found in many algorithms including ridge regression (Hoerl, 1962) and SVMs (Boser et al., 1992). In the general case, $\|f\|^2_{\mathscr{H}} = \boldsymbol{f} K^{-1} \boldsymbol{f} = \boldsymbol{\alpha}^T K \boldsymbol{\alpha}$, where $\boldsymbol{f} = [f(\boldsymbol{x}_k)]_{k=1}^m$ is the vector of predictions of the training examples, $\boldsymbol{\alpha} = [\alpha_k]_{k=1}^m$ and $K = [K(\boldsymbol{x}_h, \boldsymbol{x}_k], h = 1, \ldots, m\, k = 1, \ldots, m$. Due to the duality between RKHS and stochastic processes (Wahba, 1990), the functions in the RKHS can also be explained as a family of random variables in a Gaussian process, assuming a prior $P(f)$ proportional to $\exp(-\gamma \|f\|_{\mathscr{H}}) = \exp(-\gamma \boldsymbol{f} K^{-1} \boldsymbol{f})$ and the kernel matrix K is interpreted as a covariance matrix $K(\boldsymbol{x}_h, \boldsymbol{x}_k) = \mathrm{cov}[f(x_k), f(x_k)]$.

Risk minimization. Given a **model space** or **hypothesis space** $\mathscr{F}$ of functions $y = f(\boldsymbol{x})$, and a **loss function** $\mathscr{L}(f(\boldsymbol{x}), y)$, we want to find the function $f^* \in \mathscr{F}$ that minimizes the **expected risk** $R[f] = \int \mathscr{L}(f(\boldsymbol{x}), y)\ dP(\boldsymbol{x}, y)$. Since $P(\boldsymbol{x}, y)$ is unknown, only estimations of $R[f]$ can be computed. The simplest estimator is the average of the loss over a finite number of examples drawn according to $P(\boldsymbol{x}, y)$ called the **empirical risk**: $R_{emp} = (1/m) \sum_{k=1}^m \mathscr{L}(f(\boldsymbol{x}_k), y_k)$. The minimization of the empirical risk is the basis of many machine learning approaches to selecting f^*, but minimizing regularized risk functionals is often preferred. See **regularization**. Also, related are the **PAC learning** procedures and the method of **Structural Risk Minimization** (SRM).

Search strategy. There are *optimal search strategies*, which guarantee that the optimum of the evaluation function will be found, including the *exhaustive search* method, for discrete hyper-parameter spaces. The popular *grid search* method for continuous hyper-parameter spaces performs an exhaustive search, up to a certain precision. A related *stochastic search* method is *uniform sampling*. Uniformly sampling parameter space may be computationally expensive and inefficient. If we use a non-uniform distribution $G(\boldsymbol{\theta})$ to sample hyper-parameter space, which resembles $P(f(\boldsymbol{\theta})|D)$, the search can be made more efficient. This idea is exploited in *rejection sampling* and *importance sampling*: according to these methods a Bayesian ensemble $F(\boldsymbol{x}) = \sum_k w_k f(\boldsymbol{x}; \boldsymbol{\theta}_k)$ would use weight w_k proportional to $P(f(\boldsymbol{\theta})|D)/G(\boldsymbol{\theta})$. Because of the computational burden of (near) optimum strategies, other strategies are often employed, usually yielding only a *local optimum*. These include *sequential search strategies* such as *coordinate ascent* or descent (making small steps along coordinate axes) or *pattern search* (Momma and Bennett, 2002) (making local steps according to a certain pattern), which, by accepting only moves that improve the evaluation function, find the local optimum nearest to the starting point.

Some stochastic search methods accept moves not necessarily improving the value of the evaluation function, like simulated annealing or **Markov chain Monte Carlo (MCMC) methods**. Both methods accept all moves improving the evaluation function and some moves that do not, for example, with probability $\exp -\Delta r/T$, where T is a positive parameter ($T = 1$ for MCMC and progressively diminishes for simulated annealing). Such stochastic methods search hyper-parameter space more intensively and do not become stuck in the nearest local optimum of the evaluation function.

Semi-supervised learning. In semi-supervised learning, in addition to the labeled data, the learning machine is given a (possibly large) set of unlabeled data. Such unlabeled data may be used for training or model selection.

Structural Risk Minimization. The method of Structural Risk Minimization (SRM) provides aeans of building regularized risk functionals (see **Regularization**), using the idea of **guaranteed risk** minimization, but not requiring the calculation of the model class capacity or complexity, which is often unknown or hard to compute. In the risk minimization framework, it is not assumed that the model space includes a function or “concept”, which generated the data (see **concept space** and **hypothesis space**).

Supervised learning. Learning with teaching signal or target y.

Under-fitting. While **over-fitting** is the problem of learning the training data too well the expense of a large **generalization error**, under-fitting is the problem of having a too weak model not even capable of learning the training data and also generalizing poorly.

Unsupervised learning. Learning in the absence of teaching signal or target y.

Weighted majority algorithm. To approximate Bayesian integrals one can draw samples f_j uniformly from the model space of functions $\mathscr{F}$ and make predictions according to $\hat{E}(y|\boldsymbol{x},D) = \sum_j P(f_j|D) f_j(\boldsymbol{x})$.

References

M. Adankon and M. Cheriet. Unified framework for SVM model selection. In I. Guyon, et al., editor, *Hands on Pattern Recognition*. Microtome, 2009.

H. Akaike. Information theory and an extension of the maximum likelihood principle. In B.N. Petrov and F. Csaki, editors, *2nd International Symposium on Information Theory*, pages 267–281. Akademia Kiado, Budapest, 1973.

C. F. Aliferis, I. Tsamardinos, and A. Statnikov. HITON, a novel Markov blanket algorithm for optimal variable selection. In *2003 American Medical Informatics Association (AMIA) Annual Symposium*, pages 21–25, 2003.

C.-A. Azencott and P. Baldi. Virtual high-throughput screening with two-dimensional kernels. In I. Guyon, et al., editor, *Hands on Pattern Recognition*. Microtome, 2009.

P. L. Bartlett. For valid generalization the size of the weights is more important than the size of the network. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, page 134, Cambridge, MA, 1997. MIT Press.

A. Ben-Hur, A. Elisseeff, and I. Guyon. A stability based method for discovering structure in clustered data. In *Pacific Symposium on Biocomputing*, pages 6–17, 2002.

A. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271, December 1997.

B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *COLT*, pages 144–152, 1992.

M. Boullé. Compression-based averaging of selective naive bayes classifiers. In I. Guyon and A. Saffari, editors, *JMLR, Special Topic on Model Selection*, volume 8, pages 1659–1685, Jul 2007. URL http://www.jmlr.org/papers/volume8/boulle07a/boulle07a.pdf.

M. Boullé. Data grid models for preparation and modeling in supervised learning. In I. Guyon, et al., editor, *Hands on Pattern Recognition*. Microtome, 2009.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

G. Cawley. Leave-one-out cross-validation based model selection criteria for weighted ls-svms. In *IJCNN*, pages 1661–1668, 2006.

G. Cawley and N. Talbot. Preventing over-fitting during model selection via Bayesian regularisation of the hyper-parameters. In I. Guyon and A. Saffari, editors, *JMLR, Special Topic on Model Selection*, volume 8, pages 841–861, Apr 2007a. URL http://www.jmlr.org/papers/volume8/cawley07a/cawley07a.pdf.

G. Cawley and N. Talbot. Over-fitting in model selection and subsequent selection bias in performance evaluation. *JMLR, submitted*, 2009.

G. C. Cawley and N. L. C. Talbot. Agnostic learning versus prior knowledge in the design of kernel machines. In *Proc. IJCNN07*, Orlando, Florida, Aug 2007b. INNS/IEEE.

G.C. Cawley, G.J. Janacek, and N.L.C. Talbot. Generalised kernel machines. In *International Joint Conference on Neural Networks*, pages 1720–1725. IEEE, August 2007.

W. Chu, S. Keerthi, C. J. Ong, and Z. Ghahramani. Bayesian Support Vector Machines for feature ranking and selection. In I. Guyon, et al., editor, *Feature Extraction, Foundations and Applications*, 2006.

G. Claeskens, C. Croux, and J. Van Kerckhoven. An information criterion for variable selection in Support Vector Machines. In I. Guyon and A. Saffari, editors, *JMLR, Special Topic on Model Selection*, volume 9, pages 541–558, Mar 2008. URL http://www.jmlr.org/papers/volume9/claeskens08a/claeskens08a.pdf.

Clopinet. Challenges in machine learning, 2004-2009. URL http://clopinet.com/challenges.

C. Dahinden. An improved Random Forests approach with application to the performance prediction challenge datasets. In I. Guyon, et al., editor, *Hands on Pattern Recognition*. Microtome, 2009.

M. Debruyne, M. Hubert, and J. Suykens. Model selection in kernel based regression using the influence function. In I. Guyon and A. Saffari, editors, *JMLR, Special Topic on Model Selection*, volume 9, pages 2377–2400, Oct 2–8. URL http://www.jmlr.org/papers/volume9/debruyne08a/debruyne08a.pdf.

Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pages 148–146. Morgan Kaufmann, 1996.

J. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression, a statistical view of boosting. *Annals of Statistics*, 28:337–374, 2000.

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software (to appear)*, 2009.

P. Germain, A. Lacasse, F. Laviolette, and M. Marchand. PAC-Bayesian learning of linear classifiers. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 353–360, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1.

Y. Guermeur. VC theory of large margin multi-category classifiers. In I. Guyon and A. Saffari, editors, *JMLR, Special Topic on Model Selection*, volume 8, pages 2551–2594, Nov 2007. URL http://www.jmlr.org/papers/volume8/guermeur07a/guermeur07a.pdf.

I. Guyon. A practical guide to model selection. In J. Marie, editor, *Machine Learning Summer School*. Springer, to appear, 2009.

I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, Editors. *Feature Extraction, Foundations and Applications*. Studies in Fuzziness and Soft Computing. With data, results and sample code for the NIPS 2003 feature selection challenge. Physica-Verlag, Springer, 2006a. URL http://clopinet.com/fextract-book/.

I. Guyon, A. Saffari, G. Dror, and J. Buhmann. Performance prediction challenge. In *IEEE/INNS conference IJCNN 2006*, Vancouver, Canada, July 16-21 2006b.

I. Guyon, A. Saffari, G. Dror, and G. Cawley. Agnostic learning vs. prior knowledge challenge. In *IEEE/INNS conference IJCNN 2007*, Orlando, Florida, August 12-17 2007.

I. Guyon, C. Aliferis, G. Cooper, A. Elisseeff, J.-P. Pellet, P. Spirtes, and A. Statnikov. Design and analysis of the causation and prediction challenge. In *JMLR W&CP*, volume 3, pages 1–33, WCCI2008 workshop on causality, Hong Kong, June 3-4 2008a. URL http://jmlr.csail.mit.edu/papers/topic/causality.html.

I. Guyon, A. Saffari, G. Dror, and G. Cawley. Analysis of the IJCNN 2007 agnostic learning vs. prior knowledge challenge. In *Neural Networks*, volume 21, pages 544–550, Orlando, Florida, March 2008b.

I. Guyon, D. Janzing, and B. Schölkopf. Causality: objectives and assessment. In *NIPS 2008 workshop on causality*, volume 7. JMLR W&CP, in press, 2009a.

I. Guyon, V. Lemaire, M. Boullé, Gideon Dror, and David Vogel. Analysis of the KDD cup 2009: Fast scoring on a large orange customer database. In *KDD cup 2009, in press*, volume 8. JMLR W&CP, 2009b.

L. E. Sucar H. J. Escalante, M. Montes. Particle swarm model selection. In I. Guyon and A. Saffari, editors, *JMLR, Special Topic on Model Selection*, volume 10, pages 405–440, Feb 2009. URL http://www.jmlr.org/papers/volume10/escalante09a/escalante09a.pdf.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning, Data Mining, Inference and Prediction*. Springer Verlag, 2000.

T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. The entire regularization path for the support vector machine. *JMLR*, 5:1391–1415, 2004. URL http://jmlr.csail.mit.edu/papers/volume5/hastie04a/hastie04a.pdf.

D. Haussler, M. Kearns, and R. Schapire. Bounds on the sample complexity of Bayesian learning using information theory and the vc dimension. *Machine Learning*, 14(1):83–113, 1994. ISSN 0885-6125.

A. E. Hoerl. Application of ridge analysis to regression problems. *Chemical Engineering Progress*, 58:54–59, 1962.

C. Hue and M. Boullé. A new probabilistic approach in rank regression with optimal Bayesian partitioning. In I. Guyon and A. Saffari, editors, *JMLR, Special Topic on Model Selection*, volume 8, pages 2727–2754, Dec 2007. URL http://www.jmlr.org/papers/volume8/hue07a/hue07a.pdf.

IBM team. Winning the KDD cup orange challenge with ensemble selection. In *KDD cup 2009, in press*, volume 8. JMLR W&CP, 2009.

R. Kohavi and G. John. Wrappers for feature selection. *Artificial Intelligence*, 97(1-2):273–324, December 1997.

I. Koo and R. M. Kil. Model selection for regression with continuous kernel functions using the modulus of continuity. In I. Guyon and A. Saffari, editors, *JMLR, Special Topic on Model Selection*, volume 9, pages 2607–2633, Nov 2008. URL http://www.jmlr.org/papers/volume9/koo08b/koo08b.pdf.

G. Kunapuli, J.-S. Pang, and K. Bennett. Bilevel cross-validation-based model selection. In I. Guyon, et al., editor, *Hands on Pattern Recognition*. Microtome, 2009.

J. Langford. Tutorial on practical prediction theory for classification. *JMLR*, 6:273–306, Mar 2005. URL http://jmlr.csail.mit.edu/papers/volume6/langford05a/langford05a.pdf.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. J. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541 – 551, 1989.

R. W. Lutz. Logitboost with trees applied to the WCCI 2006 performance prediction challenge datasets. In *Proc. IJCNN06*, pages 2966–2969, Vancouver, Canada, July 2006. INNS/IEEE.

D. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4:448–472, 1992.

O. Madani, D. M. Pennock, and G. W. Flake. Co-validation: Using model disagreement to validate classification algorithms. In *NIPS*, 2005.

M. Momma and K. Bennett. A pattern search method for model selection of Support Vector Regression. In *In Proceedings of the SIAM International Conference on Data Mining*. SIAM, 2002.

R. Neal and J. Zhang. High dimensional classification with Bayesian neural networks and dirichlet diffusion trees. In I. Guyon, et al., editor, *Feature Extraction, Foundations and Applications*, 2006.

V. Nikulin. Classification with random sets, boosting and distance-based clustering. In I. Guyon, et al., editor, *Hands on Pattern Recognition*. Microtome, 2009.

T. Poggio and F. Girosi. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247(4945):978–982, February 1990.

A. Pozdnoukhov and S. Bengio. Invariances in kernel methods: From samples to objects. *Pattern Recogn. Lett.*, 27(10):1087–1097, 2006. ISSN 0167-8655.

E. Pranckeviciene and R. Somorjai. Liknon feature selection: Behind the scenes. In I. Guyon, et al., editor, *Hands on Pattern Recognition*. Microtome, 2009.

J. Reunanen. Model selection and assessment using cross-indexing. In *Proc. IJCNN07*, Orlando, Florida, Aug 2007. INNS/IEEE.

S. Rosset and J. Zhu. Sparse, flexible and efficient modeling using L1 regularization. In I. Guyon, et al., editor, *Feature Extraction, Foundations and Applications*, 2006.

S. Rosset, J. Zhu, and T. Hastie. Boosting as a regularized path to a maximum margin classifier. *Journal of Machine Learning Research*, 5:941–973, 2004.

M. Saeed. Hybrid learning using mixture models and artificial neural networks. In I. Guyon, et al., editor, *Hands on Pattern Recognition*. Microtome, 2009.

A. Saffari and I. Guyon. Quick start guide for CLOP. Technical report, Graz University of Technology and Clopinet, May 2006. URL `http://clopinet.com/CLOP/`.

D. Schuurmans and F. Southey. Metric-based methods for adaptive model selection and regularization. *Machine Learning, Special Issue on New Methods for Model Selection and Model Combination*, 48:51–84, 2001.

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

M. Seeger. PAC-Bayesian generalisation error bounds for Gaussian process classification. *JMLR*, 3:233–269, 2003. URL `http://jmlr.csail.mit.edu/papers/volume3/seeger02a/seeger02a.pdf`.

M. Seeger. Bayesian inference and optimal design for the sparse linear model. *JMLR*, 9:759–813, 2008. ISSN 1533-7928.

P. Simard, Y. LeCun, and J. Denker. Efficient pattern recognition using a new transformation distance. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 50–58, San Mateo, CA, 1993. Morgan Kaufmann.

A. Singh, R. Nowak, and X. Zhu. Unlabeled data: Now it helps, now it doesn't. In *NIPS*, 2008.

A. Smola, S. Mika, B. Schölkopf, and R. Williamson. Regularized principal manifolds. *JMLR*, 1:179–209, 2001. URL `http://jmlr.csail.mit.edu/papers/volume1/smola01a/smola01a.pdf`.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

E. Tuv, A. Borisov, G. Runger, and K. Torkkola. Feature selection with ensembles, artificial variables, and redundancy elimination. In I. Guyon and A. Saffari, editors, *JMLR, Special Topic on Model Selection*, volume 10, pages 1341–1366, Jul 2009. URL http://www.jmlr.org/papers/volume10/tuv09a/tuv09a.pdf.

L. Valiant. A theory of the learnable. *Communications of the ACM,*, 27(11):1134–1142, 1984.

V. Vapnik. *Estimation of Dependences Based on Empirical Data [in Russian]*. Nauka, Moscow, 1979. (English translation: Springer Verlag, New York, 1982).

V. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, N.Y., 1998.

V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.*, 16:264–180, 1971.

S. Vishwanathan and A. Smola. Fast kernels for string and tree matching. In *Advances in Neural Information Processing Systems 15*, pages 569–576. MIT Press, 2003. URL http://books.nips.cc/papers/files/nips15/AA11.pdf.

G. Wahba. *Spline Models for Observational Data*, volume 59 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, Philadelphia, 1990.

A. Waibel. Consonant recognition by modular construction of large phonemic time-delay neural networks. In *NIPS*, pages 215–223, 1988.

C. Watkins. Dynamic alignment kernels. In A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 39–50, Cambridge, MA, 2000. MIT Press. URL http://www.cs.rhul.ac.uk/home/chrisw/dynk.ps.gz.

P. Werbos. Backpropagation: Past and future. In *International Conference on Neural Networks*, pages 343–353. IEEE, IEEE press, 1988.

J. Weston, A. Elisseff, B. Schoelkopf, and M. Tipping. Use of the zero norm with linear models and kernel methods. *JMLR*, 3:1439–1461, 2003.

J. Wichard. Agnostic learning with ensembles of classifiers. In *Proc. IJCNN07*, Orlando, Florida, Aug 2007. INNS/IEEE.

J. Ye, S. Ji, and J. Chen. Multi-class discriminant kernel learning via convex programming. In I. Guyon and A. Saffari, editors, *JMLR, Special Topic on Model Selection*, volume 9, pages 719–758, Apr 2008. URL http://www.jmlr.org/papers/volume9/ye08b/ye08b.pdf.

J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani. 1-norm support vector machines. In *NIPS*, 2003.

Journal of Machine Learning Research 11(Jul):2079–2107, 2010 Submitted 10/09; Revised 3/10; Published 7/10

Chapter 3

On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation

Gavin C. Cawley GCC@CMP.UEA.AC.UK
Nicola L. C. Talbot NLCT@CMP.UEA.AC.UK
School of Computing Sciences
University of East Anglia
Norwich, United Kingdom NR4 7TJ

Editor: Isabelle Guyon

Abstract

Model selection strategies for machine learning algorithms typically involve the numerical optimisation of an appropriate model selection criterion, often based on an estimator of generalisation performance, such as k-fold cross-validation. The error of such an estimator can be broken down into bias and variance components. While unbiasedness is often cited as a beneficial quality of a model selection criterion, we demonstrate that a low variance is at least as important, as a non-negligible variance introduces the potential for over-fitting in model selection as well as in training the model. While this observation is in hindsight perhaps rather obvious, the degradation in performance due to over-fitting the model selection criterion can be surprisingly large, an observation that appears to have received little attention in the machine learning literature to date. In this paper, we show that the effects of this form of over-fitting are often of comparable magnitude to differences in performance between learning algorithms, and thus cannot be ignored in empirical evaluation. Furthermore, we show that some common performance evaluation practices are susceptible to a form of selection bias as a result of this form of over-fitting and hence are unreliable. We discuss methods to avoid over-fitting in model selection and subsequent selection bias in performance evaluation, which we hope will be incorporated into best practice. While this study concentrates on cross-validation based model selection, the findings are quite general and apply to any model selection practice involving the optimisation of a model selection criterion evaluated over a finite sample of data, including maximisation of the Bayesian evidence and optimisation of performance bounds.

Keywords: model selection, performance evaluation, bias-variance trade-off, selection bias, over-fitting

3.1. Introduction

This paper is concerned with two closely related topics that form core components of best practice in both the real world application of machine learning methods and the development of novel machine learning algorithms, namely model selection and performance evaluation. The majority of machine learning algorithms are based on some form of multi-level inference, where the model is defined by a set of model parameters and also a set of hyper-parameters (Guyon et al., 2009), for example in kernel learning methods the parameters correspond to the coefficients of the kernel expansion and the hyper-parameters include the regularisation parameter, the choice of kernel function and any associated kernel parameters. This division into parameters and hyper-parameters is typically performed for computational convenience; for instance in the case of kernel machines, for fixed values of the hyper-parameters, the parameters are normally given by the solution of a convex optimisation problem for which efficient algorithms are available. Thus it makes sense to take advantage of this structure and fit the model iteratively using a pair of nested loops, with the hyper-parameters adjusted to optimise a model selection criterion in the outer loop (model selection) and the parameters set to optimise a training criterion in the inner loop (model fitting/training). In our previous study (Cawley and Talbot, 2007), we noted that the variance of the model selection criterion admitted the possibility of over-fitting during model selection as well as the more familiar form of over-fitting that occurs during training and demonstrated that this could be ameliorated to some extent by regularisation of the model selection criterion. The first part of this paper discusses the problem of over-fitting in model selection in more detail, providing illustrative examples, and describes how to avoid this form of over-fitting in order to gain the best attainable performance, desirable in practical applications, and required for fair comparison of machine learning algorithms.

Unbiased and robust[1] performance evaluation is undoubtedly the cornerstone of machine learning research; without a reliable indication of the relative performance of competing algorithms, across a wide range of learning tasks, we cannot have the clear picture of the strengths and weaknesses of current approaches required to set the direction for future research. This topic is considered in the second part of the paper, specifically focusing on the undesirable optimistic bias that can arise due to over-fitting in model selection. This phenomenon is essentially analogous to the selection bias observed by Ambroise and McLachlan (2002) in microarray classification, due to feature selection prior to performance evaluation, and shares a similar solution. We show that some, apparently quite benign, performance evaluation protocols in common use by the machine learning community are susceptible to this form of bias, and thus potentially give spurious results. In order to avoid this bias, model selection must be treated as an integral part of the model fitting process and performed afresh every time a model is fitted to a new sample of data. Furthermore, as the differences in performance due to model selection are shown to be often of comparable magnitude to the difference in performance between learning algorithms, it seems no longer meaningful to evaluate the performance of machine learning algorithms in isolation, and we should instead compare learning algorithm/model selection procedure combinations. However, this means that robust unbiased performance evaluation is likely to require more rigorous and computationally intensive protocols, such a nested cross-validation or “double cross” (Stone, 1974).

None of the methods or algorithms discussed in this paper are new; the novel contribution of this work is an empirical demonstration that over-fitting at the second level of inference (i.e., model selection) can have a very substantial deleterious effect on the generalisation performance of state-of-the-art machine learning algorithms. Furthermore the demonstration that this can lead to a misleading optimistic bias in performance evaluation using evaluation protocols in common use in the machine learning community is also novel. The paper is intended to be of some tutorial value in promoting best practice in model selection and performance evaluation, however we also hope that the observation that over-fitting in model selection is a significant problem will encourage much needed algorithmic and theoretical development in this area.

The remainder of the paper is structured as follows: Section 3.2 provides a brief overview of the kernel ridge regression classifier used as the base classifier for the majority of the experimental work and Section 3.3 describes the data sets used. Section 3.4 demonstrates the importance of the variance of the model selection criterion, as it can lead to over-fitting in

1. The term “robust” is used here to imply insensitivity to irrelevant experimental factors, such as the sampling and partitioning of the data to form training, validation and test sets; this is normally achieved by computationally expensive resampling schemes, for example, cross-validation (Stone, 1974) and the bootstrap (Efron and Tibshirani, 1994).

model selection, resulting in poor generalisation performance. A number of methods to avoid over-fitting in model selection are also discussed. Section 3.5 shows that over-fitting in model selection can result in biased performance evaluation if model selection is not viewed as an integral part of the modelling procedure. Two apparently benign and widely used performance evaluation protocols are shown to be affected by this problem. Finally, the work is summarised in Section 3.6.

3.2. Kernel Ridge Regression

In this section, we provide a brief overview of the Kernel Ridge Regression (KRR) classifier (Saunders et al., 1998), also known as the Least-Squares Support Vector Machine (Suykens et al., 2002), Regularised Least Squares (Rifkin and Lippert, 2007), Regularisation Network (Poggio and Girosi, 1990) etc., used as the base classifier in most of the empirical demonstrations in the sequel. Assume we are given labeled training data, $\mathscr{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{\ell}$, where $\boldsymbol{x}_i \in \mathscr{X} \subset \mathbb{R}^d$ is a vector of input features describing the i^{th} example and $y_i \in \{-1, +1\}$ is an indicator variable such that $y_i = +1$ if the i^{th} example is drawn from the positive class, $\mathscr{C}^+$, and $y_i = -1$ if from the negative class, $\mathscr{C}^-$. Further let us assume there are ℓ^+ positive examples and $\ell^- = \ell - \ell^+$ negative examples. The Kernel Ridge Regression classifier aims to construct a linear model $f(\boldsymbol{x}) = \boldsymbol{w} \cdot \phi(\boldsymbol{x}) + b$ in a fixed feature space, $\phi : \mathscr{X} \to \mathscr{F}$, that is able to distinguish between examples drawn from $\mathscr{C}^-$ and $\mathscr{C}^+$, such that

$$\boldsymbol{x} \in \begin{cases} \mathscr{C}^+ & \text{if } f(\boldsymbol{x}) \geq 0 \\ \mathscr{C}^- & \text{otherwise} \end{cases}.$$

However, rather than specifying the feature space, $\mathscr{F}$, directly, it is induced by a kernel function, $\mathscr{K} : \mathscr{X} \times \mathscr{X} \to \mathbb{R}$, giving the inner product between the images of vectors in the feature space, $\mathscr{F}$, that is, $\mathscr{K}(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}')$. A common kernel function, used throughout this study, is the Gaussian radial basis function (RBF) kernel

$$\mathscr{K}(\boldsymbol{x}, \boldsymbol{x}') = \exp\left\{-\eta \|\boldsymbol{x} - \boldsymbol{x}'\|^2\right\}, \tag{3.1}$$

where η is a kernel parameter controlling the sensitivity of the kernel function. However, the interpretation of the kernel function as evaluating the inner product between points in an implied feature space is valid for any kernel for which the kernel matrix $\boldsymbol{K} = [k_{ij} = \mathscr{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1}^{\ell}$ is positive definite (Mercer, 1909), such that

$$\boldsymbol{a}^T \boldsymbol{K} \boldsymbol{a} > 0, \qquad \forall\, \boldsymbol{a} \neq \boldsymbol{0}.$$

The model parameters $(\boldsymbol{w}, b)$ are given by the minimum of a regularised (Tikhonov and Arsenin, 1977) least-squares loss function,

$$\mathscr{L} = \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{1}{2\lambda}\sum_{i=1}^{\ell}\left[y_i - \boldsymbol{w} \cdot \phi(\boldsymbol{x}_i) - b\right]^2, \tag{3.2}$$

where λ is a regularisation parameter controlling the bias-variance trade-off (Geman et al., 1992). The accuracy of the kernel machine on test data is critically dependent on the choice of good values for the *hyper-parameters*, in this case λ and η. The search for the optimal values for such hyper-parameters is a process known as *model selection*. The representer theorem (Kimeldorf and Wahba, 1971) states that the solution to this optimisation problem can be written as an expansion of the form

$$\boldsymbol{w} = \sum_{i=1}^{\ell} \alpha_i \phi(\boldsymbol{x}_i) \qquad \Longrightarrow \qquad f(\boldsymbol{x}) = \sum_{i=1}^{\ell} \alpha_i \mathscr{K}(\boldsymbol{x}_i, \boldsymbol{x}) + b.$$

The dual parameters of the kernel machine, $\boldsymbol{\alpha}$, are then given by the solution of a system of linear equations,

$$\begin{bmatrix} \boldsymbol{K}+\lambda\boldsymbol{I} & \boldsymbol{1} \\ \boldsymbol{1}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix} = \begin{bmatrix} \boldsymbol{y} \\ 0 \end{bmatrix}. \tag{3.3}$$

where $\boldsymbol{y} = (y_1, y_2, \dots, y_\ell)^T$, which can be solved efficiently via Cholesky factorisation of $\boldsymbol{K} + \lambda\boldsymbol{I}$, with a computational complexity of $\mathscr{O}(\ell^3)$ operations (Suykens et al., 2002). The simplicity and efficiency of the kernel ridge regression classifier makes it an ideal candidate for relatively small-scale empirical investigations of practical issues, such as model selection.

3.2.1. Efficient Leave-One-Out Cross-Validation

Cross-validation (e.g., Stone, 1974) provides a simple and effective method for both model selection and performance evaluation, widely employed by the machine learning community. Under k-fold cross-validation the data are randomly partitioned to form k disjoint subsets of approximately equal size. In the ith fold of the cross-validation procedure, the ith subset is used to estimate the generalisation performance of a model trained on the remaining $k-1$ subsets. The average of the generalisation performance observed over all k folds provides an estimate (with a slightly pessimistic bias) of the generalisation performance of a model trained on the entire sample. The most extreme form of cross-validation, in which each subset contains only a single pattern is known as leave-one-out cross-validation (Lachenbruch and Mickey, 1968; Luntz and Brailovsky, 1969). An attractive feature of kernel ridge regression is that it is possible to perform leave-one-out cross-validation in closed form, with minimal cost as a by-product of the training algorithm (Cawley and Talbot, 2003). Let $\boldsymbol{C}$ represent the matrix on the left hand side of (3.3), then the residual error for the ith training pattern in the ith fold of the leave-one-out process is given by,

$$r_i^{(-i)} = y_i - \hat{y}_i^{(-i)} = \frac{\alpha_i}{\boldsymbol{C}_{ii}^{-1}},$$

where $\hat{y}_i^{(-j)}$ is the output of the kernel ridge regression machine for the ith observation in the jth fold of the leave-one-out procedure and $\boldsymbol{C}_{ii}^{-1}$ is the ith element of the principal diagonal of the inverse of the matrix $\boldsymbol{C}$. Similar methods have been used in least-squares linear regression for many years, (e.g., Stone, 1974; Weisberg, 1985). While the optimal model parameters of the kernel machine are given by the solution of a simple system of linear equations, (3.3), some form of model selection is required to determine good values for the *hyper-parameters*, $\boldsymbol{\theta} = (\lambda, \eta)$, in order to maximise generalisation performance. The analytic leave-one-out cross-validation procedure described here can easily be adapted to form the basis of an efficient model selection strategy (cf. Chapelle et al., 2002; Cawley and Talbot, 2003; Bo et al., 2006). In order to obtain a continuous model selection criterion, we adopt Allen's Predicted REsidual Sum-of-Squares (PRESS) statistic (Allen, 1974),

$$\mathrm{PRESS}(\boldsymbol{\theta}) = \sum_{i=1}^{\ell} \left[r_i^{(-i)} \right]^2.$$

The PRESS criterion can be optimised efficiently using scaled conjugate gradient descent (Williams, 1991) or Nelder-Mead simplex (Nelder and Mead, 1965) procedures. For full details of the training and model selection procedures for the kernel ridge regression classifier, see Cawley (2006). A public domain MATLAB implementation of the kernel ridge regression classifier, including automated model selection, is provided by the Generalised Kernel Machine (GKM) (Cawley et al., 2007) toolbox.[2]

2. Toolbox can be found at http://theoval.cmp.uea.ac.uk/$\sim$gcc/projects/gkm.

3.3. Data Sets used in Empirical Demonstrations

In this section, we describe the benchmark data sets used in this study to illustrate the problem of over-fitting in model selection and to demonstrate the bias this can introduce into performance evaluation.

3.3.1. A Synthetic Benchmark

A synthetic benchmark, based on that introduced by Ripley (1996), is used widely in the next section to illustrate the nature of over-fitting in model selection. The data are drawn from four spherical bivariate Gaussian distributions, with equal probability. All four Gaussians have a common variance, $\sigma^2 = 0.04$. Patterns belonging to the positive classes are drawn from Gaussians centred on $[+0.4, +0.7]$ and $[-0.3, +0.7]$; the negative patterns are drawn from Gaussians centred on $[-0.7, +0.3]$ and $[+0.3, +0.3]$. Figure 3.1 shows a realisation of the synthetic benchmark, consisting of 256 patterns, showing the Bayes-optimal decision boundary and contours representing an a-posteriori probability of belonging to the positive class of 0.1 and 0.9. The Bayes error for this benchmark is approximately 12.38%. This benchmark is useful firstly as the Bayes optimal decision boundary is known, but also because it provides an inexhaustible supply of data, allowing the numerical approximation of various expectations.

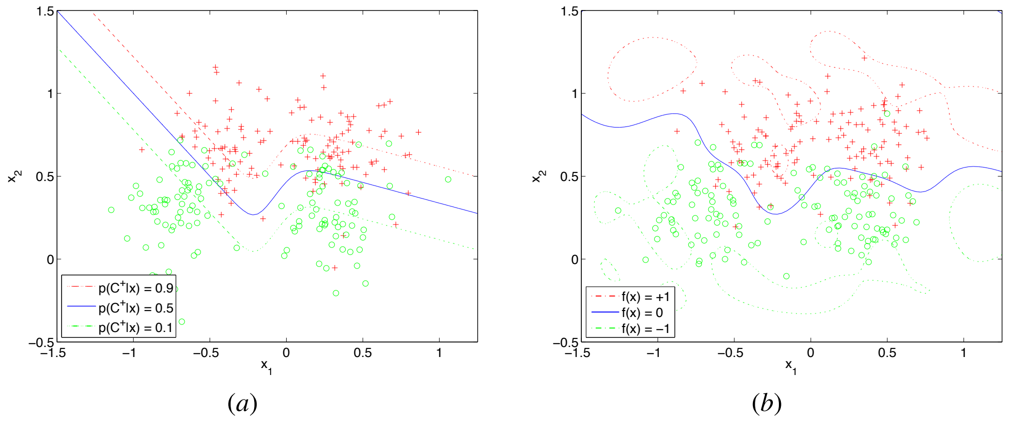


Figure 3.1: Realisation of the Synthetic benchmark data set, with Bayes optimal decision boundary (*a*) and kernel ridge regression classifier with an automatic relevance determination (ARD) kernel where the hyper-parameters are tuned so as to minimise the true test MSE (*b*).

3.3.2. A Suite of Benchmarks for Robust Performance Evaluation

In addition to illustrating the nature of over-fitting in model selection, we need to demonstrate that it is a serious concern in practical applications and show that it can result in biased performance evaluation if not taken into consideration. Table 3.1 gives the details of a suite of thirteen benchmark data sets, introduced by Rätsch et al. (2001). Each benchmark is based on a data set from the UCI machine learning repository, augmented by a set of 100 pre-defined partitions to form multiple realisations of the training and test sets (20 in the case of the larger `image` and `splice` data sets). The use of multiple benchmarks means that the evaluation is more robust as the selection of data sets that provide a good match to the inductive bias of a particular classifier becomes less likely. Likewise, the use of multiple partitions provides robustness against sensitivity to the sampling of data to form training and test sets. Results on this suite of benchmarks thus provides a reasonable indication of the magnitude of the effects of over-fitting in model selection that we might expect to see in practice.

Table 3.1: Details of data sets used in empirical comparison.

Data Set	**Training Patterns**	**Testing Patterns**	**Number of Replications**	**Input Features**
banana	400	4900	100	2
breast cancer	200	77	100	9
diabetis	468	300	100	8
flare solar	666	400	100	9
german	700	300	100	20
heart	170	100	100	13
image	1300	1010	20	18
ringnorm	400	7000	100	20
splice	1000	2175	20	60
thyroid	140	75	100	5
titanic	150	2051	100	3
twonorm	400	7000	100	20
waveform	400	4600	100	21

3.4. Over-fitting in Model Selection

We begin by demonstrating that it is possible to over-fit a model selection criterion based on a finite sample of data, using the synthetic benchmark problem, where ground truth is available. Here we use "over-fitting in model selection" to mean minimisation of the model selection criterion beyond the point at which generalisation performance ceases to improve and subsequently begins to decline. Figure 3.1(*b*) shows the output of a kernel ridge regression classifier for the synthetic problem, with the Automatic Relevance Determination (ARD) variant of the Gaussian radial basis function kernel,

$$\mathscr{K}(\boldsymbol{x}, \boldsymbol{x}') = \exp\left\{-\sum_{i=1}^{d} \eta_i (x_i - x_i')^2\right\},$$

which has a separate scaling parameter, η_i, for each feature. A much larger training set of 4096 samples was used, and the hyper-parameters were tuned to minimise the true test mean squared errors (MSE). The performance of this model, achieved an error rate of 12.50%, which suggests that a model of this form is capable of approaching the Bayes error rate for this problem, at least in principle, and so there is little concern of model mis-specification.

A further one thousand independent realisations of this benchmark were generated, each consisting of 64 samples. A kernel ridge regression classifier, based on the ARD kernel, was constructed for each realisation, with the hyper-parameters tuned so as to minimise a four-fold cross-validation estimate of the mean squared error. The true generalisation performance of each model was estimated numerically using the underlying generative model of the data

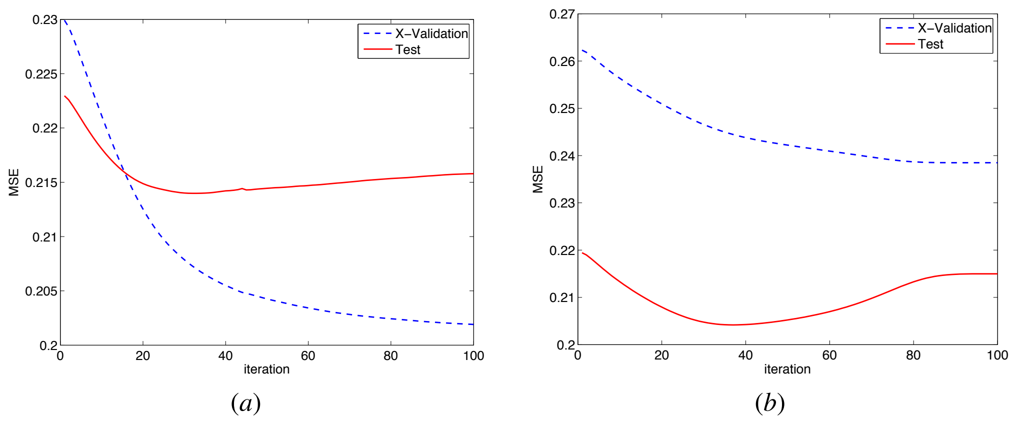


Figure 3.2: Evolution of the expected four-fold cross-validation and true test mean squared error as a function of the number of iterations (optimisation steps in the minimisation of the model selection criterion) of the model selection process, for a kernel ridge regression classifier trained on the `synthetic` benchmark data set (*a*) and (*b*) the evolution of those statistics for a particular realisation of the data set.

set. Figure 3.2(*a*) shows the expected true test and cross-validation estimates of the mean squared error averaged over all 1000 realisations of the benchmark. As would be expected, the cross-validation estimate of the mean squared error, forming the model selection criterion, is monotonically decreasing. However, the expected value of the true test MSE initially shows a decrease, as the hyper-parameters are modified in a manner that provides genuine improvements in generalisation, but after a relatively short time (approximately 30–40 iterations), the test error begins to climb slowly once more as the hyper-parameters are tuned in ways that exploit the meaningless statistical peculiarities of the sample. This produces a close analog of the classic plot used to illustrate the nature of over-fitting in training, for example, Figure 9.7 of the book by Bishop (1995). Figure 3.2 (b) shows the same statistics for one particular realisation of the data, demonstrating that the over-fitting can in some cases be quite substantial, clearly in this case some form of early-stopping in the model selection process would have resulted in improved generalisation. Having demonstrated that the classic signature of over-fitting during training is also apparent in the evolution of cross-validation and test errors during model selection, we discuss in the next section the origin of this form of over-fitting in terms of the *bias* and *variance* of the model selection criterion.

3.4.1. Bias and Variance in Model Selection

Model selection criteria are generally based on an estimator of generalisation performance evaluated over a finite sample of data, this includes resampling methods, such as split sample estimators, cross-validation (Stone, 1974) and bootstrap methods (Efron and Tibshirani, 1994), but also more loosely, the Bayesian evidence (MacKay, 1992; Rasmussen and Williams, 2006) and theoretical performance bounds such as the radius-margin bound (Vapnik, 1998). The error of an estimator can be decomposed into two components, *bias* and *variance*. Let $G(\boldsymbol{\theta})$ represent the true generalisation performance of a model with hyper-parameters $\boldsymbol{\theta}$, and $g(\boldsymbol{\theta};\mathscr{D})$ be an estimate of generalisation performance evaluated over a finite sample, $\mathscr{D}$, of n patterns. The expected squared error of the estimator can then be written in the form (Geman et al., 1992; Duda et al., 2001),

$$E_{\mathscr{D}}\left\{[g(\boldsymbol{\theta};\mathscr{D})-G(\boldsymbol{\theta})]^2\right\}=[E_{\mathscr{D}}\{g(\boldsymbol{\theta};\mathscr{D})-G(\boldsymbol{\theta})\}]^2+E_{\mathscr{D}}\left\{[g(\boldsymbol{\theta};\mathscr{D})-E_{\mathscr{D}'}\{g(\boldsymbol{\theta};\mathscr{D}')\}]^2\right\},$$

where $E_{\mathscr{D}}\{\cdot\}$ represents an expectation evaluated over independent samples, $\mathscr{D}$, of size n. The first term, the squared *bias*, represents the difference between the expected value of the estimator and the unknown value of the true generalisation error. The second term, known as the *variance*, reflects the variability of the estimator around its expected value due to the sampling of the data $\mathscr{D}$ on which it is evaluated. Clearly if the expected squared error is low, we may reasonably expect $g(\cdot)$ to perform well as a model selection criterion. However, in practice, the expected squared error may be significant, in which case, it is interesting to ask whether the bias or the variance component is of greatest importance in reliably achieving optimal generalisation.

It is straightforward to demonstrate that leave-one-out cross-validation provides an almost unbiased estimate of the true generalisation performance (Luntz and Brailovsky, 1969), and this is often cited as being an advantageous property of the leave-one-out estimator in the setting of model selection (e.g., Vapnik, 1998; Chapelle et al., 2002). However, for the purpose of model selection, rather than performance evaluation, unbiasedness *per se* is relatively unimportant, instead the primary requirement is merely for the minimum of the model selection criterion to provide a reliable indication of the minimum of the true test error in hyper-parameter space. This point is illustrated in Figure 3.3, which shows a hypothetical example of a model selection criterion that is unbiased (by construction) *(a)* and another that is clearly biased *(b)*. Unbiasedness provides the assurance that the minimum of the expected value of the model selection criterion, $E_{\mathscr{D}}\{g(\boldsymbol{\theta};\mathscr{D})\}$ coincides with the minimum of the test error, $G(\boldsymbol{\theta})$. However, in practice, we have only a finite sample of data, $\mathscr{D}_i$, over which to evaluate the model selection criterion, and so it is the minimum of $g(\boldsymbol{\theta};\mathscr{D}_i)$ that is of interest. In Figure 3.3*(a)*, it can be seen that while the estimator is unbiased, it has a high variance, and so there is a large spread in the values of θ at which the minimum occurs for different samples of data, and so $g(\boldsymbol{\theta};\mathscr{D}_i)$ is likely to provide a poor model selection criterion in practice. On the other hand, Figure 3.3*(b)* shows a criterion with lower variance, and hence is the better model selection criterion, despite being biased, as the minima of $g'(\boldsymbol{\theta};\mathscr{D}_i)$ for individual samples lie much closer to the minimum of the true test error. This demonstrates that while unbiasedness is reassuring, as it means that the form of the model selection criterion is correct *on average*, the variance of the criterion is also vitally important as it is this that ensures that the minimum of the selection criterion evaluated on a particular sample will provide good generalisation.

3.4.2. The Effects of Over-fitting in Model Selection

In this section, we investigate the effect of the variance of the model selection criterion using a more realistic example, again based on the `synthetic` benchmark, where the underlying generative model is known and so we are able to evaluate the true test error. It is demonstrated that over-fitting in model selection can cause both under-fitting and over-fitting of the training sample. A fixed training set of 256 patterns is generated, and used to train a kernel ridge regression classifier, using the simple RBF kernel (3.1), with hyper-parameter settings defining a fine grid spanning reasonable values of the regularisation and kernel parameters, λ and η respectively. The smoothed error rate (Bo et al., 2006),

$$\mathrm{SER}(\boldsymbol{\theta})=\frac{1}{2n}\sum_{i=1}^{n}[1-y_i\tanh\{\gamma f(\boldsymbol{x}_i)\}]$$

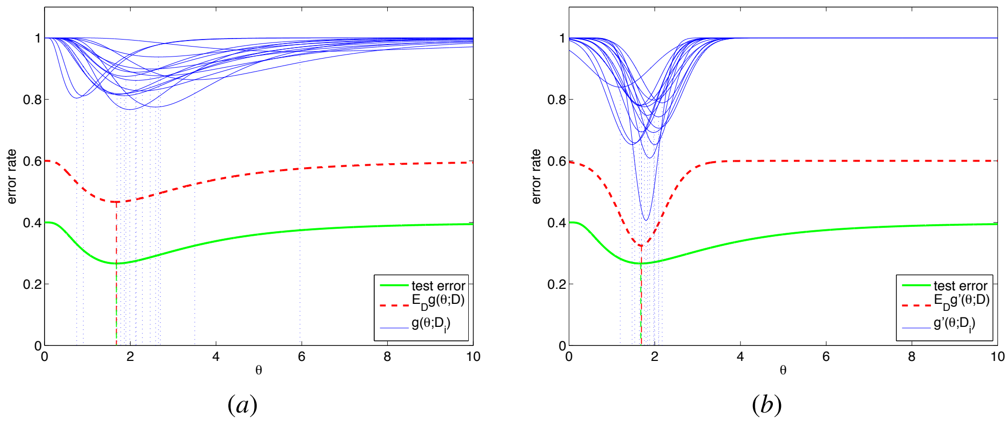
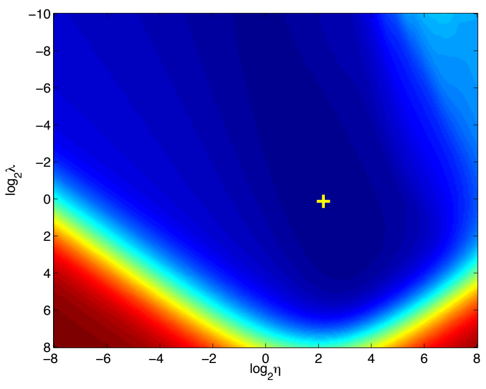


Figure 3.3: Hypothetical example of an unbiased (*a*) and a biased (*b*) model selection criterion. Note that the biased model selection criterion (*b*) is likely to provide the more effective model selection criterion as it has a lower variance, even though it is significantly biased. For clarity, the true error rate and the expected value of the model selection criteria are shown with vertical displacements of -0.6 and -0.4 respectively.

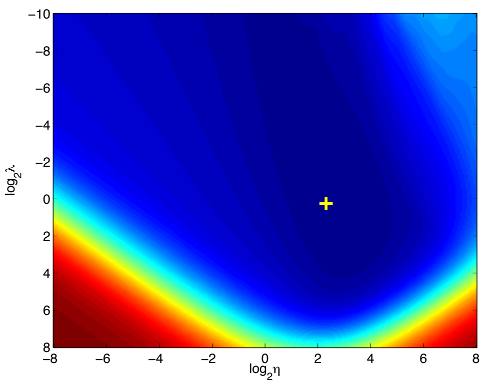
is used as the statistic of interest, in order to improve the clarity of the figures, where γ is a parameter controlling the amount of smoothing applied ($\gamma = 8$ is used throughout, however the precise value is not critical). Figure 3.4(*a*) shows the true test smoothed error rate as a function of the hyper-parameters. As these are both scale parameters, a logarithmic representation is used for both axes. The true test smoothed error rate is an approximately unimodal function of the hyper-parameters, with a single distinct minimum, indicating the hyper-parameter settings giving optimal generalisation.

In practical applications, however, the true test error is generally unknown, and so we must rely on an estimator of some sort. The simplest estimator for use in model selection is the error computed over an independent validation set, that is, the split-sample estimator. It seems entirely reasonable to expect the split-sample estimator to be unbiased. Figure 3.4(*b*) shows a plot of the mean smoothed error rate using the split-sample estimator, over 100 random validation sets, each of which consists of 64 patterns. Note that the same fixed training set is used in each case. This plot is very similar to the true smoothed error, shown in Figure 3.4(*a*), demonstrating that the split sample estimator is indeed approximately unbiased.

While the split-sample estimator is unbiased, it may have a high variance, especially as in this case the validation set is (intentionally) relatively small. Figure 3.5 shows plots of the split-sample estimate of the smoothed error rate for six selected realisations of a validation set of 64 patterns. Clearly, the split-sample error estimate is no longer as smooth, or indeed unimodal. More importantly, the hyper-parameter values selected by minimising the validation set error, and therefore the true generalisation performance, depends on the particular sample of data used to form the validation set. Figure 3.6 shows that the variance of the split-sample estimator can result in models ranging from severely under-fit (*a*) to severely over-fit (*f*), with variations in between these extremes.



(*a*)



(*b*)

Figure 3.4: Plot of the true test smoothed error rate (a) and mean smoothed error rate over 100 random validation sets of 64 samples (b), for a kernel ridge regression classifier as a function of the hyper-parameters. In each case, the minimum is shown by a yellow cross, $+$.

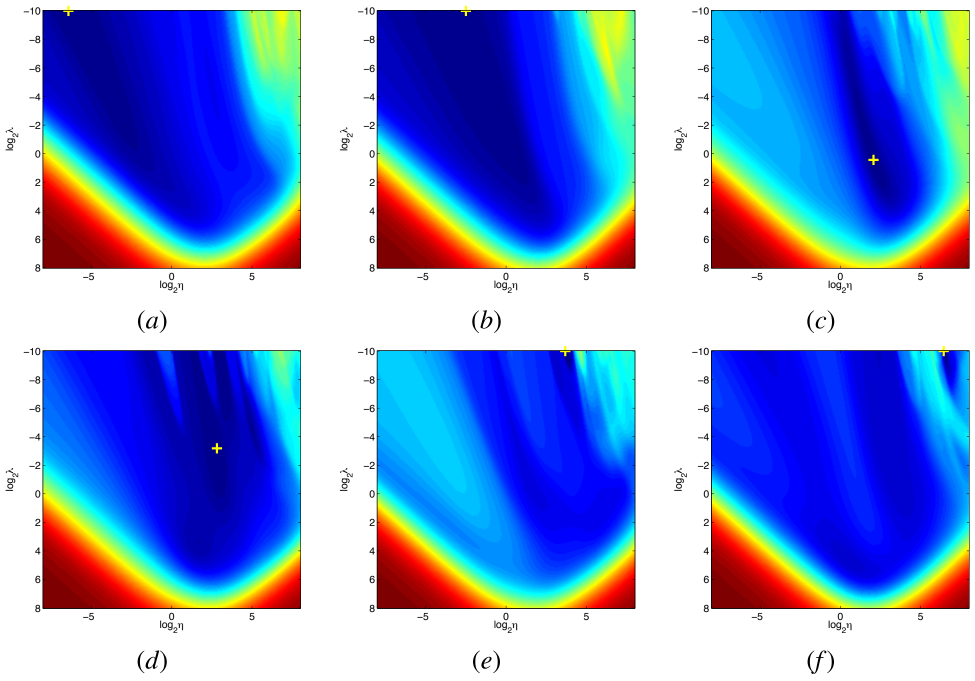


Figure 3.5: Contour plot of the split-sample estimate of the smoothed error rate for a kernel ridge regression machine as a function of the hyper-parameters, for six random realisations of the validation set. The minimum is shown by a cross, $+$.

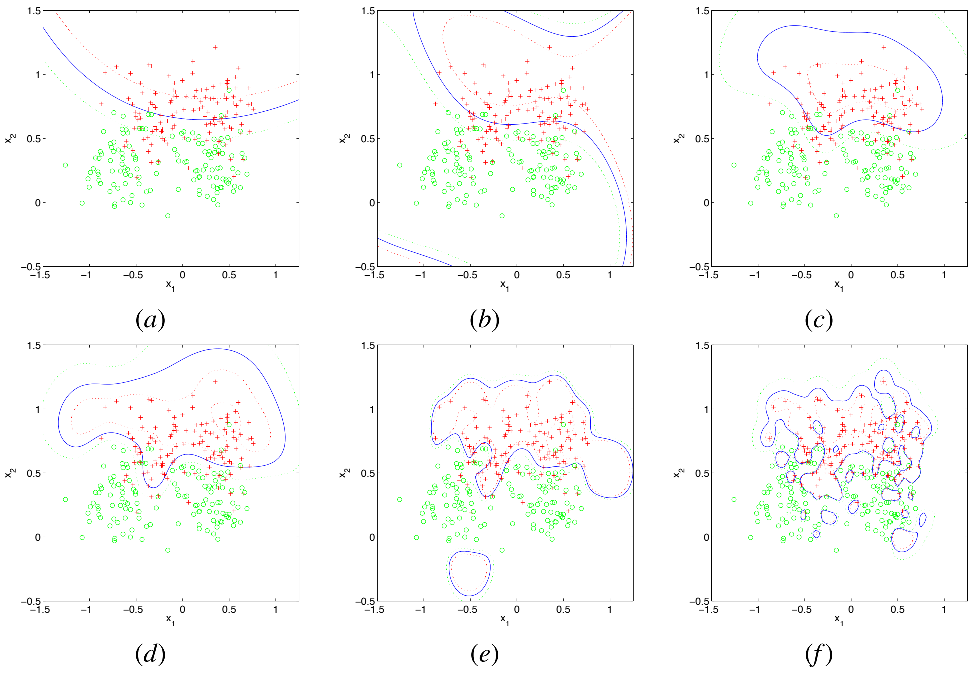


Figure 3.6: Kernel ridge regression models of the synthetic benchmark, using hyper-parameters selected according to the smoothed error rate over six random realisations of the validation set (shown in Figure 3.5). The variance of the model selection criterion can result in models ranging from under-fit, (*a*) and (*b*), through well-fitting, (*c*) and (*d*), to over-fit (*e*) and (*f*).

Figure 3.7(*a*) shows a scatter plot of the validation set and true error rates for kernel ridge regression classifiers for the synthetic benchmark, with split-sample based model selection using 100 random realisations of the validation set. Clearly, the split-sample based model selection procedure normally performs well. However, there is also significant variation in performance with different samples forming the validation set. We can also see that the validation set error is strongly biased, having been directly minimised during model selection, and (of course) should not be used for performance estimation.

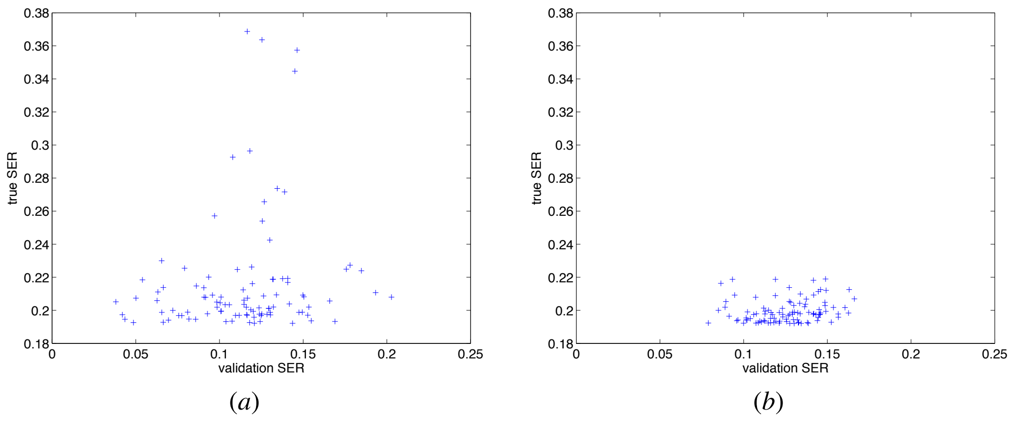


Figure 3.7: Scatter plots of the true test smoothed error rate as a function of the validation set smoothed error rate for 100 randomly generated validation sets of (*a*) 64 and (*b*) 256 patterns.

Note that in this section we have deliberately employed a split-sample based model selection strategy with a relatively high variance, due to the limited size of the validation set. A straightforward way to reduce the variance of the model selection criterion is simply to increase the size of the validation sample over which it is evaluated. Figure 3.8 shows the optimal hyper-parameter settings obtained using 100 realisations of validation sets of 64 (*a*) and 256 (*b*) samples. It can be clearly seen that the use of a larger validation set has resulted in a tighter clustering of hyper-parameter values around the true optimum, note also that the hyper-parameters are concentrated along the bottom of a broad valley in hyper-parameter space, so even when the selected values are different from the optimal value, they still lie in positions giving good generalisation. This is further illustrated in Figure 3.7(*b*), where the true smoothed error rates are much more tightly clustered, with fewer outliers, for the larger validation sets than obtained using smaller validation sets, shown in Figure 3.7(*a*).

The variation in performance for different realisations of the benchmark suggests that evaluation of machine learning algorithms should always involve multiple partitions of the data to form training/validation and test sets, as the sampling of data for a single partition of the data might arbitrarily favour one classifier over another. This is illustrated in Figure 3.9, which shows the test error rates for Gaussian Process and Kernel Logistic Regression classifiers (GPC and KLR respectively), for 100 random realisations of the `banana` benchmark data set used in Rätsch et al. (2001) (see Section 3.5.1 for details). On 64 realisations of the data GPC out-performs KLR, but on 36 KLR out-performs GPC, even though the GPC is better on average (although the difference is not statistically significant in this case). If the classifiers had been evaluated on only one of the latter 36 realisations, it might incorrectly be concluded that the

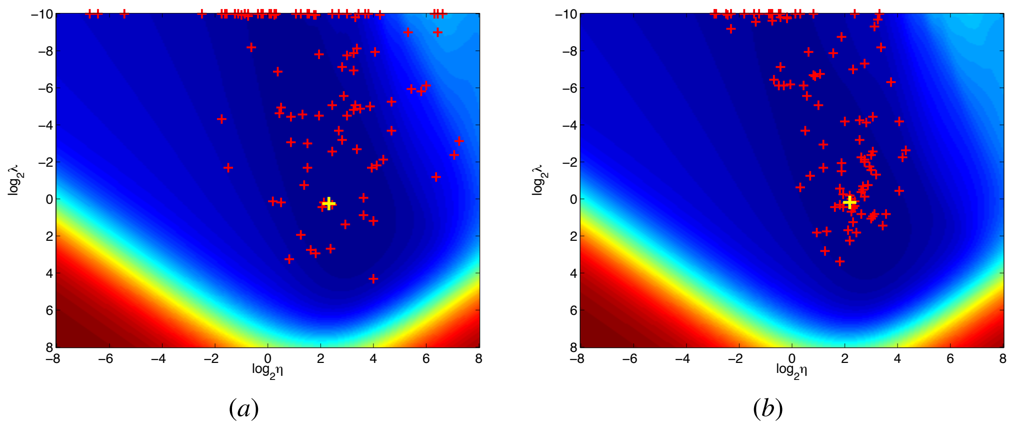


Figure 3.8: Contour plot of the mean validation set smoothed error rate over 100 randomly generated validation sets of (*a*) 64 and (*b*) 256 patterns. The minimum of the mean validation set error is marked by a yellow cross, and the minimum for each realisation of the validation set marked by a red cross.

KLR classifier is superior to the GPC for that benchmark. However, it should also be noted that a difference in performance between two algorithms is unlikely to be of *practical* significance, even if it is *statistically* significant, if it is smaller than the variation in performance due to the random sampling of the data, as it is probable that a greater improvement in performance would be obtained by further data collection than by selection of the optimal classifier.

3.4.3. Is Over-fitting in Model Selection Really a Genuine Concern in Practice?

In the preceding part of this section we have demonstrated the deleterious effects of the variance of the model selection criterion using a synthetic benchmark data set, however this is not sufficient to establish that over-fitting in model selection is actually a genuine concern in practical applications or in the development of machine learning algorithms. Table 3.2 shows results obtained using kernel ridge regression (KRR) classifiers, with RBF and ARD kernel functions over the thirteen benchmarks described in Section 3.3.2. In each case, model selection was performed independently for each realisation of each benchmark by minimising the PRESS statistic using the Nelder-Mead simplex method (Nelder and Mead, 1965). For the majority of the benchmarks, a siginicantly lower test error is achieved (according to the Wilcoxon signed ranks test) using the basic RBF kernel; the ARD kernel only achieves statistical superiority on one of the thirteen (`image`). This is perhaps a surprising result as the models are nested, the RBF kernel being a special case of the ARD kernel, so the optimal performance that can be achieved with the ARD kernel is guaranteed to be at least equal to the performance achievable using the RBF kernel. The reason for the poor performance of the ARD kernel in practice is because there are many more kernel parameters to be tuned in model selection and so many degrees of freedom available in optimising the model selection criterion. If the criterion used has a non-negligible variance, this includes optimisations exploiting the statistical peculiarities of the particular sample of data over which it is evaluated, and hence there will be more scope for over-fitting. Table 3.2 also shows the mean value of the PRESS statistic, following model selection, the fact that the majority of ARD models display a lower value for the PRESS statistic

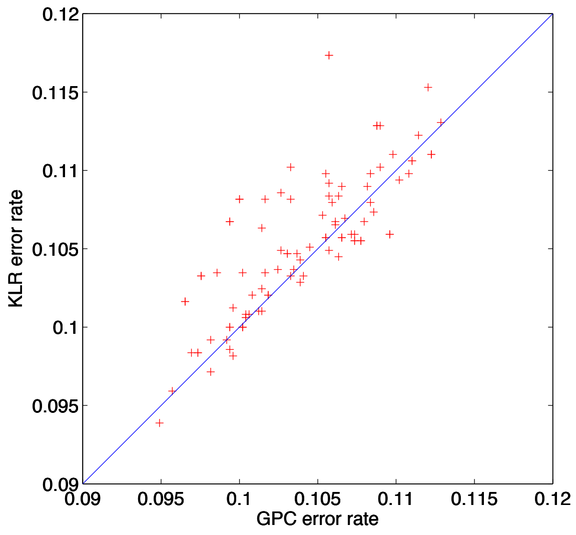


Figure 3.9: Scatter plots of the test set error for Gaussian process and Kernel Logistic regression classifiers (GPC and KLR respectively) for 100 realisations of the `banana` benchmark.

than the corresponding RBF model, while exhibiting a higher test error rate, is a strong indication of over-fitting the model selection criterion. This is a clear demonstration that over-fitting in model selection can be a significant problem in practical applications, especially where there are many hyper-parameters or where only a limited supply of data is available.

Table 3.3 shows the results of the same experiment performed using expectation-propagation based Gaussian process classifiers (EP-GPC) (Rasmussen and Williams, 2006), where the hyper-parameters are tuned independently for each realisation, for each benchmark individually by maximising the Bayesian evidence. While the leave-one-out cross-validation based PRESS criterion is known to exhibit a high variance, the variance of the evidence (which is also evaluated over a finite sample of data) is discussed less often. We find again here that the RBF covariance function often out-performs the more general ARD covariance function, and again the test error rate is often negatively correlated with the evidence for the models. This indicates that over-fitting the evidence is also a significant practical problem for the Gaussian process classifier.

3.4.4. Avoiding Over-fitting in Model Selection

It seems reasonable to suggest that over-fitting in model selection is possible whenever a model selection criterion evaluated over a finite sample of data is directly optimised. Like over-fitting in training, over-fitting in model selection is likely to be most severe when the sample of data is small and the number of hyper-parameters to be tuned is relatively large. Likewise, assuming additional data are unavailable, potential solutions to the problem of over-fitting the model selection criterion are likely to be similar to the tried and tested solutions to the problem of over-fitting the training criterion, namely regularisation (Cawley and Talbot, 2007), early stopping (Qi et al., 2004) and model or hyper-parameter averaging (Cawley, 2006; Hall and Robinson, 2009). Alternatively, one might minimise the number of hyper-parameters, for instance by

Table 3.2: Error rates of kernel ridge regression (KRR) classifier over thirteen benchmark data sets (Rätsch et al., 2001), using both standard radial basis function (RBF) and automatic relevance determination (ARD) kernels. Results shown in bold indicate an error rate that is statistically superior to that obtained with the same classifier using the other kernel function, or a PRESS statistic that is significantly lower.

Data Set	**Test Error Rate**		**PRESS**	
	RBF	**ARD**	**RBF**	**ARD**
banana	10.610 ± 0.051	10.638 ± 0.052	60.808 ± 0.636	60.957 ± 0.624
breast cancer	**26.727** ± **0.466**	28.766 ± 0.391	70.632 ± 0.328	**66.789** ± **0.385**
diabetis	**23.293** ± **0.169**	24.520 ± 0.215	146.143 ± 0.452	**141.465** ± **0.606**
flare solar	34.140 ± 0.175	34.375 ± 0.175	267.332 ± 0.480	**263.858** ± **0.550**
german	**23.540** ± **0.214**	25.847 ± 0.267	228.256 ± 0.666	**221.743** ± **0.822**
heart	**16.730** ± **0.359**	22.810 ± 0.411	42.576 ± 0.356	**37.023** ± **0.494**
image	2.990 ± 0.159	**2.188** ± **0.134**	74.056 ± 1.685	**44.488** ± **1.222**
ringnorm	**1.613** ± **0.015**	2.750 ± 0.042	28.324 ± 0.246	**27.680** ± **0.231**
splice	10.777 ± 0.144	9.943 ± 0.520	186.814 ± 2.174	**130.888** ± **6.574**
thyroid	4.747 ± 0.235	4.693 ± 0.202	9.099 ± 0.152	**6.816** ± **0.164**
titanic	22.483 ± 0.085	22.562 ± 0.109	48.332 ± 0.622	**47.801** ± **0.623**
twonorm	**2.846** ± **0.021**	4.292 ± 0.086	**32.539** ± **0.279**	35.620 ± 0.490
waveform	**9.792** ± **0.045**	11.836 ± 0.085	61.658 ± 0.596	**56.424** ± **0.637**

Table 3.3: Error rates of expectation propagation based Gaussian process classifiers (EP-GPC), using both standard radial basis function (RBF) and automatic relevance determination (ARD) kernels. Results shown in bold indicate an error rate that is statistically superior to that obtained with the same classifier using the other kernel function or evidence that is significantly higher.

Data Set	**Test Error Rate**		**-Log Evidence**	
	RBF	**ARD**	**RBF**	**ARD**
banana	**10.413** ± **0.046**	10.459 ± 0.049	116.894 ± 0.917	**116.459** ± **0.923**
breast cancer	**26.506** ± **0.487**	27.948 ± 0.492	110.628 ± 0.366	**107.181** ± **0.388**
diabetis	**23.280** ± **0.182**	23.853 ± 0.193	230.211 ± 0.553	**222.305** ± **0.581**
flare solar	34.200 ± 0.175	**33.578** ± **0.181**	394.697 ± 0.546	**384.374** ± **0.512**
german	**23.363** ± **0.211**	23.757 ± 0.217	359.181 ± 0.778	**346.048** ± **0.835**
heart	**16.670** ± **0.290**	19.770 ± 0.365	73.464 ± 0.493	**67.811** ± **0.571**
image	2.817 ± 0.121	**2.188** ± **0.076**	205.061 ± 1.687	**123.896** ± **1.184**
ringnorm	**4.406** ± **0.064**	8.589 ± 0.097	121.260 ± 0.499	**91.356** ± **0.583**
splice	11.609 ± 0.180	**8.618** ± **0.924**	365.208 ± 3.137	**242.464** ± **16.980**
thyroid	4.373 ± 0.219	4.227 ± 0.216	25.461 ± 0.182	**18.867** ± **0.170**
titanic	22.637 ± 0.134	22.725 ± 0.133	78.952 ± 0.670	**78.373** ± **0.683**
twonorm	**3.060** ± **0.034**	4.025 ± 0.068	45.901 ± 0.577	**42.044** ± **0.610**
waveform	**10.100** ± **0.047**	11.418 ± 0.091	105.925 ± 0.954	**91.239** ± **0.962**

treating kernel parameters as simply parameters and optimising them at the first level of inference and have a single regularisation hyper-parameter controlling the overall complexity of the model. For very small data sets, where the problem of over-fitting in both learning and model selection is greatest, the preferred approach would be to eliminate model selection altogether and opt for a fully Bayesian approach, where the hyper-parameters are integrated out rather than optimised (e.g., Williams and Barber, 1998). Another approach is simply to avoid model selection altogether using an ensemble approach, for example the Random Forest (RF) method (Breiman, 2001). However, while such methods often achieve state-of-the-art performance, it is often easier to build expert knowledge into hierarchical models, for example through the design of kernel or covariance functions, so unfortunately approaches such as the RF are not a panacea.

While the problem of over-fitting in model selection is of the same nature as that of over-fitting at the first level of inference, the lack of mathematical tractability appears to have limited the theoretical analysis of model selection via optimisation of a model selection criterion. For example, regarding leave-one-out cross-validation, Kulkarni et al. (1998) comment "In spite of the practical importance of this estimate, relatively little is known about its properties. *The available theory is especially poor when it comes to analysing parameter selection based on minimizing the deleted estimate*." (our emphasis). While some asymptotic results are available (Stone, 1977; Shao, 1993; Toussaint, 1974), these are not directly relevant to the situation considered here, where over-fitting occurs due to optimising the values of hyper-parameters using a model selection criterion evaluated over a finite, often quite limited, sample of data. Estimates of the variance of the cross-validation error are available for some models (Luntz and Brailovsky, 1969; Vapnik, 1982), however Bengio and Grandvalet (2004) have shown there is no unbiased estimate of the variance of (k-fold) cross-validation. More recently bounds on the error of leave-one-out cross-validation based on the idea of *stability* have been proposed (Kearns and Ron, 1999; Bousquet and Elisseeff, 2002; Zhang, 2003). In this section, we have demonstrated that over-fitting in model selection is a genuine problem in machine learning, and hence is likely to be an area that could greatly benefit from further theoretical analysis.

3.5. Bias in Performance Estimation

Avoiding potentially significant bias in performance evaluation, arising due to over-fitting in model selection, is conceptually straightforward. The key is to treat both training *and* model selection together, as integral parts of the model fitting procedure and ensure they are never performed separately at any point of the evaluation process. We present two examples of potentially biased evaluation protocols that do not adhere to this principle. The scale of the bias observed on some data sets is much larger than difference in performance between learning algorithms, and so one could easily draw incorrect inferences based on the results obtained. This highlights the importance of this issue in empirical studies. We also demonstrate that the magnitude of the bias depends on the learning and model selection algorithms involved in the comparison and that combinations that are more prone to over-fitting in model selection are favored by biased protocols. This means that studies based on potentially biased protocols are not internally consistent, even if it is acknowledged that a bias with respect to other studies may exist.

3.5.1. An Unbiased Performance Evaluation Methodology

We begin by describing an unbiased performance protocol, that correctly accounts for any over-fitting that may occur in model selection. Three classifiers are evaluated using an unbiased protocol, in which model selection is performed separately for each realisation of each data set.

This is termed the "internal" protocol as the model selection process is performed independently within each fold of the resampling procedure. In this way, the performance estimate includes a component properly accounting for the error introduced by over-fitting the model selection criterion. The classifiers used were as follows: RBF-KRR—kernel ridge regression with a radial basis function kernel, with model selection based on minimisation of Allen's PRESS statistic, as described in Section 3.2. RBF-KLR—kernel logistic regression with a radial basis function kernel and model selection based on an approximate leave-one-out cross-validation estimate of the log-likelihood (Cawley and Talbot, 2008). EP-GPC—expectation-propagation based Gaussian process classifier, with an isotropic squared exponential covariance function, with model selection based on maximising the marginal likelihood (e.g., Rasmussen and Williams, 2006). The mean error rates obtained using these classifiers under an unbiased protocol are shown in Table 3.4. In this case, the mean ranks of all methods are only minimally different, and so there is little if any evidence for a statistically significant superiority of any of the classifiers over any other. Figure 3.10 shows a critical difference diagram (Demšar, 2006), providing a graphical illustration of this result. A critical difference diagram displays the mean rank of a set of classifiers over a suite of benchmark data sets, with cliques of classifiers with statistically similar performance connected by a bar. The critical difference in average ranks required for a statistical superiority of one classifier over another is also shown, labelled "CD".

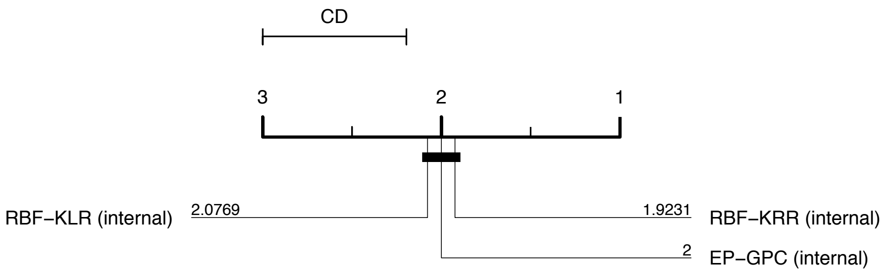


Figure 3.10: Critical difference diagram (Demšar, 2006) showing the average ranks of three classifiers with internal model selection protocol.

It is not unduly surprising that there should be little evidence for any statistically significant superiority, as all three methods give rise to structurally similar models. The models though differ significantly in their model selection procedures, the EP-GPC is based on stronger statistical assumptions, and so can be expected to excel where these assumptions are justified, but poorly where the model is mis-specified (e.g., the ringnorm benchmark). The cross-validation based model selection procedures, on the other hand, are more pragmatic and being based on much weaker assumptions might be expected to provide a more consistent level of accuracy.

3.5.2. An Example of Biased Evaluation Methodology

The performance evaluation protocol most often used in conjunction with the suite of benchmark data sets, described in Section 3.3.2, seeks to perform model selection independently for only the first five realisation of each data set. The median values of the hyper-parameters over these five folds are then determined and subsequently used to evaluate the error rates for each realisation. This "median" performance evaluation protocol was introduced in the same paper that popularised this suite of benchmark data sets (Rätsch et al., 2001) and has been widely

Table 3.4: Error rate estimates of three classifiers over a suite of thirteen benchmark data sets: The results for each method are presented in the form of the mean error rate over test data for 100 realisations of each data set (20 in the case of the `image` and `splice` data sets), along with the associated standard error.

Data Set	**GPC (internal)**	**KLR (internal)**	**KRR (internal)**
banana	10.413 ± 0.046	10.567 ± 0.051	10.610 ± 0.051
breast cancer	26.506 ± 0.487	26.636 ± 0.467	26.727 ± 0.466
diabetis	23.280 ± 0.182	23.387 ± 0.180	23.293 ± 0.169
flare solar	34.200 ± 0.175	34.197 ± 0.170	34.140 ± 0.175
german	23.363 ± 0.211	23.493 ± 0.208	23.540 ± 0.214
heart	16.670 ± 0.290	16.810 ± 0.315	16.730 ± 0.359
image	2.817 ± 0.121	3.094 ± 0.130	2.990 ± 0.159
ringnorm	4.406 ± 0.064	1.681 ± 0.031	1.613 ± 0.015
splice	11.609 ± 0.180	11.248 ± 0.177	10.777 ± 0.144
thyroid	4.373 ± 0.219	4.293 ± 0.222	4.747 ± 0.235
titanic	22.637 ± 0.134	22.473 ± 0.103	22.483 ± 0.085
twonorm	3.060 ± 0.034	2.944 ± 0.042	2.846 ± 0.021
waveform	10.100 ± 0.047	9.918 ± 0.043	9.792 ± 0.045

adopted (e.g., Mika et al., 1999; Weston, 1999; Billings and Lee, 2002; Chapelle et al., 2002; Chu et al., 2003; Stewart, 2003; Mika et al., 2003; Gold et al., 2005; Peña Centeno and D., 2006; Andelić et al., 2006; An et al., 2007; Chen et al., 2009). The original motivation for this protocol was that the internal model selection protocol was prohibitively expensive using workstations available (Rätsch, 2006), which was perfectly reasonable at the time, but is no longer true.[3] The use of the median, however, can be expected to introduce an optimistic bias into the performance estimates obtained using this "median" protocol. Firstly all of the training data comprising the first five realisations have been used during the model selection process for the classifiers used in every fold of the re-sampling. This means that some of the test data for each fold is no longer statistically "pure" as it has been seen during model selection. Secondly, and more importantly, the median operation acts as a variance reduction step, so the median of the five sets of hyper-parameters is likely to be better on average than any of the five from which it is derived. Lastly, as the hyper-parameters are now fixed, there is no longer scope for over-fitting the model selection criterion due to peculiarities of the sampling of data for the training and test partitions in each realisation.

We begin by demonstrating that the results using the internal and median protocols are not commensurate, and so the results obtained using different methods are not directly comparable. Table 3.5 shows the error rate obtained using the RBF-KRR classifier with the internal and median performance evaluation protocols and the resulting bias, that is, the difference between the mean error rates obtained with the internal and median protocols. It is clearly seen that the median protocol introduces a positive bias on almost all benchmarks (`twonorm` and `waveform` being the exceptions) and that the bias can be quite substantial on some benchmarks. Indeed, for several benchmarks, `breast cancer`, `german`, `heart` and `thyroid` in particular, the bias is larger than the typical difference in performance between classifiers evaluated using an unbiased protocol. Demšar (2006) recommends the Wilcoxon signed ranks test for determina-

3. All of the experimental results presented in this paper were obtained using a single modern Linux workstation.

Table 3.5: Error rate estimates of three classifiers over a suite of thirteen benchmark data sets: The results for each method are presented in the form of the mean error rate over test data for 100 realisations of each data set (20 in the case of the image and splice data sets), along with the associated standard error.

Data Set	**KRR (internal)**	**KRR (median)**	**Bias**
banana	10.610 ± 0.051	10.384 ± 0.042	0.226 ± 0.034
breast cancer	26.727 ± 0.466	26.377 ± 0.441	0.351 ± 0.195
diabetis	23.293 ± 0.169	23.150 ± 0.157	0.143 ± 0.074
flare solar	34.140 ± 0.175	34.013 ± 0.166	0.128 ± 0.082
german	23.540 ± 0.214	23.380 ± 0.220	0.160 ± 0.067
heart	16.730 ± 0.359	15.720 ± 0.306	1.010 ± 0.186
image	2.990 ± 0.159	2.802 ± 0.129	0.188 ± 0.095
ringnorm	1.613 ± 0.015	1.573 ± 0.010	0.040 ± 0.010
splice	10.777 ± 0.144	10.763 ± 0.137	0.014 ± 0.055
thyroid	4.747 ± 0.235	4.560 ± 0.200	0.187 ± 0.100
titanic	22.483 ± 0.085	22.407 ± 0.102	0.076 ± 0.077
twonorm	2.846 ± 0.021	2.868 ± 0.017	-0.022 ± 0.014
waveform	9.792 ± 0.045	9.821 ± 0.039	-0.029 ± 0.020

tion of the statistical significance of the superiority of one classifier over another over multiple data sets. Applying this test to the data shown for EP-GPC (internal), RBF-KLR (internal) and RBF-KRR (median), from Tables 3.4 and 3.5, reveals that the RBF-KRR (median) classifier is statistically superior to the remaining classifiers, at the 95% level of significance. A critical difference diagram, summarising this result is shown in Figure 3.12. However, the difference in performance is entirely spurious as it is purely the result of reducing the effects of over-fitting in model selection and does not reflect the true operational performance of the combination of classifier and model selection method. It is clear then that results obtained using the internal and median protocols are not directly comparable, and so reliable inferences cannot be drawn by comparison of results from different studies, using biased and unbiased protocols.

3.5.2.1. Is the Bias Solely due to Inadvertent Re-use of Test Samples?

One explanation for the observed bias of the median protocol is that some of the training samples for the first five realisations of the benchmark, which have been used in tuning the hyper-parameters, also appear in the test sets for other realisations of the benchmark used for performance analysis. In this section, we demonstrate that this inadvertent re-use of test samples is not the only cause of the bias. One hundred replications of the internal and median protocol were performed using the `synthetic` benchmark, for which an inexhaustible supply of i.i.d. data is available. However, in this case in each realisation, 100 training sets of 64 patterns and a large test set of 4096 samples were generated, all mutually disjoint. This means the only remaining source of bias is the amelioration of over-fitting in model selection by the reduction of variance by taking the median of the hyper-parameters over the first five folds (cf. Hall and Robinson, 2009). Figure 3.11 shows the mean test errors for the internal and median protocols over 100 replications, showing a very distinct optimistic bias in the median protocol (statistically highly significant according to the Wilcoxon signed ranks test, $p < 0.001$), even though there is absolutely no inadvertent re-use of test data.

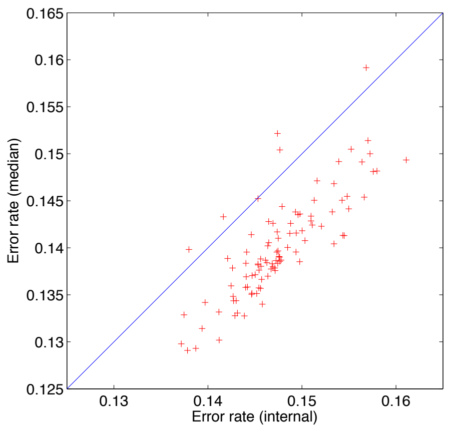


Figure 3.11: Mean error rates for the internal and median evaluation protocols for the `synthetic` benchmark, without inadvertent re-use of test data.

3.5.2.2. IS THE MEDIAN PROTOCOL INTERNALLY CONSISTENT?

Having established that the median protocol introduces an optimistic bias, and that the results obtained using the internal and median protocols do not give comparable results, we next turn our attention to whether the median protocol is internally consistent, that is, does the median protocol give the correct rank order of the classifiers? Table 3.6 shows the performance of three classifiers evaluated using the median protocol; the corresponding critical difference diagram is shown in Figure 3.13. In this case the difference in performance between classifiers is not statistically significant according to the Friedman test, however it can clearly be seen that the bias of the median protocol has favored one classifier, namely the RBF-KRR, much more strongly than the others. It seems feasible then that the bias of the median protocol may be sufficient in other cases to amplify a small difference in performance, due perhaps to an accidentally favorable choice of data sets, to the point where it spuriously appears to be statistically significant. This suggests that the median protocol may be unreliable and perhaps should be deprecated.

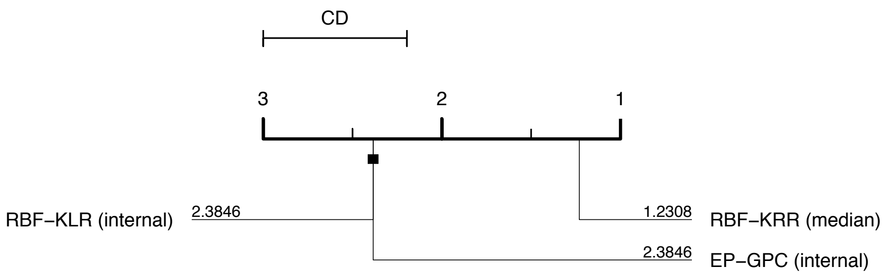


Figure 3.12: Critical difference diagram (Demšar, 2006) showing the average ranks of three classifiers, EP-GPC and RBF-KLR with internal model selection protocol and RBF-KLR using the optimistically biased median protocol (cf. Figure 3.10).

Table 3.6: Error rate estimates of three classifiers over a suite of thirteen benchmark data sets: The results for each method are presented in the form of the mean error rate over test data for 100 realisations of each data set (20 in the case of the image and splice data sets), along with the associated standard error.

Data Set	**EP-GPC (median)**	**RBF-KLR (median)**	**RBF-KRR (median)**
banana	10.371 ± 0.045	10.407 ± 0.047	10.384 ± 0.042
breast cancer	26.117 ± 0.472	26.130 ± 0.474	26.377 ± 0.441
diabetis	23.333 ± 0.191	23.300 ± 0.177	23.150 ± 0.157
flare solar	34.150 ± 0.170	34.212 ± 0.176	34.013 ± 0.166
german	23.160 ± 0.216	23.203 ± 0.218	23.380 ± 0.220
heart	16.400 ± 0.273	16.120 ± 0.295	15.720 ± 0.306
image	2.851 ± 0.102	3.030 ± 0.120	2.802 ± 0.129
ringnorm	4.400 ± 0.064	1.574 ± 0.011	1.573 ± 0.010
splice	11.607 ± 0.184	11.172 ± 0.168	10.763 ± 0.137
thyroid	4.307 ± 0.217	4.040 ± 0.221	4.560 ± 0.200
titanic	22.490 ± 0.095	22.591 ± 0.135	22.407 ± 0.102
twonorm	3.241 ± 0.039	3.068 ± 0.033	2.868 ± 0.017
waveform	10.163 ± 0.045	9.888 ± 0.042	9.821 ± 0.039

Table 3.7: Results of a statistical analysis of the bias introduced by the median protocol into the test error rates for RBF-KRR and RBF-EP-GPC, using the Wilcoxon signed ranks test.

Data Set	**RBF-KRR bias**	**RBF-EP-GPC bias**	**Wilcoxon p-value**
banana	0.226 ± 0.034	0.043 ± 0.012	< 0.05
breast cancer	0.351 ± 0.195	0.390 ± 0.186	0.934
diabetis	0.143 ± 0.074	-0.053 ± 0.051	< 0.05
flare solar	0.128 ± 0.082	0.050 ± 0.090	0.214
german	0.160 ± 0.067	0.203 ± 0.051	0.458
heart	1.010 ± 0.186	0.270 ± 0.120	< 0.05
image	0.188 ± 0.095	-0.035 ± 0.032	0.060
ringnorm	0.040 ± 0.010	0.006 ± 0.002	< 0.05
splice	0.014 ± 0.055	0.002 ± 0.014	0.860
thyroid	0.187 ± 0.100	0.067 ± 0.064	0.159
titanic	0.076 ± 0.077	0.147 ± 0.090	0.846
twonorm	-0.022 ± 0.014	-0.180 ± 0.032	< 0.05
waveform	-0.029 ± 0.020	-0.064 ± 0.022	0.244

Next, we perform a statistical analysis to determine whether there is a statistically significant difference in the magnitude of the biases introduced by the median protocol for different classifiers, for each benchmark data set.[4] First the bias introduced by the use of the median protocol was computed for the RBF KRR and RBF EP-GPC classifiers as the difference between

4. We are grateful to an anonymous reviewers for suggesting this particular form of analysis.

the test set error estimated by the internal and median protocols. The Wilcoxon signed rank test was then used to determine whether there is a statistically significant difference in the bias, over the 100 realisations of the benchmark (20 in the case of the `image` and `splice` benchmarks) . The results obtained are shown in Table 3.7, the p-value is below 0.05 for five of the thirteen benchmarks, indicating that in each case the median protocol is significantly biased in favour of the RBF KRR classifier. Clearly, as the median protocol does not impose a commensurate bias on the estimated test error rates for different classifiers, it does not provide a reliable protocol for comparing the performance of machine learning algorithms.

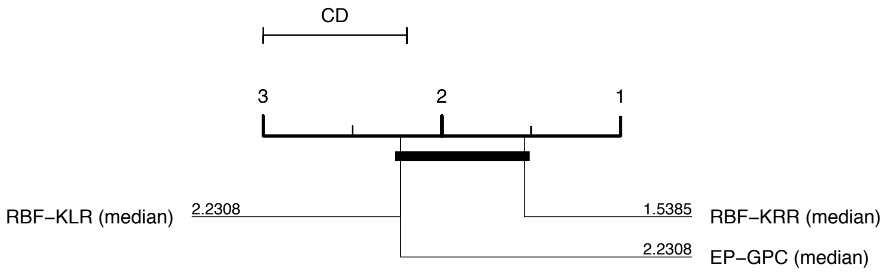


Figure 3.13: Critical difference diagram showing the average ranks of three classifiers with the median model selection protocol (cf. Figure 3.10).

In the final illustration of this section, we show that the magnitude of the bias introduced by the median protocol is greater for model selection criteria with a high variance. This means the median protocol favors most the least reliable model selection procedures and as a result does not provide a reliable indicator even of relative performance of classifier-model selection procedures combinations. Again the RBF-KRR model is used as the base classifier, however in this case a repeated split-sample model selection criterion is used, where the data are repeatedly split at random to form disjoint training and validation sets in proportions 9:1, and the hyper-parameters tuned to optimise the average mean-squared error over the validation sets. In this way, the variance of the model selection criterion can be controlled by varying the number of repetitions, with the variance decreasing as the number of folds becomes larger. Figure 3.14(*a*) shows a plot of the average ranks of EP-GPC and RBF-KLR classifiers, with model selection performed as in previous experiments, and RBF-KRR with repeated split-sample model selection, as a function of the number of folds. In each case the unbiased internal evaluation protocol was used. Clearly if the number of folds is small (five or less), the RBF-KRR model performs poorly, due to over-fitting in model selection due to the high variance of the criterion used. However, as the number of folds increases, the variance of the model selection criterion falls, and the performances of all three algorithms are very similar. Figure 3.14(*b*) shows the corresponding result using the biased median protocol. The averaging of hyper-parameters reduces the apparent variance of the model selection criterion, and this disguises the poor performance of the RBF-KRR model when the number of folds is small. This demonstrates that the bias introduced by the median protocol favors most the worst model selection criterion, which is a cause for some concern.

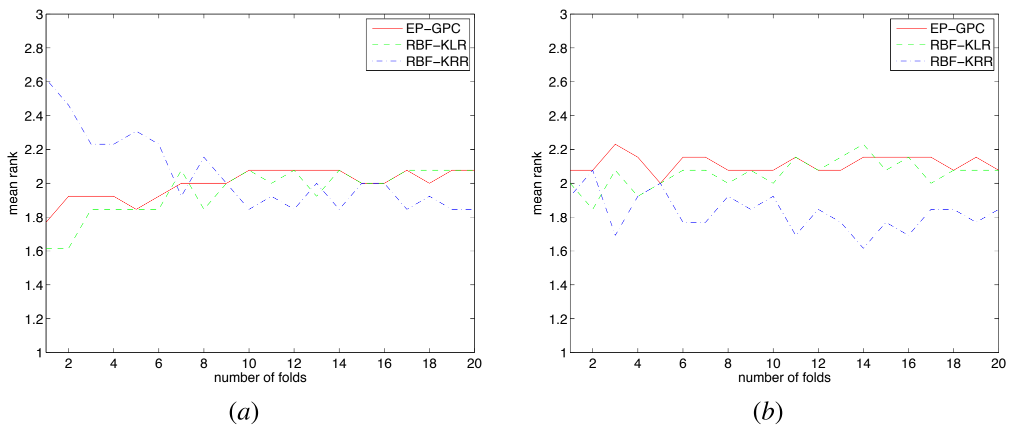


Figure 3.14: Mean ranks of three classifiers as a function of the number of folds used in the repeated split sample model selection procedure employed by the kernel ridge regression (RBF-KRR) machine, using (*a*) the unbiased *internal* protocol and (*b*) the biased *median* protocol.

3.5.3. Another Example of Biased Evaluation Methodology

In a biased evaluation protocol, occasionally observed in machine learning studies, an initial model selection step is performed using all of the available data, often interactively as part of a “preliminary study”. The data are then repeatedly re-partitioned to form one or more pairs of random, disjoint design and test sets. These are then used for performance evaluation *using the same fixed set of hyper-parameter values*. This practice may seem at first glance to be fairly innocuous, however the test data are no longer statistically pure, as they have been “seen” by the models in tuning the hyper-parameters. This would not present a serious problem were it not for the danger of over-fitting in model selection, which means that in practice the hyper-parameters will inevitably be tuned to an extent in ways that take advantage of the statistical peculiarities of this particular set of data rather than only in ways that favor improved generalisation. As a result the hyper-parameter settings retain a partial “memory” of the data that now form the test partition. We should therefore expect to observe an optimistic bias in the performance estimates obtained in this manner.

Table 3.8 shows a comparison of 10-fold cross-validation estimates of the test error rate, for kernel ridge regression with a Gaussian radian basis function kernel, obtained using protocols where the model selection stage is either *external* or *internal* to the cross-validation procedure. In the external protocol, model selection is performed once using the entire design set, as described above. In the internal protocol, the model selection step is performed separately in each fold of the cross-validation. The internal cross-validation procedure therefore provides a more realistic estimate of the performance of the combination of model selection and learning algorithm that is actually used to construct the final model. The table also shows the relative bias (i.e., the mean difference between the internal and external cross-validation protocols). The external protocol clearly exhibits a consistently optimistic bias with respect to the more rigorous internal cross-validation protocol, over all thirteen benchmarks. Furthermore, the bias is statistically significant (i.e., larger than twice the standard error of the estimate) for all benchmarks, apart from `splice` and `twonorm`. In many cases, the bias is of similar magnitude to the

Table 3.8: Error rate estimates for kernel ridge regression over thirteen benchmark data sets, for model selection schemes that are internal and external to the cross-validation process. The results for each approach and the relative bias are presented in the form of the mean error rate over for 100 realisations of each data set (20 in the case of the image and splice data sets), along with the associated standard error.

Data Set	External	Internal	Bias
banana	10.355 ± 0.146	10.495 ± 0.158	0.140 ± 0.035
breast cancer	26.280 ± 0.232	27.470 ± 0.250	1.190 ± 0.135
diabetis	22.891 ± 0.127	23.056 ± 0.134	0.165 ± 0.050
flare solar	34.518 ± 0.172	34.707 ± 0.179	0.189 ± 0.051
german	23.999 ± 0.117	24.217 ± 0.125	0.219 ± 0.045
heart	16.335 ± 0.214	16.571 ± 0.220	0.235 ± 0.073
image	3.081 ± 0.102	3.173 ± 0.112	0.092 ± 0.035
ringnorm	1.567 ± 0.058	1.607 ± 0.057	0.040 ± 0.014
splice	10.930 ± 0.219	11.170 ± 0.280	0.240 ± 0.152
thyroid	3.743 ± 0.137	4.279 ± 0.152	0.536 ± 0.073
titanic	22.167 ± 0.434	22.487 ± 0.442	0.320 ± 0.077
twonorm	2.480 ± 0.067	2.502 ± 0.070	0.022 ± 0.021
waveform	9.613 ± 0.168	9.815 ± 0.183	0.203 ± 0.064

typical difference observed between competitive learning algorithms (cf. Table 3.4). In some cases, for example, `banana` and `thyroid` benchmarks, the bias is of a surprising magnitude, likely to be large enough to conceal even the true difference between even state-of-the-art and uncompetitive learning algorithms. This clearly shows that the external cross-validation protocol exhibits a consistent optimistic bias, potentially of a very substantial magnitude even when the number of hyper-parameters is small (in this case only two), and so should not be used in practice.

3.6. Conclusions

In this paper, we have discussed the importance of bias and variance in model selection and performance evaluation, and demonstrated that a high variance can lead to over-fitting in model selection, and hence poor performance, even when the number of hyper-parameters is relatively small. Furthermore, we have shown that a potentially severe form of selection bias can be introduced into performance evaluation by protocols that have been adopted in a number of existing empirical studies. Fortunately, it seems likely that over-fitting in model selection can be overcome using methods that have already been effective in preventing over-fitting during training, such as regularisation or early stopping. Little attention has so far been focused on over-fitting in model selection, however in this paper we have shown that it presents a genuine pitfall in the practical application of machine learning algorithms and in empirical comparisons. In order to overcome the bias in performance evaluation, model selection should be viewed as an integral part of the model fitting procedure, and should be conducted independently in each trial in order to prevent selection bias and because it reflects best practice in operational use. Rigorous performance evaluation therefore requires a substantial investment of processor time in order to evaluate performance over a wide range of data sets, using multiple randomised partitionings of the available data, with model selection performed separately in each trial. However, it is straightforward to fully automate these steps, and so requires little manual involvement. Performance evaluation according to these principles requires repeated training of models using different sets of hyper-parameter values on different samples of the available data, and so is also well-suited to parallel implementation. Given the recent trend in processor design towards multi-core designs, rather than faster processor speeds, rigorous performance evaluation is likely to become less and less time-consuming, and so there is little justification for the continued use of potentially biased protocols.

Acknowledgments

The authors would like to thank Gareth Janacek, Wenjia Wang and the anonymous reviewers for their helpful comments on earlier drafts of this paper, and the organisers and participants of the WCCI-2006 Performance Prediction Challenge and workshop that provided the inspiration for our work on model selection and performance prediction. G. C. Cawley is supported by the Engineering and Physical Sciences Research Council (EPSRC) grant EP/F010508/1 - Advancing Machine Learning Methodology for New Classes of Prediction Problems.

References

D. M. Allen. The relationship between variable selection and prediction. *Technometrics*, 16: 125–127, 1974.

C. Ambroise and G. J. McLachlan. Selection bias in gene extraction on the basis of microarray gene-expression data. *Proceedings of the National Academy of Sciences*, 99(10):6562–6566, May 14 2002. doi: 10.1073/pnas.102102699.

S. An, W. Liu, and S. Venkatesh. Fast cross-validation algorithms for least squares support vector machines and kernel ridge regression. *Pattern Recognition*, 40(8):2154–2162, August 2007. doi: 10.1016/j.patcog.2006.12.015.

E. Andelić, M. Schafföner, M. Katz, S. E. Krüger, and A. Wendermuth. Kernel least-squares models using updates of the pseudoinverse. *Neural Computation*, 18(12):2928–2935, December 2006. doi: 10.1162/neco.2006.18.12.2928.

Y. Bengio and Y. Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. 5:1089–1105, 2004.

S. A. Billings and K. L. Lee. Nonlinear Fisher discriminant analysis using a minimum squared error cost function and the orthogonal least squares algorithm. *Neural Networks*, 15(2):263–270, March 2002. doi: 10.1016/S0893-6080(01)00142-3.

C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

L. Bo, L. Wang, and L. Jiao. Feature scaling for kernel Fisher discriminant analysis using leave-one-out cross validation. *Neural Computation*, 18(4):961–978, April 2006. doi: 10.1162/neco.2006.18.4.961.

O. Bousquet and A. Elisseeff. Stability and generalization. *Journal of Machine Learning Research*, 2:499–526, 2002.

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, October 2001. doi: 10.1023/A:1010933404324.

G. C. Cawley. Leave-one-out cross-validation based model selection criteria for weighted LS-SVMs. In *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks (IJCNN-06)*, pages 1661–1668, Vancouver, BC, Canada, July 16–21 2006. doi: 10.1109/IJCNN.2006.246634.

G. C. Cawley and N. L. C. Talbot. Efficient leave-one-out cross-validation of kernel Fisher discriminant classifiers. *Pattern Recognition*, 36(11):2585–2592, November 2003. doi: 10.1016/S0031-3203(03)00136-5.

G. C. Cawley and N. L. C. Talbot. Preventing over-fitting during model selection via Bayesian regularisation of the hyper-parameters. *Journal of Machine Learning Research*, 8:841–861, April 2007.

G. C. Cawley and N. L. C. Talbot. Efficient approximate leave-one-out cross-validation for kernel logistic regression. *Machine Learning*, 71(2–3):243–264, June 2008. doi: 10.1007/s10994-008-5055-9.

G. C. Cawley, G. J. Janacek, and N. L. C. Talbot. Generalised kernel machines. In *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks (IJCNN-07)*, pages 1720–1725, Orlando, Florida, USA, August 12–17 2007. doi: 10.1109/IJCNN.2007.4371217.

O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1–3):131–159, January 2002. doi: 10.1023/A:1012450327387.

H. Chen, P. Tino, and X. Yao. Probabilistic classification vector machines. *IEEE Transactions on Neural Networks*, 20(6):901–914, June 2009. doi: 10.1109/TNN.2009.2014161.

W. Chu, S. S. Keerthi, and C. J. Ong. Bayesian trigonometric support vector classifier. *Neural Computation*, 15(9):2227–2254, September 2003. doi: 10.1162/089976603322297368.

J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. John Wiley and Sons, second edition, 2001.

B. Efron and R. J. Tibshirani. *Introduction to the bootstrap*. Monographs on Statistics and Applied Probability. Chapman & Hall, 1994.

S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, January 1992. doi: 10.1162/neco.1992.4.1.1.

C. Gold, A. Holub, and P. Sollich. Bayesian approach to feature selection and parameter tuning for support vector machine classifiers. *Neural Networks*, 18(5):693–701, July/August 2005. doi: 10.1016/j.neunet.2005.06.044.

I. Guyon, A. Saffari, G. Dror, and G. Cawley. Model selection: Beyond the Bayesian/frequentist divide. *Journal of Machine Learning Research*, 11:61–87, 2009.

P. Hall and A. P. Robinson. Reducing the variability of crossvalidation for smoothing parameter choice. *Biometrika*, 96(1):175–186, March 2009. doi: doi:10.1093/biomet/asn068.

M. Kearns and D. Ron. Algorithmic stability and sanity-check bounds for leave-one-out cross-validation. *Neural Computation*, 11(6):1427–1453, August 1999. doi: 10.1162/089976699300016304.

G. S. Kimeldorf and G. Wahba. Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33:82–95, 1971.

S. R. Kulkarni, G. Lugosi, and S. S. Venkatesh. Learning pattern classification — a survey. *IEEE Transactions on Information Theory*, 44(6):2178–2206, October 1998.

P. A. Lachenbruch and M. R. Mickey. Estimation of error rates in discriminant analysis. *Technometrics*, 10(1):1–12, February 1968.

A. Luntz and V. Brailovsky. On estimation of characters obtained in statistical procedure of recognition (in Russian). *Techicheskaya Kibernetica*, 3, 1969.

D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, May 1992. doi: 10.1162/neco.1992.4.3.415.

J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London, Series A*, 209: 415–446, 1909.

S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K.-R. Müller. Fisher discriminant analysis with kernels. In *Neural Networks for Signal Processing IX, Proceedings of the 1999 IEEE Signal Processing Society Workshop*, pages 41–48, Maddison, WI, USA, 21–25 August 1999. doi: 10.1109/NNSP.1999.788121.

S. Mika, G. Rätsch, J. Weston, B. Schölkpf, and K.-R. Müller. Contructing descriptive and discriminative nonlinear features: Rayleigh coefficients in kernel feature spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):623–628, May 2003. doi: 10.1109/TPAMI.2003.1195996.

J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7: 308–313, 1965.

T. Peña Centeno and Lawrence N. D. Optimising kernel parameters and regularisation coefficients for non-linear discriminant analysis. *Journal of Machine Learning Research*, 7: 455–491, February 2006.

T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, September 1990. doi: 10.1109/5.58326.

Y. Qi, T. P. Minka, R. W. Picard, and Z. Ghahramani. Predictive automatic relevance determination by expectation propagation. In *Proceedings of the Twenty First International Conference on Machine Learning (ICML-04)*, pages 671–678, Banff, Alberta, Canada, July 4–8 2004.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, 2006.

G. Rätsch, 2006. Personal communication.

G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for AdaBoost. *Machine Learning*, 42(3): 287–320, March 2001. doi: 10.1023/A:1007618119488.

R. M. Rifkin and R. A. Lippert. Notes on regularized least squares. Technical Report MIT-CSAIL-TR-2007-025, Computer Science and Artificial Intelligence Laboratory, MIT, May 2007.

B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

C. Saunders, A. Gammerman, and V. Vovk. Ridge regression learning algorithm in dual variables. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML-98)*, pages 515–521. Morgan Kaufmann, 1998.

J. Shao. Linear model selection by cross-validation. *Journal of the American Statistical Society*, 88:486–494, 1993.

I. Stewart. On the optimal parameter choice for ν-support vector machines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10):1274–1284, October 2003. doi: 10.1109/TPAMI.2003.1233901.

M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society, Series B (Statistical Methodology)*, 36(2):111–147, 1974.

M. Stone. Asymptotics for and against cross-validation. *Biometrika*, 64(1):29–35, April 1977. doi: 10.1093/biomet/64.1.29.

J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vanderwalle. *Least squares support vector machine*. World Scientific Publishing Company, Singapore, 2002. ISBN 981-238-151-1.

A. N. Tikhonov and V. Y. Arsenin. *Solutions of ill-posed problems*. John Wiley, New York, 1977.

G. Toussaint. Bibliography on estimation of misclassification. *IEEE Transactions on Information Theory*, IT-20(4):472–479, July 1974.

V. N. Vapnik. *Estimation of dependences based on empirical data*. Springer, 1982.

V. N. Vapnik. *Statistical learning theory*. Adaptive and learning systems for signal processing, communications and control series. Wiley, 1998.

S. Weisberg. *Applied linear regression*. Probability and Mathematical Statistics. John Wiley & Sons, second edition, 1985.

J. Weston. Leave-one-out support vector machines. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 727–733, San Fransisco, CA, USA, 1999. Morgan Kaufmann.

C. K. I. Williams and D. Barber. Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351, December 1998. doi: 10.1109/34.735807.

P. M. Williams. A Marquardt algorithm for choosing the step size in backpropagation learning with conjugate gradients. Technical Report CSRP-229, University of Sussex, February 1991.

T. Zhang. Leave-one-out bounds for kernel machines. *Neural Computation*, 15(6):1397–1437, June 2003. doi: 10.1162/089976603321780326.

Part II

Data representation

Overview

Every pattern recognition problem starts with data encoding and preprocessing. In our challenges, we alleviated the task of the participants by providing data already preprocessed as feature vectors. Here are some examples of feature coding we used:

- Categorical variables were represented with a simple disjunctive code. For 4 levels, were get 4 features taking values 1 0 0 0, 0 1 0 0, 0 0 1 0, or 0 0 0 1.
- Images of handwritten digits were represented as pixel maps, after centering and scaling the digit.
- Pharmaceutical molecules were represented as binary features indicating presence or absence of certain groups of atoms.
- Texts from newsgroups were represented as features indicating the frequency of appearance of word.

In this part of the book, we selected a few techniques employed by the participants to improve upon the data representations provided in the challenges, to illustrate various aspects of preprocessing.

In Chapter 4, **Mehreen Saeed proposes to use clustering methods** to simplify the data representation and reduce dimensionality before performing supervised learning. **It has been a long time debate whether it is beneficial to perform unsupervised preprocessing** to reduce data dimensionality. In addition to clustering, such techniques include Principal Component Analysis (PCA), Independent Component Analysis (ICA) and many other principal subspace methods, linear or non-linear. While space dimensionality reduction has gained a lot of popularity in the recent years, in supervised learning challenges, their have not prevailed as preprocessing methods. This seems to be largely due to the fact that modern supervised learning techniques are robust against overfitting and embed their own "implicit" space dimensionality reduction, without performing "hard decisions" discarding dimensions at an early stage. For instance, the popular "ridge regression" method penalizes dimensions corresponding to small eigen values of the data correlation matrix, thus performing an implicit selection according to principal components, like PCA. Yet the excellent performance of Mehreen Saeed in the ALvsPK challenge (first on NOVA, third on SYLVA in the agnostic track) reveal that well conducted unsupervised learning may yield good results, with the additional benefit of gaining in data understanding and ease of visualization.

In Chapter 5, **Marc Boullé presents advanced discretization techniques** providing a unified framework for representing data as piecewise constant distributions, including methods for optimally discretizing continuous variables and for grouping values of variables, which are already discrete (including categorical variables). The paper follows a new methodology based on data dependent Bayesian priors. Discretization may be performed as a preprocessing step to classification techniques requiring discrete variables, such as the Naive Bayes algorithm. The benefits of discretization include data compression, which may play a role in overfitting avoidance, similarly to space dimensionality reduction. In this paper, "data grids" are obtained in a supervised manner and may be used directly for classification, or as preprocessing to other classifiers. Data grids lend themseves to deriving simple rules of classification, like decision trees, facilitating the understanding of the classification process. Marc Boullé consistently obtained good results with his methods, ranking first on ADA and SYLVA in the performance prediction challenge.

In Chapter 6, **Chloé-Agathe Azencott and Pierre Baldi shows the benefit using low level representations.** In the ALvsPK challenge, the participants of the "agnostic learning" (AL) track used the data representations provided by the organizers (all low-level feature representations) and those of the "prior knowedge" (PK) track constructed their own representation, starting from raw data, and using their own domain knowledge. All the top ranking participant in the PK track ended up using low level representation, namely many easy-to-extract features not incorporating a lot of domain knowledge. To win first place on the HIVA dataset in the prior knowledge track, Azencott and Baldi used features encoding molecular connectivity, detecting the presence of certain molecule subgraphs. In contrast, other participants who used higher level features crafted by phamacology experts, did not obtain as good results. Similarly, on NOVA (text processing) the winner in the PK track used a bag-of-word representation, a variant of the low-level representation proposed in the AL track by the organizers. No use was made of word semantics nor grammatical constructs.

Chapter 4

Hybrid Learning Using Mixture Models and Artificial Neural Networks

Mehreen Saeed MEHREEN.SAEED@NU.EDU.PK
Department of Computer Science
National University of Computer and Emerging Sciences
Lahore Campus, Pakistan.

Editor: Isabelle Guyon, Gavin Cawley, Gideon Dror and Amir Saffari

Abstract

This chapter describes a hybrid approach to learning using mixture models and artificial neural networks. Mixture models provide a semi-parametric approach for density estimation of data. We show how these mixtures can be used for feature transformation, producing a huge reduction in the dimensionality of the initial input space. The transformed features are fed into a neural network for classification. We have explored the potential of using Bernoulli mixture models for binary data and Gaussian mixtures for continuous data. The hybrid learning model was applied to five datasets which were launched as part of the "Agnostic vs. Prior Knowledge" challenge organized by the International Joint Conference on Neural Networks in 2007. Our model achieved the best result on the NOVA dataset in the agnostic learning track and good results on the other four datasets.

Keywords: Bernoulli mixture models, Gaussian mixture models, artificial neural networks, hybrid learning, dimensionality reduction

4.1. Introduction

Hybrid learning involves the integration of an unsupervised learning technique with a supervised learning method. Learning takes place in two stages. In the first stage an unsupervised method is used to determine data clusters. The data clusters can be determined using any appropriate clustering method, e.g., partitional or agglomerative technique. In the second stage a transformation of data is performed using the learned clusters and a supervised learning algorithm is used to learn a function that discriminates between different class labels (Alpaydin, 2005). The supervised layer can be built from any suitable learning method like neural networks, support vector machines and decision trees, etc.

The use of hybrid models for learning is not new. A radial basis function (RBF) network is an example of a hybrid model that uses local RBF functions at the input layer and its output is used in supervised learning of labels or classes (Moody and Darken, 1989). Mixture of experts is another example where a function is approximated using very simple local approximation functions (Jacobs et al., 1991). The potential of combining generative models with discriminative classifiers has also been discussed by Jaakkola and Haussler (1998), Ulusoy and Bishop (2005) and Lasserre et al. (2006). They argue that both models have different properties and characteristics and therefore their advantages can be exploited by combining them into a hybrid model.

This chapter describes how we can construct a hybrid learning model by building the unsupervised layer using mixture models and the supervised layer using artificial neural networks.

Mixture models involving Gaussian distributions have been used extensively for density estimation in both supervised and unsupervised pattern classification. Our learning approach is not restricted to Gaussian mixture models, as used traditionally, but also uses Bernoulli mixtures to learn the data clusters. We argue that Gaussian mixtures are not suitable for all types of data. When the data is discrete or binary a different probability distribution is more appropriate for its density estimation. Also, adding a supervised learning technique on top of the mixture models gives us dimensionality reduction and improves classification accuracy. Hence, we propose to combine a generative model with a discriminative classifier.

Using the hybrid learning approach we model the binary features in an unconventional manner. The importance of binary features cannot be denied as in many machine learning problems different nominal/categorical attributes are converted into numeric data which is often a feature vector of binary values. So, typically if there is an l-category attribute, it is converted into l numbers where one of the l numbers is 1 and the rest are $0s$. For example, a three-category attribute such as small, medium, large will be represented by $(0,0,1)$, $(0,1,0)$ and $(1,0,0)$ using the above scheme. This scheme is normally known to give better results as compared to using a single number to represent a nominal attribute (Chang and Lin, 2001). Instead of using the traditional Gaussian mixture models, we use Bernoulli mixture models when the data is binary and Gaussian mixture models when the data is composed of continuous features.

The use of Bernoulli mixture models for solving different problems involving binary variables is not new. The basic formula for a Bernoulli mixture model was proposed by Duda and Hart (1973). They have been successfully used for OCR tasks by Juan and Vidal (2004) and Grim et al. (2000). They have been used in supervised text classification tasks (for example, Juan and Vidal, 2002). Mixture models including Bernoulli mixture models have also been used for supervised dimensionality reduction task (Sajama and Orlitsky, 2005).

The organization of this chapter is as follows: In Section 4.2 of this chapter we give an overview of the mixture models and the expectation maximization technique used to estimate the parameters of these models. We describe how these mixtures can be used for classification. In this section we also give a brief overview of artificial neural networks. In Section 4.3 we discuss our hybrid approach for combining mixture models with a discriminative classifier such as an artificial neural network. This section also details how our approach can be used for feature transformation and dimensionality reduction. The simulation results of applying this technique on various datasets are presented in Section 4.4 and the final conclusions are drawn in Section 4.5.

4.2. Mixture Models and Expectation Maximization Algorithm

In this section we describe mixture density models and the use of expectation maximization algorithm for finding the parameters of these models. Before we explain mixture models we would like to point out that expectation maximization (EM) is a general optimization technique for finding maximum likelihood solutions for models that use hidden or latent variables. The name expectation maximization was coined by Dempster et al. and today it is used in many learning applications in the computer vision, natural language processing, psychometrics, etc., domains. Wikipedia (`http://www.wikipedia.org/`) describes EM as a description of a class of related algorithms or a 'meta algorithm' which is used to devise particular algorithms. For example, Baum-Welch algorithm is an example of EM which is used for maximum likelihood estimation in hidden Markov models. Generally, we can say that EM is an iterative method in which the likelihood of the entire data or some subset of data, increases. Duda et al. have used it for estimating the parameters of a distribution from a training set that has missing data (2000). Bishop has described the use of this algorithm for estimating parameters of mixture densities and Bayesian linear regression (2006). In this book chapter we will restrict our use of EM for data clustering and finding the parameters of mixture models.

A finite mixture model assumes that the data is generated by a set of parametric probability distributions instead of being generated by a single distribution. If we have labeled data then we can generate mixture distributions for each class label. To keep the notation simple we suppress the class indices from the equations and assume that right now we are dealing with just one class, and therefore, only one set of mixture distributions. We will deal with multiple classes later.

Suppose we have a sample of training data, $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$, consisting of m input vectors. Each input vector $\mathbf{x}$ is an n-dimensional vector of attributes, hence, $\mathbf{x}_k = \{x_{k1}, x_{k2}, \ldots, x_{kn}\}$. If we want to estimate D mixture components from this data, then a finite mixture model is described by a probability (density) function given by:

$$p(\mathbf{x}) = \sum_{d=1}^{D} P(d)p(\mathbf{x}|d)$$

Here $P(d)$ is the prior of each mixture and $p(\mathbf{x}|d)$ is its component-conditional probability (density) function. This model is termed a generative model which selects the d^{th} component distribution with probability $P(d)$, such that $\sum_d P(d) = 1$, and then generates $\mathbf{x}$ according to $p(\mathbf{x}|d)$.

Learning the parameters of a finite mixture model is a statistical parameter estimation problem. We use expectation maximization (EM) algorithm to estimate these parameters from a sample of training data X. The expectation maximization algorithm determines the parameters of a model by maximizing the log likelihood function of data given by:

$$\mathscr{L}(\Theta|X) = \sum_{k=1}^{m} \log\left(\sum_{d=1}^{D} P(d)p(\mathbf{x}_k|d)\right) \tag{4.1}$$

Here Θ denotes the parameters of the EM algorithm. It consists of the priors, $P(i)$, of each mixture and the parameters, θ_i, of each mixture distribution, i.e.,

$$\Theta = \{P(i), \theta_i\}_{i=1}^{D}$$

The EM algorithm assumes that the observed data is incomplete and associates a vector of latent variables $\mathbf{z}_k = \{z_{k1}, z_{k2}, \ldots, z_{kD}\}$ with each data point. The latent variables are indicator variables, with $z_{ki} = 1$ indicating that the i^{th} mixture component generated the k^{th} data point.

The EM optimization takes place iteratively in two steps. In step 1, also called the expectation step (E-Step), we estimate the expected values of the hidden variables assuming that the model parameters θ_i are known. In step 2, also called the maximization step (M-Step), we estimate the parameter values θ_i to maximize the likelihood of data, given by Equation (4.1), on the basis of the latent variables calculated in the E-step. This is done iteratively until the parameters converge to stable values.

The form of E-step is the same for more or less all distributions and it is given by:

$$z_{kd} = \frac{P(d)p(\mathbf{x}_k|d)}{\sum_{j=1}^{D} P(j)p(\mathbf{x}_k|j)} \qquad (\forall d, 1 \le d \le D, \forall k, 1 \le k \le m)$$

The M-step determines the maximum likelihood estimate of the priors, of each distribution, as given below:

$$P(d) = \frac{1}{m}\sum_{k=1}^{m} z_{kd} \qquad (\forall d, 1 \le d \le D)$$

Also, in this step the parameters of the particular probability distribution are estimated. These parameters depend on the probability (density) function being used. We describe the M-step for Bernoulli and Gaussian mixtures next.

4.2.1. Multivariate Bernoulli Mixtures

A Bernoulli mixture model assumes that each component of the model is an n-dimensional multivariate Bernoulli probability distribution, each component or mixture having its own set of parameters. The form of this distribution for a single vector $\mathbf{x}_k \in \{0,1\}^n$ in the d^{th} distribution is given by (Bishop, 2006):

$$p(\mathbf{x}_k|d) = \prod_{i=1}^{n} p_{di}^{x_{ki}} (1-p_{di})^{1-x_{ki}}$$

Here $p_{di} \in [0,1]$ is the probability of success of the i^{th} component of vector $\mathbf{x}_k$ for the d^{th} mixture, i.e., $p_{di} = p(x_{ki} = 1|d), \forall k, 1 \le k \le m, \forall i, 1 \le i \le n, \forall d, 1 \le d \le D$. Also, we are assuming that the n-dimensional vector $\mathbf{x}$ has n independent component attributes so that the overall probability is the product of the independent uni-dimensional Bernoulli probability functions. Here the parameter θ to be determined is the probability of success for each attribute of vector $\mathbf{x}$, i.e., $\theta = \mathbf{p}$.

To start the EM algorithm we initialize the probabilities with random values. The M-step finds the maximum likelihood estimate of the probability of success of each vector component as given below:

$$\mathbf{p}_d = \frac{\sum_{k=1}^{m} z_{kd}\mathbf{x}_k}{\sum_{k=1}^{m} z_{kd}} \qquad (\forall d, 1 \le d \le D)$$

In the experiments described in Section 4.4 we have used the Laplacian prior to smooth the probability estimates, hence, the probability values are estimated as below:

$$\mathbf{p}_d = \frac{1+\sum_{k=1}^{m} z_{kd}\mathbf{x}_k}{2+\sum_{k=1}^{m} z_{kd}} \qquad (\forall d, 1 \le d \le D)$$

4.2.2. Multivariate Gaussian Mixtures

The n-dimensional multivariate Gaussian distribution has two parameters to be determined, namely, mean vector and covariance matrix, i.e., $\theta = (\boldsymbol{\mu}; \boldsymbol{\Sigma})$. The form of the d^{th} multivariate Gaussian mixture component for a vector $\mathbf{x}_k \in R^n$ is given by:

$$p(\mathbf{x}_k|d) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}_d|}} e^{-(\mathbf{x}_k-\boldsymbol{\mu}_d)^t \boldsymbol{\Sigma}_d^{-1} (\mathbf{x}_k-\boldsymbol{\mu}_d)}$$

Here $\boldsymbol{\mu}_d$ and $\boldsymbol{\Sigma}_d$ are the mean vector and the covariance matrix for the d^{th} mixture respectively. The M-step estimates these parameters as given below:

$$\begin{aligned}
\boldsymbol{\mu}_d &= \frac{\sum_{k=1}^{m} z_{kd}\mathbf{x}_k}{\sum_{k=1}^{m} z_{kd}} \\
\boldsymbol{\Sigma}_d &= \frac{\sum_{k=1}^{m} z_{kd}(\mathbf{x}_k-\boldsymbol{\mu}_d)(\mathbf{x}_k-\boldsymbol{\mu}_d)^t}{\sum_{k=1}^{m} z_{kd}} \qquad (\forall d, 1 \le d \le D)
\end{aligned}$$

We have restricted the covariance matrix to a diagonal matrix of the variances of individual features to prevent singularity during computation.

4.2.3. Classification Using Mixture Models

Finding the mixture components is an unsupervised learning technique that finds the various clusters within data. Each cluster is represented by a particular probability distribution. Now we extend our terminology to include the class labels assigned to each example point. Suppose we have a set of C labeled classes $Q = \{q_1, q_2, \ldots, q_C\}$ with prior for the c^{th} class being $P(q_c)$ then the class conditional probability function for the c^{th} class having D_c mixture components is given by:

$$p(\mathbf{x}|q_c) = \sum_{d=1}^{D_c} p(\mathbf{x}|d, q_c)P(d|q_c)$$

The posterior probabilities for each class are calculated using Bayes' Rule and the vector $\mathbf{x}$ is assigned the label of the class which has the maximum posterior probability, i.e.,

$$\begin{aligned} y'(\mathbf{x}) &= \arg\max_{q_c} P(q_c)p(\mathbf{x}|q_c) \\ &= \arg\max_{q_c} P(q_c)\sum_{d=1}^{D_c} P(d|q_c)p(\mathbf{x}|d, q_c) \end{aligned} \tag{4.2}$$

We can estimate the class priors from the data as the ratio of training examples of that class to the total number of training examples. From Equation (4.2) we can see that the winning class is determined by the class conditional probabilities $p(\mathbf{x}|q_c)$ and the class priors $P(q_c)$. Also, the class conditional probabilities are determined by the weighted sum of the conditional probability functions of the mixture component for that class $p(\mathbf{x}|d, q_c)$ weighted by the prior of that mixture with respect to that class $P(d|q_c)$. Hence, classification takes place using a linear discriminant function ψ of $p(\mathbf{x}|d, q_c)$ and the weights $P(d|q_c)$, i.e.,

$$y'(\mathbf{x}) = \psi\Big(p(\mathbf{x}|d, q_c), P(d|q_c)\Big) \tag{4.3}$$

4.2.4. Artificial Neural Networks

An artificial neural network (ANN) is a model of learning inspired by the biological neural networks. It consists of interconnected group of artificial neurons that act as non-linear processing units and exhibit a complex global behavior. ANNs have been successfully applied to a wide variety of applications involving regression, classification or data processing. There is an immense amount of literature available on neural networks. The reader is referred to any standard text book on machine learning, e.g., Bishop (2006), Alpaydin (2005) and Mitchell (1997). In this section, we restrict the discussion of neural networks to the models of classification that we have used for our work.

An ANN consists of several layers of neurons. Each neuron takes inputs from neurons in the preceding layer and produces an output via a non-linear activation function. So if a neuron i receives n inputs $\{x_j\}_{j=1}^{n}$ from the previous layer, then its output o_i is given by:

$$o_i = f(\sum_{j=1}^{n} w_{ji}x_j)$$

where $f(.)$ represents a non-linear activation function such as the sigmoid function and w_{ji} is the weight or strength of the connection between neuron i and the j^{th} neuron in its predecessor layer.

An ANN always has an input layer, an output layer and can have multiple hidden layers. For our work we employed fully connected feedforward nets with just one hidden layer. Each neuron acts as a local processing unit but together the entire network is capable of modeling complex relationships between the inputs and the outputs. The training of the network takes place by iteratively adjusting the weights or connection strengths, based upon some error criterion. We trained the network using back propagation training algorithm guided by scaled conjugate gradient descent search. Also, we determined the network structure, i.e., the number of units in the hidden layer via cross validation. The simulations were run using the implementation of neural networks provided by Challenge Learning Object Package (CLOP) library (Saffari and Guyon, 2006).

4.3. Hybrid Learning

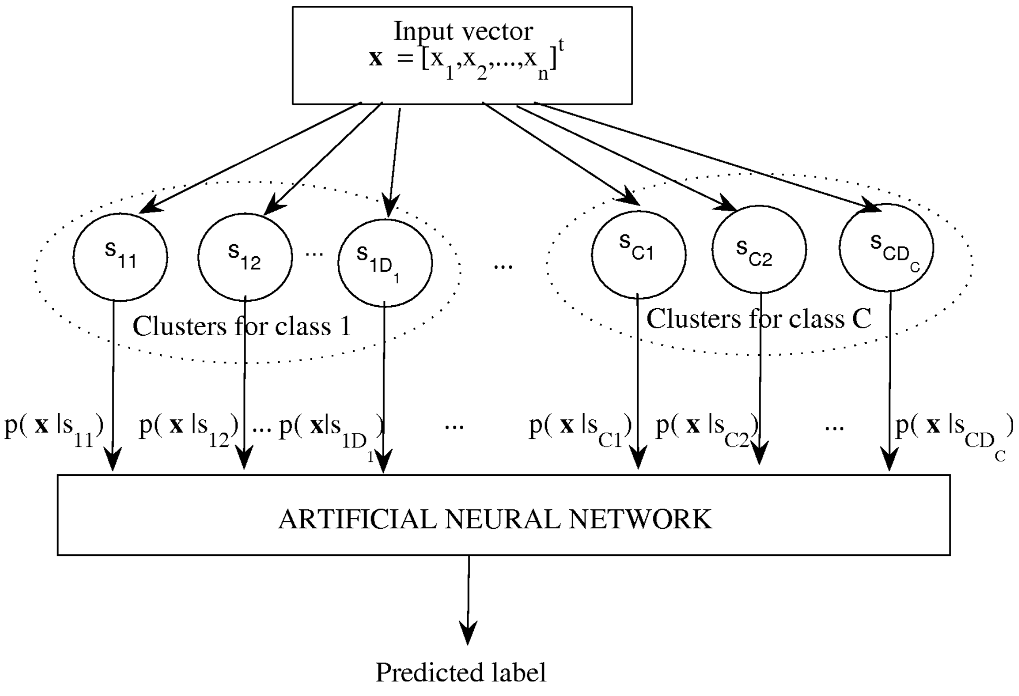


Figure 4.1: Hybrid learning model

Section 4.2.3 describes a mechanism for classification if only mixture models are used. We propose to combine the mixture modeling approach with a neural network model for hybrid learning. Our hybrid learning model is shown in Figure 4.1. The first step involved in this approach is the determination of data clusters using expectation maximization described in Section 4.2. For C classes, a set S of data clusters is determined:

$$S = \{s_{ij}\}_{i=1,j=1}^{i=C,j=Di}$$

Each member, s_{ij}, of S represents the j^{th} cluster or sub-class for the i^{th} class, e.g., the i^{th} class has D_i clusters associated with it, given by: $\mathbf{s_i} = \{s_{i1}, s_{i2}, \ldots, s_{iD_i}\}$. Next, the conditional probability $p(\mathbf{x}|s_{ij})$ of the vector $\mathbf{x}$ is determined in each cluster. This, in effect, is equivalent to performing a transformation $T : X \longmapsto \Phi$ of the input vector $\mathbf{x}_k \in R^n$ ($\mathbf{x}_k \in \{0,1\}^n$ in case of a binary feature vector) to a new feature vector $\phi_k \in [0,1]^N$, where the vector ϕ_k is given by:

$$\phi_k = [\{p(\mathbf{x}_k|s_{ij}\}_{i=1,j=1}^{i=C,j=Di}]^t$$

Also, $N = \sum_{i=1}^{C} D_i$. The transformed set of feature vectors Φ is then used to train a discriminative classifier for learning the class separations. Hence, in this hybrid approach we propose that the label of the class is some function of the class conditional probabilities, $p(\mathbf{x}_k|s_{ij})$, determined by each mixture component. Any suitable discriminative classifier can be trained to learn this function. In our present work, we mainly focused on using artificial neural networks in the supervised layer to perform classification on the transformed vector. However, any suitable supervised learning algorithm can perform this step.

In the hybrid learning approach that we have used, we are assuming that the data for each class is being generated by a set of sub-classes represented by mixture models. This method is similar to RBF networks, where the input to the neural network is a set of RBF functions, the centers of which can be found using a suitable clustering method. The novel thing about this approach is that binary data is being modeled by multivariate Bernoulli mixtures and Gaussians are being used to model the continuous data. In case the input data has both continuous and binary attributes we split the input vector $\mathbf{x}$ into two vectors, $\mathbf{v}_1$ and $\mathbf{v}_2$, i.e., $\mathbf{x} = [\mathbf{v}_1\ \mathbf{v}_2]$. Vector $\mathbf{v}_1$ is composed of only binary attributes and $\mathbf{v}_2$ consists of only continuous attributes, i.e., $\mathbf{v}_1 \in \{0,1\}^a$, $\mathbf{v}_2 \in R^b$ and $a+b=n$. Vectors $\mathbf{v}_1$ are used to determine Bernoulli mixtures for all classes and $\mathbf{v}_2$ are used to determine Gaussian mixtures for all classes. Also, instead of using a linear model of classification as depicted by Equation (4.3), we are using a non-linear discriminant function g to determine the classification, i.e.,

$$y'(\mathbf{x}) = g(p(\mathbf{x}|d,q_c))$$

This non-linear mapping is found using a discriminative classifier. An important characteristic of this model is the assumption that the data of each class is being generated by different source distributions instead of just one distribution. This is a realistic and reasonable assumption for real life data. This enables us to capture the various interesting characteristics and properties of data. Also, an immense reduction in dimensionality of the input vector results via this approach (described later in Section 4.3.1). In this chapter we don't address the problem of how many mixture components can fall in one class. For the current set of experiments, we determine this number using a cross validation approach. The details of the simulations are given in Section 4.4.

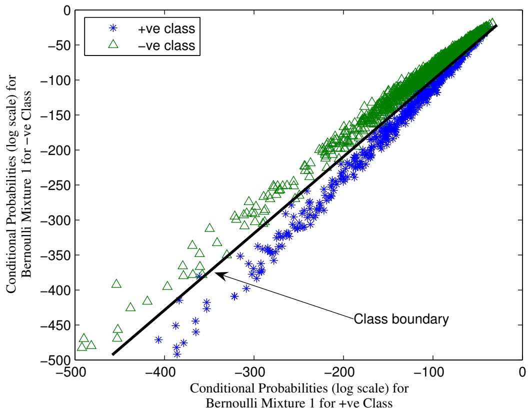
4.3.1. Transformation of Input Space and Dimensionality Reduction

The hybrid learning approach, described in Section 4.3, is appealing because it not only gives us improved accuracy of classification but it gives us means for input transformation and dimensionality reduction. Once we have the mixture models we determine the class conditional probabilities of each data point with respect to the mixture densities of all classes. If the total number of mixture components for all the classes is $N = \sum_{i=1}^{C} D_i$ then we have performed a transformation of data equivalent to:

$$T : X \longmapsto \Phi \qquad X \in \mathscr{R}^n \qquad \Phi \in \mathscr{R}^N$$

If the inequality $N < n$ holds true, then we have reduced the size of our feature vector for input to a discriminative classifier.

The fact that the hybrid learning approach achieves a huge dimensionality reduction of data is particularly useful in creating different views of data and making the data more manageable



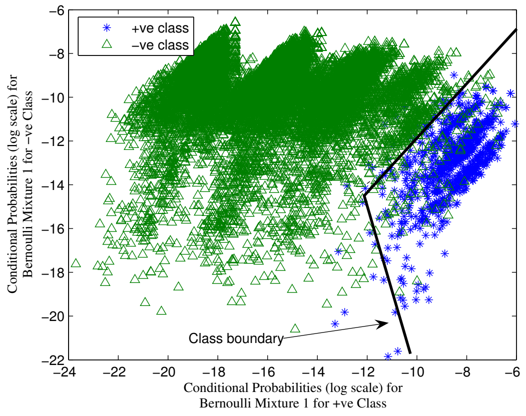


Figure 4.2: Plots for a subset of training data for Nova (left) and SYLVA (right) datasets after feature transformation. A possible class boundary has been marked by hand.

for further processing. For example, Figure 4.2 shows the plot of a subset of training and validation data for two datasets, namely, the NOVA dataset and the SYLVA dataset used in IJCNN's agnostic learning track. These datasets are described in Section 4.4. For the NOVA dataset we had 7,131 features after initial input attribute elimination and we generated 2 Bernoulli mixtures for the positive class and 2 for the negative class. The graph shows the plot of conditional probabilities for mixture 1 of positive class against that of the negative class. It is very interesting to see that there is a clear linear separation of the transformed features for both the negative and positive class for the NOVA dataset. The same two features are plotted for the SYLVA dataset. Again, we can see a clear separation of the two class labels when the transformed features are used.

Table 4.1 shows the percentage reduction of dimensionality of the input space for various datasets. Their detailed simulation results are presented in Section 4.4. The table shows the total input attributes and the attributes left after initial attribute elimination step (described in Section 4.4.2). Columns 4 and 5 show the number of mixture distributions generated for the positive and negative class labels. The total mixture distributions is N which determine the size of the new feature space. The last column shows the reduction in dimensionality of the input space with respect to the total attributes after elimination. This percentage is as low as 99.9% for the NOVA dataset.

The optimum number of mixtures (Column 4 and 5 of Table 4.1) for each dataset was either determined by cross validation or the EM algorithm automatically determined this number. On datasets like ADA and GINA we had to use cross validation. In case of NOVA and SYLVA datasets, the EM algorithm found some Bernoulli mixtures with almost zero priors. In such cases we retained only those mixtures with non-zero priors. For these datasets we observed that the number of non-zero prior mixtures did not change with a changing value of D specified by the user (D is the number, input to the EM algorithm, specifying the number of mixtures to be generated). Hence we can conclude from our observations that for certain types of datasets EM algorithm determines the optimum number of clusters present in the data.

Table 4.1: Dimensionality reduction on various datasets. n is the total number of initial attributes, N is the number of transformed features. Columns 4 and 5 show the mixtures for positive and negative class separately

Dataset	**n**	**After elimination**	**Bernoulli mixtures**	**Gaussian mixtures**	**N**	**Reduction**
ADA	48	48	3+6	15+15	39	19%
GINA	970	433	35+35	-	70	83%
HIVA	1617	574	8+7	-	15	97%
NOVA	16969	7131	2+2	-	4	99.9%
SYLVA	216	216	1+1	3+3	8	96%

4.4. Experimental Results

In this section we describe in detail the results obtained using various datasets. The datasets were used for the "Agnostic Learning vs. Prior Knowledge" challenge, organized by IJCNN (2007a). There are 5 datasets, namely, ADA, GINA, NOVA, SYLVA and HIVA. All these datasets have two possible class labels. For our simulations we have used the datasets from the agnostic learning track. As we are using the data from the agnostic learning track, we don't have any feature information. We give a brief summary of these datasets, as given by Guyon et al. (2007), in Table 4.2. It can be seen that the above datasets are taken from different sources and can be used as a test bed for various learning algorithms. Also, because these datasets were launched as part of a machine learning competition, they serve as a benchmark for comparison between different techniques. The evaluation of results described in this section is based on the balanced error rate (BER) which is the average of the error rates on positive and negative class labels (Guyon et al., 2007).

Table 4.2: Datasets used (Guyon et al., 2007) (Column 4, 5 and 6 show the total examples in the training, validation and test sets. The last column shows the percentage of positive examples)

Dataset (domain)	Type	Attributes	Train	Valid	Test	Pos%
ADA (marketing)	mixed	48	4,147	415	41,471	24.8%
GINA (handwriting)	continuous	970	3,153	315	31,532	49.2%
HIVA (medicine)	binary	1,617	3,845	384	38,449	3.5%
NOVA (text-mining)	binary	16,969	1,754	175	17,537	28.5%
SYLVA (ecology)	mixed	216	13,086	1,309	1,30,857	6.2%

4.4.1. Overall Hybrid Learning Model

The overall learning model is shown in Figure 4.3. The first step involved is that of attribute elimination. We call it attribute elimination to distinguish it from feature transformation and dimensionality reduction. The next step is the estimation of mixture densities for various classes and feature transformation. The transformed features are used for training with a discriminative classifier which gives us the predicted label. To run the simulations we added objects to the Challenge Learning Object Package (CLOP) library (Saffari and Guyon, 2006). For pre-processing, cross-validation and neural network learning we used the objects provided by CLOP. For mixture modeling we wrote our code in C++ and provided an interface to it in Matlab.

4.4.2. Initial Attribute Elimination

We adopted a very simple counting procedure for reducing the dimensionality of the input space. There are many raw data features which don't convey substantial information about their class and seem redundant. The attributes, whether continuous or binary, for which the percentage of non-zero entries for both the positive and negative classes, was less than a certain threshold were omitted. This threshold was determined empirically using cross validation. The rule for the initial elimination of attributes is then given by the following:

$$\begin{array}{lll} \text{If} & \dfrac{\sum_{k=1}^{m} I(x_{k,i} \neq 0 \; and \; y_k = +1)}{\sum_{k=1}^{m} I(y_k = +1)} \leq \text{threshold } \textit{and} \\ & \dfrac{\sum_{k=1}^{m} I(x_{k,i} \neq 0 \; and \; y_k = -1)}{\sum_{k=1}^{m} I(y_k = -1)} \leq \text{threshold} \\ \text{then} & \text{eliminate input variable } x_i \end{array} \tag{4.4}$$

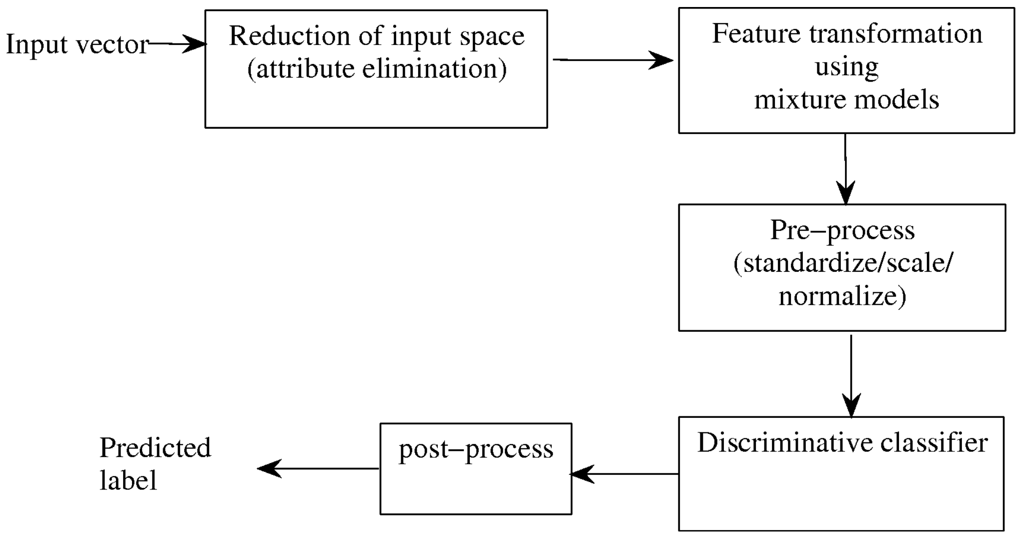

Input vector
Reduction of input space (attribute elimination)
Feature transformation using mixture models
Pre–process (standardize/scale/ normalize)
Discriminative classifier
post–process
Predicted label

Figure 4.3: Overall model

where the function $I(R) = 1$ when the rule R holds. Although, this method of feature elimination may seem like an ad-hoc method at the first glance, it has an intuitive appeal and justification based on information theory. For all the datasets we have a two way classification and if we consider only two possibilities for the value of an input attribute (zero or non-zero), the information gain, Δ_i, for attribute i ($\forall i, 1 \leq i \leq n$) is given by ($E$ is the entropy):

$$\Delta_i = E(X) - \frac{\sum_{k=1}^{m} I(x_{k,i} \neq 0)}{m} E_{i(x_{k,i}\neq 0)} - \frac{\sum_{k=1}^{m} I(x_{k,i} = 0)}{m} E_{i(x_{k,i}=0)} \tag{4.5}$$

Hence, the features that satisfy Rule (4.4) will have low information gain. Note that not all features with low information gain as given by Equation (4.5) will be removed in this step. Apart from ADA and SYLVA, this method of feature elimination when applied to the datasets yielded good results.

4.4.3. Simulations for Finding Mixture Parameters and Number of Clusters

The EM algorithm was written in C++ to determine the parameters of the Gaussian mixture models and Bernoulli mixture models. The number of mixture components, D, to be determined is pre-defined by the user. The EM algorithm doesn't give us any means for determining the optimum number of mixture distributions or clusters. When running this algorithm for Bernoulli mixture models it was observed that the priors of some of the mixture components were reduced to zero after some iterations. For example, we started with the initial parameter $D = 10$ on the NOVA dataset for the positive class and the number of resulting mixtures with non-zero priors was 2. For most of the simulations we observed that the mixtures with non-zero priors was 2, even with different starting values of D. Similarly, in SYLVA's case, most of the simulations resulted in only one or two mixture distributions with non-zero priors. Hence, in such cases the EM algorithm results in a fewer number of mixture distributions compared to the initially specified number. For our hybrid learning model we only retain the distributions with non-zero priors. This gives us a way of determining the number of clusters needed for each class. This is shown in detail for different datasets in the next sections.

4.4.4. NOVA Dataset

Table 4.3: Results for the NOVA dataset. The second column (initial Bernoullies) shows the value of D, the initial clusters, specified by the user for the EM algorithm. The third column (Bernoulli mixtures) shows the number of clusters left with non-zero priors after running the EM algorithm. For the last row entry, a layer of 5 boosting units was used, each unit represented by 2-layer neural network having 25 hidden units.

No.	**Initial Bernoullies**	**Bernoulli mixtures**	**NN/w units**	**Train BER**	**Valid BER**	**Test BER**
1.	15+15	2+2	15	0.038	0.028	0.046
2.	20+20	2+3	20	0.037	0.028	0.048
3.	20+20	2+3	15	0.037	0.028	0.048
4.	20+20	2+3	8	0.037	0.028	0.049
5.	10+10	2+3	25 boost (5)	0.037	0.028	0.050

Here, we present the results achieved for agnostic learning with the NOVA dataset. The NOVA dataset has 16,969 binary features. Table 4.3 shows the results of the entries submitted to the challenge. The initial step of feature elimination described in Section 4.4.2 was applied to the dataset with a threshold value of 0.3%. This eliminates 9,838 features leaving behind only 7,131 features. These features were used to generate Bernoulli mixture models for feature transformation. The second column shows the initial D parameter input to the EM algorithm for the number of Bernoulli mixtures (see Section 4.4.3). The third column shows how many clusters were left with non-zero priors. For this dataset we almost always ended up with 2 or 3 mixtures, depending upon the initial values of the EM algorithm. The features were pre-processed using the 'shift-n-scale' and 'standardize' functions from CLOP. The resulting data was then trained using a 2 layer neural network. Column 4 shows the number of units comprising the hidden layer. The final labels were post-processed using the bias option with transduction as provided in CLOP. The topmost entry is the winning entry for the NOVA dataset in the agnostic learning track. The last entry shows the results obtained by using a boosting layer of 5 units, each unit being composed of a neural network having 25 hidden units. There is not much difference in accuracy for different parameter values.

4.4.5. SYLVA Dataset

Table 4.4 shows the results obtained on the SYLVA dataset. The topmost entry was ranked third in the agnostic learning track. The SYLVA dataset has both binary and continuous attributes for which we generated Bernoulli and Gaussian mixtures respectively. It was interesting to observe that for this dataset we always ended up with either 1 or 2 Bernoulli mixtures with non-zero priors, depending upon the initial parameters used by the EM algorithm (see Section 4.4.3). This observation leads us to conclude that the positive and negative examples can be modeled by just 1 or 2 Bernoulli distributions for the binary attributes. Section 4.3.1 shows the separation obtained by the features resulting from the Bernoulli mixtures. After generating Bernoulli mixtures the features were pre-processed using the standardize, shift-n-scale and normalize options provided by CLOP. Also, the labels generated by the neural network model were post-processed using the bias option 1. The first 3 entries show the results obtained by using a 2-layer neural network at the supervised layer. The fourth entry shows the results obtained by using support vector machines as the discriminative classifier, where the kernel was calculated using Kullback-Leibler (KL) divergence. The fifth entry uses a boosting layer of 20 units, each unit being made of 2-layer neural network with 5 hidden units. Neural networks achieve a higher accuracy as compared to the other two discriminative methods. However, the difference in accuracy is not very significant.

Table 4.4: Results for the SYLVA dataset. Entry 5 shows a supervised layer made from boosting 20 units, each unit being composed of a 2-layer neural network having 5 hidden units.

No.	**Bernoulli mixtures**	**Gaussian mixtures**	**NN/w units**	**Other classifier**	**Train BER**	**Valid BER**	**Test BER**
1.	1+1	3+3	3	-	0.0072	0.0053	0.0094
2.	1+1	6+6	10	-	0.0069	0.0061	0.0096
3.	1+1	6+6	10	-	0.0078	0.0065	0.0104
4.	1+1	2+2	-	SVM	0.0094	0.0057	0.0124
5.	2+2	6+6	-	5 boost (20)	0.0083	0.0053	0.0131

4.4.6. GINA Dataset

Table 4.5: Results for the GINA dataset

No.	**Bernoulli mixtures**	**NN/W units**	**Boosting units**	**Train BER**	**Valid BER**	**Test BER**
1.	35+35	15	5	0.010	0	0.049
2.	25+25	20	5	0.014	0.006	0.050
3.	25+25	15	5	0.014	0.006	0.051
4.	15+15	15	5	0.023	0.013	0.052
5.	20+20	20	5	0.010	0.006	0.052

The results obtained on the GINA dataset are given in Table 4.5. For this dataset the attribute elimination threshold was kept at 40%, (method described in Section 4.4.2) which reduced our initial attributes from 970 to 433. This dataset has continuous attributes. As we want to fit Bernoulli mixtures to this dataset, we treated all the non-zero entries in this dataset as success and all the zero values as failure, resulting in binary attributes. Even though this results in information loss but surprisingly it still gives us good results. For pre-processing, shift-n-scale and standardize methods of CLOP were used. Also, the output labels of the neural network were post processed using the bias algorithm with the transduction option. The topmost entry was ranked fifth in the agnostic learning track. For the supervised layer we used the boosting method, with each unit being made of a 2-layer neural network. Boosting gave us better results as compared to using a single 2-layer neural network. The difference in results for various values of the parameters is not very significant in this case.

4.4.7. HIVA Dataset

The results obtained on the HIVA dataset are shown in Table 4.6. The threshold used for the initial attribute elimination step was 10%. So the initial input space was reduced from 1617 to 574. The inputs to the neural network were first pre-processed using shift-n-scale, standardize and normalize options from CLOP. Also post-processing of output labels was done using bias option 4. For the second entry we used boosting with 20 units, each unit being composed of a 2-layer neural network having 3 hidden units. For this dataset also, there is not much difference between the error rates, when using different values of initial parameters.

Table 4.6: Results on the HIVA dataset

No.	Bernoulli mixtures	NN/w units	Train BER	Valid BER	Test BER
1.	8+7	7	0.218	0.237	0.305
2.	9+4	3 boost (20)	0.172	0.185	0.309
3.	8+7	20	0.151	0.110	0.309
4.	8+7	40	0.148	0.132	0.317
5.	8+7	35	0.166	0.147	0.319

4.4.8. ADA Dataset

Table 4.7 shows the results obtained on the ADA dataset. We found this dataset the most difficult to train as this was the only dataset on which a simple neural network with 2 layers performed as well as the other methods (row number 2, 3 and 4). The hybrid learning approach doesn't seem to give much benefit in this case.

Table 4.7: Results on the ADA dataset

No.	Eliminate threshold	Bernoulli mixtures	Gaussian mixtures	NN/w units	Train BER	Valid BER	Test BER
1.	0	3+6	15+15	15	0.180	0.190	0.181
2.	0	-	-	7	0.174	0.188	0.181
3.	0	-	-	7	0.177	0.188	0.181
4.	0	-	-	3	0.170	0.174	0.185
5.	1%	7+8	15+15	7	0.190	0.201	0.187

4.4.9. Comparison With Other Methods

In this section we compare the performance of the hybrid learning model with other methods. First we will discuss the results obtained from the individual mixture model, individual neural network model and the hybrid model. Then we'll compare the classification results of the hybrid model with the best models on the challenge datasets.

Table 4.8 shows the results of applying three different algorithms to the challenge datasets. The third column shows the BER when only mixture models are used for classification (details are given in Section 4.2.3). BER obtained by applying neural networks is shown in the fourth column and the last column shows the BER of the hybrid learning model. All the error rates

Table 4.8: Comparison of hybrid learning approach with individual mixture model and neural network model. The table shows the balanced error rate of classification on the test set of the various datasets

No.	Dataset	Mixture models	Neural network	Hybrid learning
1.	ADA	0.264	0.1871	0.181
2.	GINA	0.1379	0.1173	0.049
3.	HIVA	0.3493	0.2961	0.305
4.	NOVA	0.1241	0.1453	0.0456
5.	SYLVA	0.0156	0.0119	0.0094

pertain to the test cases of the corresponding datasets. We can see from the table that for GINA, NOVA and SYLVA the hybrid learning model achieves a significant improvement in accuracy as compared to the individual mixture models or the neural network model. For ADA, the performance of the hybrid approach is almost the same as the neural network model but better than the individual mixture model. In case of HIVA, the mixture models have the worst accuracy and the neural networks perform slightly better than the hybrid model.

The results of the "Agnostic vs. Prior Knowledge" (2007a) challenge can be found on http://clopinet.com/isabelle/Projects/agnostic/Results.html. Table 4.9 shows a comparison of our method with the winning entries in the agnostic learning track. This comparison is based on the balanced error rates on the test dataset. It can be seen from the table that apart from ADA and HIVA, our method achieves good results on the datasets. The winner of the overall agnostic learning track is Lutz (2006) who has used boosting techniques for classification. His entries are also ranked best on the ADA, GINA and SYLVA datasets. The last row of the table shows an overall comparison of our method with the winning entry. We can see that the difference in test BERs for the overall result of the two methods is not very significant.

4.4.10. Discussion of Results

In this section we presented the results obtained by applying hybrid learning model on the five different datasets of the agnostic track of the "Agnostic vs. Prior Knowledge" challenge. All the five datasets have been taken from different domains and some have both binary and continuous features and some have only binary features. Our approach for solving the classification problem on these datasets consists of mainly three stages. The first step involves the elimination of attributes which do not have a high information gain. In the second step we use finite Gaussian or Bernoulli mixture models for unsupervised clustering of data. Using these mixture models we transform the original input space into a low dimensional probability space. In the third phase, the transformed features are fed to an artificial neural network for classification.

The main motivation behind our technique is to use unsupervised technique to capture the important properties of data using mixture models and classify the data using the non-linear discriminative function provided by artificial neural networks. Hence, the model combines the generative power of mixture models with the discriminative ability of neural networks. Mixture models provide a semi-parametric approach for modeling the density of data when the distribution of data is not unimodal. We have successfully demonstrated their ability to attain a high reduction in the original dimensionality of data, hence making the data more manageable for

Table 4.9: Comparison of hybrid learning approach with other methods (IJCNN, 2007b). Note that the best entry for a dataset is not necessarily the best overall entry.

	Best method		**Hybrid learning**	
Dataset	**Method**	**Test BER**	**Test BER**	**Rank**
ADA	LogitBoost with trees (Lutz, 2006)	0.166	0.181	9
GINA	LogitBoost/Doubleboost (Lutz, 2006)	0.0339	0.0495	5
HIVA	RBF SVM (Franc, 2007)	0.2827	0.305	10
NOVA	Hybrid learning (Saeed, 2007)	0.0456	0.0456	1
SYLVA	LogitBoost with trees (Lutz, 2006)	0.0062	0.0094	3
Overall	LogitBoost with trees (Lutz, 2006)	0.1117	0.1194	6

input to a classifier. The reduction in dimensionality is as low as 99.9% on the NOVA dataset, 97% and 96% on the HIVA and SYLVA datasets respectively.

Empirical results demonstrate that the hybrid learning model is a simple yet, effective method for agnostic learning where the attributes involved are raw low level features. We conducted experiments to compare the BER of the hybrid learning algorithm with individual mixture models and neural network models and found that the combined model performs significantly better than the individual models in case of GINA, SYLVA and NOVA. The performance of this method is almost the same as neural networks in case of ADA and slightly worse than neural networks in case of HIVA. In all cases the hybrid model has a much better accuracy rate as compared to the mixture model classifiers.

The effectiveness of the hybrid learning model has been supported by empirical results on the five challenge datasets. According to the August, 2007 ranking of the challenge participants our method achieved the best performance on the NOVA dataset, was ranked third on the SYLVA dataset, fifth on the GINA dataset and ninth, tenth on the ADA and HIVA datasets respectively. In the overall performance one of our entries was ranked sixth. The best participant had a BER of 11.17% and our entry had an overall BER of 11.94%. The difference between the BER of these two entries is not very significant.

4.5. Conclusions

In this chapter we have presented a hybrid learning method using mixture models and artificial neural networks. The hybrid learning approach, described in this chapter, combines the generative model with a discriminative approach. This method is particularly intuitive as it first models the data using mixture models. The mixture models assume that the entire data is generated from various sources and, hence, this method can capture the various properties of data using different component distributions. These mixture models are used to perform a feature transformation of the input vector. The transformed features are then used to train a discriminative classifier. In this chapter we have also shown how mixture models can be used for feature transformation and dimensionality reduction of the input space.

Our hybrid learning approach was applied to 5 datasets which were launched as part of the “Agnostic vs. Prior Knowledge” challenge, by IJCNN 2007, in the agnostic learning track. This method has the winning entry on the NOVA dataset. Also, our entries were ranked third and fifth for the SYLVA and GINA datasets. The entries sent to the challenge show that this method is comparable to the other methods. Work is on going to devise methods for finding the optimum number of mixture models for different types of datasets. Also, we are exploring the potential of other mixture distributions to be used in the hybrid learning approach.

Acknowledgments

I would like to thank Dr. Haroon Babri and Mr. Kashif Javed of University of Engineering and Technology, Lahore, for reviewing and providing valuable comments and feedback on this chapter. I also thank Isabelle Guyon for all her technical help during the challenge.

References

Agnostic learning vs. prior knowledge challenge, 2007a. See http://www.agnostic.inf.ethz.ch.

Agnostic learning vs. prior knowledge competition results, 2007b. See http://clopinet.com/isabelle/Projects/agnostic/Results.html.

Ehem Alpaydin. *Introduction to Machine Learning*. Prentice-Hall of India Private Limited, 2005.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Arthur Dempster, Nan Laird, and Donald Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley and Sons, inc, 2000.

Vojtech Franc. *Modified multi-class SVM formulation; Efficient LOO computation*, 2007. Fact sheet available at http://clopinet.com/isabelle/Projects/agnostic/.

Jiri Grim, Pavel Pudil, and Petr Somol. Multivariate structural Bernoulli mixtures for recognition of handwritten numerals. In *Proceedings of International Conference on Pattern Recognition (ICPR’00)*, 2000.

Isabelle Guyon, Amir Saffari, Gideon Dror, and Gavin Cawley. Agnostic learning vs. prior knowledge challenge. In *Proceedings of International Joint Conference on Neural Networks*, August 2007.

Tommi S. Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *Proceedings of the 1998 conference on Advances in neural information processing systems II*, pages 487–493, 1998.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixture of local experts. *Neural Computation*, 3:79–87, 1991.

Alfons Juan and Enrique Vidal. On the use of Bernoulli mixture models for text classification. *Pattern Recognition*, 35(12):2705–2710, December 2002.

Alfons Juan and Enrique Vidal. Bernoulli mixture models for binary images. In *Proceedings of 17th International Conference on Pattern Recognition (ICPR-04)*, 2004.

Julia A. Lasserre, Christopher M. Bishop, and Thomas P. Minka. Principled hybrids of generative and discriminative models. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, New York, 2006.

Roman W. Lutz. Logitboost with trees applied to the WCCI 2006 performance prediction challenge datasets. In *Proceedings of International Joint Conference on Neural networks*, pages 2966–2969, Vancouver, Canada, July 2006. Available at `http://stat.ethz.ch/~lutz/publ`.

Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

John Moody and Christian Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

Mehreen Saeed. *Hybrid approach for learning*, 2007. Fact sheet available at `http://clopinet.com/isabelle/Projects/agnostic/`.

Amir Saffari and Isabelle Guyon. *Quick Start Guide For CLOP*, May 2006. Available at `http://ymer.org/research/files/clop/QuickStartV1.0.pdf`.

Sajama and Alon Orlitsky. Supervised dimensionality reduction using mixture models. In *Proceedings of the 22nd international conference on machine learning*, pages 768–775, Bonn, Germany, 2005.

Ilkay Ulusoy and Christopher M. Bishop. Generative versus discriminative methods for object recognition. In *Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition, CVPR*, San Diego, 2005.

Chapter 5

Data Grid Models for Preparation and Modeling in Supervised Learning

Marc Boullé MARC.BOULLE@ORANGE-FTGROUP.COM
France Telecom R&D
2, avenue Pierre Marzin
22300 Lannion, France

Editor: Isabelle Guyon

Abstract

This paper introduces a new method to automatically, rapidly and reliably evaluate the class conditional probability of any subset of variables in supervised learning. It is based on a partitioning of each input variable into intervals in the numerical case and into groups of values in the categorical case. The cross-product of the univariate partitions forms a multivariate partition of the input representation space into a set of cells. This multivariate partition, called data grid, is a piecewise constant nonparametric estimator of the class conditional probability. The best data grid is searched using a Bayesian model selection approach and an efficient combinatorial algorithm.

We also extend data grids to joint density estimation in unsupervised learning and apply this extension to the problem of coclustering the instances and variables of a sparse binary dataset.

We finally present three classification techniques, exploiting the maximum a posteriori data grid, an ensemble of data grids, or a coclustering data grid, and report results in the Agnostic Learning vs. Prior Knowledge Challenge, where our method achieved the best performance on two of the datasets. These experiments demonstrate the value of using data grid models in machine learning tasks, for conditional density estimation, data preparation, supervised classification, clustering and rule based explanation.

5.1. Introduction

Univariate partitioning methods have been studied extensively in the past, mainly in the context of decision trees (Kass, 1980; Breiman et al., 1984; Quinlan, 1993; Zighed and Rakotomalala, 2000). Supervised discretization methods split the numerical domain into a set of intervals and supervised value grouping methods partition the input values into groups. Fine grained partitions allow an accurate discrimination of the output values, whereas coarse grain partitions tend to be more reliable. When the size of the partition is a free parameter, the trade-off between information and reliability is an issue. In the MODL approach, supervised discretization (Boullé, 2006) (or value grouping (Boullé, 2005)) is considered as a nonparametric model of dependence between the input and output variables. The best partition is found using a Bayesian model selection approach.

In this paper, we describe an extension of the MODL approach to the supervised bivariate case for pairs of input variables (Boullé, 2007a)[1], and introduce its generalization to any subset

1. The method is available as a shareware, downloadable at http://perso.rd.francetelecom.fr/boulle/

of variables of any types, numerical, categorical or mixed types. Each input variable is partitioned, into intervals in the numerical case and into groups of values in the categorical case. The cross-product of the univariate partitions forms a multi-dimensional data grid. The correlation between the cells of this data grid and the output values allows the joint predictive information to be quantified. The trade-off between information and reliability is established using a Bayesian model selection approach. We also extend these models to the unsupervised case, where the data grids are nonparametric models of dependence between all the variables, with a piecewise constant estimation of the joint probability distribution. Sophisticated algorithms are necessary to explore the search space of data grid models. They have to strike a balance between the quality of the optimization and the computation time. Several optimization heuristics, including greedy search, meta-heuristic and post-optimization, are introduced to efficiently search the best possible data grid.

The paper is organized as follows. Section 5.2 summarizes the MODL method in the univariate supervised discretization and value grouping cases. Section 5.3 extends the approach to the multivariatiate case and Section 5.4 describes the generalization of such models to unsupervised learning and coclustering. Section 5.5 presents the optimization algorithms. Section 5.6 evaluates the data grid models on artificial datasets. Section 5.7 reports experiments performed on the agnostic learning vs. prior knowledge challenge datasets (Guyon et al., 2007) and analyzes their interest for classification and explanation. Finally, Section 5.8 gives a summary and discusses future work.

5.2. The MODL Supervised Discretization and Value Grouping Methods

For the convenience of the reader, this section summarizes the MODL approach in the univariate case, detailed in (Boullé, 2006) for supervised discretization, and in (Boullé, 2005) for supervised value grouping.

5.2.1. Discretization

The objective of supervised discretization is to induce a list of intervals which partitions the numerical domain of a continuous input variable, while keeping the information relative to the output variable. A trade-off must be found between information quality (homogeneous intervals in regard to the output variable) and statistical quality (sufficient sample size in every interval to ensure generalization).

In the MODL approach, the discretization is turned into a model selection problem. First, a space of discretization models is defined. The parameters of a specific discretization model are the number of intervals, the bounds of the intervals and the frequencies of the output values in each interval. Then, a prior distribution is proposed on this model space. This prior exploits the hierarchy of the parameters: the number of intervals is first chosen, then the bounds of the intervals and finally the frequencies of the output values. The prior is uniform at each stage of the hierarchy. Finally, we assume that the multinomial distributions of the output values in each interval are independent from each other. A Bayesian approach is applied to select the best discretization model, which is found by maximizing the probability $p(Model|Data)$ of the model given the data. Using the Bayes rule and since the probability $p(Data)$ is constant under varying the model, this is equivalent to maximizing $p(Model)p(Data|Model)$.

Let N be the number of instances, J the number of output values, I the number of input intervals. N_i denotes the number of instances in the interval i and N_{ij} the number of instances of output value j in the interval i. In the context of supervised classification, the number of instances N and the number of classes J are supposed to be known. A discretization model M is then defined by the parameter set $\left\{I, \{N_i\}_{1 \le i \le I}, \{N_{ij}\}_{1 \le i \le I, 1 \le j \le J}\right\}$.

Using the definition of the model space and its prior distribution, Bayes formula can be used to calculate the exact prior probabilities of the models and the probability of the data given a model. Taking the negative log of the probabilities, this provides the evaluation criterion given in Formula 5.1.

$$\log N + \log \binom{N+I-1}{I-1} + \sum_{i=1}^{I} \log \binom{N_i+J-1}{J-1} + \sum_{i=1}^{I} \log \frac{N_i!}{N_{i1}!N_{i2}!\ldots N_{iJ}!} \tag{5.1}$$

The first term of the criterion stands for the choice of the number of intervals and the second term for the choice of the bounds of the intervals. The third term corresponds to the parameters of the multinomial distribution of the output values in each interval and the last term represents the conditional likelihood of the data given the model, using a multinomial term. Therefore "complex" models with large numbers of intervals are penalized.

Once the evaluation criterion is established, the problem is to design a search algorithm in order to find a discretization model that minimizes the criterion. In (Boullé, 2006), a standard greedy bottom-up heuristic is used to find a good discretization. In order to further improve the quality of the solution, the MODL algorithm performs post-optimizations based on hill-climbing search in the neighborhood of a discretization. The neighbors of a discretization are defined with combinations of interval splits and interval merges. Overall, the time complexity of the algorithm is $O(JN \log N)$.

The MODL discretization method for supervised classification provides the most probable discretization given the data. Extensive comparative experiments report high performance (Boullé, 2006).

5.2.2. Value Grouping

Categorical variables are analyzed in a way similar to that for numerical variables, using a partitioning model of the input values. In the numerical case, the input values are constrained to be adjacent and the only considered partitions are the partitions into intervals. In the categorical case, there are no such constraints between the values and any partition into groups of values is possible. The problem is to improve the reliability of the estimation of the class conditional probabilities owing to a reduced number of groups of values, while keeping the groups as informative as possible. Producing a good grouping is harder with large numbers of input values since the risk of overfitting the data increases. In the extreme situation where the number of values is the same as the number of instances, overfitting is obviously so important that efficient grouping methods should produce one single group, leading to the elimination of the variable.

Again, let N be the number of instances, V the number of input values, J the number of output values and I the number of input groups. N_i denotes the number of instances in the group i, and N_{ij} the number of instances of output value j in the group i. The Bayesian model selection approach is applied like in the discretization case and provides the evaluation criterion given in Formula 5.2. This formula has a similar structure as that of Formula 5.1. The two first terms correspond to the prior distribution of the partitions of the input values, into groups of values in Formula 5.2 and into intervals in Formula 5.1. The two last terms are the same in both formula.

$$\log V + \log B(V,I) + \sum_{i=1}^{I} \log \binom{N_i+J-1}{J-1} + \sum_{i=1}^{I} \log \frac{N_i!}{N_{i1}!N_{i2}!\ldots N_{iJ}!} \tag{5.2}$$

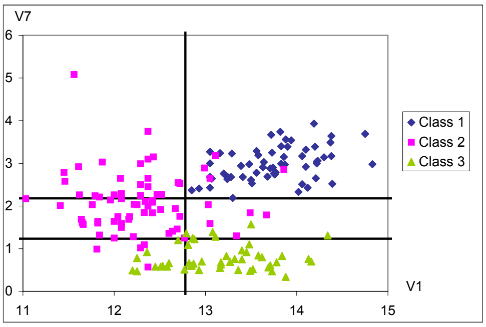
$B(V,I)$ is the number of divisions of V values into I groups (with eventually empty groups). When $I = V$, $B(V,I)$ is the Bell number. In the general case, $B(V,I)$ can be written as $B(V,I) = \sum_{i=1}^{I} S(V,i)$, where $S(V,i)$ is the Stirling number of the second kind (see Abramowitz and Stegun, 1970), which stands for the number of ways of partitioning a set of V elements into i nonempty sets. In (Boullé, 2005), a standard greedy bottom-up heuristic is proposed to find a good partition of the input values. Several pre-optimization and post-optimization steps are incorporated, in order to both ensure an algorithmic time complexity of $O(JN\log(N))$ and obtain accurate value groupings.

5.3. Supervised Data Grids Models for any Subset of Variables

In this section, we describe the extension of the MODL approach to pairs of variables introduced in (Boullé, 2007a) and generalize it to any subset of input variables variables for supervised learning, in the numerical, categorical and mixed type case. We first introduce the approach using an illustrative example for the case of supervised bivariate discretization, then summarizes the principles of the extension in the general case, and present the evaluation criterion of such models. Finally, we relate our modeling approach to information theory and discuss the robustness of our method.

5.3.1. Interest of the joint partitioning of two input variables

Figure 5.1 gives a multiple scatter plot (per class value) of the input variables V1 and V7 of the wine dataset (Blake and Merz, 1996). This diagram shows the conditional probability of the output values given the pair of input variables. The V1 variable taken alone cannot separate Class 1 from Class 3 for input values greater than 13. Similarly, the V7 variable is a mixture of Class 1 and Class 2 for input values greater than 2. Taken jointly, the two input variables allow a better separation of the class values.



]2.18, +∞[	(0, 23, 0)	(59, 0, 4)
]1.235, 2.18]	(0, 35, 0)	(0, 5, 6)
]-∞, 1.235]	(0, 4, 11)	(0, 0, 31)
V7xV1	]-∞, 12.78]	]12.78, +∞[

Figure 5.1: Multiple scatterplot (per class value) of the input variables V1 and V7 of the wine dataset. The optimal MODL supervised bivariate partition of the input variables is drawn on the multiple scatterplot, and the triplet of class frequencies per data grid cell is reported in the right table

Extending the univariate case, we partition the dataset on the cross-product of the input variables to quantify the relationship between the input and output variables. Each input variable is partitioned into a set of *parts* (intervals in the numerical case). The cross-product of the univariate input partitions defines a *data grid*, which partitions the instances into a set of *data* *cells*. Each data cell is defined by a pair of parts. The connection between the input variables and the output variable is evaluated using the distribution of the output values in each cell of the data grid. It is worth noting that the considered partitions can be factorized on the input variables. For instance in Figure 5.1, the V1 variable is discretized into 2 intervals (one bound at 12.78) and the V7 variable into 3 intervals (two bounds at 1.235 and 2.18). The instances of the dataset are distributed in the resulting bidimensional data grid. In each cell of the data grid, we consider the empirical distribution of the output values. For example, the cell defined by the intervals $]12.78, +\infty[$ on V1 and $]2.18, +\infty[$ on V7 contains 63 instances. These 63 instances are distributed on 59 instances for Class 1 and 4 instances for Class 3. Coarse grain data grids tend to be reliable, whereas fine grain data grids allow a better separation of the output values. In our example, the MODL optimal data grid is drawn on the multiple scatter plot on Figure 5.1.

5.3.2. Principles of the Extension to Data Grid Models

The MODL approach has been studied in the case of univariate supervised partitioning for numerical variables (Boullé, 2006) and categorical variables (Boullé, 2005). The extension to the multivariate case applies the same principles as those described in Section 5.3.1. Each input variable is partitioned, into intervals in the numerical case and into groups of values in the categorical case. Taking the cross-product of the univariate partitions, we obtain a data grid of input cells, the content of which characterizes the distribution of the output values. Compared to the bivariate case, we introduce a new level in the hierarchy of the model parameters, related to variable selection. Indeed, a multivariate data grid model implicitly handles variables selection, where the selected variables that bring predictive information are partitioned in at least two parts. The other variables, the partition of which consists of one single part, can be considered as irrelevant and discarded.

The space of multivariate data grid models is very large. Selecting the best model is a difficult task, both from a model selection and optimization point of view. In our approach, we:

1. precisely define the parameters of the data grid models,
2. define a prior on the model parameters,
3. establish an analytic criterion to evaluate the posterior probability of each model
4. design sophisticated optimization algorithm to search the maximum a posteriori (MAP) model.

Our space of models is data dependent: we exploit the input data in order to define the model parameters and restrict their range. Note that our space of models is both nonparametric (the number of parameters increase with the size of the data) and finite (each parameter is discrete with a range bounded according to the input data). To select the best model, we adopt a Bayesian approach and define a prior distribution on the model parameters. Following the principle of parsimony, our prior exploits the hierarchy of the parameters and is uniform at each stage of this hierarchy. We then obtain an analytic formula that evaluates the exact posterior probability of each data grid model. Finally, we exploit the combinatorial algorithm described in Section 5.5 to efficiently search the space of data grid models.

5.3.3. Evaluation Criterion for Supervised Data Grids

We present in Definition 5.1 a family of multivariate partitioning models and select the best model owing to a Bayesian model selection approach.

Definition 5.1 *A data grid classification model is defined by a subset of selected input variables, for each selected variable by a univariate partition, into intervals in the numerical case* *and into groups of values in the categorical case, and by a multinomial distribution of the output values in each cell of the data grid resulting from the cross-product of the univariate partitions.*

Notation.

- N: number of instances,
- Y: output variable,
- J: number of output values,
- $X_1, \ldots, X_K$: input variables,
- K: number of input variables,
- $\mathcal{K}$: set of input variables ($|\mathcal{K}| = K$),
- $\mathcal{K}_n$: subset of numerical input variables,
- $\mathcal{K}_c$: subset of categorical input variables,
- $V_k, k \in \mathcal{K}_c$: number of values of the categorical input variable X_k,
- K_s: number of selected input variables,
- $\mathcal{K}_s$: subset of selected input variables ($|\mathcal{K}_s| = K_s$),
- I_k: number of parts (intervals or groups of values) in the univariate partition of input variable X_k,
- $N_{i_1 i_2 \ldots i_K}$: number of instances in the input data cell $(i_1, i_2, \ldots, i_K)$,
- $N_{i_1 i_2 \ldots i_K j}$: number of instances of output value j in the input data cell $(i_1, i_2, \ldots, i_K)$.

Like the bivariate case, presented in Section 5.3.1, any input information is used to define the family of the model. For example, the numbers of instances per cell $N_{i_1 i_2 \ldots i_K}$ do not belong to the parameters of the data grid model: they are derived from the definition of the univariate partitions of the selected input variables and from the dataset. These numbers of instances allow the specification of the multinomial distribution of the output values in each input cell to be constrained.

We now introduce in Definition 5.2 a prior distribution on the parameters of the data grid models.

Definition 5.2 *The hierarchical prior of the data grid models is defined as follows:*

- *the number of selected input variables is uniformly distributed between* 1 *and* K*,*
- *for a given number* K_S *of selected input variables, the subsets of* K_S *variables are uniformly distributed (with replacement),*
- *the numbers of input parts, are independent from each other, and uniformly distributed between* 1 *and* N *for numerical variables, between* 1 *and* V_k *for categorical variables,*
- *for each numerical input variable and for a given number of intervals, every partition into intervals is equiprobable,*
- *for each categorical input variable and for a given number of groups, every partition into groups is equiprobable,*
- *for each cell of the input data grid, every distribution of the output values is equiprobable,*
- *the distributions of the output values in each cell are independent from each other.*

Applying the MODL approach, this prior exploits the hierarchy of the parameters and is uniform at each stage of this hierarchy.

For the variable selection parameters, we reuse the prior introduced by Boullé (2007b) in the case of the selective naïve Bayes classifier. We first choose the number of variables and second the subset of selected variables. For the number of selected variables K_s, we adopt a uniform prior between 0 and K variables, representing $(K+1)$ equiprobable alternatives. For the choice of the K_s variables, we assign the same probability to every subset of K_s variables. The number of combinations $\binom{K}{K_s}$ seems the natural way to compute this prior, but it has the disadvantage of being symmetric. Beyond $K/2$ variables, adding variables is favored. As we prefer simpler models, we propose to use the number of combinations with replacement $\binom{K+K_s-1}{K_s}$, which leads to a prior probability decreasing with the number of variables.

For the specification of each univariate partition, we reuse the prior presented by Boullé (2006) for supervised discretization of numerical variables and by Boullé (2005) for supervised value grouping of categorical variables (see Section 5.2). We apply the Bayesian model selection approach and obtain the evaluation criterion of a data grid model in Formula 5.3.

$$\begin{aligned}
&\log(K+1)+\log\binom{K+K_s-1}{K_s}\\
&+\sum_{k\in\mathcal{K}_s\cap\mathcal{K}_n}\left(\log N+\log\binom{N+I_k-1}{I_k-1}\right)+\sum_{k\in\mathcal{K}_s\cap\mathcal{K}_c}\left(\log V_k+\log B(V_k,I_k)\right)\\
&+\sum_{i_1=1}^{I_1}\sum_{i_2=1}^{I_2}\dots\sum_{i_K=1}^{I_K}\log\binom{N_{i_1 i_2\dots i_K}+J-1}{J-1}\\
&+\sum_{i_1=1}^{I_1}\sum_{i_2=1}^{I_2}\dots\sum_{i_K=1}^{I_K}\left(\log N_{i_1 i_2\dots i_K}!-\sum_{j=1}^{J}\log N_{i_1 i_2\dots i_K j}!\right)
\end{aligned} \tag{5.3}$$

The first line in Formula 5.3 corresponds to the prior for variable selection. As in the univariate case, the second line is related to the prior probability of the discretization parameters (like in Formula 5.1) for the selected numerical input variables and to that of the value grouping parameters (like in Formula 5.2) for the selected categorical input variables. The binomial terms in the third line represent the choice of the multinomial distribution of the output values in each cell of the input data grid. The multinomial terms in the last line represent the conditional likelihood of the output values given the data grid model.

5.3.4. Relation with Information Theory

Let us first introduce the null model $M_\emptyset$, where no input variable is selected. The null model is composed of a single cell containing all the instances. Applying Formula 5.3, the cost $c(M_\emptyset)$ of the null model (its value according to evaluation criterion 5.3) reduces to

$$c(M_\emptyset)=\log(K+1)+\log\binom{N+J-1}{J-1}+\log\frac{N!}{N_1!N_2!\dots N_J!},$$

where N_j denotes the frequency of the output value j. This corresponds to the posterior probability of a multinomial model of the output variable, independently of any input variable. To get an asymptotic evaluation of the cost of the null model, we now introduce the Shannon entropy $H(Y)$ (Shannon, 1948) of the output variable, $H(Y)=-\sum_{j=1}^{J}p_j\log p_j$, where p_j if the prior probability of the output value j. Using the approximation $\log N! = N(\log N-1)+O(\log N)$ based on Stirling's formula, the cost of the null model is asymptotically equivalent to N times the Shannon entropy of the output variable:

$$c(M_\emptyset) = NH(Y) + O(\log N). \tag{5.4}$$

As the negative log of a probability can be interpreted as a coding length (Shannon, 1948), our model selection technique is closely related to the minimum description length (MDL) approach (Rissanen, 1978; Hansen and Yu, 2001; Grünwald et al., 2005), which aims to approximate the Kolmogorov complexity (Li and Vitanyi, 1997) for the coding length of the output data. The Kolmogorov complexity is the length of the shortest computer program that encodes the output data given the input data.

Overall, our prior approximates the Kolmogorov complexity of the data grid model given the input data and our conditional likelihood encodes the output values given the data grid model. In our approach, the choice of the null model corresponds to the lack of predictive information. The coding length of the null model is asymptotically equivalent to the Shannon entropy the output data (cf. Formula 5.4), which corresponds to a basic encoding of the output data, with no use of the input data. This is close to the idea of Kolmogorov, who considers data to be random if its algorithmic complexity is high, that is if it cannot be compressed significantly. This makes our approach very robust, since detecting predictive information using data grid models is necessarily related to a coding length better than that of the null model, thus to non random patterns according Kolmogorov's definition of randomness. This robustness has been confirmed using extensive experiments in the univariate case (Boullé, 2006, 2005), and is evaluated in the multivariate case in Section 5.6.

5.4. Data Grid Models for Coclustering of Instances and Variables

In (Boullé, 2008b), we have investigated the extension of data grid models to unsupervised learning, in order to evaluate the joint probability distribution of any subset of variables, numerical or categorical. In Section 5.4.1, we present a detailed description of these models in the case of two categorical variables. In Section 5.4.2, we show how to apply such bivariate categorical models to the problem of coclustering the instances and variables of a dataset, as a data preparation technique for supervised learning.

5.4.1. Bivariate Value Grouping of Categorical Variables

In this section, we focus on the case of two categorical variables. We introduce unsupervised bivariate data grid models and their evaluation criterion. We then show how such models can be interpreted as nonparametric models of the correlation between the variables.

5.4.1.1. PRESENTATION

Our objective is to provide a joint description of two categorical variables Y_1 and Y_2, as illustrated in Figure 5.2. In the case of categorical variables with many values, the contingency table between the variables is sparse and does not allow the identification of reliable correlations. Standard statistical tests rely on approximations which are valid only asymptotically. For example, the chi-square test requires an expected frequency of at least 5 in each cell of the contingency table (Cochran, 1954), which does not permit its application in sparse cases. Grouping the values of each variable allows the cell frequencies to be raised (at the expense of potentially mixing interesting patterns), and gives greater confidence in the observed correlation. However, since many grouping models might be considered, there is a risk of overfitting the data. The issue is to find a trade-off between the quality of the density estimation and the generalization ability, on the basis of the granularity of the grid.

D	∅	•	∅	•
C	•	∅	•	∅
B	∅	•	∅	•
A	•	∅	•	∅
	a	b	c	d

{B, D}	∅	●
{A, C}	●	∅
	{a, c}	{b, d}

Figure 5.2: Example of joint density for two categorical variables Y_1 having 4 values a, b, c, d and Y_2 having 4 values A, B, C, D. The initial contingency table on the left contains instances only for one half of the cells (tagged as •), and the remaining cells are empty. After the bivariate value grouping, the preprocessed contingency table on the right provides a synthetic description of the correlation between Y_1 et Y_2.

5.4.1.2. FORMALIZATION

Our objective is to describe the joint distribution of the data, which turns into describing the value of the instances for each variable. We introduce a family of unsupervised partitioning models, based on groups of values for each variable and on a multinomial distribution of all the instances on the cells of the resulting data grid. This family of models is formalized in Definition 5.3.

Definition 5.3 *An unsupervised bivariate value grouping model is defined by:*

- *a number of groups for each variable,*
- *for each variable, the repartition of the values into the groups of values,*
- *the distribution of the instances of the data sample among the cells of the resulting data grid,*
- *for each variable and each group, the distribution of the instances of the group on the values of the group.*

Notation.

- Y_1, Y_2: variables (both considered as output variables)
- V_1, V_2: number of values for each variable (assumed as prior knowledge)
- N: number of training instances
- $D = \{D_1, D_2, \ldots, D_n\}$: training instances
- J_1, J_2: number of groups for each variable
- $G = J_1 J_2$: number of cells in the resulting data grid
- $j_1(v_1), j_2(v_2)$: index of the group containing value v_1 (resp. v_2)
- m_{j_1}, m_{j_2}: number of values in group j_1 (resp. j_2)
- n_{v_1}, n_{v_2}: number of instances for value v_1 (resp. v_2)
- N_{j_1}: number of instances in the group j_1 of variable Y_1
- N_{j_2}: number of instances in the group j_2 of variable Y_2
- $N_{j_1 j_2}$: number of instances in the cell (j_1, j_2) of the data grid

We assume that the numbers of values V_1 and V_2 per categorical variable are known in advance and we aim to model the joint distribution of the finite data sample of size N on these values. The family of models introduced in Definition 5.3 is completely defined by the parameters describing the partition of the values into groups of values

$$J_1, J_2, \{j_1(v_1)\}_{1 \leq v_1 \leq V_1}, \{j_2(v_2)\}_{1 \leq v_2 \leq V_2},$$

by the parameters of the multinomial distribution of the instances on the data grid cells

$$\{N_{j_1 j_2}\}_{1 \leq j_1 \leq J_1, 1 \leq j_2 \leq J_2},$$

and by the parameters of the multinomial distribution of the instances of each group on the values of the group

$$\{n_{v_1}\}_{1 \leq v_1 \leq V_1}, \{n_{v_2}\}_{1 \leq v_2 \leq V_2}.$$

The numbers of values per groups m_{j_1} and m_{j_2} are derived from the specification of the partitions of the values into groups: they do not belong to the model parameters. Similarly, the number of instances in each group can be deduced by adding the cell frequencies in the rows or columns of the grid, according to $N_{j_1} = \sum_{j_2=1}^{J_2} N_{j_1 j_2}$ and $N_{j_2} = \sum_{j_1=1}^{J_1} N_{j_1 j_2}$.

In order to select the best model, we apply a Bayesian approach, using the prior distribution on the model parameters described in Definition 5.4.

Definition 5.4 *The prior for the parameters of an unsupervised bivariate value grouping model are chosen hierarchically and uniformly at each level:*

- *the numbers of groups J_1 and J_2 are independent from each other, and uniformly distributed between 1 and V_1 for Y_1, between 1 and V_2 for Y_2,*
- *for a given number of groups J_1 of Y_1, every partition of the V_1 values into J_1 groups is equiprobable,*
- *for a given number of groups J_2 of Y_2, every partition of the V_2 values into J_2 groups is equiprobable,*
- *for a data grid of given size (J_1, J_2), every distribution of the N instances on the $G = J_1, J_2$ cells of the grid is equiprobable,*
- *for a given group of a given variable, every distribution of the instances of the group on the values of the group is equiprobable.*

Taking the negative log of the probabilities, this provides the evaluation criterion given in Theorem 5.5.

Theorem 5.5 *An unsupervised bivariate value grouping model distributed according to a uniform hierarchical prior is Bayes optimal if the value of the following criteria is minimal*

$$\begin{aligned}
&\log V_1 + \log V_2 + \log B(V_1, J_1) + \log B(V_2, J_2) \\
&+ \log \binom{N+G-1}{G-1} + \sum_{j_1=1}^{J_1} \log \binom{N_{j_1} + m_{j_1} - 1}{m_{j_1} - 1} + \sum_{j_2=1}^{J_2} \log \binom{N_{j_2} + m_{j_2} - 1}{m_{j_2} - 1} \\
&+ \log N! - \sum_{j_1=1}^{J_1} \sum_{j_2=1}^{J_2} \log N_{j_1 j_2}! \\
&+ \sum_{j_1=1}^{J_1} \log N_{j_1}! + \sum_{j_2=1}^{J_2} \log N_{j_2}! - \sum_{v_1=1}^{V_1} \log n_{v_1}! - \sum_{v_2=1}^{V_2} \log n_{v_2}!
\end{aligned} \tag{5.5}$$

The first line in Formula 5.5 relates to the prior distribution of the group numbers J_1 and J_2 and to the specification the partition of the values in groups for each variable. These terms are the same as in the case of the MODL supervised univariate value grouping method (Boullé, 2005), summarized in Section 5.2.2. The second line in Formula 5.5 represents the specification of the parameters of the multinomial distribution of the N instances on the G cells of the data grid, followed by the specification of the multinomial distribution of the instances of each group on the values of the group. The third line stands for the likelihood of the distribution of the instances on the data grid cells, by the mean of a multinomial term. The last line corresponds to the likelihood of the distribution of the values locally to each group, for each variable.

5.4.1.3. Interpretation

The null model $M_\emptyset$ contains one single cell and Formula 5.5 reduces to

$$\begin{aligned} c(M_\emptyset) = & \log V_1 + \log V_2 + \log \binom{N+V_1-1}{V_1-1} + \log \binom{N+V_2-1}{V_2-1} \\ & + \log \frac{N!}{n_{v_1}! n_{v_2}! \ldots n_{V_1}!} + \log \frac{N!}{n_{v_1}! n_{v_2}! \ldots n_{V_2}!} \end{aligned} \tag{5.6}$$

which corresponds to the posterior probability of the multinomial model for the distribution of the instances on the values, for each variable. This means that each variable is described independently.

More complex data grid models allow a nonparametric description of the correlation between the variables, by the means of cells where groups of values are correlated. The penalization of the model is balanced by a shorter description of each variable given the model. The best trade-of is searched using a Bayesian model selection approach.

Example with two identical categorical variables. Let us consider two identical categorical variables $Y_1 = Y_2$ and the maximum data grid model M_{Max} with as many groups as values ($J_1 = V_1$), as illustrated in Figure 5.3. The evaluation criterion of the data grid is equal to

$$c(M_{Max}) = 2\log V_1 + 2\log B(V_1, V_1) + \log \binom{N+V_1^2-1}{V_1^2-1} + \log \frac{N!}{n_{v_1}! n_{v_2}! \ldots n_{V_1}!} \tag{5.7}$$

d	$\emptyset$	$\emptyset$	$\emptyset$	●
c	$\emptyset$	$\emptyset$	●	$\emptyset$
b	$\emptyset$	●	$\emptyset$	$\emptyset$
a	●	$\emptyset$	$\emptyset$	$\emptyset$
	a	b	c	d

Figure 5.3: Bivariate value grouping data grid with as many groups as values for two identical categorical variables $Y_1 = Y_2$, having four values a, b, c and d.

If we compare $c(M_\emptyset)$ in Formula 5.6 to $c(M_{Max})$ in Formula 5.7 in the case of two identical categorical variables, we observe an overhead in the prior terms of the maximum model (specification of the value grouping with Bell numbers and specification of the distribution of the N instances on the ${V_1}^2$ cells of the grid). On the other hand, the likelihood term is divided by a factor two: since the correlation between the variables is perfectly detected using the data grid model, describing the joint distribution of the data given the model reduces to describing the distribution of one single variable.

Let us now compare Formulae (5.6) and (5.7) in the asymptotic case. The multinomial term for the distribution of the values of a categorical variable can be approximated with

$$\log \frac{N!}{n_{v_1}!n_{v_2}!\dots n_{V_1}!} \approx NH(Y_1),$$

where $H(Y_1)$ is the Shannon entropy of variable Y_1 (Shannon, 1948). In the case of the null model having one single cell, we get

$$c(M_\emptyset) \approx 2(V_1 - 1)\log N + 2NH(Y_1).$$

In the case of the maximum model with as many groups as values, we obtain

$$c(M_{Max}) \approx (V_1^2 - 1)\log N + NH(Y_1).$$

The maximum model, which detects the correlation between the variables, will thus be preferred as soon as there are enough instances compared to the number of values. It is worth noting that Formulae (5.6) and (5.7) allow us to select the best model in the non-asymptotic case.

5.4.2. Coclustering of Instances and Variables

In this section, we first introduce the application of unsupervised bivariate data grids to the coclustering problem, and then describe how to build a classifier on the basis of a coclustering model.

5.4.2.1. COCLUSTERING

A coclustering (Hartigan, 1972) is the simultaneous clustering of the rows and columns of a matrix. In case of sparse binary datasets, coclustering is an appealing data preparation technique to identify the correlation between clusters of instances and clusters of variables (Bock, 1979).

Let us consider a sparse binary dataset with N instances, K variables and V non-null values. A sparse dataset can be represented in tabular format, with two columns and V rows. This corresponds to a new *dataset* with two *variables* named "Instance ID" and "Variable ID" where each *instance* is a couple of values (Instance ID, Variable ID), like in Figure 5.4.

$$\begin{array}{c|c|c|c|c|c|c}
 & V_1 & V_2 & V_3 & V_4 & V_5 & \dots \\ \hline
I_1 & 0 & 1 & 0 & 0 & 0 & \dots \\ \hline
I_2 & 0 & 0 & 1 & 1 & 0 & \dots \\ \hline
I_3 & 0 & 1 & 0 & 0 & 0 & \dots \\ \hline
I_4 & 0 & 0 & 0 & 1 & 1 & \dots \\ \hline
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}
\quad \longrightarrow \quad
\begin{array}{|c|c|}
\multicolumn{1}{c}{InstanceID} & \multicolumn{1}{c}{VariableID} \\ \hline
I_1 & V_2 \\ \hline
I_2 & V_3 \\ \hline
I_2 & V_4 \\ \hline
I_3 & V_2 \\ \hline
I_4 & V_4 \\ \hline
I_4 & V_5 \\ \hline
\vdots & \vdots
\end{array}$$

Figure 5.4: Sparse binary dataset: from the sparse (instances × variables) table to the dense bivariate representation.

The application of bivariate unsupervised data grid models forms groups of instance *IDs* and groups of variable *IDs*, so as to maximize the correlation between instances and variables.

We expect to find "natural" patterns both in the space of instances and in the space of variables. It is worth noting that the clusters retrieved by data grid models are non-overlapping, since they form a partition of the whole dataset.

5.4.2.2. Application to Semi-Supervised Learning

We apply a semi-supervised learning approach (Chapelle et al., 2006) to exploit all the data from the train, validation and test datasets. In the first step, all of the instances are processed without any output label to identify the "natural" clusters of instances owing to the data grid coclustering technique. In a second step, the available labeled instances are used to describe the output distribution in each cluster of instances. The label of a test instance is then predicted according to the output distribution of its cluster.

Preprocessing the data with semi-supervised coclustering makes sense under the assumption that the "natural" clusters are correlated with the output values (predefined clusters). We expect this assumption to be true for some datasets, especially in the pattern recognition domain.

5.5. Optimization Algorithm for Multivariate Data Grids

The space of data grid models is so large that straightforward algorithms will almost surely fail to obtain good solutions within a practicable computational time.

Given that the MODL criterion is optimal, the design of sophisticated optimization algorithms is both necessary and meaningful. In this section, we describe such algorithms. They finely exploit the sparseness of the data grids and the additivity of the MODL criterion, and allow a deep search in the space of data grid models with $O(KN)$ memory complexity and $O(N\sqrt{N}\log N \max(K, \log N))$ time complexity.

5.5.1. Greedy Bottom-Up Heuristic

Let us first focus on the case of numerical input variables. The optimization of a data grid is a combinatorial problem. For each input variable X_k, there are 2^N possible univariate discretizations, which represents $\left(2^N\right)^K$ possible multivariate discretizations. An exhaustive search over the whole space of models is unrealistic.

We describe in Algorithm 5.1 a greedy bottom up merge heuristic (GBUM) to optimize the data grids. The method starts with the maximum data grid M_{Max}, which corresponds to the finest possible univariate partitions, based on single value parts, intervals or groups. It evaluates all the merges between adjacent parts for any variables (ties are broken randomly), and performs the best merge if the evaluation criterion decreases after the merge. The process is iterated until no further merge can decrease the criterion.

Each evaluation of a data grid requires $O(N^K)$ time, since the initial data grid model M_{Max} contains N^K cells. Each step of the algorithm relies on $O(N)$ evaluations of interval merges times the number K of variables. There are at most $O(KN)$ steps, since the data grid becomes equal to the null model $M_\emptyset$ (one single cell) once all the possible merges have been performed. Overall, the time complexity of the algorithm is $O(K^2N^2N^K)$ using a straightforward implementation of the algorithm. However, the GBUM algorithm can be optimized in $O(K^2N\log N)$ time, as shown in next section and demonstrated in (Boullé, 2008a) in the bivariate case.

5.5.2. Optimized Implementation of the Greedy Heuristic

The optimized algorithm mainly exploits the sparseness of the data and the additivity of the evaluation criterion. Although a data grid may contain $O(N^K)$ cells, at most N cells are non

Algorithm 5.1: Greedy Bottom-Up Merge heuristic (GBUM)

```
Require: $M$ {Initial data grid solution}
Ensure: $M^*, c(M^*) \leq c(M)$ {Final solution with improved cost}
 1: $M^* \leftarrow M$
 2: while improved solution do
 3:    {Evaluate all the merges between adjacent parts}
 4:    $c^* \leftarrow \infty, m^* \leftarrow \emptyset$
 5:    for all Variable $X_k \in \mathscr{K}$ do
 6:       for all Merge $m$ between two adjacent parts of variable $X_k$ do
 7:          $M' \leftarrow M^* + m$ {Evaluate merge $m$ on data grid $M^*$}
 8:          if $c(M') < c^*$ then
 9:             $c^* \leftarrow c(M'), m^* \leftarrow m$
10:          end if
11:       end for
12:    end for
13:    {Perform best merge}
14:    if $c^* < c(M^*)$ then
15:       $M^* \leftarrow M^* + m^*$
16:    end if
17: end while
```

empty. Thus, each evaluation of a data grid can be performed in O(N) time owing to a specific algorithmic data structure.

The additivity of the evaluation criterion means that it can be decomposed according to Definition 5.6 on the hierarchy of the components of the data grid: grid size, variables, parts and cells.

Definition 5.6 *An evaluation criterion $c(M)$ of a data grid model M is additive if it can be decomposed as a sum of terms according to*

$$c(M) = c^{(G)}(\mathscr{I}) + \sum_{k=1}^{K} c^{(V)}(X_k, I_k) + \sum_{k=1}^{K} \sum_{i_k=1}^{I_k} c^{(P)}\left(P_{i_k}^{(k)}\right) + \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_k=1}^{I_K} c^{(C)}(C_{i_1 i_2 \ldots i_K})$$

where

- *the grid criterion $c^{(G)}(\mathscr{I})$ relies only on the sizes $\mathscr{I} = \{I_1, I_2, \ldots, I_K\}$ of the univariate partitions of the data grid,*
- *the variable criterion $c^{(V)}(X_k, I_k)$ relies only on features of the input variable X_k and on the number of parts I_k of its partition,*
- *the part criterion $c^{(P)}\left(P_{i_k}^{(k)}\right)$ for each part $P_{i_k}^{(k)}$ of the univariate partition of the input variable X_k relies only on features of the part,*
- *the cell criterion $c^{(C)}(C_{i_1 i_2 \ldots i_K})$ for each cell $C_{i_1 i_2 \ldots i_K}$ of the data grid relies only on features of the cell, and is null for empty cells.*

One can easily check that the evaluation criteria introduced in Formula 5.3 or Formula 5.5 are additive. Using this additivity property, all the merges between adjacent parts can be evaluated in O(N) time. Furthermore, when the best merge is performed, the only merges that need to be re-evaluated for the next optimization step are the merges that share instances with the best merge. Since the data grid is sparse, the number of partial re-evaluations of the criterion is limited by the number of instances, not by the number of cells in the data grids. Sophisticated algorithmic data structures and algorithms, detailed in (Boullé, 2008a), are necessary to exploit these optimization principles and guarantee a time complexity of $O(K^2 N \log N)$.

5.5.3. Post-Optimization

The greedy heuristic is time efficient, but it may fall into a local optimum. First, the greedy heuristic may stop too soon and produce too many parts for each input variable. Second, the boundaries of the intervals may be sub-optimal since the merge decisions of the greedy heuristic are never reconsidered. We propose to reuse the post-optimization algorithms described in (Boullé, 2006) in the case of univariate discretization.

In a first stage called *exhaustive merge*, the greedy heuristic merge steps are performed without referring to the stopping condition until the data grid consists of one single cell. The best encountered data grid is then memorized. This stage allows escaping local minima with several successive merges and needs $O(K^2 N \log N)$ time.

In a second stage called *greedy post-optimization*, a hill-climbing search is performed in the neighborhood of the best data grid. This search alternates the optimization on each input variable. For each given input X_k, we freeze the partition of all the other input variables and optimize the partition of X_k. Since a multivariate additive criterion turns out to be an univariate additive criterion once all except one univariate partitions are frozen, we reuse the post-optimization algorithms described in (Boullé, 2006) for univariate discretizations. This process is repeated for all variables until no further improvement can be obtained. This algorithm converges very quickly in practice and requires only a few steps.

We summarize the post-optimization of data grids in Algorithm 5.2.

Algorithm 5.2: Post-optimization of a Data Grid

```
Require: $M$ {Initial data grid solution}
Ensure: $M^*; c(M^*) \leq c(M)$ {Final solution with improved cost}
1: $M^* \leftarrow$ call exhaustive merge $(M)$
2: while improved solution do
3:     {Take a random permutation of $\mathcal{K}$}
4:     for all Variable $X_k \in \mathcal{K}$ do
5:         Freeze the univariate partition of all the variables except $X_k$
6:         $M^* \leftarrow$ call univariate post-optimization $(M^*)$ for variable $X_k$
7:     end for
8: end while
```

5.5.4. Meta-Heuristic

Since the GBUM algorithm is time efficient, it is then natural to apply it several times in order to better explore the search space. This is done according to the *variable neighborhood search* (VNS) meta-heuristic introduced by Hansen and Mladenovic (2001), which consists of applying the primary heuristic to a random neighbor of the solution. If the new solution is not better, a bigger neighborhood is considered. Otherwise, the algorithm restarts with the new best solution and a minimal size neighborhood. The process is controlled by the maximum length of the series of growing neighborhoods to explore.

For the primary heuristic, we choose the greedy bottom-up heuristic followed by the post-optimization heuristic. In order to “purify” the randomly generated solutions given to the primary heuristic, we also incorporate a pre-optimization heuristic, that exploits the same principle as the post-optimization heuristic.

This meta-heuristic is described in Algorithm 5.3. According to the level of the neighborhood size l, a new solution M' is generated close to the current best solution. We define the structure of neighborhood by exploiting at most $K_{Max} = \log_2 N$ new variables. For each exploited variable, a random discretization is obtained with the choice of random interval bounds without replacement, with at most $I_{Max} = N^{\frac{1}{K_{Max}}}$ intervals. This heuristic choice for the maximum neighborhood size results from the analysis of Formula 5.3. In the case of two equidistributed output values, if we have selected K_{Max} variables with I_{Max} intervals per variable and exactly one instance per input cell, the cost of the model is slightly worse than that of the null model with no selected variable. This means that data grids that are too sparse are not likely to be informative according to Formula 5.3.

The VNS meta-heuristic only requires the number of sizes of neighborhood as a parameter. This can easily be turned into an anytime optimization algorithm, by calling the VNS algorithm iteratively with parameters of increasing size and stopping the optimization only when the allocated time is elapsed. In this paper, all the experiments are performed by calling the VNS algorithm with successive values of $1, 2, 4, \ldots, 2^T$ for the parameter $MaxLevel$.

In order to improve the initial solution, we choose to first optimize the univariate partition of each variable and to build the initial solution from a cross-product of the univariate partitions. Although this cannot help in case of strictly bivariate patterns (such as XOR for example), this might be helpful otherwise.

Algorithm 5.3: VNS meta-heuristic for data grid optimization

```
Require: $M$ {Initial data grid solution}
Require: $MaxLevel$ {Optimization level}
Ensure: $M^*, c(M^* \leq c(M)$ {Final solution with improved cost}
1: $Level \leftarrow 1$
2: while $Level \leq MaxLevel$ do
3:     {Generate a random solution in the neighborhood of $M^*$}
4:     $M'' \leftarrow$ random solution with $K_s = \frac{Level}{MaxLevel} \log_2 N$ new selected variables and $\frac{Level}{MaxLevel} N^{\frac{1}{K_s}}$
       new intervals per selected variable
5:     $M' \leftarrow M^* \cup M''$
6:     {Optimize and evaluate the new solution}
7:     $M' \leftarrow$ call Pre-Optimization$(M')$
8:     $M' \leftarrow$ call Greedy Bottom-Up Merge$(M')$
9:     $M' \leftarrow$ call Post-Optimization$(M')$
10:    if $c(M') < c(M^*)$ then
11:        $M^* \leftarrow M'$
12:        $Level \leftarrow 1$
13:    else
14:        $Level \leftarrow Level + 1$
15:    end if
16: end while
```

5.5.5. The Case of Categorical Variables

In the case of categorical variables, the combinatorial problem is worse still for large numbers of values V. The number of possible partitions of the values is equal to the Bell number

$B(V) = \frac{1}{e}\sum_{k=1}^{\infty}\frac{k^V}{k!}$ which is far greater than the $O(2^N)$ possible discretizations. Furthermore, the number of possible merges between parts is $O(V^2)$ for categorical variables instead of $O(N)$ for numerical variables. Specific pre-processing and post-processing heuristics are necessary to efficiently handle the categorical input variables. Mainly, the number of groups of values is bounded by $O(\sqrt{N})$ in the algorithms, and the initial and final groupings are locally improved by the exchange of values between groups. This allows us to keep an $O(N)$ memory complexity per variable and bound the time complexity by $O(N\sqrt{N}\log N)$ per categorical variable, with an overall time complexity of $O(K^2N\sqrt{N}\log N)$ for the complete greedy heuristic.

5.5.6. Summary of the Optimization Algorithms

The optimization of multivariate data grid models can be summarized as an extension of the univariate discretization and value grouping algorithms to the multivariate case.

The main heuristic is a greedy bottom-up heuristic, which starts from an initial fine grain data grid and iteratively performs the best merges between two adjacent parts of any input variable. Post-optimizations are carried out to improve the best data grid, by exploiting a local neighborhood of the solution. The main optimization heuristic (surrounded by pre-optimization and post-optimization steps) is run from several initial solutions, coming from the exploration of a global neighborhood of the best solution using a meta-heuristic.

These algorithms are efficiently implemented, on the basis of two main properties of the problem: the additivity of the criterion, which consists of a sum of independent terms related to the dimension of the data grid, the variables, the parts and the cells, and the sparseness of the data grids, which contain at most N non empty cells for $O(N^K)$ cells. Furthermore, in the meta-heuristic, we restrict to data grids with at most $K_{Max} = \log_2 N$ variables, which reduces the time complexity of the main greedy heuristic.

Sophisticated algorithms, detailed in (Boullé, 2008a), are necessary to make the most of these problem properties and to reach the following algorithmic performance:

- $O(KN)$ memory complexity for K variables and N instances,
- $O(KN\log N\max(K,\log N))$ if all the input variables are numerical,
- $O(KN\sqrt{N}\log N\max(K,\log N))$ in the general case of numerical variables and categorical variables having large number of input values ($V \geq \sqrt{N}$).

5.6. Experiments on Artificial Datasets

In the bivariate case, the data grid models have been intensively evaluated on artificial and real datasets in (Boullé, 2007a). In this section, we evaluate the multivariate data grid models on artificial datasets, where the true data distribution is known. Two patterns are considered: noise and multivariate XOR. This enables the evaluation of both the reliability of the method and its rate of convergence for the detection of complex patterns. We also analyze the effect of each part of the algorithm and study the limits of the method.

5.6.1. The Noise Pattern

The purpose of the noise pattern experiment is to evaluate the noise resistance of the method, under varying sample size and the number of input variables. The noise pattern consists of an output variable independent from the input variables. The expected data grid contains one single cell, meaning that the output distribution is independent from the input variables. The output variable is equidistributed on two values. The experiment is performed on a set of sample sizes ranging from 2 to 1000 instances, for 1, 2 and 10 numerical input variables uniformly distributed on the [0, 1] numerical domain. The criterion evaluated is the number of cells in the data grid. In order to obtain reliable results, the experiment is performed one million times on randomly generated training datasets for each sample size and number of input variables. In order to study the impact of variable selection in the prior distribution of the models (terms $\log(K+1) + \log\binom{K+K_s-1}{K_s}$ in Formula 5.3), we perform the experiment with and without the variable selection terms. Figure 5.5 presents the mean cell number for each sample size and number of input variable, with and without the prior for variable selection.

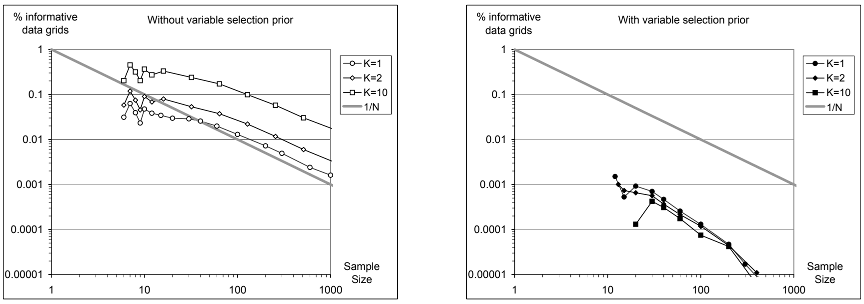


Figure 5.5: Percentage of informative data grids having more than one cell, for 1, 2 and 10 numerical input variable independent from the target variable, with and without prior for variable selection.

The results demonstrate the robustness of the approach: very few data grids are wrongly detected as informative, and the percentage of false detection rapidly decreases with the sample size. However, without prior for variable selection, the percentage of false detection grows almost linearly with the number of input variables. This makes sense since a set of K variables can be detected as an informative multivariate data grid if at most one of the K variables is detected as an informative univariate discretization.

When the prior for variable selection is accounted for, the percentage of wrongly informative models falls down by two orders of magnitude, and the rates of false detection are rapidly consistent for the different numbers of input variables. The selection prior significantly strengthens the robustness of the method and makes it almost independent from the number of variables in the representation space.

5.6.2. The Multivariate XOR Pattern

The purpose of the XOR pattern experiment is to evaluate the capacity of the method to detect complex correlations between the input variables. The pattern consists of an output variable which depends upon the input variables according to a XOR schema, as illustrated in Figure 5.6. All the input variables are uniformly distributed on the [0, 1] numerical domain. For each input variable, we compute a Boolean index according to whether the input value is below or beyond 0.5, and the output value is assigned a Boolean value related to the parity of the sum of the input indexes, which corresponds to a XOR pattern.

We first present a theoretical threshold of detection for the XOR pattern, then illustrate the behavior of the algorithms for this problem, and finally report experimental results on this complex pattern detection problem.

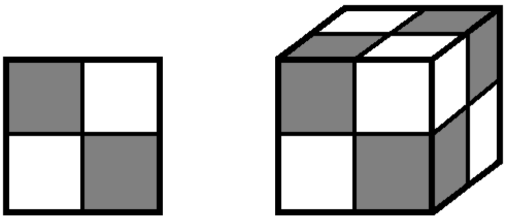


Figure 5.6: Multivariate XOR pattern in dimension 2 and 3.

5.6.2.1. Theoretical Detection Threshold

Let us consider K input variables, K_s of which represent a multivariate XOR pattern related to the output variable. The expected multivariate discretization for this pattern consists of a data grid model M_G with K_s selected input variables, each of which is discretized into two intervals. The data grid model M_G contains $G = 2^{K_s}$ cells. In order to obtain a closed formula, let us assume that these cells contain the same number $N_G = N/G$ of instances. Let us evaluate the null model $M_\emptyset$, reduced to one single cell, and the expected XOR data grid model M_G. According to Formula 5.3, we get

$$c(M_\emptyset) = \log(K+1) + \log(N+1) + \log\frac{N!}{N_1!N_2!}, \quad (5.8)$$

$$c(M_G) = \log(K+1) + \log\binom{K+K_s-1}{K_s-1} + K_s\log N + K_s\log(N+1) + G\log(N_G+1). \quad (5.9)$$

For $N_G = 1$, the null model is always preferred: one instance per cell is not enough to detect the multivariate pattern.

For small values of K and for $K_s = K$, we perform a numerical simulation to compute the minimum cell frequency N_G such that the cost $c(M_G)$ of the multivariate XOR model is lower than that of the null model. The results, reported in Figure 5.7, indicate that at least ten instances per cell, representing overall forty instances, are necessary to detect the bi-dimensional XOR pattern. This cell frequency threshold decreases with the number of input variables, and falls down to two instances per cell when the number of input variables is beyond ten. Let us notice that in spite of a very small cell frequency threshold, the whole dataset frequency threshold still grows exponentially with the number of variables.

We now extend these simulation results in the asymptotic case, assuming that each cell contains exactly $N_G = N/G$ instances. From Equations (5.8) and (5.9), we get

$$c(M_G) = c(M_\emptyset) + \log\binom{K+K_s-1}{K_s-1} + (2K_s-1)\log N - N\left(\log 2 - \frac{1}{N_G}\log(N_G+1)\right) + O(\log N).$$

This implies that for $N_G \geq 2$, the multivariate XOR model has an asymptotically lower cost than that of the null model, even when the total number K of input variables exceeds the number K_s of informative input variables.

Overall, about 2^{K+1} instances are sufficient to detect K-dimensional informative patterns, which correspond to 2 instances per cell. Since this is close from the theoretical detection threshold, this means that for a dataset consisting of N instances, it might be difficult to detect patterns exploiting more than $\log_2 N$ informative dimensions.

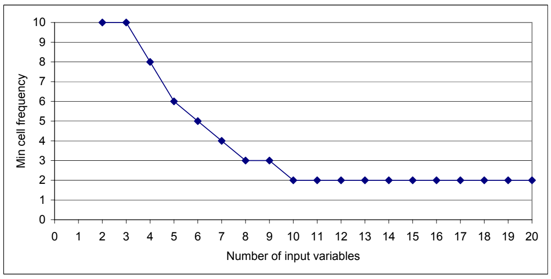


Figure 5.7: Min cell frequency necessary to detect a multivariate XOR pattern using a data grid model. For example, for a 5-dimensional XOR, 6 instances per cell, or $192 = 2^5 * 6$ instances in the sample, allow to detect the pattern using a data grid of 32 cells.

5.6.2.2. EMPIRICAL ANALYSIS OF THE ALGORITHMS

Let us first analyze the behavior of the greedy bottom-up heuristic presented in Section 5.5.1. This heuristic starts with the maximum data grid, which contains $O(N^K)$ cells for at most N non-empty cells. During the whole merge process, $O(KN)$ merges are necessary to transform the maximum data grid with N^K elementary cells into the null data grid with one single cell. During the first $(K-1)N$ merges, most of the merges between adjacent intervals concern merges between two empty adjacent cells or merges between one non-empty cell and one empty cell. When the data grid is too sparse, most interval merges do not involve "collisions" between non-empty cells. According to Formula 5.3, the only cell merges that have an impact on the likelihood of the data grid model are the "colliding" cell merges. This means that at the beginning of the greedy bottom-up heuristic, the earlier merges are guided only by the prior distribution of the models, not by their likelihood. These "blind" merges are thus likely to destroy potentially interesting patterns.

To illustrate this behavior, we perform an experiment with the basic greedy heuristic described in Algorithm 5.1 on a bi-dimensional XOR pattern. According to Formulas 5.8 and 5.9, about 40 instances are sufficient to detect the pattern. However, the greedy bottom-heuristic fails to discover the XOR pattern when the number of instance is below 1000.

The algorithms presented in Section 5.5 enhance the basic greedy heuristic using a random initialization, a pre-processing step, the greedy bottom-up merge heuristic and a post-processing step, as illustrated in Figure 5.8. The random initialization produces a dense enough data grid with at least one instance per cell on average. This is achieved by selecting at most $K_s = \log_2 N$ input variables and N^{1/K_s} parts per variable. The purpose of the pre-processing step is to "purify" the initial solution, since a random solution is likely to be blind to informative patterns. This pre-processing consists in moving the boundaries of the initial data grid, in order to get "cleaner" initial cells, as illustrated in Figure 5.8. The greedy merge heuristic is then applied on this dense cleaned data grid, and the merges are guided by the data, since the data grid is sufficiently dense. The role of the post-processing step is to improve the final solution, by exploring a local neighborhood of the solution consisting of interval splits, merges and moves of interval boundaries.

All these steps are repeated several times in the VNS meta-heuristic described in Section 5.5.4, which generates several random initial data grids of varying size. The only opti-

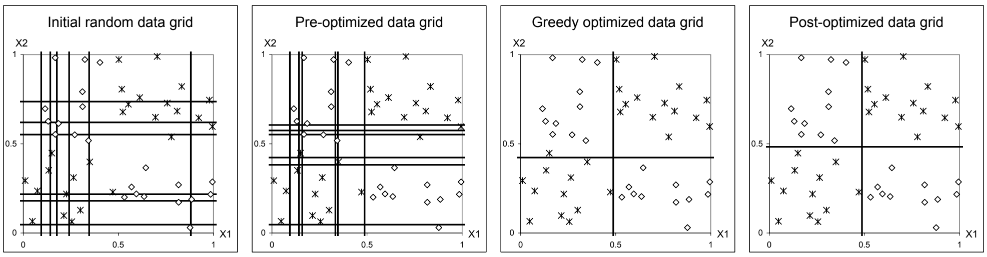


Figure 5.8: Main steps in the optimization algorithm: a random initial solution is first generated to start with a dense enough data grid, then cleaned during a pre-processing step, optimized with the greedy bottom-up merge heuristic and improved during the post-processing step.

mization parameter relates to the number of iterations in the meta-heuristic, which controls the intensity of the search.

A quantitative evaluation of the effect of each part of the algorithm is reported in (Boullé, 2008a) in the case of bivariate XOR patterns. The greedy heuristic alone is likely to be misled by the sparsity of the data and needs a very large number of instances to discover the true patterns. The meta-heuristic, which starts multiple times from dense enough random initial solutions, manages to approximate the true patterns with about 100 times fewer instances than the greedy heuristic. However, the random initialization process is not likely to produce candidate data grids with accurate boundaries. This is corrected by the pre-optimization and post-optimization heuristics.

All of the algorithmic components are useful in achieving an effective search of the space of data grids and efficiently detecting informative patterns. Using these algorithms, the empirical threshold for the detection of simple XOR patterns reaches the theoretical threshold, even with one single iteration in the meta-heuristic. For example, bi-dimensional randomly generated patterns require only 40 instances to be detected, and 5-dimensional XOR pattern only 200 instances. In the following sections, we study the detection of more complex XOR patterns, which require more intensive search.

5.6.2.3. Detection of a Complex Patterns with Few Instances

In this experiment, we study the detection of a 10-dimensional XOR pattern in a 10-dimensional input space. The experiment is performed on a set of sample sizes ranging from 1000 to 10000 instances, and repeated 100 times for each sample size. We evaluate the empirical detection threshold for the VNS meta-heuristic, with optimization parameters T, where VNS(T) performs around 2^T iterations of the algorithm from a variety of random initial data grids. Figure 5.9 reports the average computation time for each sample size and for parameters of the VNS meta-heuristic ranging from $T = 1$ to $T = 12$. We also report the threshold related to the sample size and computation time, among which the XOR pattern is detected in 50% of the cases.

The results show that the empirical detection threshold is close to the theoretical threshold: the pattern is never detected with 1000 instances but frequently detected with only 1500 instances, which is less than 2 instances per cell of the 10-dimensional XOR pattern. However, when the instance number is close to the theoretical threshold, the problem of finding the correct 10 variable splits among N^{10} possible XOR patterns and $(2^N)^{10}$ potential multivariate

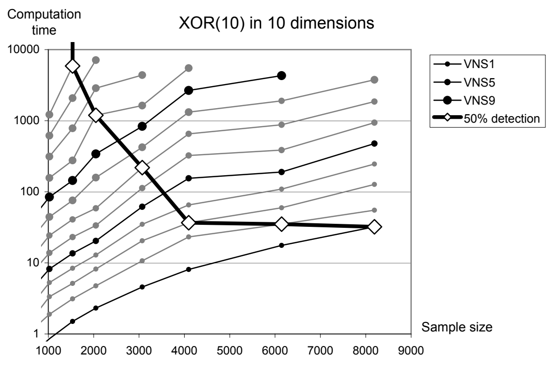


Figure 5.9: Study of the algorithm for the detection of the 10-dimensional XOR pattern.

discretizations is very hard. In this case, detecting the pattern requires much more time than when the instance number is large enough or when the pattern is simpler. For example, detecting the pattern with only 1500 instances requires about one hundred times more computation time than with 4000 instances

5.6.2.4. FINDING A NEEDLE IN A HAYSTACK

In this experiment, we study the detection of a 5-dimensional XOR pattern in a 10-dimensional input space. We use the same protocol as in the previous case, and report the results in Figure 5.10.

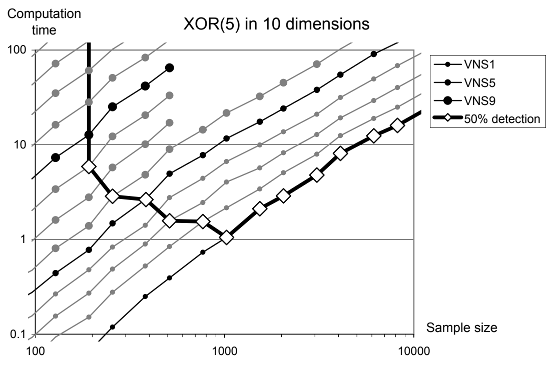


Figure 5.10: Study of the algorithm for the detection of the 5-dimensional XOR pattern, hidden in a 10-dimensional input space.

The results show that about 200 instances are sufficient to detect this pattern, which is consistent with the theoretical threshold. However, whereas the 5-dimensional XOR pattern is easily detected even within one or two iterations in the VNS meta-heuristic, the search in that 10-dimensional input space requires much more intensive search.

Apart from of the problem of finding the correct XOR boundaries, which is a difficult task, the problem of variable selection complicates the detection of the pattern. The optimization algorithm is restricted to the exploration of dense data grids, which consist of $K_x \leq \max(\log_2 N, K)$ dimensions. Finding the XOR pattern requires us to select a subset of K_x input variables among K, which is a superset of the K_s informative variables. The probability that such a subset contains the informative variable is $\binom{K_x}{Ks}/\binom{K}{Ks}$. For example, for the detection of a 5-dimensional XOR ($K_s = 5$) with 256 instances ($K_x = \log_2 256 = 8$), the probability of finding a potentially good subset is 100% for $K = 5$, 22% for $K = 10$, 0.36% for $K = 20$ and 0.04% for $K = 30$.

We performed an experiment to detect the 5-dimensional XOR in 20 dimensions with samples of size 256. The result confirms that there are enough instances for a reliable detection of the pattern, but the computational time necessary to detect the pattern in 50% of the cases amounts to about one hundred times that in 10 dimensions. This result, consistent with the ratio 22/0.36, illustrates the problem of finding a needle in a haystack.

Overall, the evaluation criterion given in Formula 5.3 is able to reliably differentiate informative patterns from noise with very few instances. The detection of complex patterns is a combinatorial problem, that is hard to solve when the number of instances is close to the detection threshold or when the informative patterns are hidden in large dimensional input spaces. Our optimization algorithm succeeds in reliably and efficiently detecting information, with performance close to the theoretical detection threshold.

5.7. Evaluation on the Agnostic Learning vs. Prior Knowledge Challenge

In this section, we first summarize the evaluation protocol of the challenge, then describe how classifiers are built from data grid models, and finally report the results from a performance and understandability point of view.

5.7.1. The Agnostic Learning vs. Prior Knowledge Challenge

The purpose of the challenge (Guyon, 2007; Guyon et al., 2007) is to assess the real added value of prior domain knowledge in supervised learning tasks. Five datasets coming from different domains are selected to evaluate the performance of agnostic classifiers vs. prior knowledge classifiers. These datasets come into two formats, as shown in Table 5.1. In the agnostic format, all the input variables are numerical. In the prior knowledge format, the input variables are both categorical and numerical for three datasets and have a special format in the two other datasets: chemical structure or text. The evaluation criterion is the test balanced error rate (BER).

Table 5.1: Challenge datasets with their prior and agnostic format.

Name	Domain	Num. ex. train/valid/test	Prior features	Agnostic features
Ada	Marketing	4147/415/41471	14	48
Gina	Handwritting reco.	3153/315/31532	784	970
Hiva	Drug discovery	3845/384/38449	Chem. struct.	1617
Nova	Text classification	1754/175/17537	Text	16969
Sylva	Ecology	13086/1309/130857	108	216

5.7.2. Building Classifiers from Data Grid Models

In this section, we describe three ways of building classifiers from data grid models.

5.7.2.1. DATA GRID

In this evaluation of data grid models, we consider one individual supervised data grid, the MAP one. We build a classifier from a data grid model by first retrieving the cell related to a test instance, and predicting the output conditional probabilities of the retrieved cell. For empty cells, the conditional probability used for the prediction is that of the entire grid.

Data grid models can be viewed as a feature selection method, since the input variables whose partition reduces to a single part can be ignored. The purpose of this experiment is to focus on understandable models and evaluate the balance between the number of selected variables and the predictive performance.

5.7.2.2. DATA GRID ENSEMBLE

In this evaluation, we focus on the predictive performance rather than on understandability, by means of averaging the prediction of a large number of classifiers. This principle was successfully exploited in Bagging (Breiman, 1996) using multiple classifiers trained from re-sampled datasets. This was generalized in Random Forests (Breiman, 2001), where the subsets of variables are randomized as well. In these approaches, the averaged classifier uses a voting rule to classify new instances. Unlike this approach, where each classifier has the same weight, the Bayesian Model Averaging (BMA) approach (Hoeting et al., 1999) weights the classifiers according to their posterior probability. The BMA approach has stronger theoretical foundations, but it requires both to be able to evaluate the posterior probability of the classifiers and to sample their posterior distribution.

In the case of data grid models, the posterior probability of each model is given by an analytic criterion. Concerning the problem of sampling the posterior distribution of data grid models, we have to strike a balance between the quality of the sampling and the computation time. We adopt a pragmatic choice by just collecting all the data grids evaluated during training, using the optimization algorithms introduced in Section 5.5. We keep all the local optima encountered in the VNS meta-heuristic and eliminate the duplicates.

An inspection of the data grids collected reveals that their posterior distribution is so sharply peaked that averaging them according to the BMA approach almost reduces to the MAP model. In this situation, averaging is useless. The same problem has been noticed by Boullé (2007b) in the case of averaging Selective Naive Bayes models. To find a trade-off between equal weights as in bagging and extremely unbalanced weights as in the BMA approach, we exploit a logarithmic smoothing of the posterior distribution called compression-based model averaging (CMA), like that introduced in (Boullé, 2007b).

To summarize, we collect the data grid models encountered during the data grid optimization algorithm and weight them according to a logarithmic smoothing of their posterior probability to build a data grid ensemble classifier.

5.7.2.3. COCLUSTERING

The coclustering method introduced in Section 5.4 applies on binary sparse datasets. Whereas the supervised data grids are limited in practice to a small number of selected input variables (see Section 5.6), the coclustering data grids are able to account for all the input variables.

The coclustering data grid is trained on all the available input data (train, validation and test), then the available labeled instances are used to predict the output distribution in each cluster of instances. In the case where a test instance belongs to a cluster with no labeled instance, we iteratively merge this unlabeled cluster so as to keep the coclustering evaluation criterion as low as possible, until at least one labeled cluster is encountered.

5.7.3. Evaluation of Supervised Data Grids

We first analyze the classification performance of supervised data grids, then focus on their benefit for understandability.

5.7.3.1. Classification Results

To evaluate the supervised data grid models, we use all the datasets in their agnostic format and only three of them in their prior format (the ones that come in a tabular format). In the case of the Sylva dataset in its prior format, we replace each subset (per record) of 40 binary SoilType variables by one single categorical variable with 40 values. The resulting dataset has only 30 variables instead of 108.

The data grid techniques are able to predict the output conditional probabilities for each test instance. When the evaluation criterion is the classification accuracy, predicting the class with the highest conditional probability is optimal. This is not the case for the BER criterion used in the challenge. We post-process each trained classifier by optimizing the probability threshold in order to maximize the BER. This optimization is performed directly on the train dataset.

Our four submissions related to supervised data grid models are named *Data Grid (MAP)* and *Data Grid (CMA)* in the prior or agnostic track and dated from February 27, 2007 for the challenge March 1st, 2007 milestone. The classifiers are trained with the any time optimization algorithm described in Section 5.5 using VNS(12) parameter. About 4000 data grids are evaluated, needing around one hour optimization time per dataset. Tables 5.2 and 5.4 report our results in the agnostic and prior track.

Table 5.2: Best challenge results versus our supervised data grid methods results for the datasets in the agnostic track.

Dataset	Winner	Best BER	Data Grid (CMA)	Data Grid (MAP)
Ada	Roman Lutz	0.166	0.1761	0.2068
Gina	Roman Lutz	0.0339	0.1436	0.1719
Hiva	Vojtech Franc	0.2827	0.3242	0.3661
Nova	Mehreen Saeed	0.0456	0.1229	0.2397
Sylva	Roman Lutz	0.0062	0.0158	0.0211

Table 5.3: Best challenge results versus our supervised data grid methods results for the datasets in the agnostic track.

Table 5.4: Best challenge results versus our supervised data grid methods results for the Gina, Hiva and Nova datasets in the prior track.

Dataset	Winner	Best BER	Data Grid (CMA)	Data Grid (MAP)
Ada	Marc Boullé	0.1756	0.1756	0.2058
Gina	Vladimir Nikulin	0.0226	0.1254	0.1721
Sylva	Roman Lutz	0.0043	0.0228	0.0099

The data grid classifiers obtain good results on the Ada and Sylva datasets, especially on the prior track, with a winning submission for the Ada dataset. The other datasets contain very large numbers of variables, which explains the poor performance of the data grid models. Since individual data grid models are essentially restricted to about $\log_2 N$ selected variable, they cannot exploit much of the information contained in the representation space. This is analyzed in Section 5.7.3.2.

The data grid ensemble classifiers confirm the benefits of compression-based model averaging. They obtain a very significant improvement of the BER criterion compared to the individual data grid classifiers. This focus on predictive performance is realized at the expense of understandability, since each trained data grid ensemble averages several hundreds of elementary data grid models.

However, even data grid ensembles fail to achieve competitive performance for datasets with large numbers of variables. A close inspection reveals that although about 4000 data grids are evaluated for each dataset, only a few hundreds (≈ 500) of different solutions are retrieved. Removing the duplicates significantly improves the performances, but there is still too much redundancy between data grids to produce an efficient ensemble classifier. Furthermore, a few hundred redundant classifiers, each with only $\approx \log_2 N$ variables, is not enough to exploit all the variables (think of Nova with 17000 variables for example). In future work, we plan to improve our meta-heuristic in order to better explore the search space and to collect a set of data grid solutions with better diversity.

5.7.3.2. UNDERSTANDABILITY

Let us now focus on understandability and inspect the number of selected variables in each trained data grid model. In the agnostic track, the MAP data grid exploits only 5 variables for Ada, 5 for Gina, 4 for Hiva, 8 for Nova and 8 for Sylva. In the prior track, the MAP data grid exploits 6 variables for Ada, 7 for Gina and 4 for Sylva. These numbers of variables are remarkably small w.r.t. the BER performance of the models.

Table 5.5: Most frequent cells in the best individual data grid model for the Ada dataset in the prior track.

ID	relationship	occupation	education number	age	capital gain	capital loss	frequency	% class 1
1	Married	Low	≤ 12	> 27	≤ 4668	≤ 1805	736	22.1%
2	Not married	Low	≤ 12	> 27	≤ 4668	≤ 1805	577	3.1%
3	Not married	High	≤ 12	> 27	≤ 4668	≤ 1805	531	5.8%
4	Married	High	≤ 12	> 27	≤ 4668	≤ 1805	489	41.3%
5	Married	High	> 12	> 27	≤ 4668	≤ 1805	445	68.5%
6	Not married	Low	≤ 12	≤ 27	≤ 4668	≤ 1805	425	0.2%
7	Not married	High	≤ 12	≤ 27	≤ 4668	≤ 1805	316	0.6%
8	Not married	High	> 12	> 27	≤ 4668	≤ 1805	268	20.5%
9	Not married	High	> 12	≤ 27	≤ 4668	≤ 1805	112	0.9%
10	Married	Low	≤ 12	≤ 27	≤ 4668	≤ 1805	96	5.2%
11	Married	High	> 12	> 27	> 5095	≤ 1805	93	100.0%
12	Married	Low	> 12	> 27	≤ 4668	≤ 1805	50	24.0%

In Table 5.5, we summarize the MAP data grid trained using the 4562 train+valid instances of the Ada dataset in the prior track. This data grid selects six variables among 14 and obtains a 0.2068 test BER. The selected variables are relationship, occupation, education number, age, capital gain and capital loss, which are partitioned into 2, 2, 2, 2, 3 and 3 groups or intervals. The relationship variable is grouped into Married = {Husband, Wife} vs. Not Married = {Not-in-family, Own-child, Unmarried, Other-relative}, and the occupation into Low = {Craft-repair, Other-service, Machine-op-inspct, Transport-moving, Handlers-cleaners, Farming-fishing, Priv-house-serv} vs. High = {Prof-specialty, Exec-managerial, Sales, Adm-clerical, Tech-support, Protective-serv, Armed-Forces}. Overall, the data grid contains $144 = 2*2*2*2*3*3$ cells, but 57 of them are non empty and the twelve most frequent cells reported in Table 5.5 contains 90% of the instances.

Each cell of the data grid can directly be interpreted as a decision rule. For example, the most frequent cell is described by Rule 1, with a support of 736 instances.

Rule 1:	IF	relationship $\in$ Married = {Husband, Wife}
		occupation $\in$ Low = {Craft-repair, Other-service, Machine-op-inspct,...}
		education number ≤ 12
		age > 27
		capital gain ≤ 4668
		capital loss ≤ 1805
	THEN	P(class=1) = 22.1%

The whole data grid forms a set of rules (Mitchell, 1997) which corresponds to a partition (not a coverage) of the training set. Since all rules exploit the same variables with the same univariate partitions, interpretation is much easier. For example, rule 5 (ID cell=5 in Table 5.5) has a large support of 445 instances with 68.5% of class 1. Rule 4 with 41.3% of class 1 only differs in the education number variable (≤ 12 vs. > 12), and rule 8 with 20.5% of class 1 in the relationship variable (Not married vs. Married).

5.7.4. Evaluation of Coclustering Data Grids

We first inspect the dimension of the data grids resulting from the coclustering method introduced in Section 5.4.2, then analyze its performance results and finally present its interest for understandability in the case of the Nova text corpus.

To evaluate the coclustering data grid models, we consider three datasets (Gina, Hiva and Nova) as sparse binary datasets. For the Gina dataset, the binary representation is obtained from the prior format by replacing each non zero value by 1. The Hiva dataset is used directly in its agnostic binary format. For the Nova dataset, we exploit the prior format in order to get insights on the understandability of the models. We preprocess the Nova text format by keeping all words of at least three characters, converting them to lowercase, truncating them to at most seven characters, and keeping the most frequent resulting words (≥ 8) so as to get a manageable bag-of-words representation (we keep the most frequent 19616 words using this frequency threshold). This preprocessing is very similar to that for the agnostic track, except that we do not exclude the 2000 most frequent words.

5.7.4.1. Dimensionality Reduction

The coclustering method exploits all the available unlabeled data to represent the initial binary matrix (instances $\times$ variables) which is potentially sparse into a denser matrix with clusters of instances related to clusters of variables. It is worth noting that the space of coclustering models is very large. For example, in the case of the Nova dataset, the number of ways of partitioning both the text and the words, based on the Bell number, is greater than 10^{120000}. To

Table 5.6: Properties of the (instances × variables) matrix for the Gina, Hiva and Nova datasets, in their initial and coclustering representation.

Dataset	Initial representation				Coclustering representation			
	Inst.	Var.	Size	Sparseness	Inst. cl.	Var. cl.	Size	Sparseness
Gina	35000	784	$2.74\ 10^7$	19.2%	480	125	$6.00\ 10^4$	79.1%
Hiva	42673	1617	$6.90\ 10^7$	9.1%	1230	210	$2.58\ 10^5$	52.2%
Nova	17537	19616	$3.44\ 10^8$	0.6%	207	1058	$2.19\ 10^5$	84.3%

obtain the best possible coclustering according to our MAP approach, we allocated several days of computation time to our anytime optimization heuristic.

In Table 5.6, we recall the properties of each dataset in its initial representation and present its pre-processed representation after the coclustering. The datasets are initially represented using very larges matrices, with up to hundreds of millions of cells. Their sparseness vary from less than 1% to about 20%. The number of non-zero elements (one variable activated for one instance) is about five million for Gina, six million for Hiva and two million for Nova. Once the coclustering is performed, we get dense representations with numbers of cells reduced by a factor of one hundred to one thousand.

5.7.4.2. Classification Results

In order to evaluate the quality of the representation, we train classifiers using the train and validation labeled instances to learn the distribution of the labels in each cluster of instances.

Table 5.7: Best challenge results vs. our coclustering method results for the Gina, Hiva and Nova datasets.

Dataset	Prior track		Agnostic track		Coclustering
	Winner	Best BER	Winner	Best BER	BER
Gina	Vladimir Nikulin	0.0226	Roman Lutz	0.0339	0.0516
Hiva	Chloé Azencott	0.2693	Vojtech Franc	0.2827	0.3127
Nova	Jorge Sueiras	0.0659	Mehreen Saeed	0.0456	0.0370

We recall in Table 5.7 the BER results of the challenge winner in the agnostic and prior track (see Guyon et al., 2007), and present our results obtained with the semi-supervised coclustering method (submission named "Data Grid(Coclustering)", dated 2007-02-27 for Gina and Hiva and 2007-09-19 for Nova). It is noteworthy that for datasets with large numbers of variables, the coclustering technique obtains far better performance than the supervised technique, with BER results about three times better on the Gina and Nova datasets. This comes from the ability of the coclustering to exploit all the variables, whereas each supervised data grid is restricted to a subset of about $\log_2 N$ variables.

Overall, the supervised coclustering method obtains good predictive performance, competitive with that of most of the challenge participants. On the Gina dataset, which is not very sparse, the BER is over twice as high as that of the leader. In the case of the Nova dataset, which is very sparse, the predictive performance significantly outperforms that of the winners and reaches the best reference result of the organizers, which is remarkable since our clusters were learned without using any class label.

5.7.4.3. UNDERSTANDABILITY

The assumption that the "natural" patterns identified by the coclustering are correlated with the classes looks true in the challenge datasets. Since we obtain many more patterns than classes, it is interesting to provide an interpretation of our coclusters.

The Gina dataset comes from the MNIST dataset (LeCun and Cortes, 1998). The task, which is handwritten digit recognition, consists of predicting the value of a digit from an image representation of $28 * 28$ pixels. The coclustering method identifies about one hundred clusters of pixels (regions) and five hundred clusters of images ("natural" shapes), each of them distributed similarly on the regions. Although the classification BER is only 0.0516 (about twice as high as that of the winner), it is interesting to notice that each digit (among the ten possible output digits) comes under a large variety of shapes. This is discovered without any domain knowledge and could be the foundation for adequate preprocessing.

In the case of the Hiva, further investigation with a domain specialist would be necessary to understand the meaning of the clusters of instances and variables.

The Nova dataset comes from the 20-Newsgroup dataset (Mitchell, 1999). The original task is to classify the texts into 20 classes (atheism, graphics, forsale, autos, motorcycles, baseball, hockey, crypt, electronics, med, space, religion.christian, politics.guns, politics.mideast, politics.misc, religion.misc). In the challenge, the classification task was a binary one, with two groups of classes (politics or religion vs. others). The coclustering method identifies about one thousand clusters of words (vocabulary themes) and two hundred clusters of texts ("natural" topics), each of them distributed similarly on the themes.

The distribution of the 17537 texts in the 207 clusters of texts (topics) is reasonably balanced. On the other hand, the repartition of the 19616 words in the 1058 clusters of words (themes) is not balanced at all. About 150 themes are singletons, like for example *the*, *and*, *for*, *that*, *this*, *have*, *you*. These are frequent words with low semantic, and even slightly different distribution of the topics on these singleton themes are significant and might be helpful for classification. For example, observing one of the singleton themes *say*, *why* or *who* approximately doubles the conditional probability of being in the challenge positive class (politics or religion).

A correlation study between the themes and the 20 original labels available on the train dataset reveals that the most informative themes are:

- *hockey, playoff, nhl, penguin, devils, pens, leafs, bruins, islande, goalie, mario, puck,...*
- *team, season, league, fans, teams, rangers, detroit, montrea, wins, scored, coach,...*
- *clipper, encrypt, nsa, escrow, pgp, crypto, wiretap, privacy, cryptog, denning,...*
- *dod, bike, motorcy, ride, riding, bikes, ama, rider, helmet, yamaha, harley, moto,...*
- *basebal, sox, jays, giants, mets, phillie, indians, cubs, yankees, stadium, cardina,...*
- *bible, scriptu, teachin, biblica, passage, theolog, prophet, spiritu, testame, revelat,...*
- *christi, beliefs, loving, rejecti, obedien, desires, buddhis, deity, strive, healed,...*
- *windows, dos, apps, exe, novell, ini, borland, ver, lan, desqvie, tsr, workgro, sdk,...*
- *pitcher, braves, pitch, pitchin, hitter, inning, pitched, pitches, innings, catcher,...*
- *car, cars, engine, auto, automob, mileage, autos, cactus, pickup, alarm, sunroof,...*

About one third of the theme are detected as informative with respect to the original labels. The partition of the words is very fine grained, so that many themes are hard to interpret, whereas other ones clearly capture semantics, such as:

- *book, books, learnin, deals, booksto, encyclo, titled, songs, helper*
- *cause, caused, causes, occur, occurs, causing, persist, excessi, occurin*
- *importa, extreme, careful, essenti, somewha, adequat*
- *morning, yesterd, sunday, friday, tuesday, saturda, monday, wednesd, thursda,...*
- *receive, sent, placed, returne, receivi, sends, resume*

Overall, our coclustering preprocessing method is able to produce a precise and reliable summary of the corpus of texts, which is demonstrated by the very good classification performance reported in Table 5.7.

5.8. Conclusion

The supervised data grid models introduced in this paper are based on a partitioning model of each input variable, into intervals for numerical variables and into groups of values for categorical variables. The cross-product of the univariate partitions, called a data grid, allows the quantification of the conditional information relative to the output variable. We have detailed this technique in the multivariate case, with a Bayesian approach for model selection and sophisticated combinatorial algorithms to efficiently search the model space.

We have also presented the principles of the extension of data grid models to unsupervised learning to evaluate the joint probability distribution of the variables. We have detailed the case of two categorical variables and applied it to the problem of coclustering the instances and variables of a sparse binary dataset.

In extensive artificial experiments, we have shown that our technique is able to reliably detect complex patterns. Our experiments quantify the limits of the approach, with data grid models limited to about $\log_2$ variables, and provides insights into the relation between the complexity of the patterns and the required computation time necessary to detect them.

We have introduced three ways of building classifiers from data grids and experimented them on the Agnostic Learning vs. Prior Knowledge challenge. This preliminary evaluation looks promising since our method was first on two of the datasets, one within the challenge deadline and the other one using a later submission. The analysis of the results demonstrates that the data grid models are of considerable interest for data understandability and data preparation.

Overall, the supervised data grids obtain good performance on datasets with small numbers of variables, while the coclustering data grids perform well on sparse binary datasets with very large numbers of variables. In future research, we plan to investigate how to better exploit the potential of these models to build more powerful classifiers. Apart from improving the optimization algorithms and building ensemble classifiers based on a wider diversity of data grid models, we intend to further explore the problem of conditional or joint density estimation.

Whereas the naïve Bayes strategy (Langley et al., 1992) factorizes the multivariate density estimation on univariate estimations, our strategy with the data grid models directly estimates the multivariate joint density, which encounters a limit in the number of variables considered. Between these two opposite strategies, other approaches have been considered, based on a relaxation of the naive Bayes assumption. This is the case for example in semi-naive Bayesian classifiers (Kononenko, 1991) or in Bayesian network classifiers (Friedman et al., 1997). In this context, we expect data grid models to be promising building blocks of future better multivariate density estimators.

References

M. Abramowitz and I. Stegun. *Handbook of mathematical functions*. Dover Publications Inc., New York, 1970.

C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1996. http://www.ics.uci.edu/mlearn/MLRepository.html.

H. Bock. Simultaneous clustering of objects and variables. In E. Diday, editor, *Analyse des Données et Informatique*, pages 187–203. INRIA, 1979.

M. Boullé. A Bayes optimal approach for partitioning the values of categorical attributes. *Journal of Machine Learning Research*, 6:1431–1452, 2005.

M. Boullé. MODL: a Bayes optimal discretization method for continuous attributes. *Machine Learning*, 65(1):131–165, 2006.

M. Boullé. Optimal bivariate evaluation for supervised learning using data grid models. *Advances in Data Analysis and Classification*, 2007a. submitted.

M. Boullé. Compression-based averaging of selective naive Bayes classifiers. *Journal of Machine Learning Research*, 8:1659–1685, 2007b.

M. Boullé. Bivariate data grid models for supervised learning. Technical Report NSM/R&D/TECH/EASY/TSI/4/MB, France Telecom R&D, 2008a. `http://perso.rd.francetelecom.fr/boulle/publications/BoulleNTTSI4MB08.pdf`.

M. Boullé. Multivariate data grid models for supervised and unsupervised learning. Technical Report NSM/R&D/TECH/EASY/TSI/5/MB, France Telecom R&D, 2008b. `http://perso.rd.francetelecom.fr/boulle/publications/BoulleNTTSI5MB08.pdf`.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. California: Wadsworth International, 1984.

O. Chapelle, B. Schölkopf, and A. Zien. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

W.G. Cochran. Some methods for strengthening the common chi-squared tests. *Biometrics*, 10 (4):417–451, 1954.

J. Connor-Linton. Chi square tutorial, 2003. `http://www.georgetown.edu/faculty/ballc/webtools/web_chi_tut.html`.

N. Friedman, D. Geiger, and M. Goldsmidt. Bayesian network classifiers. *Machine Learning*, 29:131–163, 1997.

P.D. Grünwald, I.J. Myung, and M.A. Pitt. *Advances in minimum description length : theory and applications*. MIT Press, 2005.

I. Guyon. Agnostic learning vs. prior knowledge challenge, 2007. `http://clopinet.com/isabelle/Projects/agnostic/`.

I. Guyon, A.R. Saffari, G. Dror, and G. Cawley. Agnostic learning vs. prior knowledge challenge. In *International Joint Conference on Neural Networks*, pages 829–834, 2007.

M.H. Hansen and B. Yu. Model selection and the principle of minimum description length. *J. American Statistical Association*, 96:746–774, 2001.

P. Hansen and N. Mladenovic. Variable neighborhood search: principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

J.A. Hartigan. Direct clustering of a data matrix. *Journal of the American Statistical Association*, 67(337):123–129, 1972.

J.A. Hoeting, D. Madigan, A.E. Raftery, and C.T. Volinsky. Bayesian model averaging: A tutorial. *Statistical Science*, 14(4):382–417, 1999.

G.V. Kass. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29(2):119–127, 1980.

I. Kononenko. Semi-naive Bayesian classifier. In Y. Kodrato, editor, *Sixth European Working Session on Learning (EWSL91)*, volume 482 of *LNAI*, pages 206–219. Springer, 1991.

P. Langley, W. Iba, and K. Thompson. An analysis of Bayesian classifiers. In *10th national conference on Artificial Intelligence*, pages 223–228. AAAI Press, 1992.

Y. LeCun and C. Cortes. The MNIST database of handwritten digits, 1998. http://yann.lecun.com/exdb/mnist/.

M. Li and P.M.B. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, Berlin, 1997.

T.M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

T.M. Mitchell. The 20 newsgroup dataset, 1999. http://kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html.

J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

C.E. Shannon. A mathematical theory of communication. Technical Report 27, Bell systems technical journal, 1948.

D.A. Zighed and R. Rakotomalala. *Graphes d'induction*. Hermes, France, 2000.

Chapter 6

Virtual High-Throughput Screening with Two-Dimensional Kernels

Chloé-Agathe Azencott CAZENCOT@ICS.UCI.EDU
Department of Computer Science
Institute for Genomics and Bioinformatics
University of California, Irvine
Irvine, CA 92697-3435, USA

Pierre Baldi PFBALDI@ICS.UCI.EDU
Department of Computer Science
Institute for Genomics and Bioinformatics
University of California, Irvine
Irvine, CA 92697-3435, USA

Abstract

High-Throughput Screening (HTS) is an important technology that relies on massively testing large numbers of compounds for their activity on a given assay in order to identify potential drug leads in the early stages of a drug discovery pipeline. However, because identification of drug leads by HTS is very costly, it is of great interest to develop computational methods for virtual HTS (VHTS), in order to prioritize the compounds to be screened and identify a relatively small, but highly promising, subset from a screening library that can be tested more economically. Here we develop statistical machine learning methods, based on two-dimensional spectral kernels for small molecules and extended-connectivity molecular substructures (ECFPs), to address this task. We apply them to the HIVA dataset of the Agnostic Learning versus Prior Knowledge Challenge and obtain the best results with a balanced error rate of 0.2693 and an area under the ROC curve of 0.7643 on the testing set.

Keywords: virtual high-throughput screening, drug discovery, drug screening, kernels, HTS, SVM

6.1. Introduction: The Virtual High-Throughput Screening Problem

High-Throughput Screening (HTS) is an approach to drug discovery developed in the 1980's in the pharmaceutical industry that allows to massively test large numbers (up to millions) of compounds for their activity on a given assay in order to identify potential drug leads. Nowadays, it is possible to screen up to 100,000 molecules per day in a single HTS facility. This process, however, requires a considerable amount of resources and capital investment, for instance in terms of robotics, molecular libraries, and the amount of relevant protein that must be produced. A widely circulated figure is that HTS screening costs on the order of one dollar per compound, a price that cannot be afforded by most academic laboratories.

The *in silico* approach to HTS, also called virtual HTS (VHTS), attempts to computationally select from a list of molecular compounds only those most likely to possess the properties required to positively satisfy a given assay. When the 3D structure of a target protein is known, the most common approach to VHTS is docking, which consists in scoring the compatibility of each small molecule in the screening library with respect to the known, or putative, binding pockets of the protein target. When the 3D structure of the targets is not known, or to further validate the results of a docking experiment, other computational methods must be used. In many cases, an initial list of positive and negative compounds may be known from previous, possibly small-scale, screening experiments. Therefore, in these cases, one is interested in using statistical machine learning or other methods to build a good molecular predictor and possibly clarify what are the desirable properties a molecule should have in order to positively satisfy the conditions of a given assay. The development of good VHTS methods is essential if one is to drastically reduce the number of compounds that must be experimentally assayed and reduce the time and cost of HTS.

Among the five datasets offered by the IJCNN-07 Agnostic Learning versus Prior Knowledge Challenge [1], we decided to focus on the HIVA set derived from the DTP AIDS Antiviral Screen program made available by the National Cancer Institute (NCI)[2]. This dataset contains assay results for 42,678 chemicals tested for their activity against the AIDS virus and provides a reasonable benchmark for the development of VHTS algorithms.

As in most chemoinformatics applications, such as the storage and search of large databases of small molecules (Chen et al., 2005; Swamidass and Baldi, 2007) or the prediction of their physical, chemical, and biological properties (Swamidass et al., 2005; Azencott et al., 2007), the issues of molecular data structures and representations play an essential role (Leach and Gillet, 2005). These representations and data structures are essential to define "molecular similarity", which in turn is crucial for developing efficient methods both to search the databases and predict molecular properties using kernel methods. Leveraging previous work in our group and in the literature, here we use SVMs in combination with 2D spectral representations of small molecules with Tanimoto and MinMax kernels to address the VHTS problem and tackle the HIVA challenge.

6.2. Molecular Data Representation

Small molecules are often described by graphs (King, 1983; Bonchev, 1991; McNaught and Wilinson, 1997), where vertices represent atoms and edges represent bonds. Other representations, such as one-dimensional SMILES strings (Weiniger et al., 1989) or three-dimensional descriptions based on the atomic coordinates, have been developed. Previous studies (Swamidass et al., 2005; Azencott et al., 2007) in our group as well as in other groups suggest, however, that these representations do not lead for now to better predictive performance. In this regard, it is worth noting for SMILES strings that the information they contain is identical to the information contained in the bond graphs. For 3D-based representations, the majority of the coordinates must be predicted, since only a relatively small fraction of molecular structures have been empirically solved. Furthermore, the 2D representation of molecules as graphs is also the representation of choice that underlies the structural similarity search algorithms of chemical databases such as ChemBank (Strausberg and Schreiber, 2003), ChemMine (Girke et al., 2005), or ChemDB (Chen et al., 2005, 2007).

6.2.1. Molecular Graphs

We describe a molecule as a labeled graph of bonds. Labels on the vertices represent the atom types and labels on the edges characterize the bonds. More precisely, vertices are labeled according to one of the following schemes:

1. `http://www.agnostic.inf.ethz.ch/index.php`
2. `http://dtp.nci.nih.gov/docs/aids/aids_data.html`

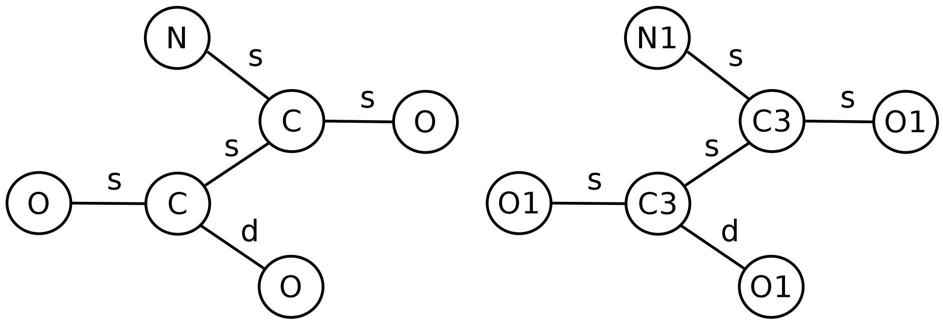


Figure 6.1: Example of molecular graphs. The vertices represent atoms, labeled with the Element scheme on the left and the Element-Connectivity scheme on the right. Bonds are represented by edges, labeled "s" for simple bonds and "d" for double bonds. Note that by convention Hydrogen atoms are ignored.

- E: Element. Each atom is simply labeled by its symbol (e.g. C for carbon, O for oxygen, N for nitrogen)
- EC: Element-Connectivity. Each atom is labeled by its symbol together with the number of atoms it is bonded to (e.g. C3 for a carbon with three atoms attached)

The bonds are simply labeled according to their type (e.g. single, double).

Figure 6.1 gives an example of the two-dimensional representation of a molecule as a graph.

From these graphs, a number of features can be extracted, such as the presence/absence or number of occurrences of particular functional groups. A more recent and general trend, however, has been to define features in terms of labeled subgraphs, such as labeled paths (Swamidass et al., 2005; Azencott et al., 2007) or labeled trees (Mahé et al., 2006), and to combinatorially extract and index all such features to represent molecules using large feature vectors, also known as fingerprints. While in other work we have compared the use of different features and have tried several of them on the HIVA challenge, here we focus on a class of shallow labeled trees, also known as extended-connectivity features in the literature (Hassan et al., 2006).

6.2.2. Extended-Connectivity Molecular Features

The concept of molecular connectivity (Razinger, 1982; Kier and Hall, 1986) leads to the idea of extended-connectivity substructures (Rogers and Brown, 2005; Hassan et al., 2006), which are labeled trees rooted at each vertex of the molecular graph. A depth parameter d controls the depth of the trees (Figure 6.2). For a given tree, this algorithm recursively labels each tree node (or atom) from the leaf nodes to the root, appending to each parent's label the labels of its children in the tree. Each resulting vertex label is then considered as a feature. For the labeling process to be unique, the vertices of the graph need to be ordered in a unique canonical way. This ordering is achieved using Morgan's algorithm (Morgan, 1965).

We extract extended-connectivity substructures of depth d up to 2, where the depth indicates the maximum distance, measured in number of bonds, to the root of each labeled tree. For example, a depth of two indicates that the label for a given atom will be composed of the labels for the neighboring atoms which are connected to it by at most two bonds. Other depths (3 to 6) have been tested but did not lead to any performance improvement.

Figure 6.2 shows an example of extended-connectivity labeling.

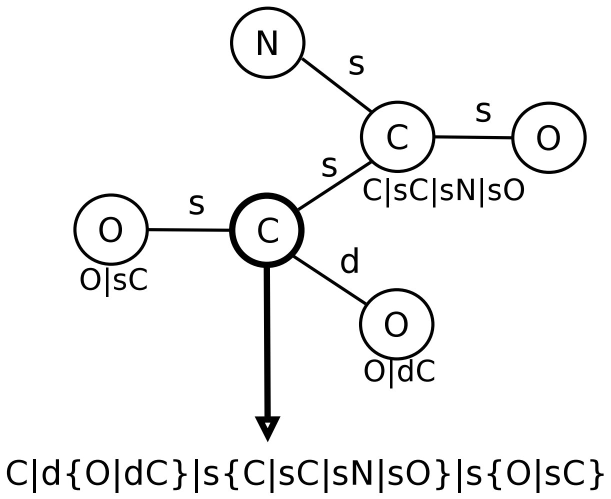


Figure 6.2: The extended-connectivity label of depth *d* up to 2 of the C atom circled in bold is given by the labels of depth up to 1 of its three neighboring atoms: (1) an O atom to which it is connected by a double bond, (2) a C atom to which it is connected by a single bond, and (3) an O atom to which it is connected by a single bond. If the EC scheme was to be used, the resulting label would be: `C3|d{O1|dC3}|s{C3|sC3|sN1|sO1}|s{O1|sC3}`.

6.2.3. Molecular Fingerprints

The molecular features are computed across the whole dataset. Each molecule can then be represented as a vector of fixed size N, where N is the total number of features found. For a given molecule, each component of the vector is set to 1 if the corresponding feature is present in the chemical, and 0 otherwise. We also use count vectors where each component of the vector is set to c, where c is the number of times the corresponding feature appears in the chemical. These feature vectors are actually extensions of traditional chemical fingerprints (Flower, 1998; Raymond and Willett, 2001).

The spectral or combinatorial approach to molecular fingerprints can easily be automated and has several advantages: (1) it alleviates the need for relying on expert knowledge, that may itself be incomplete, to select relevant molecular descriptors; (2) it produces a fixed-size representation for data of varying size.; and (3) it has been shown to be effective in the literature. Furthermore these long vectors, which have on the order of 100,000 components for the HIVA dataset, are also very sparse and can be efficiently compressed, in lossy or even lossless fashion (Baldi et al., 2007), to reduce their dimensionality and improve storage and computational complexity.

6.3. Support Vector Machines for virtual HTS

6.3.1. Kernels For Molecules

To define kernels for chemical compounds, we need to introduce a similarity measure between molecular fingerprints. Here we use the MinMax and Tanimoto similarity measures.

If $f_1 = (f_{1,1}, \ldots, f_{1,N})$ and $f_2 = (f_{2,1}, \ldots, f_{2,N})$ are two count fingerprints, the MinMax similarity measure (Swamidass et al., 2005; Ralaivola et al., 2005) is defined by

$$K(f_1, f_2) = \frac{\sum_i \min(f_{1,i}, f_{2,i})}{\sum_i \max(f_{1,i}, f_{2,i})} \tag{6.1}$$

In the case of binary fingerprints the MinMax measure reduces to the Tanimoto similarity measure defined by

$$K(f_1, f_2) = \frac{f_1 \cap f_2}{f_1 \cup f_2} \tag{6.2}$$

Both similarity measures have been shown (Swamidass et al., 2005) to be semi-definite positive and satisfy Mercer's kernel conditions. Thus the MinMax and Tanimoto kernels can be applied in combination with an SVM optimization framework to derive a molecular predictor in VHTS experiments.

6.3.2. Implementation

The HIVA dataset contains 42,678 examples. The associated pair-to-pair kernel matrix being rather large, an online implementation of SVM is desirable. Here we use the SVMTorch (Collobert and Bengio, 2001) implementation, which allows on-line learning and is thus well suited for our purpose.

Besides their size, one of the other issues with HTS datasets is that they are often highly unbalanced, usually containing far more negative than positive examples. This is the case of the HIVA dataset, which has about 28 times as many negative examples as positive examples. Without any further processing, this will negatively affect the predictor and bias it towards negative examples.

The most straightforward method to deal with class unbalance is to control the sensitivity (or C parameter) of the SVM (Veropoulos et al., 1999; Shin and Cho, 2003). By assigning a higher sensitivity to the under-represented class, one increases the coefficients of the corresponding support vectors, thus biasing the classifier towards the minority class. We first tested this method, which did not lead to significant improvements.

Another way of compensating for the small amount of positive examples is to re-sample the data, so as to train the SVM on a balanced set of examples. In this work we focus on oversampling, which consists in replicating the under-represented class so as to get a more balanced number of examples. This method has been widely studied in the literature (Estabrooks et al., 2004; Orriols and Bernad-Mansilla, 2005).

If m is the number of training examples and m_+ the number of positive training examples, we randomly split the negative data in $\frac{m}{m_+}$ subsets of about m_+ examples and build $\frac{m}{m_+}$ classifiers, each trained on a set composed of one of the negative subsets together with the m_+ positive examples. Each individual classifier produces a value of +1 if its prediction is positive and -1 if its prediction is negative. Then these values are added, and the final decision is made by comparing the resulting sum to a threshold. As this method overcompensates and leads to a bias favoring the positive class, the decision threshold has to be adjusted to a value greater than 0. To address this problem, we apply this method using 10-fold cross-validation over the training set and select the threshold that leads to the best performance on the training set. An SVM trained according to this algorithm will further be referred to as an oversampled SVM.

Eventually, we run a 10-fold cross-validation over the training set for each combination of labeling scheme, representation by bits or counts, and oversampling or not, and retain as best models the ones leading to optimal performance.

6.3.3. Performance Measures

The SVM classifiers associate a prediction value to each of the data points. We then order the values, thus ranking the data points, and set a threshold so as to separate predicted actives from predicted inactives. A number of performance measures can then be used in order to assess the performance and compare different methods.

The Agnostic Learning versus Prior Knowledge Challenge focused on the balanced error rate (BER) and area under the ROC curve (AUC) measures.

If $m_- = m - m_+$ is the number of negative examples, TP the number of true positives, TN the number of true negatives and FP the number of false positives, the BER is defined by

$$BER = 1 - \frac{1}{2}\left(\frac{TP}{m_+} + \frac{TN}{m_-}\right) \tag{6.3}$$

and the AUC is the area under the ROC curve defined by plotting the true positive rate $\frac{TP}{m_+}$ against the false positive rate $\frac{FP}{m_-}$ for each confidence value.

While these measures allow one to compare all the predictors to each other (especially in the Agnostic Learning track), they may not provide an optimal way of assessing VHTS methods. Indeed, these performance metrics do not address the "early recognition problem", in the sense that they do not quantify how efficient a given classifier is at retrieving active compounds *early*, i.e. at the top of the ranked list. High-enrichment for positives in the top of the list is highly desirable in VHTS, especially in conditions where only few compounds can be empirically tested.

An enrichment curve, representing the percentage of true positives captured as a function of the percentage of the ranked list screened, can be used to judge the ability of a predictor to recover active compounds early.

Whereas enrichment curves provide a graphical means for evaluating early recognition across many thresholds, capturing this property in a single numerical value is also desirable as a summary and to allow for easy comparison of several predictors. Truchon and Bayly (2007) develop this idea and propose a performance measure called Boltzmann-Enhanced Discrimination of Receiver Operating Characteristic (BEDROC) which partly addresses this issue.

The notion of BEDROC measure stems from the study of various virtual screening metrics, including the area under the enrichment curve (AUEC). If $TP(x)$ denotes the true positive rate, then the AUEC is defined by

$$AUEC = \int_0^1 TP(x)dx \tag{6.4}$$

The AUEC can be interpreted as the probability that an active compound will be ranked better than a compound selected at random by a uniform distribution. Therefore, in order to address the early recognition problem, Truchon and Bayly (2007) introduce the concept of weighted AUEC (wAUEC), defined by

$$wAUEC = \frac{\int_0^1 TP(x)w(x)dx}{\int_0^1 w(x)dx} \tag{6.5}$$

where $w(x)$ is a weighting probability distribution. The wAUEC is the probability that an active compound will be ranked better than a compound that would come from the probability distribution function w. By choosing for w an exponential distribution $w(x) = C(\alpha)e^{-\alpha x}$, which has higher values for low values of x, one gives a higher importance to the active compounds recognized at the top of the ranked list.

In the general case, the theoretical extreme values of the AUEC and the wAUEC measures depend on the number of actives and inactives of the problem being considered and differ from the usual 0 and 1 values associating for instance with the AUC measure. Note that the AUC is simply a scaled version of the AUEC, obtained through the following linear transformation:

$$AUC = \frac{AUEC - AUEC_{min}}{AUEC_{max} - AUEC_{min}} \tag{6.6}$$

Truchon and Bayly (2007) define the BEDROC by a similar scaling of the wAUEC:

$$BEDROC = \frac{wAUEC - wAUEC_{\mathrm{min}}}{wAUEC_{\mathrm{max}} - wAUEC_{\mathrm{min}}} \tag{6.7}$$

Therefore, the BEDROC measure can be seen as a generalization of the AUC metric that takes early recognition into account.

If $\alpha.\frac{m_+}{m} \ll 1$ and $\alpha \neq 0$, then the BEDROC measure is approximately equal to the wAUEC measure, and can be interpreted as the probability that an active compound will be ranked better than a compound selected at random from an exponential probability distribution function of parameter α.

Formally, if for every k in $[1,\ldots,m_+]$ we let r_k be the ranking of the k-th active compound, then the BEDROC metric can be estimated by

$$BEDROC \approx \frac{1}{\alpha\frac{m_+}{m}} \left(\frac{\sum_{k=1}^{m_+} e^{-\alpha.(r_k/N)}}{\frac{1-e^{-\alpha}}{e^{\alpha/m}-1}} \right) + \frac{1}{1+e^{-\alpha}} \tag{6.8}$$

In what follows, we use a typical value of $\alpha = 1$ for the early recognition parameter.

6.4. Results

The Agnostic Learning versus Prior Knowledge Challenge is run using a training set composed of 4,229 compounds randomly selected in the HIVA dataset, and a blind test set composed of the remaining 38,449 compounds. We optimize our models by 10-fold cross-validation on the training set and then evaluate their performance on the test set. The aim of the challenge is to reach the lowest possible BER on the testing set.

Table 6.1 reports the 10-fold cross-validation BER and AUC over the training set as well as the final performance of several of the tested methods. Combining molecular fingerprints with an Element labeling of atoms and a count-based fingerprint representation, together with an oversampled SVM framework, lead to the best entry among all competing groups for the HIVA dataset in the Prior Knowledge track, with a BER of 0.2693. The best 10-fold cross-validated BER on the training set, with a value of 0.1975, is achieved by the same method. We compare these results to those obtained by the winner of the Performance Prediction Challenge (Guyon et al., 2006), where the dataset was the same, but split in training and testing sets in a different fashion, and to the best results in the Agnostic Learning track[3], as well as to the second best results in the Prior Knowledge track. These second best results, with a BER of 0.2782,

3. available from http://clopinet.com/isabelle/Projects/agnostic/Results.html

have been obtained by S. Joshua Swamidass, also from our laboratory, by applying a neural-network-based approach to the same molecular fingerprints. This approach will be described elsewhere and has its own advantages, for instance in terms of speed. Both top entries in the Prior Knowledge track achieve better performance than the best entry in the Agnostic Learning track.

Table 6.1: 10-fold cross-validation BER and AUC over the HIVA training set, as well as final BER and AUC for several methods. (*) Winning entry. Best performance in bold and second best performance in italics. 'E' and 'EC' refer to the labeling schemes introduced in Section 6.2.1; 'binary' and 'counts' refer to the vector representations defined in Section 6.2.3; and 'oversampled' refers to an SVM trained on a balanced dataset obtained by replicating the underrepresented class as exposed in Section 6.3.2.

Method	Training set		Test set	
	BER	**AUC**	**BER**	**AUC**
E, binary (not oversampled)	0.2249	0.8293	0.2816	0.7550
E, binary (oversampled)	*0.1980*	0.8511	*0.2765*	0.7611
E, counts (not oversampled)	0.2238	0.8294	0.2799	0.7576
E, counts (oversampled) (*)	**0.1975**	0.8523	**0.2693**	0.7643
EC, binary (not oversampled)	0.2174	0.8338	0.2828	0.7673
EC, binary (oversampled)	0.2030	0.8413	0.2860	0.7595
EC, counts (not oversampled)	0.2189	0.8358	0.2826	0.7626
EC, counts (oversampled)	0.1993	0.8450	0.2820	0.7650
Second Best (Prior Knowledge)	0.2168	0.8198	0.2782	0.7072
Best (Agnostic Learning)	-	-	0.2827	0.7707
Performance Prediction Challenge	-	-	0.2757	0.7671

The 10-fold cross-validated enrichment curves over the training set for several methods are displayed on Figure 6.3. Close to the origin, the highest enrichment on these curves is clearly observed when using a non-oversampled SVM. This region is further magnified in Figure 6.4 which focuses on the first 10% of the ranked list. It suggests that a slightly better ability at early recognition is attained with the model derived from binary fingerprints using the element labeling scheme.

The actual enrichment curves obtained on the testing set are displayed on Figure 6.5. Here again, the best early recognition ability is clearly observed for non-oversampled SVM. Figure 6.6, which focuses on the first 10% of these enrichment curves, suggests that the model derived from count fingerprints obtained with the element labeling scheme has the best ability to recover actives at the top of the ranked list.

Table 6.2 presents the 10-fold cross-validation BEDROC over the HIVA training set as well as the final BEDROC of several methods. The best final BEDROC of 0.507 is also obtained with molecular fingerprints combined with an Element labeling of atoms and a count-based fingerprint representation, but together with an non-oversampled SVM framework. This method, which corresponds to the enrichment curve with the steepest slope before 5%, achieves a 10-fold cross-validated BEDROC of 0.609 on the training set, just behind the best value of 0.610 obtained when using a binary fingerprint representation instead of the count-based one.

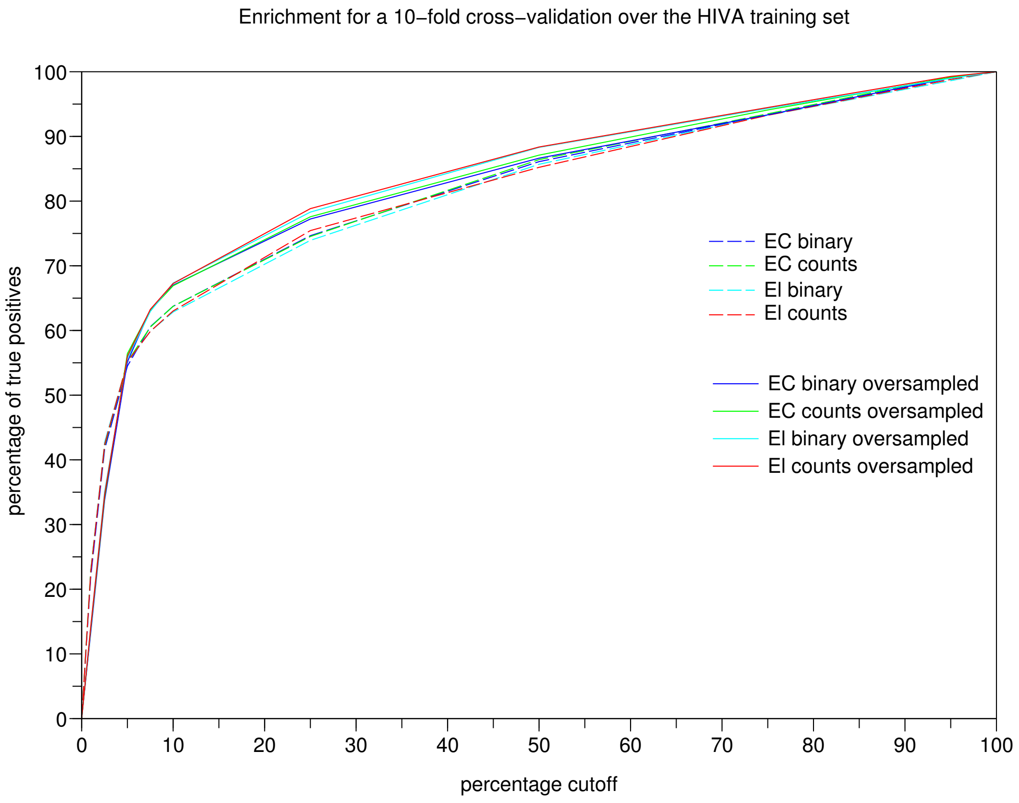


Figure 6.3: Ten-fold cross-validation enrichment curves over the HIVA training set for several methods. ‘E’ and ‘EC’ refer to the labeling schemes introduced in Section 6.2.1; ‘binary’ and ‘counts’ refer to the vector representations defined in Section 6.2.3; and ‘oversampled’ refers to an SVM trained on a balanced dataset obtained by replicating the underrepresented class as exposed in Section 6.3.2.

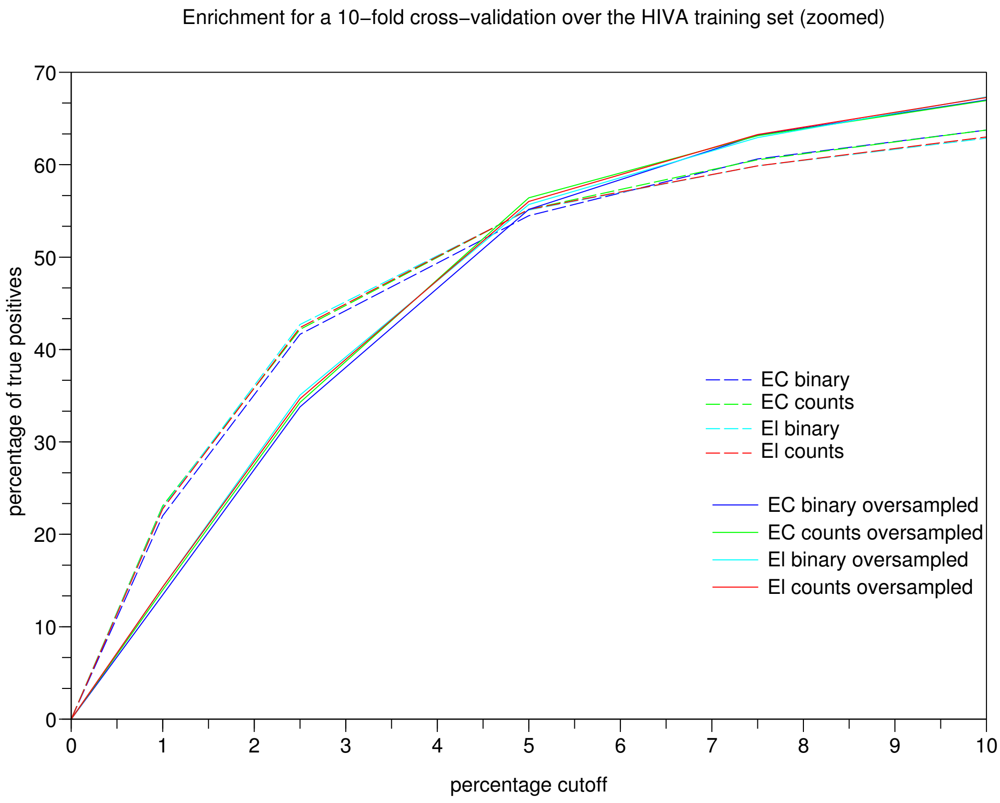


Figure 6.4: Ten-fold cross-validation enrichment curves, limited to the first 10% of the ranked list, over the HIVA training set for several methods. ‘E’ and ‘EC’ refer to the labeling schemes introduced in Section 6.2.1; ‘binary’ and ‘counts’ refer to the vector representations defined in Section 6.2.3; and ‘oversampled’ refers to an SVM trained on a balanced dataset obtained by replicating the underrepresented class as exposed in Section 6.3.2.

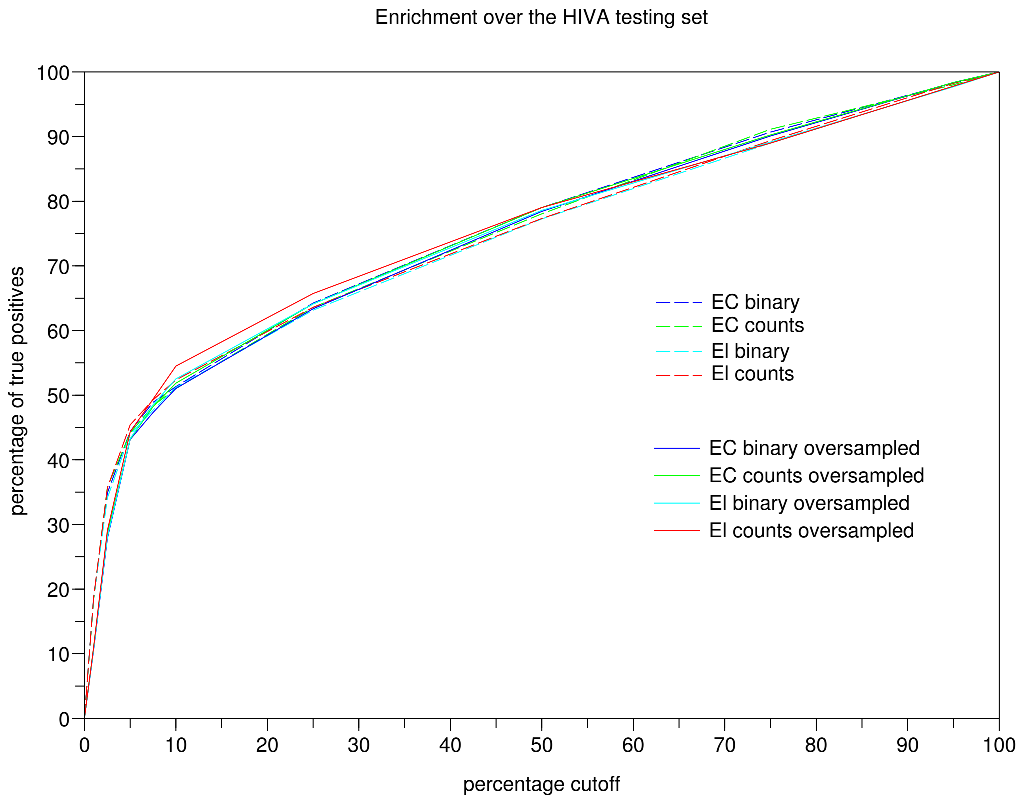


Figure 6.5: Actual enrichment curves over the HIVA testing set for several methods. 'E' and 'EC' refer to the labeling schemes introduced in Section 6.2.1; 'binary' and 'counts' refer to the vector representations defined in Section 6.2.3; and 'oversampled' refers to an SVM trained on a balanced dataset obtained by replicating the underrepresented class as exposed in Section 6.3.2.

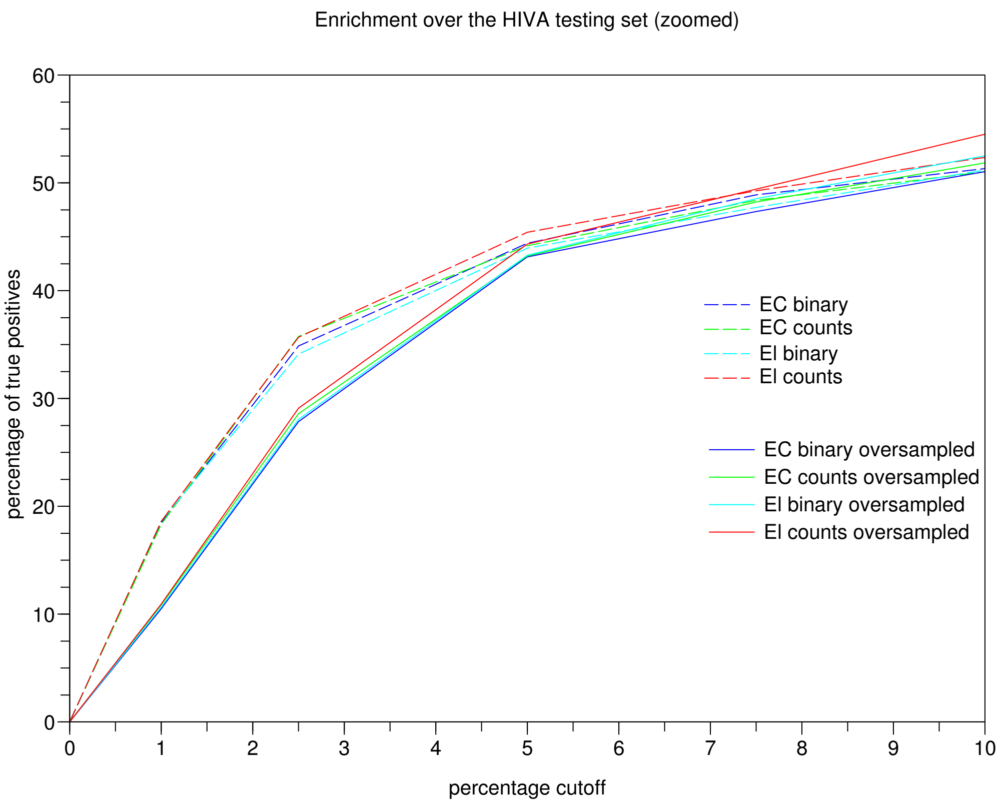


Figure 6.6: Actual enrichment curves, limited to the first 10% of the ranked list, over the HIVA testing set for several methods. 'E' and 'EC' refer to the labeling schemes introduced in Section 6.2.1; 'binary' and 'counts' refer to the vector representations defined in Section 6.2.3; and 'oversampled' refers to an SVM trained on a balanced dataset obtained by replicating the underrepresented class as exposed in Section 6.3.2.

Table 6.2: 10-fold cross-validation BEDROC over the training set as well as final BEDROC for several methods. (*) Winning entry. Best performance in bold and second best performance in italics. 'E' and 'EC' refer to the labeling schemes introduced in Section 6.2.1; 'binary' and 'counts' refer to the vector representations defined in Section 6.2.3; and 'oversampled' refers to an SVM trained on a balanced dataset obtained by replicating the underrepresented class as exposed in Section 6.3.2.

Method	Training set **BEDROC**	Test set **BEDROC**
E, binary (not oversampled)	**0.610**	*0.495*
E, binary (oversampled)	0.580	0.454
E, counts (not oversampled)	*0.609*	**0.507**
E, counts (oversampled) (*)	0.581	0.465
EC, binary (not oversampled)	0.606	0.499
EC, binary (oversampled)	0.573	0.446
EC, counts (not oversampled)	0.602	0.500
EC, counts (oversampled)	0.573	0.454
Second Best (Prior Knowledge)	0.607	0.483

6.5. Discussion

By defining feature vectors that capture molecular structural information, we have developed a kernel leading to the best results on the HIVA dataset in the Agnostic Learning versus Prior Knowledge Challenge.

The extended-connectivity molecular fingerprints present the advantage of being built automatically, without the need for human curation and expert knowledge. The results obtained with these representations are superior to those obtained using the set of binary molecular descriptors computed using the ChemTK package[4] which were offered in the Agnostic Learning track. Also, one of the challenge participants tried to collaborate with chemists to define meaningful features, but did not manage to get better results than using the Agnostic Learning features.

Overall, the results suggest that extended-connectivity fingerprints yield efficient molecular representations that can be successfully applied to a variety of chemoinformatics problems, from the prediction of molecular properties to the clustering of large libraries of compounds. These fingerprints are actually implemented in the current version of the ChemDB database (Chen et al., 2007) and routinely used to search compounds.

We also notice that the model selection method adopted here, although somewhat naïve being based only on the cross-validation performance over the training set, still allows us to efficiently choose the top classifiers and rank first in the competition. This is especially interesting because the test set is about nine times larger than the training set, raising concern of over-fitting. It may be of some interest to combine our features with the best methods of the Agnostic Learning track to see whether any further improvements can be derived.

Other extensions of this work include applying our best methods to other virtual HTS datasets. An important observation in this context is that the methods yielding best BER performance do not yield best BEDROC performance. This is because optimizing for early recognition is not equivalent to optimizing for overall classification. The enrichment curves, which are

4. http://www.sageinformatics.com

systematically steeper for low thresholds when using non-oversampled SVM, corroborate this observation. More precisely, it appears that oversampling improves the global performance of the classifier in terms of BER but not the early recognition in terms of BEDROC. This suggests that putting more emphasis on the positive training examples reduces the bias of the SVM, but also leads to assigning higher prediction values to some of the negative points. It is therefore critical to carefully choose which performance measure to optimize with regards to the specific problem being tackled and the resources available to conduct laboratory experiments to confirm the computational prediction.

Acknowledgments

Work supported by NIH Biomedical Informatics Training grant (LM-07443-01), NSF MRI grant (EIA-0321390), NSF grant 0513376, and a Microsoft Faculty Innovation Award to PB. We would like also to acknowledge the OpenBabel project and OpenEye Scientific Software for their free software academic licenses.

References

C.-A. Azencott, A. Ksikes, S. J. Swamidass, J. H. Chen, L. Ralaivola, and P. Baldi. One- to Four-Dimensional Kernels for Virtual Screening and the Prediction of Physical, Chemical, and Biological Properties. *J. Chem. Inf. Model*, 47(3):965–974, 2007.

P. Baldi, R. W. Benz, D. S. Hirshberg, and S. J. Swamidass. Lossless Compression of Chemical Fingerprints Using Integer Entropy Codes Improves Storage and Retrieval. *J. Chem. Inf. Model.*, 2007.

Danail Bonchev. *Chemical Graph Theory: Introduction and Fundamentals*. Taylor & Francis, 1991. ISBN 0856264547.

J. Chen, S. J. Swamidass, Y. Dou, J. Bruand, and P. Baldi. ChemDB: A Public Database Of Small Molecules And Related Chemoinformatics Resources. *Bioinformatics*, 21:4133–4139, 2005.

Jonathan H. Chen, Erik Linstead, S. Joshua Swamidass, Dennis Wang, and Pierre Baldi. ChemDB Update - Full-Text Search and Virtual Chemical Space. *Bioinformatics*, 2007. doi: 10.1093/bioinformatics/btm341. URL http://bioinformatics.oxfordjournals.org/cgi/content/abstract/btm341v1.

R. Collobert and S. Bengio. SVMTorch: Support Vector Machines for Large-Scale Regression Problems. *J. Mach. Learn. Res.*, 1:143–160, Sep. 2001 2001. http://www.idiap.ch/learning/SVMTorch.html.

A. Estabrooks, T. Jo, and N. Japkowicz. A Multiple Resampling Method for Learning From Imbalanced Data Set. *Computational Intelligence*, 20(1), 2004.

D. R. Flower. On the Properties of Bit String-Based Measures of Chemical Similarity. *J. Chem. Inf. Comput. Sci.*, 38:378–386, 1998.

T. Girke, L.-C. Chen, and N. Raikhel. ChemMine. A Compound Mining Database For Chemical Genomics. *Plant Physiol.*, 138:573–577, 2005. URL http://bioinfo.ucr.edu/projects/PlantChemBase/search.php.

I. Guyon, A. Saffari, G. Dror, and J. M. Buhman. Performance Prediction Challenge. In *IEEE/INNS conference IJCNN 2006, Vancouver July 16-21*, 2006.

M. Hassan, R. D. Brown, S. Varma-O'Brien, and D. Rogers. Cheminformatics Analysis and Learning in a Data Pipelining Environment. *Molecular Diversity*, 10:283–299, 2006.

Lemont B Kier and Lower H Hall. *Molecular connectivity in structure-activity analysis*. Wiley, New York, 1986. ISBN 0-471-90983-1.

R.B. King. *Chemical Applications of Topology and Graph Theory*. Elsevier, October 1983. ISBN 0444422447.

A. R. Leach and V. J. Gillet. *An Introduction to Chemoinformatics*. Springer, 2005.

P. Mahé, L. Ralaivola, V. Stoven, and J.-P. Vert. The Pharmacophore Kernel for Virtual Screening with Support Vector Machines. *J. Chem. Inf. Model.*, 46:2003–2014, 2006.

Alan D. McNaught and Andrew Wilinson. Molecular Graph, 1997. URL http://www.iupac.org/publications/compendium/index.html.

H.L. Morgan. The Generation of Unique Machine Description for Chemical Structures - A Technique Developed at Chemical Abstracts Service. *Journal of Chemical Documentation*, 5:107–113, 1965.

A. Orriols and E. Bernad-Mansilla. The Class Imbalance Problem in Learning Classifier Systems: A Preliminary Study. In *Proceedings of the 2005 Workshops on Genetic and Evolutionary Computation (Washington, D.C., June 25 - 26, 2005)*, pages 74–78, New York, NY, 2005. ACM Press.

L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi. Graph Kernels for Chemical Informatics. *Neural Netw.*, 18(8):1093–1110, 2005.

J.W. Raymond and P. Willett. Effectiveness of Graph-Based and Fingerprint-Based Similarity Measures for Virtual Screening of 2D Chemical Structure Databases. *J. Comput.-Aided Mol. Des.*, 16:59–71, 2001.

Razinger. Extended Connectivity in Chemical Graphs. *Theoretical Chemistry Accounts: Theory, Computation, and Modeling (Theoretical Chimica Acta)*, 61:581–586, 1982. doi: 10.1007/BF02394734. URL http://dx.doi.org/10.1007/BF02394734.

David Rogers and Robert D. Brown. Using Extended-Connectivity Fingerprints with Laplacian-Modified Bayesian Analysis in High-Throughput Screening Follow-Up. *Journal of Biomolecular Screening*, 10:682–686, October 2005. doi: 10.1177/1087057105281365. URL http://jbx.sagepub.com/cgi/content/abstract/10/7/682.

H. Shin and S. Cho. How to Deal With Large Datasets, Class Imbalance and Binary Output in SVM Based Response Model. In *Proceedings of the Korean Data Mining Conference*, pages 93–107, 2003. Best Paper Award.

R.L. Strausberg and S.L. Schreiber. From Knowing To Controlling: A Path From Genomics To Drugs Using Small Molecule Probes. *Science*, 300:294–295, 2003. URL http://chembank.broad.harvard.edu/.

S. J. Swamidass, J. H. Chen, J. Bruand, P. Phung, L. Ralaivola, and P. Baldi. Kernels for Small Molecules and the Predicition of Mutagenicity, Toxicity, and Anti-Cancer Activity. *Bioinformatics*, 21(Supplement 1):i359–368, 2005. Proceedings of the 2005 ISMB Conference.

S.J. Swamidass and P. Baldi. Bounds and Algorithms for Exact Searches of Chemical Fingerprints in Linear and Sub-Linear Time. *Journal of Chemical Information and Modeling*, 47 (2):302–317, 2007.

J.-F. Truchon and C. I. Bayly. Evaluating Virtual Screening Methods: Good and Bad Metrics for the "Early Recognition" Problem. *J. Chem. Inf. Model.*, 47(2):488 –508, 2007.

K. Veropoulos, C. Campbell, and N. Cristianini. Controlling the Sensitivity of Support Vector Machines. In *Proceedings of the International Joint Conference on AI*, pages 55–60, 1999.

D. Weiniger, A. Weiniger, and J.L. Weiniger. SMILES. 2. Algorithm for Generation of Uniques SMILES Notation. *J. Chem. Inf. Comput. Sci.*, 29:97–101, 1989.

Part III

Robust Parameter Estimation

Overview

Preventing overfitting or monitoring the **fit versus robustness tradeoff** has been the name of the game in machine learning and statistics for the past few decades. Several robust parameter estimation methods in the generalized linear model and kernel method families have emerged, stemming from statistical learning theory. Such methods optimize a two-part cost function. The first part is the "training error", that is the average loss over all traning examples. Loss functions include the hinge loss of Support Vector Machines (SVMs) for pattern recognition and the square loss for regression (also sometimes used for pattern recognition). The second part of the cost function is a term penalizing complex solutions, such as the norm of the weight vector. More generally, the penalty may be a norm of function in a Hilbert space containing the family of models considered.

In this part, three chapters illustrate methods derived from such approaches. In Chapter 7, **Mathias M. Adankon and Mohamed Cheriet reformulate the classical SVM optimization problem** to incorporate the box constraint as an extra kernel parameter, which facilitates performing hyper-parameter optimization with gradient descent and, in some intances, reduced the number of hyper-paramenters to be optimized. In Chapter 8, **Erinija Pranckeviciene and Ray Somorjai explore the possibilities offered by a 1-norm regularizer,** as opposed to the classical 2-norm regularizer generally used for SVMs. Such approaches provide an embedded method of feature selection, since the contraints thus imposed on the weight vector drive some weights to exactly zero. All these methods are not exempt of hyperparameter selection. Bounds on the generalization error are often used to carry out hyperparameter selection in SVMs and related kernel methods. Chapter 7 uses the radius-margin bound, while Chapter 8 uses transvariation intensity (another measure of average margin error). In Chapter 9, **Michiel Debruyne, Mia Hubert, and Johan A.K. Suykens propose a closed-form approximation of the leave-one-out-error based on the influence function.** See also Chapter 13, Part V, which describes a method for regularizing the leave-one-out error estimated by the PRESS statistic for LSSVM classifiers. This last method won overall second place in the performance prediction challenge and yielded best reference performance in the ALvsPK challenge (agnostic track).

Chapter 7

Unified Framework for SVM Model Selection

Mathias M. Adankon MATHIAS@LIVIA.ETSMTL.CA

Mohamed Cheriet MOHAMED.CHERIET@ETSMTL.CA

Synchromedia Laboratory for Multimedia Communication in Telepresence
École de Technologie Supérieure, University of Quebec
1100 Notre-Dame West, Montreal, Quebec, Canada, H3C 1K3

Editor: Isabelle Guyon, Gavin Cawley, Gideon Dror and Amir Saffari.

Abstract

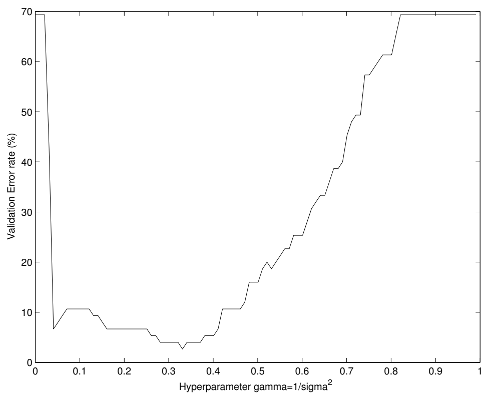
Model selection for support vector machines (SVMs) involves tuning SVM hyperparameters, such as C, which controls the amount of overlap, and the kernel parameters. Several criteria developed for doing so do not take C into account. In this paper, we propose a unified framework for SVM model selection which makes it possible to include C in the definition of the kernel parameters. This makes tuning hyperparameters for SVMs equivalent to choosing the best kernel parameter values. We tested this approach using empirical error and radius margin criteria. Our experiments on the Challenge Benchmarks dataset show promising results which confirm the usefulness of our method.

Keywords: Model Selection, SVM, Support vector machine, hyperparameter, kernel.

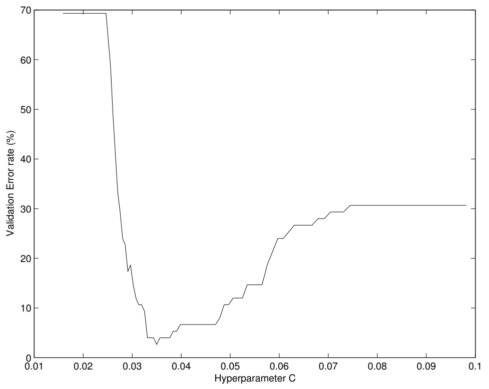
7.1. Introduction

Support vector machines (SVMs) are particular classifiers which are based on the margin maximization principle (Vapnik, 1998). They perform structural risk minimization, which was introduced to machine learning by Vapnik (Vapnik, 1982, 1992) and which has yielded excellent generalization performance. However, the generalization capacity of the SVM depends on hyperparameters such as C and the kernel parameters. The hyperparameter C is a regularization parameter which controls the trade-off between training error minimization and margin maximization. As an illustration, Figure 7.1 shows the variation of the error rate on a validation set versus the variation of the Gaussian kernel with a fixed value of C and Figure 7.2 shows the variation of the error rate on the validation set versus the variation of the hyperparameter C with a fixed value of the RBF kernel parameter. In each case, we resolve the binary problem described by the "Thyroid" data taken from the UCI benchmark. Clearly, the best performance is achieved with an optimum choice of the kernel para-meter and of C.

Several methods (Wahba et al., 1999; Vapnik, 1998; Jaakkola and Haussler, 1999; Joachims, 2000; Opper and Winther, 1999, 2000; Chapelle and Vapnik, 1999; Vapnik and Chapelle, 2000; Gold and Sollich, 2005; Adankon and Cheriet, 2007) have been developed for choosing the best hyperparameter values. In 2001, Chapelle et al. (Chapelle et al., 2001) proposed for the first time an automatic method for selecting hyperparameters for SVMs using certain criteria which approximate the error of the leave-one-out (LOO) procedure. These criteria are called Span bound and radius-margin bound. Duan et al. have shown in (Duan et al., 2003) that radius-margin bound gives good prediction for L2-SVM, but its practical viability is still not very



(*a*) Validation error rate for different values of the variance of the RBF kernel for binary problem



(*b*) Validation error rate for different values of the hyperparameter C for binary problem

Figure 7.1: Impact of SVM hyperparameters on the classifier generalization

satisfactory for L1-SVM. Then, in 2003, Kai-Min et al. (Chung et al., 2003) proposed modified radius-margin bound for L1-SVM.

Recently, Ayat et al. (Ayat et al., 2005) have proposed a new criterion based on the empirical error, where an empirical estimate of the generalization error is minimized through a validation set. This criterion is a linear function which does not require the resolution of another quadratic problem except for SVM training.

However, certain criteria, like empirical error, cannot be applied to tuning the hyperparameter C because the approximation used to compute the gradient is not tractable. In this paper, we propose a new formulation for the L1-SVM. With this formulation, the hyperparameter C is considered as a parameter of the kernel function. Hence, when a given criterion used to optimize the hyperparameters is not tractable with C, we can use this new formulation to improve model selection. Furthermore, the unified framework makes it possible to reduce the number of hyperparameters in certain cases.

This chapter is organized as follows. In Section 7.2, we describe the new formulation for the dual SVM problem, the baseline of the unified framework. In Section 7.3, we describe the various properties of this new formulation, and, in Section 7.4, the advantage of the unified framework for model selection. In Section 7.5, we provide an application example of the unified framework for model selection using the empirical error and radius-margin criteria. In Section 7.6, we present the experimental results and, in the last section, we conclude the paper.

7.2. New Formulation

We first consider a binary classification problem. Let us consider a dataset $\{(x_1,y_1),\dots,(x_\ell,y_\ell)\}$ with $x_i \in \mathscr{R}^d$ and $y_i \in \{-1,1\}$. Nonlinear SVM classifiers use the kernel trick to produce nonlinear boundaries. The idea behind kernels is to map training data nonlinearly into a higher-dimensional feature space via a mapping function Φ and to construct a separating hyperplane which maximizes the margin. The construction of the linear decision surface in this feature space only requires the evaluation of dot products $\phi(x_i)\cdot\phi(x_j) = k(x_i,x_j)$, where the application

$k : \mathscr{R}^d \times \mathscr{R}^d \to \mathscr{R}$ is called the kernel function (Boser et al., 1992; Scholkopf and Smola, 2002; Cristianini and Shawe-Taylor, 2000; Shawe-Taylor and Cristianini, 2004).

The decision function given by an SVM is :

$$y(x) = \text{sign}[w'\phi(x) + b], \tag{7.1}$$

where w and b are found by resolving the following optimization problem which expresses the maximization of the margin $1/\|w\|$ and the minimization of the training error :

$$\min_{w,b,\xi} \frac{1}{2} w'w + C\sum_{i=1}^{\ell} \xi_i \tag{7.2}$$

$$\text{subject to } : y_i[w'\phi(x_i) + b] \geq 1 - \xi_i \quad \forall i = 1, ..., \ell \tag{7.3}$$

$$\xi_i \geq 0 \quad \forall i = 1, ..., \ell. \tag{7.4}$$

The Lagrangian of the previous problem[1] is :

$$\mathscr{L} = \frac{1}{2} w'w + C\sum_{i=1}^{\ell} \xi_i - \sum_{i=1}^{\ell} \alpha_i [y_i(w'\phi(x_i) + b) - 1 + \xi_i] - \sum_{i=1}^{\ell} \lambda_i \xi_i \tag{7.5}$$

with the Lagrange multipliers $\alpha_i \geq 0$ and $\lambda_i \geq 0$ for all $i = 1, ..., \ell$.

When, we apply the differentiation theorem w.r.t. the Lagrangian, we obtain :

$$y(x) = \text{sign}[\sum_{i=1}^{\ell} \alpha_i y_i k(x_i, x) + b], \tag{7.6}$$

with α solution of :

$$\text{maximize} : W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \tag{7.7}$$

$$\text{subject to: } \sum_{i=1}^{\ell} \alpha_i y_i = 0 \quad \textit{and} \quad 0 \leq \alpha_i \leq C, i = 1, ..., \ell.$$

In the feature space, the optimal separating hyperplane for the SVM is defined by :

$$f(z) = \sum_{j=1}^{NVS} \alpha_j y_j k(x_j, z) + b. \tag{7.8}$$

In Equation (7.8) $j = 1, \ldots, NVS$ are the Support Vector indices corresponding to non-zero α_j.

The new formulation we propose in this paper is defined by using the following change of variables used in Chung et al. (2003) :

$$\alpha_i = C\tilde{\alpha}_i, \quad \forall i = 1, ..., \ell. \tag{7.9}$$

The QP problem expressed by (7.7) becomes:

$$\max_{\tilde{\alpha}} W(\tilde{\alpha}) = \sum_{i=1}^{\ell} C\tilde{\alpha}_i - \frac{1}{2} \sum_{i,j=1}^{\ell} C^2 \tilde{\alpha}_i \tilde{\alpha}_j y_i y_j k(x_i, x_j) \tag{7.10}$$

$$\text{subject to :} \quad \sum_{i=1}^{l} C\tilde{\alpha}_i y_i = 0 \tag{7.11}$$

$$0 \leq C\tilde{\alpha}_i \leq C, \quad i = 1, ..., \ell \tag{7.12}$$

1. Problem (7.2) expresses the L1-SVM formulation.
We also have the L2-SVM defined by $\min_{w,b,\xi}(\frac{1}{2}w'w + C\sum_{i=1}^{\ell} \xi_i^2)$

In Equations (7.11) and (7.12), we can simplify the positive real C, and so we have :

$$\sum_{i=1}^{l} \tilde{\alpha}_i y_i = 0, \tag{7.13}$$

$$0 \leq \tilde{\alpha}_i \leq 1, \quad i = 1, ..., \ell. \tag{7.14}$$

Then, the constraints become independent of the hyperparameter C.

Let us consider equation (7.10) which defines the objective function of the quadratic problem for SVMs :

$$\begin{aligned} W(\tilde{\alpha}) &= \sum_{i=1}^{\ell} C\tilde{\alpha}_i - \frac{1}{2} \sum_{i,j=1}^{\ell} C^2 \tilde{\alpha}_i \tilde{\alpha}_j y_i y_j k(x_i, x_j) \\ &= C \Big[\sum_{i=1}^{\ell} \tilde{\alpha}_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \tilde{\alpha}_i \tilde{\alpha}_j y_i y_j C k(x_i, x_j) \Big]. \end{aligned}$$

Since the real C is strictly positive, we can conclude that maximizing W w.r.t. $\alpha = (\alpha_1, ..., \alpha_\ell)'$ is equivalent to maximizing W/C w.r.t. $\tilde{\alpha} = (\tilde{\alpha}_1, ..., \tilde{\alpha}_\ell)'$.

To complete the new formulation, we use :

$$\tilde{k}(x_i, x_j) = C k(x_i, x_j) \tag{7.15}$$

and we can write :

$$W(\tilde{\alpha})/C = \sum_{i=1}^{\ell} \tilde{\alpha}_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \tilde{\alpha}_i \tilde{\alpha}_j y_i y_j \tilde{k}(x_i, x_j) \tag{7.16}$$

We reformulate now the quadratic problem expressed in Equation (7.7) as follows:

$$\text{maximize : } W_m(\tilde{\alpha}) = \sum_{i=1}^{\ell} \tilde{\alpha}_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \tilde{\alpha}_i \tilde{\alpha}_j y_i y_j \tilde{k}(x_i, x_j) \tag{7.17}$$

$$\text{subject to: } \sum_{i=1}^{l} \tilde{\alpha}_i y_i = 0 \quad \textit{and} \quad 0 \leq \tilde{\alpha}_i \leq 1, i = 1, ..., \ell$$

The preceding equation defines the new formulation where the constraints on the parameters $\tilde{\alpha}$ are independent of hyperparameter C which is included in the new kernel function $\tilde{k}$.

When we consider Equation (7.17), the Karush-Kuhn-Tucker conditions give:

- $0 < \tilde{\alpha}_i < 1$ if the point is on the margin
- $\tilde{\alpha}_i = 1$ if the point is misclassified w.r.t. the margin
- $\tilde{\alpha}_i = 0$ if the point is correctly classified outside the margin

7.3. Properties of the New Formulation

7.3.1. Hyperplane equation using the New Formulation

The separating hyperplane for the SVM that is used to define the decision boundary in a classification problem is expressed by (7.8). However, when we use the new formulation with the changes of variables proposed in Section 7.2, we have :

$$\begin{aligned} f(z) &= \sum_{j=1}^{\ell} y_j \alpha_j k(x_j, z) + b \\ &= \sum_{j=1}^{\ell} y_j \tilde{\alpha}_j C k(x_j, z) + b \end{aligned}$$

and we obtain the following equation :

$$f(z) = \sum_{j=1}^{\ell} y_j \tilde{\alpha}_j \tilde{k}(x_j, z) + b. \tag{7.18}$$

Consequently, when we resolve the problem of (7.17) which gives $\tilde{\alpha}$, we do not need to compute the parameters α for evaluating the equation of the separating hyperplane.

7.3.2. Properties of $\tilde{k}$

Since the hyperparameter C is strictly positive, then the function $\tilde{k} : (x_i, x_j) \mapsto Ck(x_i, x_j)$ is a Mercer kernel when the kernel k satisfies the Mercer condition. We have :

$$\tilde{k}(x_i, x_j) = Ck(x_i, x_j) = C\phi(x_i).\phi(x_j). \tag{7.19}$$

Hence, the nonlinear mapping $\tilde{\phi}$ resulting from the new formulation is given by the following expression:

$$\tilde{\phi} : x \mapsto \sqrt{C}\phi(x). \tag{7.20}$$

The mapping $\tilde{\phi}$ can be expressed as a composition of two transformations as follows:

$$\tilde{\phi}(x) = h[\phi(x)] = (h \circ \phi)(x), \tag{7.21}$$

where h is the homothety transformation with ratio $\sqrt{C}$.

We know that a homothety is the particular similarity transformation. As such, it preserves angles and ratios of lengths. It also preserves orientation. Consequently, the kernel function $\tilde{k}$ obtained from $\tilde{\phi}$ maintains the various properties of similarity of k.

7.3.3. Kernels definition using the New Formulation

When we use the new formulation, the new kernel function $\tilde{k}$ obtained from the Gaussian kernel is expressed with the hyperparameter C as the second parameter of the kernel function.

$$\tilde{k}(x, y) = C\exp(-\frac{\|x-y\|^2}{\sigma^2}) = C\exp(-a\|x-y\|^2), \tag{7.22}$$

where a is positive real.

We can define the following kernel to replace the Gaussian kernel, when we use the unified framework :

$$\tilde{k}(x,y) = \exp(-a\|x-y\|^2 + b) \tag{7.23}$$

In this new kernel, we have two parameters, the first a replace the inverse of the width while the second kernel parameter b is equal to $\ln(C)$. This latter parameter controls the sparseness of the data in the feature space, and makes it possible to control the overlap between the classes. When b is too large, the patterns tend to be similar if the distance $\|x-y\|$ is small, because $\tilde{k}(x,y) \approx \exp(b)$.

For certain kernel functions, the hyperparameter C does not appear, i.e. the new kernel function $\tilde{k}$ does not have an extra parameter. We can illustrate this when using the polynomial kernel with three parameters :

$$k(x,y) = (ax.y+b)^n,$$

where a and b are positive reals and n the degree.

We have

$$\begin{aligned} \tilde{k}(x,y) &= C(ax.y+b)^n \\ &= (C^{1/n})^n(ax.y+b)^n \\ &= (C^{1/n}ax.y+C^{1/n}b)^n. \end{aligned}$$

Then,

$$\tilde{k}(x,y) = (\tilde{a}x.y+\tilde{b})^n, \tag{7.24}$$

where $\tilde{a} = C^{1/n}a$ and $\tilde{b} = C^{1/n}b$.

Another example of a kernel for which we do not need an extra parameter to define the new kernel function is the KMOD kernel (Ayat et al., 2002a) :

$$k(x,y) = a\left[\exp\left(\frac{\gamma^2}{\|x-y\|^2+\sigma^2}\right) - 1\right],$$

and

$$\tilde{k}(x,y) = \tilde{a}\left[\exp\left(\frac{\gamma^2}{\|x-y\|^2+\sigma^2}\right) - 1\right], \tag{7.25}$$

where $\tilde{a} = Ca$.

Table 7.1 shows the definition of the kernel for the unified framework corresponding to the popular kernel functions. We do not add an extra parameter to build the new kernel function, other than for the Gaussian and the Linear kernel.

Table 7.1: Common kernel definition for the unified framework

RBF (Gaussian)	$\tilde{k}(x,y) = \exp(-a\|x-y\|^2 + b)$
Linear	$\tilde{k}(x,y) = ax.y$
Polynomial	$\tilde{k}(x,y) = (ax.y + b)^n$
Laplacian	$\tilde{k}(x,y) = \exp(-a\|x-y\| + b)$
Multi-quadratic	$\tilde{k}(x,y) = (a\|x-y\| + b)^{1/2}$
Inverse multi-quadratic	$\tilde{k}(x,y) = (a\|x-y\| + b)^{-1/2}$
KMOD	$\tilde{k}(x,y) = a\left[\exp\left(\frac{\gamma^2}{\|x-y\|^2+\sigma^2}\right) - 1\right]$

7.4. Advantages of the Unified Framework for Model Selection

The various methods developed for automatic model selection for SVMs use algorithms based on gradient descent. The criteria suggested are regarded as objective functions to be optimized. But some of the criteria which that we cited in the introduction are not tractable with respect to the variable C, an example of which is the empirical error (Ayat et al., 2005).

To overcome these problems, we can use the new formulation, because it is certain that the kernel function will be derivable with respect to C. Also, the hyperparameter C is not directly related to the constraints defining the optimization problem, and as such no longer has an influence on the objective function convexity resulting from the selection criteria.

Another advantage of the new formulation for model selection is the reduction in a number of hyperparameters. For example, for the polynomial kernel and KMOD, the number of kernel parameters remains unchanged in spite of the inclusion of C. Thus, the number of hyperparameters is reduced to the number of kernel parameters. This makes easier as well the model selection manually as automatically. The reduction of the hyperparameter number reduces the search space and by this the optimization algorithm for model selection is accelerated and gives more accurate results. Then, our unified framework can be applied with any model selection criterion.

7.5. Application of the Unified Framework for Model Selection

7.5.1. The empirical error criterion

In this section, we describe the optimization of the SVM kernel parameters using the empirical error (Ayat et al., 2002b, 2005). This criterion was first developed to tune only kernel parameters but with our unified framework, it is possible to tune both the kernel parameters and the hyperparameters C, because we include C in the definition of the new kernel $\tilde{k}$.

Let us define $t_i = (y_i + 1)/2$. The empirical error is given by the following expression:

$$E_i = |t_i - \hat{p}_i|, \tag{7.26}$$

where $\hat{p}_i$ is the estimated posterior probability corresponding to the data example x_i.

The estimated posterior probability is determined by :

$$\hat{p}_i = \frac{1}{1 + \exp(A.f_i + B)}, \tag{7.27}$$

where $f_i = f(x_i)$ and the parameters A and B are fitted after minimizing the cross-entropy error as proposed by Platt (Platt, 2000).

The use of the model developed by Platt to estimate this probability makes it possible to quantify the distance from one observation to the hyperplane determined by the SVM using a continuous and derivable function. Indeed, the probability estimate makes it possible to calibrate the distance $f(x_i)$ between 0 and 1 with the following properties:

- the observations of the positive class which are well classified and located away from the margin have probabilities considered to be very close to 1;
- the observations of the negative class which are well classified and located away from the margin have probabilities considered to be very close to 0;
- and the observations located in the margin have probabilities considered to be proportional to $f(x_i)$.

Thus, with the empirical error criterion, only the misclassified observations and those located in the margin determined by the SVM are very important, since the other observations give almost null errors. Consequently, minimization of the empirical error involves the reduction of the support vectors (observations being in the margin). In other words, minimization of the empirical error makes it possible to select hyperparameters defining a margin containing fewer observations. We then construct a machine with fewer support vectors, which reduces the complexity of the classifier. The results of the tests reported in (Ayat et al., 2002b) confirm this property of the SVM constructed using the empirical error.

In fact, we have :

$$|t_i - \hat{p}_i| = \begin{cases} \hat{p}_i & \text{if } y_i = -1 \\ 1 - \hat{p}_i & \text{if } y_i = 1 \end{cases}$$

Then :
$E_i \to 0$ when $\hat{p}_i \to 0$ for $y_i = -1$ and $\hat{p}_i \to 1$ for $y_i = 1$
Consequently :
$E_i \to 0$ if $f(x_i) < -1$ for $y_i = -1$ and $f(x_i) > 1$ for $y_i = 1$

We note that minimization of the empirical error forces the maximum of the observations to be classified away from the margin, which makes this criterion useful for regularizing the maximization of the margin for SVMs.

We assume that the kernel function depends on one or several parameters, encoded within the vector $\theta = (\theta_1, \ldots, \theta_n)$. These parameters are optimized by a gradient descent minimization algorithm (Bengio, 2000) where the objective function is $E = \sum E_i$ (see Algorithm 7.1). The convergence is reached when the best fitness value is not improved after a specified number of iterations.

The derivative of the empirical error with respect to θ is evaluated using the validation dataset. If we assume N to be the size of the validation dataset; then :

$$\frac{\partial E}{\partial \theta} = \frac{\partial}{\partial \theta}\left(\frac{1}{N}\sum_{i=1}^{N} E_i\right) = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial E_i}{\partial \theta}, \tag{7.28}$$

with

$$\frac{\partial E_i}{\partial \theta} = \frac{\partial E_i}{\partial f_i}.\frac{\partial f_i}{\partial \theta}.$$

* Computation of $\frac{\partial E_i}{\partial f_i}$

$$\frac{\partial E_i}{\partial f_i} = \frac{\partial E_i}{\partial \hat{p}_i}.\frac{\partial \hat{p}_i}{\partial f_i},$$

where

$$\frac{\partial E_i}{\partial \hat{p}_i} = \frac{\partial |t_i - \hat{p}_i|}{\partial \hat{p}_i} = \begin{cases} -1 & \text{if } t_i = 1 \\ +1 & \text{if } t_i = 0 \end{cases}$$

and

$$\frac{\partial \hat{p}_i}{\partial f_i} = -A\hat{p}_i(1-\hat{p}_i).$$

Then $\frac{\partial E_i}{\partial f_i}$ is equal to:

$$\frac{\partial E_i}{\partial f_i} = Ay_i\hat{p}_i(1-\hat{p}_i). \tag{7.29}$$

* Computation of $\frac{\partial f_i}{\partial \theta}$

$$\frac{\partial f_i}{\partial \theta} = \sum_{j=1}^{NVS} y_j \left[\frac{\partial \tilde{k}(x_j,x_i)}{\partial \theta}\tilde{\alpha}_j + \frac{\partial \tilde{\alpha}_j}{\partial \theta}\tilde{k}(x_j,x_i) \right] + \frac{\partial b}{\partial \theta}. \tag{7.30}$$

This derivative is composed of two parts. We may include the bias b into the parameter vector $\tilde{\alpha}$ as $(\tilde{\alpha}_1, \ldots, \tilde{\alpha}_{NVS}, b)$. We then use the following approximation proposed by Chapelle et al. (Chapelle et al., 2001).

$$\frac{\partial \tilde{\alpha}}{\partial \theta} = -H^{-1}\frac{\partial H}{\partial \theta}\tilde{\alpha}^T, \tag{7.31}$$

where

$$H = \begin{pmatrix} K^Y & Y \\ Y^T & 0 \end{pmatrix}. \tag{7.32}$$

In Equation (7.32), H represents the Hessian matrix of the SVM objective called the modified Gram Schmidt matrix. Its components K^Y_{ij} are equal to $y_i y_j \tilde{k}(x_i,x_j)$ and Y is a vector of size $NVS \times 1$ containing support vector labels y_i.

Algorithm 7.1: SVM model selection using the empirical error criterion

```
Input Training set, Validation set, kernel type, learning rate $\eta$
Output Hyperplane $< \tilde{\alpha}, b >$, optimal kernel parameters $\theta$
Initialize the kernel parameters
while convergence is not reached do
    - Train SVM with current kernel parameters
    - Estimate A and B for the sigmoid
    - Estimate the probability of the error
    - Compute the gradient of the error
    - Correct the kernel parameters according to the gradient as $\theta^{t+1} = \theta^t - \eta \frac{\partial E}{\partial \theta}$
end while
```

7.5.2. The radius-margin criterion

The radius-margin criterion is a bound of the leave-one-out (LOO) error. In Vapnik (1998), it was shown that the following bound, called radius-margin bound holds:

$$LOO_{Error} \leq 4R^2 \parallel w \parallel \tag{7.33}$$

where $\| w \|$ is the solution of (2) and R is the radius of the smallest sphere containing all the samples (training points) in the feature space. In practice, the radius is the solution for the following quadratic problem:

$$R^2 = \max_{\beta} \sum_{i=1}^{\ell} \beta_i \tilde{k}(x_i, x_i) - \sum_{i,j=1}^{\ell} \beta_i \beta_j \tilde{k}(x_i, x_j) \tag{7.34}$$

$$\text{subject to } \sum_{i=1}^{\ell} \beta_i = 1; \beta \geq 0, i = 1, \ldots \ell$$

In Chapelle and Vapnik (1999), the radius-margin criterion is minimized for the first time by the gradient descent algorithm for finding the kernel parameter and C. The experimental results obtained show that this criterion is a good one to use for SVM model selection. However, it has also been shown that it only performs for the L2-SVM (Duan et al., 2003). So, another expression for the radius-margin criterion is proposed for the L1-SVM. In this study, we use the modified radius-margin criterion proposed in Chung et al. (2003) which is expressed as follows:

$$RM = (R^2 + \Delta/C)(\| w \|^2 + 2C \sum_{i=1}^{\ell} \xi_i) \tag{7.35}$$

where Δ is a positive constant. For our unified framework, we set $C = 1$.

The computation of the gradient of the RM is given by:

$$\frac{\partial RM}{\partial \theta} = \frac{\partial (R^2 + \Delta)}{\partial \theta} (\| w \|^2 + 2C \sum_{i=1}^{\ell} \xi_i) + \frac{\partial (\| w \|^2 + 2\sum_{i=1}^{\ell} \xi_i)}{\partial \theta} (R^2 + \Delta) \tag{7.36}$$

Using Equation (7.34), we obtained :

$$\frac{\partial (R^2 + \Delta)}{\partial \theta} = \sum_{i=1}^{\ell} \beta_i \frac{\partial \tilde{k}(x_i, x_i)}{\partial \theta} - \sum_{i,j=1}^{\ell} \beta_i \beta_j \frac{\partial \tilde{k}(x_i, x_j)}{\partial \theta} \tag{7.37}$$

The expression $\| w \|^2 + 2\sum_{i=1}^{\ell} \xi_i$ is equivalent to $\sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \tilde{\alpha}_i \tilde{\alpha}_j y_i y_j \tilde{k}(x_i, x_j)$ in dual space. Thus, we have :

$$\frac{\partial (\| w \|^2 + 2\sum_{i=1}^{\ell} \xi_i)}{\partial \theta} = - \sum_{i,j=1}^{\ell} \tilde{\alpha}_i \tilde{\alpha}_j y_i y_j \frac{\partial \tilde{k}(x_i, x_j)}{\partial \theta} \tag{7.38}$$

Algorithm 7.2 shows in detail how to minimize the RM criterion with the gradient descent strategy.

7.6. Experiments and Results

As mentioned early on in this paper, our method was designed to be independent of the type of model selection criterion and kernel function. In this section, we conduct an experiment with our unified framework and two model selection criteria, and test two different kernel functions. We also show the impact of kernel choice.

Algorithm 7.2: SVM model selection using modified radius-margin criterion

```
Input Training set, kernel type, learning rate $\eta$
Output Hyperplane $< \tilde{\alpha}, b >$, optimal kernel parameters $\theta$
Initialize the kernel parameters
while convergence is not reached do
    - Train SVM with current kernel parameters
    - Found the radius by solving (7.34)
    - Compute the gradient of RM
    - Correct the kernel parameters according to the gradient as $\theta^{t+1} = \theta^t - \eta \frac{\partial RM}{\partial \theta}$
end while
```

7.6.1. Datasets and Experimental Setup

We used the following Challenge Benchmark datasets: ADA, GINA, HIVA, NOVA and SYLVA. Each dataset is partitioned into three subsets for training, validation and test, describing the binary classification problem. Table 7.2 provides more information about the five datasets. We kept all the datasets intact without performing any preprocessing on the feature values.

Table 7.2: Description of the Challenge Benchmark datasets

Datasets	ADA	GINA	HIVA	NOVA	SYLVA
Features Number	48	970	1617	16969	216
Training samples	4147	3153	3845	1754	13086
Validation samples	415	315	384	175	1308
Test samples	41471	31532	38449	17537	130858
Positive percent	24.8	49.2	3.5	28.5	6.2

We used Joachims' algorithm, called SVMlight to train the SVMs, and we adjusted the bias by minimizing the balanced error on the validation set. We defined the modified RBF kernel as the *user kernel* in the file *kernel.h* with two parameters. Since the definition of the polynomial kernel for the unified framework remains unchanged, we used the same one as defined in SVMlight. We fixed the degree of the polynomial kernel to 3, while the parameters a and b were tuned using the given criterion.

For model selection, we used the empirical error and radius-margin criteria described in Section 7.5. The empirical error was estimated on the validation set while the radius-margin was computed on the training set. Technically, we minimize each criterion by using gradient descent algorithm. However, sometimes our problem is not convex, in which case, we use many starting points to overcome it. We can also use the simple function *fminsearch* implemented in Matlab with different starting points.

7.6.2. Results and Discussion

Tables 7.3, 7.4 and 7.5 present the results obtained using of each of the two model selection criterion with either the polynomial or the RBF kernel and the various datasets.

We note that the performance of the classifier built depends on the type of kernel used. It should also be noted that the RBF kernel does not perform for all the datasets, and the polynomial kernel gives good results on some of them. For example, in the case of the ADA dataset, the polynomial kernel performs better than the RBF kernel. This confirms that the kernel type

Table 7.3: Performance of our method using the empirical error criterion with the polynomial kernel

Datasets	Hyperparameters $(\ln(a), \ln(b))$	Balanced Error		
		Training	Validation	Test
ADA	$(-10.4756, -0.9891)$	0.172	0.1868	0.1821
GINA	$(-14.0872, -0.7335)$	0	0.0319	0.0581
HIVA	$(6.3600, 1.2200)$	0	0.2676	0.3226
NOVA	$(-7.3730, 1.0154)$	0.0054	0.036	0.0538
SYLVA	$(-15.3893, -7.10^{-5})$	0.0073	0.0115	0.0225

Table 7.4: Performance of our method using the empirical error criterion with the RBF kernel

Datasets	Hyperparameters $(\ln(a), \ln(b))$	Balanced Error		
		Training	Validation	Test
ADA	$(-13.0529, -92.0120)$	0.2433	0.2491	0.256
GINA	$(-15.5310, 1.1393)$	0	0.0352	0.0574
HIVA	$(-4.0321, 0.0399)$	0.0023	0.2618	0.2959
NOVA	$(-5.1427, -0.1427)$	0.0032	0.042	0.0574
SYLVA	$(-15.4850, -0.7109)$	0.0095	0.0127	0.0189

Table 7.5: Performance of our method using the radius-margin criterion with the RBF kernel

Datasets	Hyperparameters $(\ln(a), \ln(b))$	Balanced Error		
		Training	Validation	Test
ADA	$(-10.5620, -107.8668)$	0.221	0.2756	0.2546
GINA	$(-15.2356, 2.8698)$	0	0.0414	0.0581
HIVA	$(-0.9344, 2.1061)$	0	0.3929	0.4634
NOVA	$(-3.4891, 2.5964)$	0	0.088	0.101
SYLVA	$(-12.9651, 2.5168)$	0.000	0.0135	0.0244

is specific to the problem [2]. Figure 7.2 shows the balanced error obtained on a test set for the two types of kernels when we used the empirical error criterion.

With the SVM, as with other kernel classifiers, the choice of kernel corresponds to choosing a function space for learning. The kernel determines the functional form of all possible solutions. Thus, the choice of kernel is very important in the construction of a good machine. So, in order to obtain a good performance from the SVM classifier, we need first to design or choose a type of kernel and then optimize the SVM's hyperparameters to improve the classifier's generalization capacity.

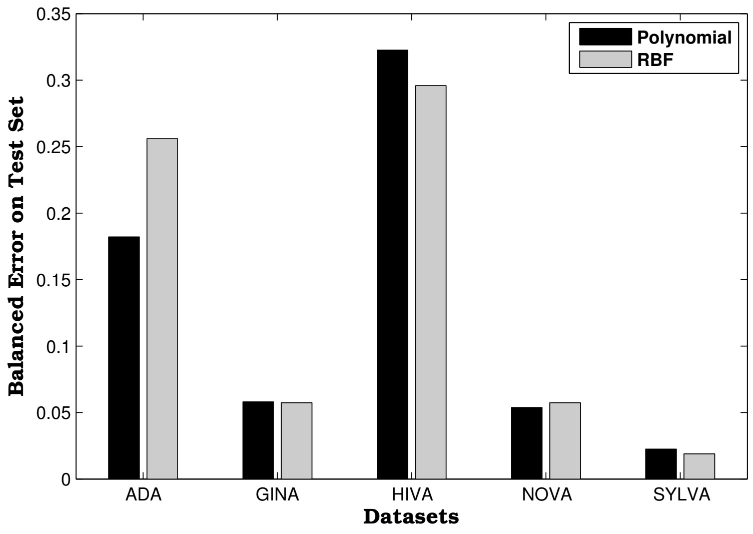


Figure 7.2: Comparison of the kernel performances for empirical error criterion

In Table 7.6, we report other results on the same Benchmark obtained from the Challenge Website and from the referenced papers. First, we compare our result with the summary results obtained from all entries reported by the Challenge Organizers in Guyon et al. (August 2007). Since we did our test on "Agnostic Learning" datasets, we plot the comparison figure with the MIN AL BER and the MEDIAN AL BER. For each dataset, our framework method performs better than the latter and worse than the former. The best entries in this category were obtained by Roman Lutz, who used boosting techniques (Lutz, July 2006) and Gavin Cawley who used Least Squares SVM (Cawley, July 2006; Cawley and Talbot, August 2007). Second, the comparison with other methods based on SVM shows the performance of our framework. Figure 7.4 illustrates how our method performs in comparison with other SVM classifiers built by using various techniques : Wei Chu and Chapelle had the best entries with the SVM classifier during the first challenge (Performance Prediction Challenge) and Franc Vojtech was an individual dataset winner with the HIVA dataset during the second challenge. Our results, obtained by empirical error minimization, are similar to those obtained by the classical grid search method

2. We confirm the well-known result (the choice of the kernel is important) by our experimental results in order to point out the usefulness of different types of kernels. Because, the RBF kernel is used most of the time, at the expense of the others; but, the RBF kernel is based on the distance and is not dependent on the direction like polynomial kernel.

Table 7.6: Performance of other methods on Challenge Benchmark : Min AL BER is the best BER on test and Median AL BER represents the median for all entries reported in Guyon et al. (August 2007); Chu (2006) used SVM/GPC with feature normalized to have variance=1 and feature pruning on GINA and HIVA; Olivier Chapelle used L2-SVM RBF kernel with feature normalized to have variance=1 and model selection done by minimizing the leave-one-out error; Franc Vojtech used also the RBF SVM with hyperparameters tuned using LOO BER.

Datasets	Balanced Error				
	Min AL BER	Median AL BER	Wei Chu	O. Chapelle	F. Vojtech
ADA	0.166	0.195	0.1899	0.184	0.2037
GINA	0.033	0.068	0.0381	0.068	0.0552
HIVA	0.271	0.305	0.2905	0.2918	0.2827
NOVA	0.046	0.081	0.048	0.0737	0.0877
SYLVA	0.006	0.014	0.01	0.0137	0.0205

(Chu, 2006), the latter being quite costly in terms of computing time and becoming intractable when there are more than two hyperparameters.

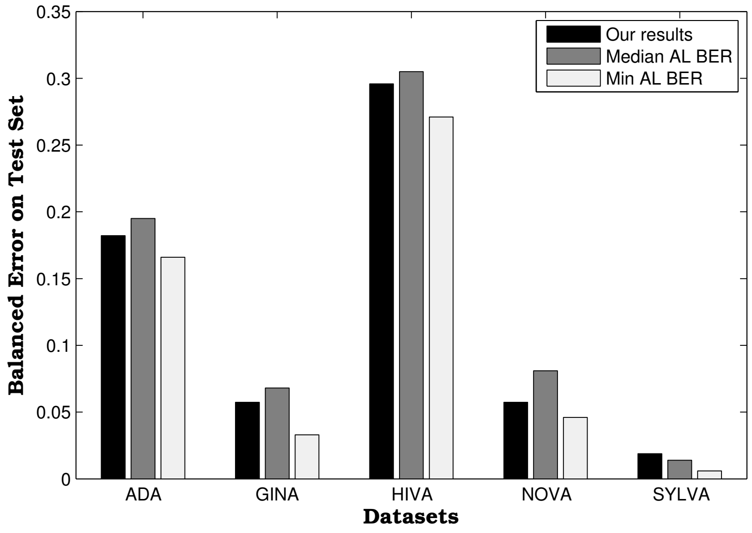


Figure 7.3: Comparison with the results of all entries

7.7. Conclusion

In this chapter, we have described a unified framework for SVM model selection. This framework makes it possible to define a new form of kernel function which includes the hyperparam-

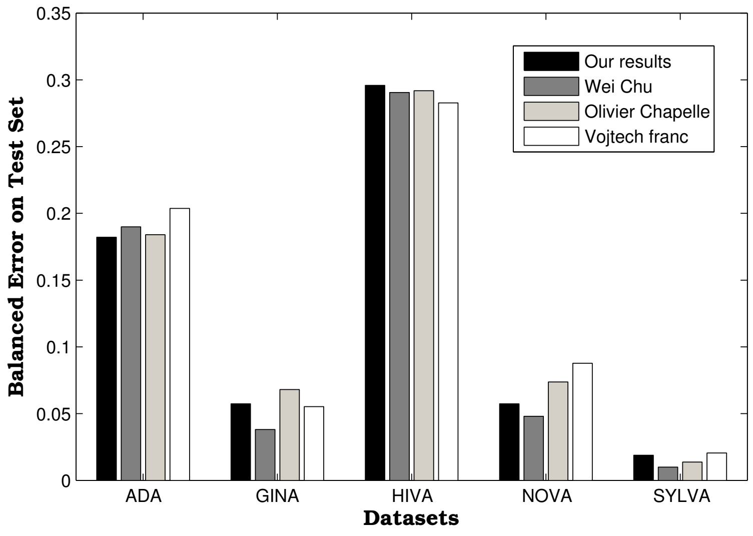


Figure 7.4: Comparison of our results with those of the other techniques using the SVM, ours having been obtained using the empirical error criterion with a polynomial kernel for ADA and an RBF kernel for the others.

eter C that controls the amount of overlap. Also, when we use certain kernel functions, such as polynomial, KMOD, Laplacian, etc., the number of hyperparameters is reduced. Consequently, with this framework, model selection for SVMs becomes easy, and is equivalent to tuning the parameters of the newly defined kernel. We applied our model selection method using the empirical error and radius-margin criteria, and obtained promising results on the Challenge dataset. As pointed out in the literature and confirmed in the Experiments and Results section, the choice of kernel function is very important in all kernel machines, and especially in the SVM. Thus, the question of how to choose the best kernel function for a given dataset is as important as how to optimize the kernel parameters. So, it is not enough to choose any kernel, optimize its parameters and wait for the designed classifier to perform well. The complete way to build an SVM classifier for a given problem is to first choose an appropriate kernel function and then carefully tune its parameters. This procedure will enable good performance. From there, the important issue may become how to choose an appropriate kernel function for the given data. Finding a way to do this will be an interesting direction for our future work.

Acknowledgments

We would like to thank Isabelle Guyon for her help during our experiments on the Challenge datasets, the anonymous reviewers for their helpful comments and the NSERC of Canada for the financial support.

References

Mathias M. Adankon and Mohamed Cheriet. Optimizing resources in model selection for support vector machines. *Pattern Recognition, in Computer Science*, 40(3):953–963, 2007.

N. E. Ayat, M. Cheriet, and C. Y. Suen. Kmod-a two parameter svm kernel for pattern recognition. *International Conference on Pattern Recognition*, 2002a.

N. E. Ayat, M. Cheriet, and C. Y. Suen. Empirical error based optimization of svm kernels: application to digit image recognition. *International Workshop on Handwriting Recognition*, pages 292–297, 2002b.

N. E. Ayat, M. Cheriet, and C.Y. Suen. Automatic model selection for the optimization of the svm kernels. *Pattern Recognition, in Computer Science*, 38(10):1733–1745, 2005.

Y. Bengio. Gradient-based optimization of hyper-parameters. *Neural Computation*, 12(8): 1889–1900, 2000.

Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In *Computational Learing Theory*, pages 144–152, 1992.

Gavin Cawley. Leave-one-out cross-validation based model selection criteria for weighted ls-svms. In *proceedings IJCNN 2006, Vancouver, Canada*, July 2006.

Gavin Cawley and Nicola Talbot. Agnostic learning versus prior knowledge in the design of kernel machines. In *proceedings IJCNN 2007, Orlando, Florida*, August 2007.

O. Chapelle and V. Vapnik. Model selection for support vector machines. *Advances in Neural Information Processing Systems*, 1999.

O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 2001.

Wei Chu. Model selection: An empirical study on two kernel classifiers. In *proceedings IJCNN 2006, Vancouver, Canada*, 2006.

K.-M. Chung, W.-C. Kao, L.-L. Wang C.-L. Sun, and C.-J. Lin. Radius margin bounds for support vector machines with the rbf kernel. *Neural Computation*, 15:2643–2681, 2003.

Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

K. Duan, S. Keerthi, and A. N. Poo. Evaluation of simple performance measures for tuning svm hyperparameters. *Neurocomputing*, 51:41–59, 2003.

Carl Gold and Peter Sollich. Fast bayesian support vector machine parameter tuning with the nystrom method. In *IJNN'05*, pages 2820–2825, 2005.

Isabelle Guyon, Amir Saffari, Gideon Dror, and Gavin Cawley. Agnostic learning vs. prior knowledge challenge. In *proceedings IJCNN 2007, Orlando, Florida*, August 2007.

T. S. Jaakkola and D. Haussler. Probabilistic kernel regression models. *Workshop in Conference on Artificial Intelligence and Statistics*, 1999.

T. Joachims. Estimating the generalization performance of a svm efficiently. *International Conference on Machine Learning*, pages 431–438, 2000.

Roman Lutz. Logitboost with trees applied to the wcci 2006 performance prediction challenge datasets. In *proceedings IJCNN 2006, Vancouver, Canada*, July 2006.

M. Opper and O. Winther. Gaussian processes and svm: Mean field and leave-one-out. In A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 311–326. MIT Press, Cambridge, 2000.

Manfred Opper and Ole Winther. Mean field methods for classification with gaussian processes. In *the 1998 conference on Advances in neural information processing systems II*, pages 309–315. MIT Press, 1999.

J. Platt. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In A.J. Smola, P. Bartlett, B. Schoelkopf, and D. Schuurmans, editors, *Advances in Large Margin Classiers*, pages 61–74. 2000.

B. Scholkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

V. Vapnik and O. Chapelle. Bounds on error expectation for support vector machines. *Neural Computation*, 12(9), 2000.

V. N. Vapnik. *Estimation of Dependences based on Empirical Data*. Springer Verlag, Berlin, 1982.

V. N. Vapnik. Principles of risk minimization for learning theory. *Adavances in Neural Information Processing Systems 4, Morgan Kaufman, San Mateo, CA*, pages 831–838, 1992.

V. N. Vapnik. *Statistical learning theory*. John Wiley and Sons, New York, 1998.

G. Wahba, Y. Lin, and H. Zhang. Generalized approximate cross validation for support vector machines, or, another way to look at margin-like quantities. Technical report, Departement of Statistics,University of Wisconsin, February 25 1999.

Chapter 8

Liknon feature selection: Behind the scenes

Erinija Pranckeviciene ERINIJA.PRANCKEVICIENE(@MF.VU.LT, @GMAIL.COM)
Department of Human and Medical Genetics,
Vilnius University,
Santariskiu 2, LT-08661 Vilnius-21, Lithuania.

Ray Somorjai RAY.SOMORJAI@NRC-CNRC.GC.CA
Institute for Biodiagnostics,
National Research Council Canada,
435 Ellice Avenue, Winnipeg, MB, Canada.

Editor: Isabelle Guyon, Gavin Cawley, Gideon Dror and Amir Saffari

Abstract

Many real-world classification problems (biomedical among them) are represented by very sparse and high dimensional datasets. Due to the sparsity of the data, the selection of classification models is strongly influenced by the characteristics of the particular dataset under study. If the class differences are not appreciable and are masked by spurious differences arising because of the peculiarities of the dataset, then the robustness/stability of the discovered feature subset is difficult to assess. The final classification rules learned on such subsets may generalize poorly. The difficulties may be partially alleviated by choosing an appropriate learning strategy. The recent success of the linear programming support vector machine (Liknon) for feature selection motivated us to analyze Liknon in more depth, particularly as it applies to multivariate sparse data. The efficiency of Liknon as a feature filter arises because of its ability to identify subspaces of the original feature space that increase class separation, controlled by a regularization parameter related to the margin between classes. We use an approach, inspired by the concept of transvariation intensity, for establishing a relation between the data, the regularization parameter and the margin. We discuss a computationally effective way of finding a classification model, coupled with feature selection. Throughout the paper we contrast Liknon-based classification model selection to the related Svmpath algorithm, which computes a full regularization path.

Keywords: Feature selection, Linear programming, Margin, Transvariation intensity, Transvariation intensity function, Liknon, Regularization parameter C, Full regularization path.

8.1. Introduction

Certain (e.g., biomedical) classification problems, characterized by very sparse and high dimensional datasets, suffer from a generic difficulty: due to the sparsity, the learned classification models are strongly influenced by the characteristics of the investigated dataset. This is manifested by overfitting, caused by sample bias (Zucchini, 2000). Many feature selection strategies and methods (Guyon et al., 2006) and comparisons have been proposed in the literature (Kudo and Sklansky, 2000; Kohavi and John, 1997). Indeed, when the sample size is small and the dimensionality high, the feature selection procedure, driven by the optimization of some criterion that ensures increasing class separation, will adapt to the training data (Ambroise and McLachlan, 2002). "Too much selection can do more harm than good" (Zucchini, 2000). Even if there exist classification models that perform well without feature selection, for the interpretability of the results it is still important to determine the set of "markers" that provide good class discrimination via feature selection, whether filter, wrapper or embedded. If the dataset is "easy", i.e., the class differences are not masked by the noise in the data, one would expect this to be revealed by the validation of the feature selection procedure. Ideally, the classification error estimate will have low variance and the identities of discovered features will not vary appreciably across different random splits of the training data.

Investigations both by other researchers and by us suggest that a feature selection method based on linear programming, originally introduced by (Fung and Mangasarian, 2004), has the desired stability properties and is robust with respect to the sample bias. For a particular application, profiling of gene expression microarrays, for which the data dimensionality exceeds the available number of samples by orders of magnitude, the usefulness of the linear programming support vector machine named Liknon was demonstrated (Bhattacharyya et al., 2003). The method was investigated further and used in practical tasks of face recognition (Guo and Dyer, 2005). It was applied for classification of spectral data (Pranckeviciene et al., 2004). The Liknon feature selection, combined with other classification rules, was among the top-ranked methods in the Agnostic learning vs. Prior knowledge competition (Guyon et al., 2007a).

Useful insights have been gained (Cherkassky and Ma, 2006) on the role of the margin between the classes as an effective measure of the match between data complexity and the capacity of a learning rule. A practical capacity control of a linear rule via the structural risk minimization principle was suggested (Guyon et al., 1992). It is known that the regularization parameter C in kernel-based classification methods controls the tradeoff between maximizing the margin of the classifier and minimizing the margin errors of the training data (Igel, 2005). It is important to use an appropriate C value for an improved generalization performance of the classifier. Usually, the value of this parameter is determined by grid search and crossvalidation.

The formulation of Liknon as a linear programming problem provides a framework for a systematic search of C, computed from the training data. The role of the parameter in Liknon feature selection can be summarized as follows: the non-zero weights of the Liknon discriminant, identifying important features, correspond to those individual data dimensions, for which the absolute difference between the classes is greater than $1/C$. A transvariation intensity function, inspired by the concept of transvariation intensity (Montanari, 2004) applied to the training data, reveals how the parameter C relates quantitatively to the class separation by the margin through the ratio of class overlap over the class difference. This relation leads to an algorithm for the computation of C in Liknon, and a strategy for classification model selection. The closest method to the Liknon-based classification model selection is the Svmpath algorithm (Hastie et al., 2004), which computes the full regularization path for a given classification problem. Throughout the paper we briefly sketch the similarities and differences of the two methods.

The rest of the chapter is organized as follows. In Section 8.2, both Liknon-based and Svmpath-based feature and classification model selection are highlighted, using an artificial dataset, for which classes separate nonlinearly. The details of the primal and dual Liknon formulations are described in Section 8.3. We introduce the transvariation intensity function and analyze it in depth in Section 8.4 in relation to the linear SVM. We derive the relationship between C and the ratio of class overlap over class difference. We illustrate experimentally, that the Liknon and Svmpath algorithms have similar solutions with respect to this ratio. The algorithm for computing C in Liknon is the topic of Section 8.5. The NIPS 2003 feature selection (NIPS 2003 FS) (Guyon et al., 2006) and the recently organized Agnostic Learning versus Prior knowledge (ALvsPK) (Guyon et al., 2007a) competitions provided excellent platforms for controlled experiments with real-life datasets. In Section 8.6 Liknon feature selection is discussed in the context of these two challenges. For the classification experiments, the functions from

PRTools (Duin et al., 2004) and the publicly available Liknon Matlab script (Bhattacharyya et al., 2003) were used. The Matlab code for computing the range of C values from the training data is listed in the Appendix. For the comparison with the Svmpath method, implemented in R (Hastie et al., 2004), Liknon was also implemented in R.

8.2. Liknon and Svmpath based feature and classification model selection when classes separate nonlinearly: case study on artificial Banana dataset

In this section we give an overview of Liknon-based feature/classification model selection, contrasting it with a related Svmpath algorithm using a linear kernel. The efficiency of Liknon feature selection in finding ground truth features was demonstrated on the artificial dataset (Pranckeviciene et al., 2007), in which class separation was due to the difference in the means of two features that separated classes linearly. Liknon feature selection is also applicable where classes separate nonlinearly, but it would find only the "linear part" of feature relevance. The varying margin in the embedded feature selection by Liknon facilitates "retrieving" multivariate feature subsets separating the classes linearly. Liknon will not retrieve the features of classes made of multiple clusters such as checkerboard or concentric classes. A nonlinear boundary between classes should be such, that it can be approximated by linear boundary with some margin. After the feature selection step, other classifiers can be explored with the selected features, aiming to improve the classification performance. The approach consists of two parts: obtaining the most prominent features via Liknon, and using them with other classifiers, or using an ensemble of Liknon discriminants. The winning model is determined by the smallest classification error in k-fold crossvalidation. First, we explain the general computational procedure of the Liknon feature selection and then we study an example. We also apply the related Svmpath algorithm on the same data. To compare fairly the relative computational times of the Liknon and Svmpath methods, the Liknon feature/classification model selection method was re-coded in R (code is available from the first author upon request).

8.2.1. Computational procedure for Liknon-based feature selection

The computational paradigm for Liknon feature selection, using the training data, is k-fold crossvalidation. The number of folds k, usually 5 or 10, is determined by the sample size of the training data. In the outer loop, the data is divided into a training set `Training` and a validation set `Validation`. During classifier development, to account for the variance in the dataset, the `Training` set in the inner loop is randomly partitioned several times into two: a balanced *Training* set and the remaining *Monitoring* set. The classifier development is performed on the `Training` set. The `Validation` set is used for the assessment of the final classification model. In every data partition/split, the number `NM` of Liknon discriminants is identified, based on the *Training* set. The sequence of the `NM` increasing values of the regularization parameter C guides the search for the optimal discriminant. The optimality criterion is the balanced classification error rate (BER) of the *Monitoring* set, computed from the confusion matrix $confmat$ of the classifier:

$$confmat = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix}, \quad BER = \tfrac{1}{2}\left(\tfrac{FP}{(FP+TP)} + \tfrac{FN}{(FN+TN)}\right) ,$$

where TP is the number of true positives, FP is the number of false positives, FN is the number of false negatives and TN is the number of true negatives. Increasing the number of features in the discriminant leads to a gradual adaptation to the *Training* set - overfitting, as evidenced by the rapidly decreasing training error. The *Monitoring* set is used to monitor the adaptation and find the best discriminant with the minimum monitoring BER of the particular data split.

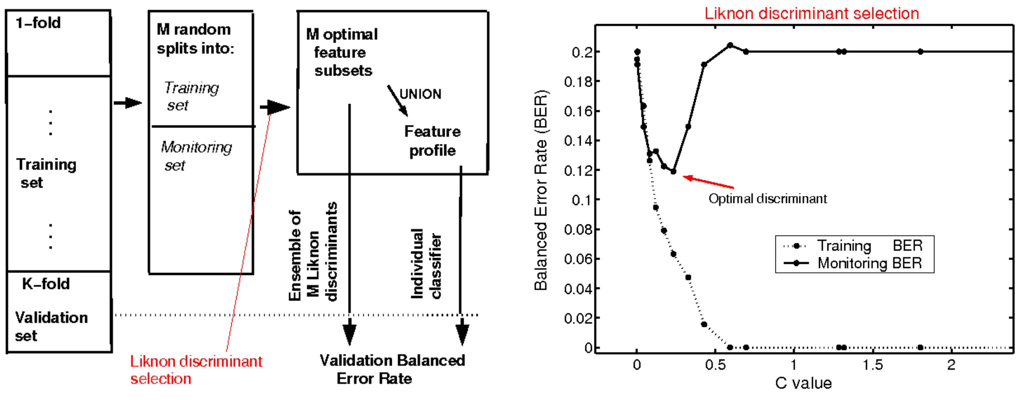


Figure 8.1: Computational scheme of Liknon-based feature/classification model selection. The left panel shows the steps in the crossvalidation procedure. The right panel shows the selection of an optimal Liknon discriminant using the monitoring set.

Every discriminant is associated with a feature subset. The number of important features is different for different discriminants. For small sample sizes for the training and monitoring sets, noisy features will occur in the model. A feature profile is created during the development, by counting the frequency of inclusion of each feature into the optimal discriminant. In the feature profile, peaking occurs for important features. Feature profiles are very important both for interpretation and for exploratory data analysis. The parameters of the presented computational procedure are the number of splits M in the inner loop and the number of Liknon discriminants `NM`. We denote the `Validation` set size as `V=V1+V2`, the balanced *Training* as `T=T1+T2` and the remaining *Monitoring* as `M=M1+M2`.

The overall scheme of the procedure is presented in Figure 8.1. The sequence of the steps in crossvalidation is shown in the left panel. In the right panel, the process of the selection of the optimal Liknon discriminant is illustrated on an artificial data set. The total number of optimization operations in the outlined procedure is $K*M*NM$, where M is the number of resamplings, NM is the number of Liknon optimizations (C values explored) and K is the number of folds.

8.2.2. Identification of useful features

We study Liknon and Svmpath using the noise-augmented Banana dataset, for which the classes separate nonlinearly. The dataset dimensionality is $D = 100$, the sample size is $N_1 + N_2 = 210 + 190$. Features 29 and 30 separate the two classes nonlinearly. The remaining 98 features are overlapping, normally distributed $N(0,1)$. The class distribution in features containing structure and no structure is presented in Figure 8.2.

The Liknon- and Svmpath- based feature/classification model selection is performed in 5-fold crossvalidation with a single random split in the inner loop. The best solutions of both methods are determined by the minimum monitoring BER. The sizes of the data subdivision in folds are `T1+T2=84+76`, `M1+M2=83+86`, `V1+V2=43+39`, and the number of the tested Liknon models is NM=50. The computations for Liknon were continued until a stable solution

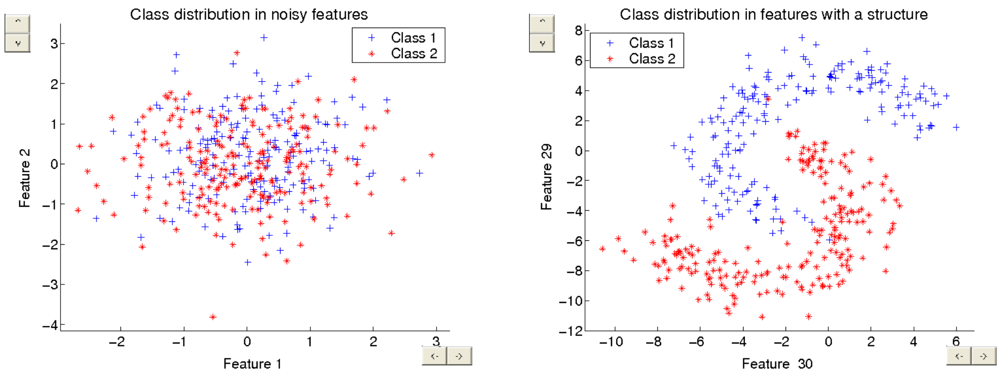


Figure 8.2: Class distribution in the Banana dataset augmented with noise. The right panel shows the nonlinear structure in features 29 and 30. The left panel shows a typical class distribution for the noisy features 1 and 2.

was reached. The feature profiles, identified in every fold, are presented in Figure 8.3 as heat maps, Liknon profiles are on the right, and Svmpath are on the left. The color-coded values of the weights of discriminants may be interpreted as indicators of feature importance. The ground truth features 29 and 30 have large weight in all folds for both algorithms.

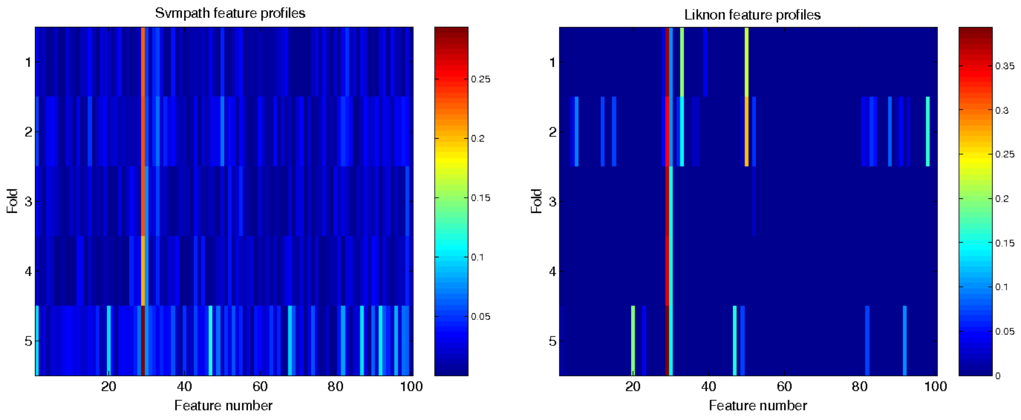


Figure 8.3: Feature profiles identified in 5 folds: Liknon on the right, Svmpath on the left.

The performances of Liknon and Svmpath methods are summarized in Table 8.1. We report the best monitoring BER, validation BER, and the validation BER of a 3nn classifier, trained using the selected features. The most important features in the Svmpath solution were those, for which the absolute value of the weight was above some threshold. Several threshold values, based on a visual examination of the Svmpath feature profiles, were explored. Time, spent by both procedures implemented in R for the computation of the best model, was estimated on a Windows-based 1.20 GHz 256 RAM PC.

Svmpath and Liknon are similar in terms of performance and Liknon's computational edge is not significant statistically. The difference is in their utility for feature selection. No large

Table 8.1: Comparison of the performances of Svmpath and Liknon.

Fold	1	2	3	4	5
Svmpath					
Monitoring BER	0.1271	0.1331	0.0888	0.0860	0.1397
Validation BER	0.1699	0.0826	0.1583	0.099	0.0489
Time (s)	15.99	17.5	16.5	18	16.99
Threshold=0.05					
3nn BER	0.0128	0.000	0.0128	0.0128	0.0385
features	3	5	4	2	23
Threshold=0.035					
3nn BER	0.0244	0.000	0.0489	0.0244	0.0489
features	8	14	6	9	38
Liknon					
Monitoring BER	0.1085	0.1266	0.0942	0.099	0.1463
Validation BER	0.1454	0.0850	0.1595	0.099	0.0620
Time (s)	6.37	12.64	14.47	9.31	8.71
3nn BER	0.0128	0.000	0.0361	0.0128	0.0244
features	6	23	3	2	9

differences are observed in monitoring and validation BER's of the best solutions of Liknon and Svmpath. None of the methods can be claimed superior in this example. The nonlinear 3nn classifier, trained on the selected feature subsets of both Liknon and Svmpath, clearly performs better in all folds on the validation sets, compared to the linear classifiers, derived from Liknon and Svmpath. Note, that the ground truth features 29 and 30 are identified in the solutions of both approaches. From the feature selection point of view, Liknon is more advantageous. Relevant features in the Liknon solution are provided by non zero weights, unlike in Svmpath. Table 8.1 shows, that the final feature subset and consequently the classification result is very sensitive to the threshold value. If the weights of the relevant features are not very distinct, choosing the threshold would present a problem, if one were to use the Svmpath solution for feature selection. Computations for Liknon with a single split in the inner loop take less time than for Svmpath. More splits in Liknon would increase the time, but still within an acceptable range. The perfect classification of the validation set in fold 2 occurs due to an accidental, split-induced data configuration. With real data, one should be cautious about a single optimistic result, which may occur just because of the overly favorable distribution of classes.

The Banana example offered several useful insights. The identified feature subsets provide information, pertinent to classification in various data "projections" in feature and sample spaces. Different classifiers perform differently in these "projections", depending on how well the feature subset represents the nature of the class separation, and the capability of the individual classifier to handle the complexity of the classification problem in the "projection". The monitoring and validation BERs of folds can be analyzed for ranking feature profiles. Liknon generates a feature profile that is optimal for linear separation. Nevertheless, we can use this feature profile as input to another classifier, realizing a nonlinear rule and gain insight into the nonlinearity of the data, yet still keeping the overall architecture simple. By processing many splits in the inner loop, we generate different feature subsets. These subsets can be accumulated into a general feature profile for another classifier, or used in the ensemble of Liknon discriminants as was done in ALvsPK challenge (Pranckeviciene et al., 2007). The justification for using this method in nonlinear separation cases still needs to be formalized and experiments need to be carried out in order to compare it with other methods, aiming to reveal possible computational or statistical advantages. In the following section the mathematical formulation of Liknon is outlined.

8.3. Liknon formulation

Liknon implements a linear rule for two-class classification:

$$y_s = \text{sign}(\mathbf{x}_s\mathbf{w}^T + w_0) \quad , \tag{8.1}$$

where $\boldsymbol{x}_s = [x_s^1, \ldots, x_s^D]$ are D-dimensional samples, $\boldsymbol{y} = [y_1, \ldots, y_N]$ is the vector of class labels, assuming values $+1$ for the positive class and -1 for the negative class, s indexes the sample number and $N = N_1 + N_2$ is the total number of samples in the two classes. The transpose is denoted by $^{\mathbf{T}}$. The * denotes the optimal solution and the ξs are slack variables. The weight vector $\boldsymbol{w}$ is obtained by solving the optimization problem:

$$(\boldsymbol{w}^*, \ldots, \xi_1^*, \ldots, \xi_N^*) = \underset{(\boldsymbol{w}, \xi_1, \ldots, \xi_N)}{\arg\min} \left(\|\boldsymbol{w}\|_1 + C\textstyle\sum_{s=1}^N \xi_s\right) \quad , \tag{8.2}$$
$$\text{s.t.:}$$
$$y_s\left(\boldsymbol{x}_s\boldsymbol{w}^T + w_0\right) + \xi_s \geq 1, \;\; \xi_s \geq 0, \;\; s = 1, \ldots, N \;\; .$$

The L_1 norm is $\|\boldsymbol{w}\|_1 = \sum_{f=1}^D |w_f|$. The formulation (8.2) is the same as for the linear SVM, except for the L_1 norm of the regularization term. The solution $\boldsymbol{w}^*$ is sparse. Because of sparsity, it is used for feature selection. Features, important for classification, are identified by large weights w_f^* of the vector $\boldsymbol{w}^*$. The regularization parameter C controls the level of sparseness. Formulation (8.2) is cast into a linear programming (LP) optimization problem, and $\boldsymbol{w}^*$ is obtained using an LP solver. The primal and dual LP optimization problems are related. In the primal optimization problem, the regularization parameter C appears in the cost function. In the dual it appears in the constraints.

8.3.1. The primal minimization problem

In order to present the minimization problem (8.2) in a form suitable for a general linear program solver, the variables in the objective function should be positive. Thus, every w_f variable is modeled by two non-negative variables u_f and v_f, a common practice in LP of changing the negative variables into positive ones (Arthanari and Dodge, 1981, page 32):

$$w_f = u_f - v_f, \; |w_f| = u_f + v_f \quad . \tag{8.3}$$

The pair of variables u_f, v_f, simultaneously satisfying conditions (8.3) is unique, given that only three choices are possible for the value of w_f: (i) $u_f = 0, \;\; v_f = 0, \;\;$ and $w_f = 0$; (ii) $u_f = 0, \;\; v_f \neq 0, \;\;$ and $w_f = -v_f$; (iii) $u_f \neq 0, \;\; v_f = 0, \;\;$ and $w_f = u_f$. The problem in (8.2) is reformulated in a form suitable for LP, by changing the variable w_f into the combination of v_f and u_f according to (8.3). The original formulation (8.2), after the change of variables, becomes

(Bhattacharyya et al., 2003):

$$(u_1^*,\ldots,u_D^*,v_1^*,\ldots,v_D^*,\xi_1^*,\ldots,\xi_N^*) = \underset{(u_1,\ldots,\xi_N)}{\arg\min} \left(\sum_{f=1}^D (u_f+v_f) + C\sum_{s=1}^N \xi_s\right),$$
$$\text{s.t.:}$$
$$\sum_{f=1}^D u_f y_s x_s^f - \sum_{f=1}^D v_f y_s x_s^f + y_s u_0 - y_s v_0 + \xi_s \geq 1, \quad \xi_s \geq 0, \quad u_f \geq 0, \quad v_f \geq 0\ ,$$
$$u_0 \geq 0, \quad v_0 \geq 0 \quad f=1,\ldots,D,\ s=1,\ldots,N\ . \tag{8.4}$$

The constant C in (8.4) is the regularization parameter.

8.3.2. Duality of linear programming

The primal and dual problems of Linear Programming are related (Papadimitriou and Steiglitz, 1982):

$$\text{minimize } J_{\min}(\boldsymbol{x}) = \boldsymbol{c}\boldsymbol{x}, \quad s.t.: \quad \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}, \quad \boldsymbol{x} \geq 0\ . \tag{8.5}$$

$$\text{maximize } J_{\max}(\boldsymbol{z}) = \boldsymbol{b}^T\boldsymbol{z}, \quad s.t.: \quad \boldsymbol{A}^T\boldsymbol{z} \leq \boldsymbol{c}, \quad \boldsymbol{z} \geq 0\ . \tag{8.6}$$

In (8.5) and (8.6), $\boldsymbol{x}$ and $\boldsymbol{z}$ denote the variables of primal and dual, $\boldsymbol{c}$ is a vector of costs and $\boldsymbol{b}$ is a vector of constraints of the primal, $J_{\min}$ and $J_{\max}$ denote the objective functions, $\boldsymbol{A}$ is a data matrix. The Liknon primal (8.4) and dual are related through the following data matrix $\boldsymbol{A}$:

$$\boldsymbol{A} = \begin{pmatrix} y_1x_1^1 & \ldots & -y_1x_1^D & y_1 & -y_1 & 1 & \ldots & 0 \\ \vdots & & \vdots & \vdots & \vdots & \ddots & & \\ \vdots & & \vdots & \vdots & \vdots & & \ddots & \\ y_Nx_N^1 & \ldots & -y_Nx_N^D & y_N & -y_N & 0 & \ldots & 1 \end{pmatrix}\ . \tag{8.7}$$

The costs $\boldsymbol{c}$ and the constraints $\boldsymbol{b}$ of (8.5) correspond to the costs and constraints of Liknon's primal minimization problem (8.4):

$$\boldsymbol{c} = [1_1,\ldots,1_{2D},0,0,C_1,\ldots,C_N] \quad \text{and} \quad \boldsymbol{b} = [1_1,\ldots,1_N]^T\ .$$

The variables of primal (8.5) and dual (8.6) correspond to the Liknon variables:

$$\boldsymbol{x} = [u_1,\ldots,u_D,v_1,\ldots,v_D,u_0,v_0,\xi_1,\ldots,\xi_N] \quad \text{and} \quad \boldsymbol{z} = [z_1,\ldots,z_N]\ .$$

For consistency with SVM terminology, for Liknon dual variables we use α instead of z.

8.3.3. The dual maximization problem

The Liknon dual is obtained straightforwardly from the primal. The costs $\boldsymbol{c}$ of the primal (8.5) become the constraints of the dual (8.6). Similarly, the constraints of the primal become the costs for the dual. Using the data matrix (8.7) of Liknon, its dual maximization problem (8.6) is formulated as follows:

$$(\alpha_1^*,\ldots,\alpha_N^*) = \underset{(\alpha_1,\ldots,\alpha_N)}{\arg\max} \left(\sum_{s=1}^N \alpha_s\right)\ ,$$
$$\text{s.t.:}$$
$$\pm y_1x_1^f\alpha_1 \pm \ldots \pm y_Nx_N^f\alpha_N \leq 1, \quad y_1\alpha_1 + \ldots + y_N\alpha_N = 0, \quad 0 \leq \alpha_s \leq C\ ,$$
$$s=1,\ldots,N, \quad f=1,\ldots,D\ . \tag{8.8}$$

The Liknon dual variables α are positive real numbers.

8.3.4. Optimality conditions

The optimal solutions of the primal and dual satisfy the optimality conditions for every feature $f = 1,\dots,D$:

$$u_f^* \left(\sum_{s=1}^N y_s x_s^f \alpha_s^* - 1 \right) = 0, \quad v_f^* \left(\sum_{s=1}^N y_s x_s^f \alpha_s^* + 1 \right) = 0, \quad \xi_s^* \left(\alpha_s^* - C \right) = 0 \ . \tag{8.9}$$

For every sample $s = 1,\dots,N$, the optimality conditions are:

$$\alpha_s^* \left(\sum_{f=1}^D y_s x_s^f (u_f^* - v_f^*) + y_s(u_0^* - v_0^*) + \xi_s^* - 1 \right) = 0 \ . \tag{8.10}$$

It is important to note that the binding constraints determine the non-zero variables of the optimal solutions. Therefore, the non-zero components w_f of $\boldsymbol{w}$ in (8.1), corresponding to the selected features, are determined solely by the constraints that become binding for u_f and v_f in (8.9).

A theoretical basis for understanding the algorithm for computing the range of C values from the training data relies on the concept of transvariation intensity.

8.4. Transvariation intensity and `margin`

Based on the concept of transvariation suggested by Gini and explained by Montanari, several class/group separation measures are defined (Montanari, 2004): transvariation area, transvariation probability and the class separation measure based on the transvariation intensity. The transvariation probability was used successfully in classification of high-dimensional data in low-dimensional projected feature spaces (Somorjai et al., 2007). We propose a modification of the transvariation intensity, to be used in our study.

8.4.1. Univariate class separation measure, based on transvariation intensity

The two classes, mapped onto a line and represented by $x_i, \ i = 1,\dots,N_1$ and $x_j, \ j = 1,\dots,N_2$ transvary around their corresponding mean values m_1 and m_2 if the sign of any of the N_1N_2 differences $x_i - x_j$ is opposite to the sign of $m_1 - m_2$, where N_1 and N_2 are the number of samples in the two classes and the indices i and j are used to distinguish the samples x_i and x_j of the two classes. Such pair is called a transvariation and the absolute difference $|x_i - x_j|$ is its intensity. The class separation measure, based on the transvariation intensity, is defined as:

$$i_{tran} = \frac{2\sum_{i=1}^{N_1}\sum_{j=1}^{N_2} |x_i - x_j|}{\sum_{i=1}^{N_1}\sum_{j=1}^{N_2} |(x_i - m_1) - (x_j - m_2)|} \ . \tag{8.11}$$

The denominator in (8.11), divided by N_1N_2, is the Gini mean difference. We will be using the expression of the denominator in (8.11), called the transvariation intensity maximum (Montanari, 2004). For the purposes of our study the sign is important. As an extension, in place of m_1 and m_2, we can take any two points, p_1 from class 1 and p_2 from class 2, or even the medians. With respect to the two arbitrary points p_1 and p_2, we can measure how well the signed interval $p_1 - p_2$ orders other points x_i and x_j of classes 1 and 2, respectively. A reference ordering on the line places the class 2 points x_j to the left, towards $-\infty$ and the class 1 points x_i to the right, towards $+\infty$. Let the pair of points (p_1, p_2) be the reference pair. For an arbitrary single pair (x_i, x_j), we compute, with respect to the reference pair (p_1, p_2), the appropriate single term in the **signed** transvariation intensity maximum of (8.11):

$$t_{ij}(p_1,p_2) = (x_i - p_1) - (x_j - p_2) = (x_i - x_j) - (p_1 - p_2) \ .$$

If the class 2 points x_j are located to the left of p_2 and the class 1 points x_i occur to the right of p_1, then the pair (x_i, x_j) is ordered by the pair (p_1, p_2) and $t_{ij}(p_1, p_2)$ is positive. If the points (x_i, x_j) are on the wrong sides of p_1 and p_2, then the $t_{ij}(p_1, p_2)$ is negative. If the ***margin*** between the classes is defined by p_1 and p_2, then the value $t_{ij}(p_1, p_2)$ is the extent by which the margin orders the pair of points (x_i, x_j). The ***margin*** is a segment on the line at a specific location, determined by the p_1 on the left and p_2 on the right. In the context of classification, the ordering may either be ***complete*** (the classes separate) or ***partial*** (there is class overlap). A complete ordering gives maximal classification accuracy. However, for the same classification accuracy, several partial orderings may exist. In margin-based classification $p_2 < p_1$. The value $t_{ij}(p_1, p_2)$ shows the amount by which the margin separates the class points.

8.4.2. Size of margin errors

In the following, $\boldsymbol{x}_s = [x_s^1, \ldots, x_s^D]$ are D-dimensional samples, the vector $\boldsymbol{y} = [y_1, \ldots, y_N]$ comprises the class labels, assuming values $+1$ for the positive class and -1 for the negative class. The indices i and j refer to samples from the positive and negative classes, respectively. $N = N_1 + N_2$ is the total number of samples in the two classes. We consider the balanced case, $N_1 = N_2$. $\boldsymbol{w}$ is a linear discriminant, onto which all multivariate points are projected. The dual variables of SVM and Liknon are denoted by α_s, assuming real, non-negative values. Let us consider the linear two-class classification rule (8.1):

$$y_s = \text{sign}(\boldsymbol{x}_s \boldsymbol{w}^T + w_0) \quad .$$

In margin-based classifiers, such as the Linear Support Vector Machine (SVM), the projected class points have to satisfy the margin requirements. Margin in SVM is specified by two positions: -1 for class 2 and $+1$ for class 1. Some points, when projected onto the discriminant, may fail to satisfy the margin requirement by an amount ξ, the slack variable:

$$\boldsymbol{x}_i \boldsymbol{w}^T + w_0 = 1 - \xi_i, \quad \boldsymbol{x}_j \boldsymbol{w}^T + w_0 = -1 + \xi_j \quad .$$

The difference between the projections and the margin is:

$$(\mathbf{x}_i - \mathbf{x}_j)\mathbf{w}^T - (1 - (-1)) = -(\xi_i + \xi_j) \quad .$$

The value of $-(\xi_i + \xi_j)$ relates to $t_{ij}(p_1, p_2)$, discussed in Section 8.4.1. It indicates quantitatively how the margin $[-1 \ , \ +1]$ separates the projected points. If both ξ_i and ξ_j are positive, meaning that the projected points fall on the wrong side of the margin, then the value $-(\xi_i + \xi_j)$ is negative.

In SVM, the margin errors are defined (Scholkopf et al., 2000) as **points** with positive slack variables $\xi_i > 0$, $\xi_j > 0$. The value of the slack variable can be interpreted as the size of the margin error. In SVM, negative values of the slack variables are not considered. If the projected pair of points falls outside the margin (i.e., classified correctly), then the margin error is zero. The negative-valued slack variables would provide information on how well the margin separated the projected class points, and the value of $-(\xi_i + \xi_j)$ would be positive. Suppose we allowed a "negative size" and counted all slacks $\xi < 0$ and $\xi \geq 0$ negative and non-negative. In this case, the size of the total margin error indicated the extent by which the margin, specified by the pair $p_1 = 1$ and $p_2 = -1$ in SVM, separated the classes, when projected onto the discriminant $\boldsymbol{w}$. We can compute the size of the total margin error by summing up all projected points $\boldsymbol{x}_i$ of class 1 and $\boldsymbol{x}_j$ of class 2 and taking the difference:

$$(\sum_{i=1}^{N_1} \mathbf{x}_i - \sum_{j=1}^{N_2} \mathbf{x}_j)\mathbf{w}^T - \frac{N}{2}(1 - (-1)) = -(\sum_{i=1}^{N_1} \xi_i + \sum_{j=1}^{N_2} \xi_j) \quad . \tag{8.12}$$

The value $\xi_{ntot} = -(\sum_{i=1}^{N_1} \xi_i + \sum_{j=1}^{N_2} \xi_j)$ in (8.12) is quantitatively related to the class separation by the margin in the following way. If there are many points on the wrong side of the margin (bad class separation), then the size of the total margin error tends to be positive since $\xi_s > 0$ for those points, and subsequently ξ_{ntot} is negative. When the margin separates the majority of the points, the size of the total margin error tends to be negative, but ξ_{ntot} positive. For classes perfectly separated by the margin, ξ_{ntot} is strictly positive.

We explained the concept of the total margin error size, using the SVM margin $[-1\ ,\ +1]$. However, any segment can play the role of margin. For some fixed $\boldsymbol{w}$, the varying margin positions p_1 and p_2 produce varying ξ_{ntot}. Equation (8.12) represents a linear equation in two variables: the margin, given by $(p_1 - p_2)$ and the size of the total margin error, given by $-\xi_{ntot}$. There is a direct link of (8.12) to the transvariation intensity function, introduced in Section 8.4.3. The transvariation intensity function represents the quantity $-\xi_{ntot}$.

8.4.3. Transvariation intensity function

We introduce the transvariation intensity function as the signed transvariation intensity maximum (the denominator in (8.11)) of the classes with respect to the arbitrary pair of D-dimensional points, specifying two locations $\boldsymbol{p}_{i_1}$ and $\boldsymbol{p}_{j_2}$ in the D-dimensional space. For balanced classes $(N_1 = N_2 = \frac{N}{2})$, the D-dimensional vectorial expression of the transvariation intensity function in the original data space is:

$$\boldsymbol{T}_o = \frac{2}{N}\sum_{i=1}^{N_1}\sum_{j=1}^{N_2}((\boldsymbol{x}_i - \boldsymbol{p}_{i_1}) - (\boldsymbol{x}_j - \boldsymbol{p}_{j_2})) = (\sum_{i=1}^{N_1}\boldsymbol{x}_i - \sum_{j=1}^{N_2}\boldsymbol{x}_j) - \frac{N}{2}(\boldsymbol{p}_{i_1} - \boldsymbol{p}_{j_2}) \ . \tag{8.13}$$

The left side of the equality represents the differences between the data points and the selected locations $\boldsymbol{p}_{i_1}$ and $\boldsymbol{p}_{j_2}$. In the linear Support Vector Machine, the optimal discriminant $\boldsymbol{w}$ is determined by a linear combination of the scaled data points:

$$\boldsymbol{w} = \sum_{i=1}^{N_1}\alpha_i\boldsymbol{x}_i - \sum_{j=1}^{N_2}\alpha_j\boldsymbol{x}_j \ . \tag{8.14}$$

By scaling the differences of the left side of (8.13), the transvariation intensity function $\boldsymbol{T}_\alpha$ is expressed in the scaled data space as:

$$\boldsymbol{T}_\alpha = \frac{2}{N}\sum_{i=1}^{N_1}\sum_{j=1}^{N_2}(\alpha_i(\boldsymbol{x}_i - \boldsymbol{p}_{i_1}) - \alpha_j(\boldsymbol{x}_j - \boldsymbol{p}_{j_2})) = (\sum_{i=1}^{N_1}\alpha_i\boldsymbol{x}_i - \sum_{j=1}^{N_2}\alpha_j\boldsymbol{x}_j) - \frac{\sum_{s=1}^{N}\alpha_s}{2}(\boldsymbol{p}_{i_1} - \boldsymbol{p}_{j_2}). \tag{8.15}$$

In the formulation of SVM (Shawe-Taylor and Christianini, 2004), the constraint $\sum_{i=1}^{N_1}\alpha_i - \sum_{j=1}^{N_2}\alpha_j = 0$ allows for factoring out the multiplier $\frac{\sum_{s=1}^{N}\alpha_s}{2}$ from the left hand side of expression (8.15). The same constraint arises in the Liknon dual formulation (8.8). The vectors $\boldsymbol{T}_o$ and $\boldsymbol{T}_\alpha$ become scalars $t = \boldsymbol{T}_o\boldsymbol{w}^T$ and $t_\alpha = \boldsymbol{T}_\alpha\boldsymbol{w}^T$ when the classes are projected onto the discriminant vector $\boldsymbol{w}$. Let us project the right-hand side of (8.13) and (8.15) onto the discriminant $\boldsymbol{w}$ given by (8.14):

$$T_o\boldsymbol{w}^T = (\sum_{i=1}^{N_1}\boldsymbol{x}_i - \sum_{j=1}^{N_2}\boldsymbol{x}_j)\boldsymbol{w}^T - \frac{N}{2}((\boldsymbol{p}_{i_1} - \boldsymbol{p}_{j_2})\boldsymbol{w}^T) \ , \tag{8.16}$$

and in the scaled space:

$$T_\alpha\boldsymbol{w}^T = (\sum_{i=1}^{N_1}\alpha_i\boldsymbol{x}_i - \sum_{j=1}^{N_2}\alpha_j\boldsymbol{x}_j)\boldsymbol{w}^T - \frac{\sum_{s=1}^{N}\alpha_s}{2}((\boldsymbol{p}_{i_1} - \boldsymbol{p}_{j_2})\boldsymbol{w}^T) \ . \tag{8.17}$$

The dot product $d = (\boldsymbol{p}_{i_1} - \boldsymbol{p}_{j_2})\boldsymbol{w}^T$ in (8.16) and (8.17) specifies the margin. For fixed discriminant $\boldsymbol{w}$, the differences between the projected class points $d_w = (\sum_{i=1}^{N_1} \boldsymbol{x}_i - \sum_{j=1}^{N_2} \boldsymbol{x}_j)\boldsymbol{w}^T$ and $d_{wa} = (\sum_{i=1}^{N_1} \alpha_i \boldsymbol{x}_i - \sum_{j=1}^{N_2} \alpha_j \boldsymbol{x}_j)\boldsymbol{w}^T$ are constant. The transvariation intensity functions (8.16) and (8.17) become linear equations in two variables: $t(d) = d_w + (\frac{N}{2})d$ and $t_a(d) = d_{wa} + (\frac{\sum_{s=1}^{N} \alpha_s}{2})d$, if the pair of data points $\boldsymbol{p}_{i_1}$ and $\boldsymbol{p}_{j_2}$ varies. In the projection, the transvariation intensity function indicates how the size of the total margin error (ξ_{ntot} described in Section 8.4.2) varies with the margin, specified by the different pairs of the projected points of the two classes.

8.4.4. Ordering the classes on the discriminant

The projected class points on the discriminant are ordered: class 2 tends to occupy the line segment towards $-\infty$ and class 1 towards $+\infty$. $t(d)$ allows the comparison of the discriminants for fixed d. Larger positive values of $t(d)$ correspond to margins that separate the two classes better. The transvariation intensity function $t(d)$ equals to zero at $d_{t(0)}$, the difference between the class centroids on the discriminant $\boldsymbol{w}$:

$$d_{t(0)} = \frac{2}{N}(\sum_{i=1}^{N_1} \boldsymbol{x}_i - \sum_{j=1}^{N_2} \boldsymbol{x}_j)\boldsymbol{w}^T \quad . \tag{8.18}$$

$t_\alpha(d)$ equals to zero at $d_{t_\alpha(0)}$, which is the difference between the centers of mass of the class points on the wrong sides of the margin on $\boldsymbol{w}$ (in SVM the margin is $[-1, \ +1]$):

$$d_{t_\alpha(0)} = \frac{2}{\sum_{s=1}^{N} \alpha_s}(\ \boldsymbol{w}\boldsymbol{w}^T \ ) \ . \tag{8.19}$$

The expression $\boldsymbol{w}$ in (8.14), when used in (8.19), gives:

$$d_{t_\alpha(0)} = \frac{2}{\sum_{s=1}^{N} \alpha_s}(\sum_{i=1}^{N_1} \alpha_i(\boldsymbol{x}_i\boldsymbol{w}^T) - \sum_{j=1}^{N_2} \alpha_j(\boldsymbol{x}_j\boldsymbol{w}^T)) \ . \tag{8.20}$$

The points outside the margin have zero α coefficients. The remaining points are the ones that determine the class overlap on $\boldsymbol{w}$, by the amount $d_{t_\alpha(0)}$ (8.20). In the following, we will use $\boldsymbol{a}$ when referring to the difference between classes:

$$\boldsymbol{a} = (\sum_{i=1}^{N_1} \boldsymbol{x}_i - \sum_{j=1}^{N_2} \boldsymbol{x}_j) \ . \tag{8.21}$$

The left panel of Figure 8.4 illustrates the linear SVM boundary that separates class 1 (squares) from class 2 (crosses), and the margins. The top panel on the right shows the partial class ordering on the discriminant $\boldsymbol{w}$. The right-bottom panel shows the transvariation intensity functions t and t_α of the class ordering shown on the right-top panel. The transvariation intensity functions illustrate the class separation with respect to the distances between the projected class points. The distances/margins that separate classes better, are characterized by positive values of the transvariation intensity functions. In the top-right panel, the centers of classes and the centers of the overlapping points are marked by stars. The segments - $d_{t(0)}$ and $d_{t_\alpha(0)}$ - are indicated by horizontal lines. The arrows from these lines lead to the bottom panel, where the transvariation intensity functions are depicted. The arrows show the positions of the critical segments $d_{t(0)}$ and $d_{t_\alpha(0)}$ in the bottom-right plot. In this example, the classes overlap. The points best separating the classes are swapped, hence the “margin” is negative and corresponds to the largest value of t and t_α.

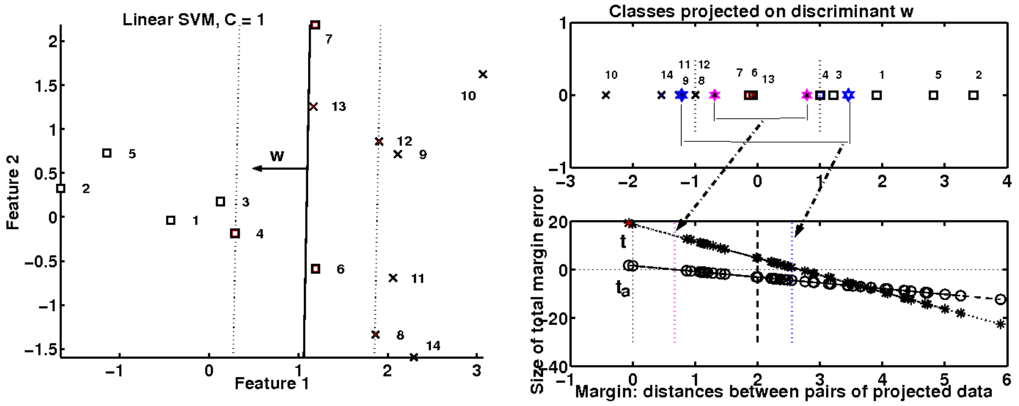


Figure 8.4: Illustration of the concepts related to the transvariation intensity functions. The left panel shows the class separation boundary (solid line) and the margins (dotted lines). The right-top panel shows the ordering of the projected class points (numbered) on the SVM discriminant. The right-bottom panel shows the transvariation functions t and t_α computed for the class ordering of the right-top panel. The asterisks (*) on t and circles (o) on t_α indicate the actual distances $(\boldsymbol{p}_{i_1} - \boldsymbol{p}_{j_2})\boldsymbol{w}^T$ between the points on the right-top panel.

8.4.5. Regularization parameter C

The class overlap (8.20) is always less than the difference between the class centroids (8.18) on the discriminant $\boldsymbol{w}$:

$$\frac{2}{\sum_{s=1}^{N} \alpha_s}(\boldsymbol{w}\boldsymbol{w}^T) \leq \frac{2}{N}(\boldsymbol{a}\boldsymbol{w}^T) \ . \tag{8.22}$$

The coefficients $\alpha_s \leq C, \ \ s = 1, \ldots, N$ are constrained by C in the original SVM formulation. Rearranging (8.22) and noting that for $\alpha_s \leq C$, the mean value $\frac{\sum_{s=1}^{N}\alpha_s}{N}$ is also less than C, we have:

$$\frac{(\boldsymbol{w}\boldsymbol{w}^T)}{(\boldsymbol{a}\boldsymbol{w}^T)} < \frac{\sum_{s=1}^{N}\alpha_s}{N} \leq C \quad \Rightarrow \quad ratio = \frac{\|\boldsymbol{w}\|}{\|\boldsymbol{a}\|\cos(\boldsymbol{w}\widehat{\ }\boldsymbol{a})} \leq C \ . \tag{8.23}$$

The *ratio* in (8.23) explains how the regularization parameter C relates to the class separation. C bounds the ratio of $\|\boldsymbol{w}\|$ over $\|\boldsymbol{a}\|\cos(\boldsymbol{w}\widehat{\ }\boldsymbol{a})$. $\|\boldsymbol{w}\|$ is the length of the normal to the separation boundary, related to the margin between the classes in the original space as $\frac{1}{\|\boldsymbol{w}\|}$. The expression $a_w = \|\boldsymbol{a}\|\cos(\boldsymbol{w}\widehat{\ }\boldsymbol{a})$ means a projection of the class difference (8.21) onto the normal $\boldsymbol{w}$, showing the alignment of the two vectors $\boldsymbol{w}$ and $\boldsymbol{a}$. An increase in class separation is associated with the increase of $\|\boldsymbol{w}\|$ and the decrease of a_w. Thus, the increasing *ratio* indicates better separation of the classes, projected onto the solved discriminant $\boldsymbol{w}$.

Both Svmpath and Liknon produce a sequence of solutions. The sequence of the solutions improving class separation proceeds through the stages of under-fitting, optimal and over-fitting with respect to the classification performance on the independent validation set. These stages can be visualized using the monitoring BER of every solution. For visualization we use a logarithmic transformation of the ratio $\frac{\|\boldsymbol{w}\|}{a_w}$, $lr = \lg_{10}(\frac{\|\boldsymbol{w}\|}{a_w})$. Figure 8.5 illustrates the performance (monitoring BER) of the sequence of the solutions of the full regularization path and Liknon

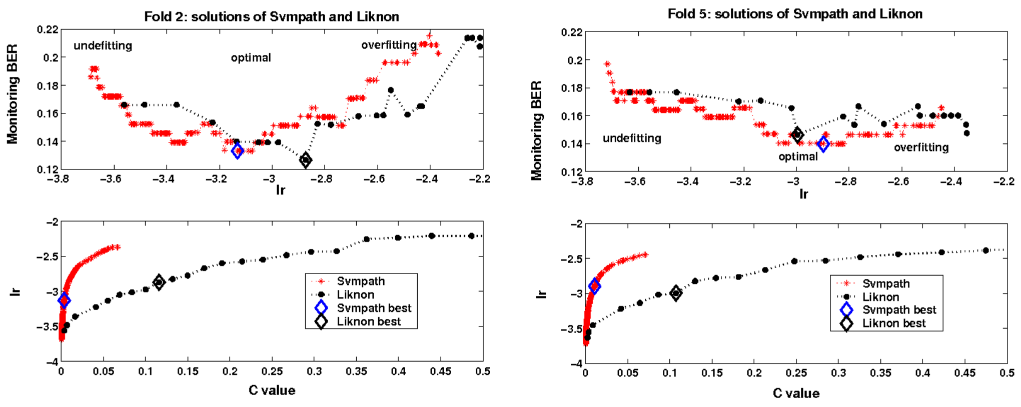


Figure 8.5: Relationship between the performance, the lr and regularization parameter C for the sequence of the solutions of the full regularization path and Liknon. We illustrate the solutions in the folds 2 and 5 of the Banana example. The best solutions are indicated by diamonds. The top panel shows the monitoring BER of every solution versus $lr = \lg_{10}(\frac{\|\boldsymbol{w}\|}{a_w})$. The bottom panel shows the plot of lr versus C.

versus the lr in folds 2 and 5 of the artificial Banana example of Section 8.2. Plots of lr versus C for every solution are shown in the left-bottom (fold 2) and the right-bottom (fold 5) panels. In the Svmpath method, C corresponds to the parameter $1/\lambda$ (Hastie et al., 2004). The solutions, producing minimum monitoring BER are indicated by diamonds. The values of the lr of the optimal solutions of both Liknon and Svmpath are similar and occur in the range of $[-3.2, -2.8]$. The increasing lr starts saturating at about -2.8, where the monitoring BER starts growing. The C values, associated with the optimal solutions of Svmpath and Liknon are not close, but the character of the dependance between the lr and C is similar for both methods. It is possible to obtain several Liknon solutions and use them for approximating the range of the optimal lr for Svmpath. The details are beyond the scope of this chapter. The explained concepts and relation (8.23) provide the basis for the derivation of the constructive algorithm in Liknon for C computation.

8.5. Liknon feature selection

Liknon simultaneously identifies a subset of useful features and a linear discriminant. Discriminants of increasing complexity in terms of large/non-zero weights are associated with small margins between classes. The margin can be decreased by increasing the value of the regularization parameter C. Here we outline the algorithm for the computation of C for Liknon.

8.5.1. The standard discriminant, given by the solution of the Liknon dual

The optimal solution $(^*)$ $\alpha^* = [\alpha_1^*, \ldots, \alpha_N^*]$ of the dual (8.8) satisfies all constraints and the optimality conditions (8.9) and (8.10). Geometrically, it also determines the direction that discriminates the classes, obtained from a linear combination of the data points. According to the optimality conditions, all coordinates of the discriminant (henceforth called the *standard discrim-*

inant) have unit length. For a set of m selected individual features $\boldsymbol{f} = (f_1, \dots, f_m), \quad m \leq D$, corresponding to the binding constraints, the set of equations

$$e^{f_k} = \sum_{i=1}^{N_1} \alpha_i^* x_i^{f_k} - \sum_{j=1}^{N_2} \alpha_j^* x_j^{f_k} = \pm 1 \quad , \tag{8.24}$$

is satisfied. Therefore, the solution of the Liknon dual identifies the standard discriminant in the m-dimensional subspace of the selected features $\boldsymbol{f}$:

$$\boldsymbol{e} = \sum_{i=1}^{N_1} \alpha_i^* \boldsymbol{x}_i - \sum_{j=1}^{N_2} \alpha_j^* \boldsymbol{x}_j, \quad \text{and} \quad \boldsymbol{e} = [\pm 1_1 \dots \pm 1_m], \tag{8.25}$$

which corresponds to (8.14) and can be interpreted as a "preimage" of the $\boldsymbol{w}$, normal to the optimal Liknon hyperplane. The discriminant $\boldsymbol{e}$ has constant length $\sqrt{m}$, determined by the dimensionality of the identified subspace.

8.5.2. Given vs. desired: margin control

The Liknon-selected feature subset of dimensionality m identifies some subspace. In this subspace we have $\boldsymbol{a}$ - the difference between the class vectors and the standard discriminant $\boldsymbol{e}$, given by the solution of the Liknon dual. The distance $\boldsymbol{a}\boldsymbol{e}^T$ between the classes projected onto $\boldsymbol{e}$ depends on the number of non-zero elements of the standard discriminant/number of selected features. The ratio of class overlap to the distance between classes, (8.23), in Liknon gives:

$$\frac{(\mathbf{e}\mathbf{e}^T)}{(\mathbf{a}\mathbf{e}^T)} < \frac{\sum_{s=1}^{N} \alpha_s}{N} \leq C \quad . \tag{8.26}$$

Thus, we can explain and visualize how the value of the parameter C controls the class separation by the distance, and influences the selection of the subspaces:

$$\frac{(\boldsymbol{a}\boldsymbol{e}^T)}{m} > \frac{1}{C} \quad .$$

The distance between the classes on the standard discriminant, determined by the solution of Liknon at a specific value of C, will be greater than $\frac{1}{C}$. We don't know in advance the standard discriminant, and apply (8.26) to the individual features to determine C.

8.5.3. Selection of C

Projection of the values of individual features f_k onto their respective elements of the standard discriminant $x_s^{f_k} e^{f_k}$ is merely a multiplication of $x_s^{f_k}$ by 1 or -1. The difference $d_{i,j}^{f_k} = (x_i^{f_k} - x_j^{f_k}) e^{f_k}$ of the feature values in the two classes specifies the margin. a^{f_k} is an element of (8.21). The transvariation intensity functions t and t_α can be written as:

$$t^{f_k}(d_{i,j}^{f_k}) = a^{f_k} e^{f_k} - (\tfrac{N}{2}) d_{i,j}^{f_k} \quad , \quad t_\alpha(d_{i,j}^{f_k}) = e^{f_k} e^{f_k} - (\tfrac{\sum_{s=1}^{N} \alpha_s}{2}) d_{i,j}^{f_k} \quad . \tag{8.27}$$

According to (8.18) and (8.19), and noting that $e^{f_k} e^{f_k} = 1$, the margins/segments for which t^{f_k} and t_α attain zero are:

$$d_{t^{f_k}(0)} = \tfrac{2}{N}(a^{f_k} e^{f_k}), \quad d_{t_\alpha(0)} = \frac{2}{\sum_{s=1}^{N} \alpha_s} \quad . \tag{8.28}$$

Relation (8.26) for an individual feature gives:

$$\frac{1}{(a^{f_k}e^{f_k})} < \frac{\sum_{s=1}^{N}\alpha_s}{N} \quad . \tag{8.29}$$

The sign of $a^{f_k}e^{f_k}$ can be ignored, because it merely indicates swapped classes. C influences the selection of the individual features by bounding the class difference:

$$|a^{f_k}| = |\sum_{i=1}^{N_1} x_i^{f_k} - \sum_{j=1}^{N_2} x_j^{f_k}| \;>\; \frac{1}{C} \quad , \tag{8.30}$$

The individual features, with a difference between classes greater than $\frac{1}{C}$, are candidates to be included in the Liknon discriminant. By arranging class differences of the individual features in descending order, we obtain the relationship $|a^{f_k}_{\max}| \geq \ldots \geq |a^{f_k}_{\min}|$. Our first approach to selecting C was setting the parameter equal to the inverse of every member of the sequence, $\frac{1}{|a^{f_k}_{\max}|} \leq \ldots \leq \frac{1}{|a^{f_k}_{\min}|}$, and solving a sequence of Liknon models (primal) with the sequence of C values so determined. However, if the number of features is large, as in microarray data, the computations become prohibitive. An algorithm for selecting fewer C values is necessary. Our second approach was to compute a histogram and select the C using the modes of the histogram. (A limitation of the histogram approach is the selection of optimal bin.)

8.5.4. Algorithm for computing C

The algorithm for computing a subset of C values is based on the transvariation intensity functions F_{trans} of the individual features. The F_{trans} are modeled by the linear equations $t^{f_k} = |a^{f_k}| - (\frac{N}{2})d$, with slopes $\frac{N}{2}$ and intercepts $|a^{f_k}|$. The variable d accounts for the distances $(x_i^{f_k} - x_j^{f_k})e^{f_k}$. The standard F_{trans}, $t_\alpha = 1 - (\frac{\sum_{l=1}^{N}\alpha_l}{2})d$ has slope $\frac{\sum_{l=1}^{N}\alpha_l}{2}$ and intercept 1. In general, the F_{trans} of the features that separate classes better are larger and they attain zero at larger d. The F_{trans} for which distances/margins separate classes better have positive values. Given the features, the increase in class separation as a function of all features for all distances d can be determined by summing the positive parts of the t^{f_k} . We call this measure t_{total}. It increases piecewise linearly as the distance d decreases. The set of NM (number of models) values of t_{total} is used for the determination of the set of distances d_s, $s = 1,\ldots,NM$, assuming the class differences $|a^s| = (\frac{N}{2})d_s$. The C_s value associated with the d_s is computed using (8.30), as $C_s = \frac{2}{Nd_s}$. Setting the C value to C_s means that we constrained the absolute class difference of the selected features to be greater than $\frac{1}{C_s}$. The smaller the value of d_s, the larger C_s, the more features are included in the Liknon discriminant.

Figure 8.6 illustrates the idea of the outlined algorithm for an artificial dataset with $D = 10$ features and $N = 100$ samples. Two features are discriminatory, the rest are fully overlapping $N(0,1)$-distributed features. The F_{trans} values of the discriminatory features are larger and attain zero at a larger d.

There are segments where t_{total} changes slowly. The more functions become positive and add to the total, the more t_{total} changes. The potentially interesting segments occur where the F_{trans} concentrate. A Matlab code for computing the range of C values is listed in the Appendix.

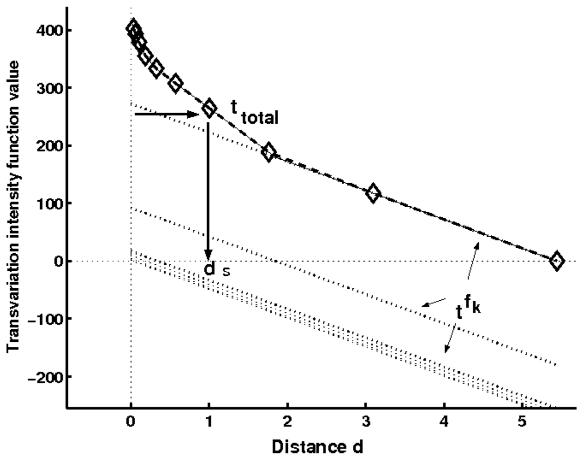


Figure 8.6: Computation of C using t_{total}. The transvariation intensity functions t^{f_k} are shown by the dotted lines. The dashed line depicts t_{total}. The diamonds indicate the positions and values of the t_{total} for which the distance d_s is determined.

8.6. Liknon feature and classification model selection on the benchmark datasets

Here we present a realistic single-run assessment of Liknon-based feature selection with respect to other feature selection methods in the NIPS 2003 FS challenge. We also present our results for the ALvsPK competition. In the ALvsPK challenge, the Liknon-based classification model selection strategy was among the top-ranked methods.

8.6.1. Liknon feature selection applied to the NIPS 2003 FS benchmark datasets

Information on the NIPS 2003 FS benchmark is on the website `www.nipsfsc.ecs.soton.ac.uk` and in publications (Guyon et al., 2006, 2007b). In the benchmark, the ground truth for features - *useful* or *probe* - is available. Probe is a “fake” feature purposefully inserted into the dataset. We tested Liknon’s utility as a filter and a wrapper in a single run, including univariate feature prefiltering. The class difference (8.21) provides information about the class separation. In prefiltering we rank the individual features by the decreasing absolute value of $|a^{f_k}|$ computed for every feature and discarding a percentage of low-$|a^{f_k}|$ value features. Liknon is solved for the remainder. The ranking of the features by (8.21) to the ranking by difference in means can be compared by the fraction of the disagreeing ranks. The percentage of disagreeing ranks of the features ordered by (8.21) and by the difference between the means of the NIPS 2003 FS training data is: ARCENE - 66.90%; DEXTER - 24.61%; DOROTHEA - 59.48%; GISETTE - 3.28%; MADELON - 5.20%. The percentage of the discarded features and C were determined in several repetitions of 5-fold crossvalidation, with a single split in the inner loop. The determined parameters were used in a `final single` Liknon run on the Training set. For DOROTHEA and GISETTE, a smaller, random, balanced training sample was taken. For the Liknon wrapper, the same discriminant was used as the final classifier. For the filter, the Liknon identified feature subset was input to 3-nearest-neighbor (3nn) and subspace (subsp) classifiers. The final model - 3nn or subsp - was the one with the smaller Validation BER. Table 8.2 summarizes the experimental situation.

Table 8.2: Experimental setup for the NIPS 2003 FS benchmark.

Dataset Data origin	ARCENE mass spectra	GISETTE handwritten digits	DEXTER text	DOROTHEA drug discovery	MADELON difficult artificial
Dimensionality	10000	5000	20000	100000	500
Total probes(%)	30	50	50.3	50	96
Train	44 + 56	3000 + 3000	150 + 150	78 + 722	1000 + 1000
Validation	44 + 56	500 + 500	150 + 150	34 + 316	300 + 300
Test	310 + 390	3250 + 3250	1000 + 1000	78 + 722	600 + 600
Discarded(%)	86	95	75	99	98
C	0.0058	0.00066	0.0035	0.2369	0.016
Classifier	3nn	3nn	subsp	subsp	3nn

Table 8.3 presents the performance of the Liknon wrapper and filter with respect to the methods in the NIPS 2003 FS benchmark that performed feature selection. Mean and median test BERs of the latter allow ranking the Liknon-based method with respect to other methods. We also present the best entries, published recently (Guyon et al., 2007b). Both Liknon filter and wrapper compare favorably with the average performance of the benchmark methods, but are worse than the recent best entries. Our results indicate better performance of Liknon as a filter than as a wrapper. Liknon features, used with 3nn and subsp classifiers performed better on all datasets, except for DEXTER.

Table 8.3: Performance comparison. Test BERs

Dataset	Mean	Median	Liknon filter	Liknon wrapper	Best recent entry
ARCENE	0.2396	0.2087	0.1711	0.1962	0.1048
DEXTER	0.1170	0.0690	0.0820	0.0820	0.0325
DOROTHEA	0.2054	0.1661	0.1961	0.2011	0.0930
GISETTE	0.0823	0.0246	0.0402	0.0742	0.0111
MADELON	0.1922	0.1245	0.1133	0.3789	0.0622

Liknon was robust in detecting the ground truth. Our fraction of features and probes and those of the best recent entries are listed in Table 8.4. The fraction of features F_{feat} is the ratio of the number of the selected features to the total number of features. The fraction of probes F_{probe} is the ratio of the probes in the selected subset to the total number of selected features. The Liknon feature subsets contain fewer features and probes than the best recent entries. Useful features were consistently identified in crossvalidation and in the feature subset obtained in the final training.

The presented computational experiment highlights the potential of a simple single Liknon application to the data. First, by univariate feature filtering, using class difference (8.21) as the criterion, most of the probes were discarded. Second, an even smaller feature subset, optimal for linear separation, was obtained by solving Liknon. It is known, that some NIPS 2003 FS

Table 8.4: Fraction of features F_{feat} and probes F_{probe}

Dataset	Our F_{feat}	Our F_{probe}	Best recent F_{feat}	Best recent F_{probe}
ARCENE	0.0041	0.00	0.14	0.04
DEXTER	0.0079	0.095	0.23	0.55
DOROTHEA	$7*(10^{-5})$	0.00	0.01	0.03
GISETTE	0.0112	0.0179	0.20	0.00
MADELON	0.02	0.00	0.04	0.00

benchmark datasets have nonlinear character. Linear classifiers such as Liknon are not optimal in such scenario. The poorer performance of the Liknon wrapper compared to the filter corroborates this fact. If the number of features is larger than the number of samples, then the number of non-zero weights in the Liknon discriminant is bounded by the number of samples, guaranteed by the nature of linear programming formulation and binding constraints. The small number of features, identified in a single Liknon run, might not be sufficient to fully capture most of the information about class separation. The feature profile, accumulated in many Liknon runs on many data splits, may overcome this limitation. Such strategy was used in the ALvsPK competition and it ranked Liknon among the top-ranked methods.

8.6.2. Liknon versus Svmpath on the NIPS2003 feature selection benchmark datasets

In this subsection the Liknon filter is contrasted with the related method of S. Rosset and J. Zhu (Rosset and Zhu, 2006) on NIPS2003 FS benchmark datasets. The method included a univariate filtering by the t-statistic and computation of the principal components (PCA) as features after prefiltering for the ARCENE and DOROTHEA datasets; Then, a regularized optimization scheme using various combinations of a loss function L and a penalty J was applied as follows.

- ARCENE: L was the Huberized hinge loss, and J was the L_2-norm penalty (linear support vector machine).
- DEXTER: L was the Huberized hinge loss, and J was the L_1-norm penalty (L_1 norm support vector machine).
- DOROTHEA: L was the Huberized hinge loss; J was the L_1-norm penalty (L_1 norm support vector machine).
- GISETTE: L was the exponential loss; J was the L_1-norm penalty.
- MADELON: L was the hinge loss, and J was the L_2-norm penalty (radial basis kernel support vector machine).

All hyper-parameters (including the number of features) were selected using 5-fold cross-validation. Liknon filter was applied to all datasets, using the same unified scheme explained in the previous subsection. The test BERs, area under the curve, AUC, number of features and probes of both approaches are summarized and contrasted in Table 8.5. These results are also available on the website `www.nipsfsc.ecs.soton.ac.uk`.

The method of S.Rosset and J.Zhu, applied to DEXTER and DOROTHEA, is comparable to the Liknon filter, since it also implements a linear SVM with L_1-norm penalty. If we compare the performances by BER and AUC, then the results of S.Rosset and J.Zhu are better on these datasets. If we compare the size and purity of the feature subsets in DEXTER and DOROTHEA,

Table 8.5: Results of S.Rosset and J.Zhu compared to Liknon-based feature/classification model selection in the NIPS2003 feature selection benchmark.

Method	Saharon Rosset and Ji Zhu			Liknon filter		
Dataset	BER	AUC	Features(Probes)	BER	AUC	Features(Probes)
ARCENE	0.1962	0.8038	3000(171)	0.1711	0.8908	41(0)
DEXTER	0.0690	0.9628	112(50)	0.0820	0.9511	158(15)
DOROTHEA	0.1569	0.8451	5207(4009)	0.1996	0.8053	7(0)
GISETTE	0.0134	0.9826	1500(0)	0.0402	0.9791	56(1)
MADELON	0.0906	0.9094	21(2)	0.1133	0.9369	10(0)

then the Liknon filter is better. In the NIPS2003 FS challenge, several methods performed better on the feature sets containing many probes. For datasets with limited number of samples, e.g., DEXTER, many fake features transform the class separation character and we actually have a different classification problem. On this feature-augmented problem some methods do better than on the original problem without probes. In general, a fair comparison is only possible if we had the reference BERs of several methods - linear and nonlinear - using a) only relevant features and b) all features. Such experiment is beyond the scope of this chapter.

8.6.3. Numerical experiments on the datasets of Agnostic Learning vs. Prior Knowledge competition

The Liknon-based classification model participated in the agnostic track of the ALvsPK challenge as a "black box", using five datasets ADA, GINA, HIVA, NOVA and SYLVA. 10-fold crossvalidation with 31 splits in the inner loop and without feature prefiltering was used to obtain an ensemble (ens) of 31 Liknon discriminants, and the profile of the identified relevant features. From the profile, a subset of features occurring more frequently than some threshold, (the thresholds were $10\%, 15\%, 20\%, \ldots, 95\%, 100\%$), was selected and used in training the following classifiers from PRTools (Duin et al., 2004): fisher linear discriminant, logistic linear classifier, quadratic classifier, subspace classifier, and 1- and 3-nearest-neighbor classifiers. On four of the five datasets, the ensembles of Liknon discriminants performed better than other rules. The dimensionality of the datasets, the sizes of the subdivisions of training data (explained in Section 8.2), the optimal threshold for the feature profile, the number of the tested Liknon models NM, and the winning classifier are summarized in Table 8.6.

Our results in the ALvsPK competition are presented in Table 8.7. We report our balanced test error rate (BER), the area under the receiver operating curve (AUC), and the same information for the winning entries. The ensemble of Liknon discriminants was superior for ADA, HIVA, NOVA and SYLVA. For GINA, the 3-nearest-neighbor rule, trained on the most prominent features, performed best. In Section 8.2, we analyzed a dataset, for which two features separate the classes nonlinearly. Similarly, in GINA, which is too complex for a linear classifier, Liknon identified features that represented the nature of class separation sufficiently well for a nonlinear rule. For SYLVA, among the individual rules, the quadratic discriminant performed comparably to the ensemble. The histogram-based approach of C selection worked best for HIVA. For the remaining datasets, the C selection algorithm presented in Section 8.5.4 gave the top-ranked results. All details of our results in the ALvsPK challenge have already been published elsewhere (Pranckeviciene et al., 2007).

Table 8.6: Setup and parameters of the ALvsPK challenge datasets

Dataset Data origin	ADA marketing	GINA handwritten digits	HIVA drug discovery	NOVA text	SYLVA ecology
Dimensionality	48	970	1617	16969	216
Threshold	55%	50%	20%	80%	20%
Training	600+600	350+350	100+100	400+400	400+400
Monitoring	2487+419	1237+1184	3572+34	842+94	11758+397
Validation	343+113	176+171	408+15	138+55	1351+89
NM	5	8	20	25	10
Winning classifier	ens	3nn	ens	ens	ens

In Table 8.7, for GINA, NOVA and SYLVA, our AUC is slightly better than those of the best entries. This arises because AUC accounts for the full range of operating points, whereas BER is based on a single confusion matrix and represents a single operating point of the classifier on the ROC curve. A larger value of AUC suggests, that the classifier may have improved classification performance when the prevalence (proportions) of the test samples in the two classes is different from the proportion given in the challenge datasets.

Table 8.7: Results on the ALvsPK challenge datasets: our method and the best entry in the agnostic track

Dataset	Our Test BER	Our Test AUC	Best Test BER	Best Test AUC
ADA	0.1818	0.8702	0.1660	0.9168
GINA	0.0533	0.9740	0.0339	0.9668
HIVA	0.2939	0.7589	0.2827	0.7707
NOVA	0.0725	0.9814	0.0456	0.9552
SYLVA	0.0190	0.9949	0.0062	0.9938

8.7. Discussion and Conclusions

Inspired by the work of Montanari (Montanari, 2004), we introduced a novel and key concept, the transvariation intensity function. It characterizes the univariate separation of two classes as a linear function of the distances between points from the classes. For the original and scaled data, projected onto some discriminant, it specifies the two distances important for class separation: the size of the class difference (8.18) and the size of class overlap (8.20). Their ratio elucidates the role of the regularization parameter as the controller of the class overlap in SVM and Liknon. The relationship between the regularization parameter C, the class difference and the class overlap provided a better understanding of how Liknon works, important in practical applications of the method. In Liknon, the class difference on the standard discriminant and individual selected features is constrained from below by $1/C$. By increasing the parameter C, we control the class overlap in individual features, allowing more features to be included into the optimal Liknon discriminant. Based on the total transvariation intensity function of the individual features, we proposed an algorithm for computing C values.

We demonstrated the efficiency of Liknon, combined with univariate feature filtering, in identifying smaller feature subsets of the ground truth features. Because of the properties of the linear programming method of Liknon, the number of non-zero components (selected features) in the solution of the primal $\boldsymbol{w}$ is bounded by the number of samples in the dataset. Liknon's limited, single-run performance, when compared to the best recent entries for the NIPS 2003 FS datasets, indicates that either the number of selected features in a single split is not sufficient for fully representing the nature of the class separation, or that the linear classifier cannot handle the complexity of the classification problem. As possible strategies, we suggest using an ensemble of Liknon discriminants trained on feature subsets of different sizes, or employing Liknon features as inputs to nonlinear classification rules. In the ALvsPK challenge, results based on these strategies were among those of the top-ranked methods. The main advantage of Liknon is its ability to identify stable features in high-dimensional, small sample size datasets. A possible disadvantage is the considerable computational load of processing many data splits in the inner crossvalidation loop.

Related to our work is the full regularization path algorithm (Hastie et al., 2004; Rosset and Zhu, 2006). It uses loss functions with quadratic terms and L_1-norm penalty. It allows computation of derivatives in order to obtain the sequence of solutions $\boldsymbol{w}$. The algorithm does not calculate the regularization parameter explicitly. Liknon has different formulation in terms of the loss function. Loss in Liknon is linear and the partial derivatives with respect to $\boldsymbol{w}$ would be constant. The linearity of Liknon enables formulating it as a linear programming problem and solving for $\boldsymbol{w}$ using the LP optimization method. Using such formulation we can derive how to compute the regularization parameter, but we can't compute the weights of the discriminant directly. Solving Liknon with a sequence of Cs and using a monitoring set to select the optimal solution $\boldsymbol{w}$ is similar to the procedure for the regularization path approach, but Liknon uses a **linear loss** and an L_1 penalty. These two algorithms obtain the sequence of the solutions in different ways.

We carried out numerical experiments on the artificial Banana example to compare the performances of the two methods. We also contrasted the Liknon filter with the relevant entry of the NIPS2003 FS challenge, the regularization path method used by S.Rosset and J. Zhu; it performed better than the Liknon filter. However, from the point of view of the purity and size of the selected feature subsets, Liknon was better. The regularization path method is more flexible than Liknon in learning nonlinear structures through the use of various loss functions. Liknon is "conservative" in adequately capturing the complexity of class separation, identifying only the subsets of interacting features that are optimal for linear class separation. Liknon-embedded feature selection is suited for problems for which user is interested in finding several important original features, the class differences arise due to the differences in class means, there are many correlated features and finally, the number of features is much larger than the number of the training samples.

Acknowledgments

We thank the reviewers and editors for constructive comments that helped us improve the manuscript. The first author thanks the organizers of Agnostic Learning vs. Prior Knowledge challenge for their effort in providing an excellent platform to test learning methods. The group of Prof. R.P.W. Duin is also acknowledged for making PRTools publicly available.

Appendix.

The Matlab function for computation of a sequence of NM values of *C* is presented in this appendix. The code implements the algorithm presented in Section 8.5.4. Given a data matrix `X` and a vector of class labels `Y`, the vector `a` of class difference is computed. A grid of `d` values is constructed out of the vector `a`. Linear transvariation functions for every feature `j` are computed as `Ftrans=[abs(a(j))-(N/2)*d]`. `Ttotal` is obtained by summing up the positive parts of the transvariation functions `Ftrans`. The logarithm of `Ttotal` is used to obtain an equally spaced grid. Using such a grid, the sequence `ds` is obtained by linear interpolation. Using the sequence `ds`, the corresponding `C` values are computed as `C=[2./(N*ds)]`. A large `C=100` is also included in the sequence.

```
function [c]=computeC(X, Y, NM)
%
% Output: c - sequence of C values
% Input: X - data matrix,  Y - class labels,  NM - number of C values
%
N = size(X,1); Na = 100;
i1 = find(Y==1); i2 = find(Y==-1);
a = sum(X(i1,:))-sum(X(i2,:));
da = (2/N)*abs(a);
d = unique([da[0:max(da)/Na:max(da)]]);
Ttotal = zeros(1,length(d));
for j = 1:length(a)
   Ftrans = [abs(a(j))-(N/2)*d];
   m = [Ftrans>0];
   Ttotal = Ttotal+Ftrans.*m;
end;
tm = max(Ttotal);tt = abs(Ttotal-tm);zeroi = find(tt==0);
if ~isempty(zeroi); Ttotal(zeroi) = []; tt(zeroi) = []; d(zeroi) = []; e
tlog = log(tt); [val,ind] = unique(tlog);
tlog = tlog(ind); d = d(ind);
t1 = min(tlog); t2 = max(tlog);
ti = [t1+(t2-t1)/NM:(t2-t1)/NM:t2];
ds = interp1(tlog,d,ti,'linear');
c = sort([2./(N*ds) 100]);
ts = tm-exp(ti);
plot(d,Ttotal,'k',ds,ts,'kd-');
return;
```

References

C.Ambroise and G.J.McLachlan. Selection bias in gene extraction on the basis of microarray gene-expression data. *PNAS*, 99(10), pages 6562-6566, 2002.

T.S. Arthanari and Y. Dodge. *Mathematical programming in statistics*. John Willey and Sons, New York, 1981.

C.Bhattacharyya, L.R.Grate, A.Rizki, D. Radisky, F.J. Molina, M.I. Jordan, M.J. Bissell and I.S. Mian. Simultaneous relevant feature identification and classification in high-dimensional spaces: application to molecular profiling data. *Signal Processing*, 83(4), pages 729-743, 2003.

V.Cherkassky and Y.Ma, Margin-based Learning and Popper's Philosophy of Inductive Learning. In M. Basu and T.K. Ho, editors. *Data complexity in pattern recognition.*. 2006.

R.P.W.Duin, P.Juszczak, P.Paclik, E.Pekalska, D.de Ridder, and D.M.J.Tax. *PRTools4 A Matlab toolbox for pattern recognition*, February, 2004.

G.Fung and O.Mangasarian. A feature selection Newton method for support vector machine classification. *Computational Optimization and Applications*, 28, pages 185–202, 2004.

G.D.Guo and C.Dyer. Learning from examples in the small sample case: face expression recognition. *IEEE Trans. on System, Man and Cybernetics - Part B*, 35(3), pages 477–488, 2005.

I.Guyon, V.Vapnik, B.Boser, L.Bottou, and S.Solla. Capacity control in linear classifiers for pattern recognition. *Proceedings of the 11th IAPR International Conference on Pattern Recognition*, IEEE, 2, pages 385–388, 1992.

I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, editors. *Feature extraction, foundations and applications*. Physica-Verlag, Springer, 2006.

I.Guyon, A.Safari, G.Dror, and G.Cawley. Agnostic learning vs. prior knowledge challenge. *Proccedings of International Joint Conference on Neural Networks IJCNN2007*, INNS/IEEE, Orlando Florida, pages 829–834, 2007a.

I.Guyon, J.Li, T.Mader, P.A.Pletsher, G.Schneider, and M.Uhr. Competitive baseline methods set new standards for the NIPS 2003 feature selection benchmark. *Pattern recognition letters*, 28, pages 1438–1444, 2007b.

T.Hastie, S.Rosset, R.Tibshirani, and J.Zhu. The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, 5, pages 1391-1415, 2004.

C.Igel. Multi-objective model selection for support vector machines. In C.A. Coello Coello, A. Hernandez Aguirre, and E. Zitzler, editors. *Evolutionary Multi-criterion Optimization, LNCS* 3410, pages 534–546, 2005.

R.Kohavi and G.H.John. Wrappers for feature subset selection. *Artificial intelligence*, 97(1-2), pages 273–324, 1997.

M.Kudo and J.Sklansky. Comparison of algorithms that select features for pattern classifiers. *Pattern recognition*, 33(1), pages 25–41, 2000.

A.Montanari. Linear discriminant analysis and transvariation. *Journal of Classification*, 21, pages 71–88, 2004.

C.H. Papadimitriou and K. Steiglitz. *Combinatorial optimization Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

E.Pranckeviciene, R.Somorjai, R.Baumgartner, and M.Jeon. Identification of signatures in biomedical spectra using domain knowledge. *AI in Medicine*, 35(3), pages 215–226, 2005.

E.Pranckeviciene, R.Somorjai, and M.N.Tran. Feature/model selection by the Linear Programming SVM combined with State-of-art classifiers: what can we learn about the data. *Proccedings of International Joint Conference on Neural Networks IJCNN2007*, INNS/IEEE, Orlando, Florida, pages 1627–1632, 2007.

S.Rosset and J.Zhu. Sparse, flexible and efficient modelling using L_1 regularization. In I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, editors. *Feature extraction, foundations and applications*, pages 379–398, 2006.

J.Shawe-Taylor and N.Chistianini. *Kernel methods for Pattern Analysis*. Cambridge University Press, 2004.

B.Scholkopf, R.C.Williamson, and P.Bartlet. New Support Vector Algorithms. *Neural Computation*, 12, pages 1207–1245, 2000.

R.L.Somorjai, B.Dolenko, and M.Mandelzweig. Direct classification of high-dimensional data in low-dimensional feature spaces- comparison of several classification methodologies. *Journal of Biomedical Informatics*, 40, pages 131–138, 2007.

W.Zucchini. An introduction to model selection. *Journal of mathematical psychology*, 44, pages 41–61, 2000.

Journal of Machine Learning Research 9(Oct):2377–2400, 2008 Submitted 8/07; Revised 6/08; Published 10/08

Chapter 9

Model Selection in Kernel Based Regression using the Influence Function

Michiel Debruyne MICHIEL.DEBRUYNE@DEXIA.COM
Mia Hubert MIA.HUBERT@WIS.KULEUVEN.BE
Department of Mathematics - LStat
K.U.Leuven
Celestijnenlaan 200B, B-3001 Leuven, Belgium

Johan A. K. Suykens JOHAN.SUYKENS@ESAT.KULEUVEN.BE
ESAT-SCD/SISTA
K.U.Leuven
Kasteelpark Arenberg 10, B-3001 Leuven, Belgium

Editor: Isabelle Guyon

Abstract

Recent results about the robustness of kernel methods involve the analysis of influence functions. By definition the influence function is closely related to leave-one-out criteria. In statistical learning, the latter is often used to assess the generalization of a method. In statistics, the influence function is used in a similar way to analyze the statistical efficiency of a method. Links between both worlds are explored. The influence function is related to the first term of a Taylor expansion. Higher order influence functions are calculated. A recursive relation between these terms is found characterizing the full Taylor expansion. It is shown how to evaluate influence functions at a specific sample distribution to obtain an approximation of the leave-one-out error. A specific implementation is proposed using a L_1 loss in the selection of the hyperparameters and a Huber loss in the estimation procedure. The parameter in the Huber loss controlling the degree of robustness is optimized as well. The resulting procedure gives good results, even when outliers are present in the data.

Keywords: kernel based regression, robustness, stability, influence function, model selection

9.1. Introduction

Quantifying the effect of small distributional changes on the resulting estimator is a crucial analysis on many levels. A simple example is leave-one-out which changes the sample distribution slightly by deleting one observation. This leave-one-out error plays a vital role for example in model selection (Wahba, 1990) and in assessing the generalization ability (Poggio et al. 2004 through the concept of stability). Most of these analyses however are restricted to the sample distribution and the addition/deletion of some data points from this sample.

In the field of robust statistics the influence function was introduced in order to analyze the effects of outliers on an estimator. This influence function is defined for continuous distributions that are slightly perturbed by adding a small amount of probability mass at a certain place. In Section 9.2 some general aspects about the influence function are gathered. Recent results about influence functions in kernel methods include those of Christmann and Steinwart (2004, 2007) for classification and regression. In Section 9.3 these results are stated and their importance is summarized. A new theoretical result concerning higher order influence functions is presented. In Section 9.4 we show how to evaluate the resulting expressions at sample distributions. Moreover we apply these influence functions in a Taylor expansion approximating the leave-one-out error. In Section 9.5 we use the approximation with influence functions to select the hyperparameters. A specific implementation is proposed to obtain robustness with a Huber loss function in the estimation step and a L_1 loss in the model selection step. The degree of robustness is controlled by a parameter that can be chosen in a data driven way as well. Everything is illustrated on a toy example and some experiments in Section 9.6.

9.2. The Influence Function

In statistics it is often assumed that a sample of data points is observed, all generated independently from the same distribution and some underlying process, but sometimes this is not sufficient. In many applications gathering the observations is quite complex, and many errors or subtle changes can occur when obtaining data. Robust statistics is a branch of statistics that deals with the detection and neutralization of such outlying observations. Roughly speaking a method is called robust if it produces similar results as the majority of observations indicates, no matter how a minority of other observations is placed. A crucial analysis in robust statistics is the behavior of a functional T, not only at the distribution of interest P, but in an entire neighborhood of distributions around P. The influence function measures this behavior. In this section we recall its definition and discuss some links with other concepts.

9.2.1. Definition

The pioneering work of Hampel et al. (1986) and Huber (1981) considers distributions $P_{\varepsilon,z} = (1-\varepsilon)P + \varepsilon\Delta_z$ where Δ_z denotes the Dirac distribution in the point $z \in \mathscr{X} \times \mathscr{Y}$, representing the contaminated part of the data. For having a robust T, $T(P_{\varepsilon,z})$ should not be too far away from $T(P)$ for any possible z and any small ε. The limiting case of $\varepsilon \downarrow 0$ is comprised in the concept of the influence function.

Definition 9.1 *Let P be a distribution. Let T be a functional $T : P \to T(P)$. Then the influence function of T at P in the point z is defined as*

$$IF(z;T,P) = \lim_{\varepsilon\to 0} \frac{T(P_{\varepsilon,z}) - T(P)}{\varepsilon}.$$

The influence function measures the effect on the estimator T when adding an infinitesimally small amount of contamination at the point z. Therefore it is a measure of the robustness of T. Of particular importance is the supremum over z. If this is unbounded, then an infinitesimally small amount of contamination can cause arbitrary large changes. For robust estimators, the supremum of its influence function should be bounded. Then small amounts of contamination cannot completely change the estimate and a certain degree of robustness is indeed present. The simplest example is the estimation of the location of a univariate distribution with density f symmetric around 0. The influence function of the mean at $z \in \mathbb{R}$ then equals the function z and is clearly unbounded. If the median of the underlying distribution is uniquely defined, that is if $f(0) > 0$, then the influence function of the median equals $\text{sign}(z)/(2f(0))$ which is bounded. The median is thus more robust than the mean.

9.2.2. Asymptotic Variance and Stability

From Definition 9.1 one can see that the influence function is a first order derivative of $T(P_{\varepsilon,z})$ at $\varepsilon = 0$. Higher order influence functions can be defined too:

Definition 9.2 *Let P be a distribution. Let T be a functional $T : P \to T(P)$. Then the k-th order influence function of T at P in the point z is defined as*

$$IF_k(z;T,P) = \frac{\partial}{\partial^k \varepsilon} T(P_{\varepsilon,z})|_{\varepsilon=0}.$$

If all influence functions exist then the following Taylor expansion holds:

$$T(P_{\varepsilon,z}) = T(P) + \varepsilon IF(z;T,P) + \frac{\varepsilon^2}{2!} IF_2(z;T,P) + \dots \tag{9.1}$$

characterizing the estimate at a contaminated distribution in terms of the estimate at the original distribution and the influence functions.

Actually this is a special case of a more general Von Mises expansion (take $Q = P_{\varepsilon,z}$):

$$T(Q) = T(P) + \int IF(x;T,P)d(Q-P)(x) + \dots$$

Now take Q equal to a sample distribution P_n of a sample $\{z_i\}$ of size n generated i.i.d. from P. Then

$$\begin{aligned} T(P_n) - T(P) &= \int IF(z;T,P)dP_n(z) + \dots \\ &= \frac{1}{n}\sum_{i=1}^{n} IF(z_i;T,P) + \dots . \end{aligned}$$

The first term on the right hand side is now a sum of n i.i.d. random variables. If the remaining terms are asymptotically negligible, the central limit theorem thus immediately shows that $\sqrt{n}(T(P_n) - T(P))$ is asymptotically normal with mean 0 and variance

$$ASV(T,P) = \int IF^2(z;T,P)dP(z).$$

Since the asymptotic efficiency of an estimator is proportional to the reciprocal of the asymptotic variance, the integrated squared influence function should be as small as possible to achieve high efficiency. Consider again the estimation of the center of a univariate distribution with density f. At a standard normal distribution the asymptotic variance of the mean equals $\int z^2 dP(z) = 1$, and that of the median equals $\int (\text{sign}(z)/(2f(0)))^2 dP(z) = 1.571$. Thus the mean is more efficient than the median at a normal distribution. However, at a Cauchy distribution for instance, this is completely different: the ASV of the median equals 2.47, but for the mean it is infinite since the second moment of a Cauchy distribution does not exist. Thus to estimate the center of a Cauchy, the median is a much better choice than the mean.

An interesting parallel can be drawn towards the concept of stability in learning theory. Several measures of stability were recently proposed in the literature. The leave-one-out error often plays a vital role, for example in hypothesis stability (Bousquet and Elisseeff, 2001), partial stability (Kutin and Niyogi, 2002) and CV$_{loo}$-stability (Poggio et al., 2004). The basic idea is that the result of a learning map T on a full sample should not be very different from the result obtained when removing only one observation. More precisely, let P be a distribution on a set $\mathscr{X} \times \mathscr{Y}$ and $T : P \to T(P)$ with $T(P) : \mathscr{X} \to \mathscr{Y} : x \to T(P)(x)$. Let P_n^{-i} denote the empirical distribution of a sample without the ith observation $z_i = (x_i, y_i) \in \mathscr{X} \times \mathscr{Y}$. Poggio et al. (2004) call the map T CV$_{loo}$-stable for a loss function $L : \mathscr{Y} \to \mathbb{R}^+$ if

$$\lim_{n\to\infty} \sup_{i\in\{1,\dots,n\}} |L(y_i - T(P_n)(x_i)) - L(y_i - T(P_n^{-i})(x_i))| \to 0 \tag{9.2}$$

for $n \to \infty$. This means intuitively that the prediction at a point x_i should not be too different whether or not this point is actually used constructing the predictor. If the difference is too large there is no stability, since in that case adding only one point can yield a large change in the result. Under mild conditions it is shown that CV$_{loo}$-stability is required to achieve good predictions. Let L be the absolute value loss and consider once again the simple case of estimating the location of a univariate distribution. Thus P_n is just a univariate sample of n real numbers $\{y_1,\dots,y_n\}$. Then the left hand side of (9.2) equals

$$\lim_{n\to\infty} \sup_{i\in\{1,\dots,n\}} |T(P_n) - T(P_n^{-i})|.$$

Let $y_{(i)}$ denote the ith order statistic. Consider T the median. Assuming that n is odd and $y_i < y_{(\frac{n+1}{2})}$ (the cases $y_i > y_{(\frac{n+1}{2})}$ and equality can easily be checked as well), we have that

$$|\mathrm{Med}(P_n) - \mathrm{Med}(P_n^{-i})| = \left| y_{(\frac{n+1}{2})} - \frac{1}{2}\left(y_{(\frac{n+1}{2})} + y_{(\frac{n+3}{2})}\right)\right| = \frac{1}{2}|y_{(\frac{n+1}{2})} - y_{(\frac{n+3}{2})}|.$$

If the median of the underlying distribution P is unique, then both $y_{(\frac{n+1}{2})}$ and $y_{(\frac{n+3}{2})}$ converge to this number and CV_{loo} stability is obtained. However, when taking the mean for T, we have that

$$|\mathbb{E}(P_n) - \mathbb{E}(P_n^{-i})| = \left| \frac{1}{n}\sum_{j=1}^{n} y_j - \frac{1}{n-1}\sum_{\substack{j=1\\ j\neq i}}^{n} y_j \right| = \left| -\frac{1}{n(n-1)}\sum_{\substack{j=1\\ j\neq i}}^{n} y_j + \frac{y_i}{n}\right|.$$

The first term in this sum equals the sample mean of P_n^{-i} divided by n and thus converges to 0 if the mean of the underlying distribution exists. The second term converges to 0 if

$$\lim_{n\to\infty} \sup_{i\in\{1,\dots,n\}} \frac{|y_i|}{n} = 0.$$

This means that the largest absolute value of n points sampled from the underlying distribution should not grow too large. For a normal distribution for instance this is satisfied since the largest observation only grows logarithmically: for example the largest of 1000 points generated from a normal distribution only has a very small probability to exceed 5. This is due to the exponentially decreasing density function. For heavy tailed distribution it can be different. A Cauchy density for instance only decreases at the rate of the reciprocal function and $\sup_{i\in\{1,\dots,n\}}|y_i|$ is of the order $O(n)$. Thus for a normal distribution the mean is CV$_{loo}$ stable, but for a Cauchy distribution it is not.

In summary note that both the concepts of influence function and asymptotic variance on one hand and CV$_{loo}$ stability on the other hand yield the same conclusions: using the sample median as an estimator is ok as long as the median of the underlying distribution is unique. Then one has CV$_{loo}$ stability and a finite asymptotic variance. Using the sample mean is ok for a normal distribution, but not for a Cauchy distribution (no CV$_{loo}$ stability and an infinite asymptotic variance).

A rigorous treatment of asymptotic variances and regularity conditions can be found in Boos and Serfling (1980) and Fernholz (1983). In any event, it is an interesting link between perturbation analysis through the influence function and variance/efficiency in statistics on one hand, and between leave-one-out and stability/generalization in learning theory on the other hand.

9.2.3. A Strategy for Fast Approximation of the Leave-one-out Error

In leave-one-out crossvalidation $T(P_n^{-i})$ is computed for every i. This means that the algorithm under consideration has to be executed n times, which can be computationally intensive. If the influence functions of T can be calculated, the following strategy might provide a fast alternative. First note that

$$P_n^{-i} = (1-(\frac{-1}{n-1}))P_n + \frac{-1}{n-1}\Delta_{z_i}.$$

Thus, taking $P_{\varepsilon,z} = P_n^{-i}$, $\varepsilon = -1/(n-1)$ and $P = P_n$, Equation (9.1) gives

$$T(P_n^{-i}) = T(P_n) + \sum_{j=1}^{\infty}(\frac{-1}{n-1})^j \frac{IF_j(z_i;T,P_n)}{j!}. \tag{9.3}$$

The right hand side now only depends on the full sample P_n. In practice one can cut off the series after a number of steps ignoring the remainder term, or if possible one can try to estimate the remainder term.

The first goal of this paper is to apply this idea in the context of kernel based regression. Christmann and Steinwart (2007) computed the first order influence function. We will compute higher order terms in (9.1) and use these results to approximate the leave-one-out estimator applying (9.3).

9.3. Kernel Based Regression

In this section we recall some definitions on kernel based regression. We discuss the influence function and provide a theorem on higher order terms.

9.3.1. Definition

Let $\mathscr{X},\mathscr{Y}$ be non-empty sets. Denote P a distribution on $\mathscr{X}\times\mathscr{Y}\subseteq\mathbb{R}^d\times\mathbb{R}$. Suppose we have a sample of n observations $(x_i,y_i)\in\mathscr{X}\times\mathscr{Y}$ generated i.i.d. from P. Then P_n denotes the corresponding finite sample distribution. A functional T is a map that maps any distribution P onto $T(P)$. A finite sample approximation is given by $T_n := T(P_n)$.

Definition 9.3 *A function $K:\mathscr{X}\times\mathscr{X}\to\mathbb{R}$ is called a kernel on $\mathscr{X}$ if there exists a $\mathbb{R}$-Hilbert space $\mathscr{H}$ and a map $\Phi:\mathscr{X}\to\mathscr{H}$ such that for all $x,x'\in\mathscr{X}$ we have*

$$K(x,x') = \langle\Phi(x),\Phi(x')\rangle.$$

We call Φ a feature map *and $\mathscr{H}$ a* feature space *of K.*

Frequently used kernels include the linear kernel $K(x_i,x_j) = x_i^t x_j$, polynomial kernel of degree p for which $K(x_i,x_j) = (\tau + x_i^t x_j)^p$ with $\tau > 0$ and RBF kernel $K(x_i,x_j) = \exp(-\|x_i - x_j\|_2^2/\sigma^2)$ with bandwidth $\sigma > 0$. By the reproducing property of $\mathscr{H}$ we can evaluate any $f\in\mathscr{H}$ at the point $x\in\mathscr{X}$ as the inner product of f with the feature map: $f(x) = \langle f,\Phi(x)\rangle$.

Definition 9.4 *Let K be a kernel function with corresponding feature space $\mathscr{H}$ and let $L:\mathbb{R}\to\mathbb{R}^+$ be a twice differentiable convex loss function. Then the functional $f_{\lambda,K}: P\to f_{\lambda,K}(P) = f_{\lambda,K,P}\in\mathscr{H}$ is defined by*

$$f_{\lambda,K,P} := \underset{f\in\mathscr{H}}{argmin}\,\mathbb{E}_P L(Y-f(X)) + \lambda\|f\|_{\mathscr{H}}^2$$

where $\lambda > 0$ is a regularization parameter.

The functional $f_{\lambda,K}$ maps a distribution P onto the function $f_{\lambda,K,P}$ that minimizes the regularized risk. When the sample distribution P_n is used, one has that

$$f_{\lambda,K,P_n} := \operatorname*{argmin}_{f \in \mathscr{H}} \frac{1}{n} \sum_{i=1}^{n} L(y_i - f(x_i)) + \lambda \|f\|_{\mathscr{H}}^2. \tag{9.4}$$

Such estimators have been studied in detail, see for example Wahba (1990), Tikhonov and Arsenin (1977) or Evgeniou et al. (2000). In a broader framework (including for example classification, PCA, CCA etc.) primal-dual optimization methodology involving least squares kernel estimators were studied by Suykens et al. (2002b). Possible loss functions include

- the least squares loss: $L(r) = r^2$.
- Vapnik's ε-insensitive loss: $L(r) = \max\{|r| - \varepsilon, 0\}$, with special case the L_1 loss if $\varepsilon = 0$.
- the logistic loss: $L(r) = -\log(4\Lambda(r)[1 - \Lambda(r)])$ with $\Lambda(r) = 1/(1 + e^{-r})$. Note that this is not the same loss function as used in logistic regression.
- Huber loss with parameter $b > 0$: $L(r) = r^2$ if $|r| \leq b$ and $L(r) = 2b|r| - b^2$ if $|r| > b$. Note that the least squares loss corresponds to the limit case $b \to \infty$.

9.3.2. Influence Function

The following proposition was proven in Christmann and Steinwart (2007).

Proposition 9.5 *Let $\mathscr{H}$ be a RKHS of a bounded continuous kernel K on $\mathscr{X}$ with feature map $\Phi : \mathscr{X} \to \mathscr{H}$. Furthermore, let P be a distribution on $\mathscr{X} \times \mathscr{Y}$ with finite second moment. Then the influence function of $f_{\lambda,K}$ exists for all $z := (z_x, z_y) \in \mathscr{X} \times \mathscr{Y}$ and we have*

$$IF(z; f_{\lambda,K}, P) = -S^{-1}\left(2\lambda f_{\lambda,K,P}\right) + L'(z_y - f_{\lambda,K,P}(z_x))S^{-1}\Phi(z_x)$$

where $S : \mathscr{H} \to \mathscr{H}$ is defined by $S(f) = 2\lambda f + \mathbb{E}_P\left[L''(Y - f_{\lambda,K,P}(X))\langle \Phi(X), f\rangle \Phi(X)\right]$.

Thus if the kernel is bounded and the first derivative of the loss function is bounded, then the influence function is bounded as well. Thus L_1 type loss functions for instance lead to robust estimators. The logistic loss as well since the derivative of this loss function equals $L'(r) = 2 - 1/(1 + e^{-r})$ which is bounded by 2. For the Huber loss $L'(r)$ is bounded by $2b$. This shows that the parameter b controls the amount of robustness: if b is very large than the influence function can become very large too. For a small b the influence function remains small. For a least squares loss function on the other hand, the influence function is unbounded ($L'(r) = 2r$): the effect of the smallest amount of contamination can be arbitrary large. Therefore it is said that the least squares estimator is not robust.

9.3.3. Higher Order Influence Functions

For the second order influence function as in Definition 9.2 the following theorem is proven in the Appendix.

Theorem 9.6 *Let P be a distribution on $\mathscr{X} \times \mathscr{Y}$ with finite second moment. Let L be a convex loss function that is three times differentiable. Then the second order influence function of $f_{\lambda,K}$*

exists for all $z := (z_x, z_y) \in \mathscr{X} \times \mathscr{Y}$ and we have

$$\begin{aligned} IF_2(z; f_{\lambda,K}, P) = & S^{-1}\Big(2\mathbb{E}_P[IF(z; f_{\lambda,K}, P)(X)L''(Y - f_{\lambda,K}(X))\Phi(X)] \\ & + \mathbb{E}_P[(IF(z; f_{\lambda,K}, P)(X))^2 L'''(Y - f_{\lambda,K,P}(X))] \\ & - 2[IF(z; f_{\lambda,K}, P)(z_x)L''(z_y - f_{\lambda,K}(z_x))\Phi(z_x)]\Big) \end{aligned}$$

where $S : \mathscr{H} \to \mathscr{H}$ is defined by $S(f) = 2\lambda f + \mathbb{E}_P\left[L''(Y - f_{\lambda,K,P}(X))\langle\Phi(X), f\rangle\Phi(X)\right]$.

When the loss function is infinitely differentiable, all higher order terms can in theory be calculated, but the number of terms grows rapidly since all derivatives of L come into play. However, in the special case that all derivatives higher than three are 0, a relatively simple recursive relation exists.

Theorem 9.7 *Let P be a distribution on $\mathscr{X} \times \mathscr{Y}$ with finite second moment. Let L be a convex loss function such that the third derivative is 0. Then the $(k+1)$th order influence function of $f_{\lambda,K}$ exists for all $z := (z_x, z_y) \in \mathscr{X} \times \mathscr{Y}$ and we have*

$$\begin{aligned} IF_{k+1}(z; f_{\lambda,K}, P) = (k+1)S^{-1}\Big(& \mathbb{E}_P[IF_k(z; f_{\lambda,K}, P)(X)L''(Y - f_{\lambda,K}(X))\Phi(X)] \\ & - [IF_k(z; f_{\lambda,K}, P)(z_x)L''(Z_y - f_{\lambda,K}(z_x))\Phi(z_x)]\Big) \end{aligned}$$

where $S : \mathscr{H} \to \mathscr{H}$ is defined by $S(f) = 2\lambda f + \mathbb{E}_P\left[L''(Y - f_{\lambda,K,P}(X))\langle\Phi(X), f\rangle\Phi(X)\right]$.

9.4. Finite Sample Expressions

Since the Taylor expansion in (9.1) is now fully characterized for any distribution P and any z, we can use this to assess the influence of individual points in a sample with sample distribution P_n. Applying Equation (9.3) with the KBR estimator f_{λ,K,P_n} from (9.4) we have that

$$f_{\lambda,K,P_n^{-i}}(x_i) = f_{\lambda,K,P_n}(x_i) + \sum_{j=1}^{\infty}\left(\frac{-1}{n-1}\right)^j \frac{IF_j(z_i; f_{\lambda,K}, P_n)(x_i)}{j!}. \tag{9.5}$$

Let us see how the right hand side can be evaluated in practice.

9.4.1. Least Squares Loss

First consider taking the least squares loss in (9.4). Denote Ω the $n \times n$ kernel matrix with i,j-th entry equal to $K(x_i, x_j)$. Let I_n be the $n \times n$ identity matrix and denote $S_n = \Omega/n + \lambda I_n$. The value of f_{λ,K,P_n} at a point $x \in \mathscr{X}$ is given by

$$f_{\lambda,K,P_n}(x) = \frac{1}{n}\sum_{i=1}^{n}\alpha_i K(x_i, x) \quad \text{with} \quad \begin{pmatrix}\alpha_1 \\ \vdots \\ \alpha_n\end{pmatrix} = S_n^{-1}\begin{pmatrix} y_1 \\ \vdots \\ y_n\end{pmatrix} \tag{9.6}$$

which is a classical result going back to Tikhonov and Arsenin (1977). This also means that the vector of predictions in the n sample points simply equals

$$\begin{pmatrix} f_{\lambda,K,P_n}(x_1) \\ \vdots \\ f_{\lambda,K,P_n}(x_n) \end{pmatrix} = H \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \tag{9.7}$$

with the matrix $H = \frac{1}{n} S_n^{-1} \Omega$, sometimes referred to as the smoother matrix.

To compute the first order influence function at the sample the expression in Proposition 9.5 should be evaluated at P_n. The operator S at P_n maps by definition any $f \in \mathcal{H}$ onto

$$S_{P_n}(f) = 2\lambda f + \mathbb{E}_{P_n} 2f(X)\Phi(X) = 2\lambda f + \frac{2}{n}\sum_{j=1}^{n} f(x_j)\Phi(x_j)$$

and thus

$$\begin{aligned}\begin{pmatrix} S_{P_n}(f)(x_1) \\ \vdots \\ S_{P_n}(f)(x_n) \end{pmatrix} &= 2\lambda \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix} + \frac{2}{n} \begin{pmatrix} K(x_1,x_1) & \dots & K(x_1,x_n) \\ \vdots & & \\ K(x_n,x_1) & & K(x_n,x_n) \end{pmatrix} \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix} \\ &= 2S_n \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix}\end{aligned}$$

which means that the matrix $2S_n$ is the finite sample version of the operator S at the sample P_n. From Proposition 9.5 it is now clear that

$$\begin{pmatrix} IF(z_i; f_{\lambda,K}, P_n)(x_1) \\ \vdots \\ IF(z_i; f_{\lambda,K}, P_n)(x_n) \end{pmatrix} = S_n^{-1}\left((y_i - f_{\lambda,K,P_n}(x_i)) \begin{pmatrix} K(x_i,x_1) \\ \vdots \\ K(x_i,x_n) \end{pmatrix} - \lambda \begin{pmatrix} f_{\lambda,K,P_n}(x_1) \\ \vdots \\ f_{\lambda,K,P_n}(x_n) \end{pmatrix} \right). \tag{9.8}$$

In order to evaluate the influence function at sample point z_i at a sample distribution P_n, we only need the full sample fit f_{λ,K,P_n} and the matrix S_n^{-1}, which is already obtained when computing f_{λ,K,P_n} (cf. Equation (9.6)). From Theorem 9.7 one sees similarly that the higher order terms can be computed recursively as

$$\begin{aligned}\begin{pmatrix} IF_{k+1}(z_i; f_{\lambda,K}, P_n)(x_1) \\ \vdots \\ IF_{k+1}(z_i; f_{\lambda,K}, P_n)(x_n) \end{pmatrix} =& (k+1) S_n^{-1} \frac{\Omega}{n} \begin{pmatrix} IF(z_i; f_{\lambda,K}, P_n)(x_1) \\ \vdots \\ IF_k(z_i; f_{\lambda,K}, P_n)(x_n) \end{pmatrix} \\ & - (k+1) IF_k(z_i; f_{\lambda,K}, P_n)(x_i) S_n^{-1} \begin{pmatrix} K(x_i,x_1) \\ \vdots \\ K(x_i,x_n) \end{pmatrix}.\end{aligned} \tag{9.9}$$

Define $[IFM_k]$ the matrix containing $IF_k(z_j; f_{\lambda,K}, P_n)(x_i)$ at entry i,j. Then (9.9) is equivalent to

$$[IFM_{k+1}] = (k+1)\left(H\, [IFM_k] - nH \bullet [IFM_k]\right)$$

with $\bullet$ denoting the entrywise matrix product (also known as the Hadamard product). Or equivalently

$$[IFM_{k+1}] = (k+1)\left(H([IFM_k] \bullet M(1-n))\right) \tag{9.10}$$

with M the matrix containing $1/(1-n)$ at the off-diagonal and 1 at the diagonal. A first idea is now to approximate the series in (9.5) by cutting it off at some step k:

$$f_{\lambda,K,P_n^{-i}}(x_i) \approx f_{\lambda,K,P_n}(x_i) + \sum_{j=1}^{k} \frac{1}{(1-n)^j j!}[IFM_j]_{i,i}. \tag{9.11}$$

However using (9.10) we can do a bit better. Expression (9.5) becomes

$$\begin{aligned} f_{\lambda,K,P_n^{-i}}(x_i) = f_{\lambda,K,P_n}(x_i) &+ \frac{1}{1-n}[IFM_1]_{i,i} + \frac{1}{1-n}[H(IFM_1 \bullet M)]_{i,i} \\ &+ \frac{1}{1-n}[H(H(IFM_1 \bullet M) \bullet M)]_{i,i} + \dots \end{aligned}$$

In every term there is a multiplication with H and an entrywise multiplication with M. The latter means that all diagonal elements remain unchanged but the non-diagonal elements are divided by $1-n$. So after a few steps the non-diagonal elements will converge to 0 quite fast. It makes sense to set the non-diagonal elements 0 retaining only the diagonal elements:

$$\begin{aligned} f_{\lambda,K,P_n^{-i}}(x_i) &\approx f_{\lambda,K,P_n}(x_i) + \sum_{j=1}^{k-1} \frac{1}{(1-n)^j j!}[IFM_j]_{i,i} + \frac{1}{(1-n)^k k!} \sum_{j=0}^{\infty} H_{i,i}^j [IFM_k]_{i,i} \\ &= f_{\lambda,K,P_n}(x_i) + \sum_{j=1}^{k-1} \frac{1}{(1-n)^j j!}[IFM_j]_{i,i} + \frac{1}{(1-n)^k k!} \frac{[IFM_k]_{i,i}}{1-H_{i,i}} \end{aligned} \tag{9.12}$$

since $H_{i,i}$ is always smaller than 1.

9.4.2. Huber Loss

For the Huber loss function with parameter $b > 0$ we have that

$$L(r) = \begin{cases} r^2 & \text{if } |r| < b. \\ 2b|r| - b^2 & \text{if } |r| > b. \end{cases}$$

and thus

$$L'(r) = \begin{cases} 2r & \text{if } |r| < b \\ 2b \operatorname{sign}(r) & \text{if } |r| > b \end{cases}, \qquad L''(r) = \begin{cases} 2 & \text{if } |r| < b \\ 0 & \text{if } |r| > b \end{cases}.$$

Note that the derivatives in $|r| = b$ do not exist, but in practice the probability that a residual exactly equals b is 0, so we further ignore this possibility. The following equation holds:

$$f_{\lambda,K,P_n}(x) = \frac{1}{n}\sum_{i=1}^{n} \alpha_i K(x_i, x) \quad \text{with} \quad 2\lambda\alpha_j = L'(y_j - \frac{1}{n}\sum_{i=1}^{n} \alpha_i K(x_i, x_j)). \tag{9.13}$$

Thus a set of possibly non-linear equations has to be solved in α. Once the solution for the full sample is found, an approximation of the leave-one-out error is obtained in a similar way as for least squares. Proposition 9.5 for P_n gives the first order influence function.

$$\begin{pmatrix} IF(z_i; f_{\lambda,K}, P_n)(x_1) \\ \vdots \\ IF(z_i; f_{\lambda,K}, P_n)(x_n) \end{pmatrix} = S_b^{-1}\left(L'(y_i - f_{\lambda,K,P_n}(x_i)) \begin{pmatrix} K(x_i, x_1) \\ \vdots \\ K(x_i, x_n) \end{pmatrix} - \lambda \begin{pmatrix} f_{\lambda,K,P_n}(x_1) \\ \vdots \\ f_{\lambda,K,P_n}(x_n) \end{pmatrix}\right)$$

with $S_b = 2\lambda I_n + \Omega \bullet B/n$ and B the matrix containing $L''(y_i - f_{\lambda,K,P_n}(x_i)$ at every entry in the ith column. Let $H_b = S_b^{-1}\Omega/n \bullet B$. Starting from Theorem 9.7 one finds analogously as (9.10) the following recursion to compute higher order terms.

$$[IFM_{k+1}] = (k+1)\,(H_b([IFM_k] \bullet M(1-n)))\,.$$

Finally one can use these matrices to approximate the leave-one-out estimator as

$$f_{\lambda,K,P_n^{-i}}(x_i) \approx f_{\lambda,K,P_n}(x_i) + \sum_{j=1}^{k-1} \frac{1}{(1-n)^j j!}[IFM_j]_{i,i} + \frac{1}{(1-n)^k k!}\frac{[IFM_k]_{i,i}}{1-[H_b]_{i,i}} \tag{9.14}$$

in the same way as in (9.12)

9.4.3. Reweighted KBR

In Equation (9.14) the full sample estimator f_{λ,K,P_n} is of course needed. For a general loss function L one has to solve Equation (9.13) to find f_{λ,K,P_n}. A fast way to do so is to use reweighted KBR with a least squares loss. Let

$$W(r) = \frac{L'(r)}{2r}. \tag{9.15}$$

Then we can rewrite (9.13) as

$$\begin{aligned} 2\lambda f_{\lambda,K,P_n}(x_k) &= \frac{1}{n}\sum_{i=1}^{n} L'(y_i - f_{\lambda,K,P_n}(x_i))K(x_i,x_k) \quad \forall 1 \leq k \leq n. \\ &= \frac{1}{n}\sum_{i=1}^{n} 2W(y_i - f_{\lambda,K,P_n}(x_i))(y_i - f_{\lambda,K,P_n}(x_i))K(x_i,x_k). \end{aligned}$$

Denoting $w_i = W(y_i - f_{\lambda,K,P_n}(x_i))$ this means that

$$\lambda f_{\lambda,K,P_n}(x_k) = \frac{1}{n}\sum_{i=1}^{n} w_i y_i K(x_i,x_k) - \frac{1}{n}\sum_{i=1}^{n} w_i f_{\lambda,K,P_n}(x_i)K(x_i,x_k) \quad \forall 1 \leq k \leq n.$$

Let I_w denote the $n \times n$ diagonal matrix with w_i at entry i,i. Then

$$\begin{pmatrix} f_{\lambda,K,P_n}(x_1) \\ \vdots \\ f_{\lambda,K,P_n}(x_n) \end{pmatrix} = \left(\frac{\Omega}{n} + \lambda I_w\right)^{-1} \frac{\Omega}{n} \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \tag{9.16}$$

and thus f_{λ,K,P_n} can be written as a reweighted least squares estimator with additional weights w_i compared to Equations (9.6) and (9.7). Of course these weights still depend on the unknown f_{λ,K,P_n}, so (9.16) only implicitly defines f_{λ,K,P_n}. It does suggest the following iterative reweighting algorithm.

1. Start with simple least squares computing (9.7). Denote the solution f^0_{λ,K,P_n}.

2. At step $k+1$ compute weights $w_{i,k} = W(y_i - f^k_{\lambda,K,P_n}(x_i))$.

3. Solve (9.16) using the weights $w_{i,k}$. Let the solution be f^{k+1}_{λ,K,P_n}.

In Suykens et al. (2002a) it is shown that this algorithm usually converges in very few steps. In Debruyne et al. (2010) the robustness of such stepwise reweighting algorithm is analyzed by calculating stepwise influence functions. It is shown that the influence function is stepwise reduced under certain conditions on the weight function.

For the Huber loss with parameter b Equation (9.15) means that the corresponding weight function equals $W(r) = 1$ if $|r| \leq b$ and $W(r) = b/|r|$ if $|r| > b$. This gives a clear interpretation of this loss function: all observations with error smaller than b remain unchanged, but the ones with error larger than b are downweighted compared to the least squares loss. This also explains the gain in robustness. One can expect better robustness as b decreases.

It would be possible to compute higher order terms of such $k-$step estimators as well. Then one could explicitly use these terms to approximate the leave-one-out error of the $k-$step reweighted estimator. In this paper however we use the reweighting only to compute the full sample estimator f_{λ,K,P_n} and we assume that it is fully converged to the solution of (9.13). For the model selection (9.14) is then used.

9.5. Model Selection

Once the approximation of $f_{\lambda,K,P_n^{-i}}$ is obtained, one can proceed with model selection using the leave-one-out principle. In the next paragraphs we propose a specific implementation taking into account performance as well as robustness.

9.5.1. Definition

The traditional leave-one-out criterion is given by

$$\text{LOO}(\lambda, K) = \frac{1}{n}\sum_{i=1}^{n} V(y_i - f_{\lambda,K,P_n^{-i}}(x_i)) \tag{9.17}$$

with V an appropriate loss function. The values of λ and of possible kernel parameters for which this criterion is minimal, are then selected to train the model. The idea we investigate is to replace the explicit leave-one-out by the approximation in (9.12) for least squares and (9.14) for the Huber loss.

Definition 9.8 *The k-th order influence function criterion at a regularization parameter $\lambda > 0$ and kernel K for Huber loss KBR with parameter b is defined as*

$$C_{IF}^{k}(\lambda,K,b) = \frac{1}{n}\sum_{i=1}^{n} V\left(y_i - f_{\lambda,K,P_n}(x_i) - \sum_{j=1}^{k-1}\frac{1}{(1-n)^j j!}[IFM_j]_{i,i} - \frac{1}{(1-n)^k k!}\frac{[IFM_k]_{i,i}}{1-[H_b]_{i,i}}\right).$$

For KBR with a least squares loss we write

$$C_{IF}^{k}(\lambda,K,\infty) = \frac{1}{n}\sum_{i=1}^{n} V\left(y_i - f_{\lambda,K,P_n}(x_i) - \sum_{j=1}^{k-1}\frac{1}{(1-n)^j j!}[IFM_j]_{i,i} - \frac{1}{(1-n)^k k!}\frac{[IFM_k]_{i,i}}{1-[H]_{i,i}}\right).$$

since a least squares loss is a limit case of the Huber loss as $b \to \infty$.

Several choices need to be made in practice. For k taking five steps seems to work very well in the experiments. If we refer to the criterion with this specific choice $k = 5$ we write C_{IF}^{5}. For V one typically chooses the squared loss or the absolute value corresponding to the mean squared error and the mean absolute error. Note that V does not need to be the same as the loss function used to compute f_{λ,K,P_n} (the latter is always denoted by L). Recall that a loss function L with bounded first derivative L' is needed to perform robust fitting. It is important to note that this result following from Proposition 9.5 holds for a fixed choice of λ and the kernel K. However, if these parameters are selected in a data driven way, outliers in the data might have a large effect on the selection of the parameters. Even if a robust estimator is used, the result could be quite bad if wrong choices are made for the parameters due to the outliers. It is thus important to use a robust loss function V as well. Therefore we set V equal to the absolute value loss function unless we explicitly state differently. In Section 9.6.1 an illustration is given on what can go wrong if a least squares loss is chosen for V instead of the absolute value.

9.5.2. Optimizing *b*

With k and V now specified, the criterion C^5_{IF} can be used to select optimal hyperparameters for a KBR estimator with L the Huber loss with parameter b. Now the final question remains how to choose b. In Section 9.4.3 it was argued that b controls the robustness of the estimator since all observations with error smaller than b are downweighted compared to the least squares estimator. Thus we want to choose b small enough such that outlying observations receive sufficiently small weight, but also large enough such that the good non outlying observations are not downweighted too much. A priori it is quite difficult to find such a good choice for b, since this will depend on the scale of the errors.

However, one can also treat b as an extra parameter that is part of the optimization, consequently minimizing C^5_{IF} for λ, K and b simultaneously. The practical implementation we propose is as follows:

1. Let Λ be a set of reasonable values for the regularization parameter λ and let $\mathscr{K}$ be a set of possible choices for the kernel K (for instance a grid of reasonable bandwidths if one considers the RBF kernel).

2. Start with L the least squares loss. Find good choices for λ and K by minimizing $C^5_{IF}(\lambda,K,\infty)$ for all $\lambda \in \Lambda$ and $K \in \mathscr{K}$. Compute the residuals r_i with respect to the least squares fit with these optimal λ and K.

3. Compute a robust estimate of the scale of these residuals. We take the Median Absolute Deviation (MAD):

$$\hat{\sigma}_{err} = \mathrm{MAD}(r_1,\ldots,r_n) = \frac{1}{\Phi^{-1}(0.75)}\mathrm{median}(|r_i - \mathrm{median}(r_i)|) \qquad (9.18)$$

   with $\Phi^{-1}(0.75)$ the 0.75 quantile of a standard normal distribution.

4. Once the scale of the errors is estimated in the previous way, reasonable choices of b can be constructed, for example $\{1,2,3\} \times \hat{\sigma}_{err}$. This means that we compare downweighting observations further away than 1, 2, 3 standard deviations. We also want to compare to the least squares fit and thus set

$$\mathscr{B} = \{\hat{\sigma}_{err}, 2\hat{\sigma}_{err}, 3\hat{\sigma}_{err}, \infty\}.$$

5. Minimize $C^5_{IF}(\lambda,K,b)$ over all $\lambda \in \Lambda$, $K \in \mathscr{K}$ and $b \in \mathscr{B}$. The optimal values of b, λ and K can then be used to construct the final fit.

9.5.3. Generalized Cross Validation

The criterion C_{IF}^5 uses influence functions to approximate the leave-one-out error. Other approximations have been proposed in the literature. In this section we very briefly mention some results that are described for example by Wahba (1990) in the context of spline regression. The following result can be proven.

Let $\tilde{P}_n^{-i}$ be the sample P_n with observation (x_i, y_i) replaced by $(x_i, f_{\lambda,K,P_n^{-i}}(x_i))$. Suppose the following conditions are satisfied for any sample P_n:

$$(i) \quad f_{\lambda,K,\tilde{P}_n^{-i}}(x_i) = f_{\lambda,K,P_n^{-i}}(x_i). \tag{9.19}$$

$$(ii) \quad \text{There exists a matrix } H \text{ such that } \begin{pmatrix} f_{\lambda,K,P_n}(x_1). \\ \vdots \\ f_{\lambda,K,P_n}(x_n) \end{pmatrix} = H \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}. \tag{9.20}$$

Then

$$f_{\lambda,K,P_n^{-i}}(x_i) = \frac{f_{\lambda,K,P_n}(x_i) - H_{i,i} y_i}{1 - H_{i,i}}. \tag{9.21}$$

For KBR with the least squares loss condition (22) is indeed satisfied (cf. Equation (9.7)), but condition (9.19) is not, although it holds approximately. Then (9.21) can still be used as an approximation of the leave-one-out estimator. The corresponding model selection criterion is given by

$$CV(\lambda, K) = \frac{1}{n} \sum_{i=1}^{n} V\left(\frac{y_i - f_{\lambda,K,P_n}(x_i)}{1 - H_{i,i}}\right). \tag{9.22}$$

We call this approximation CV. Sometimes a further approximation is made replacing every $H_{i,i}$ by $\operatorname{trace}(H)/n$. This is called Generalized Cross Validation (GCV, Wahba, 1990). Note that the diagonal elements of the hatmatrix H play an important role in the approximation with the influence function too (9.12). Both penalize small values on the diagonal of H.

For KBR with a general loss function one does not have a linear equation of the form of (22), and thus it is more difficult to apply this approximation. We shall thus use CV for comparison in the experiments only in the case of least squares.

9.6. Empirical Results

We illustrate the results on a toy example and a small simulation study.

9.6.1. Toy Example

As a toy example 50 data points were generated with x_i uniformly distributed on the interval $[2, 11]$ and $y_i = \sin(x_i) + e_i$ with e_i Gaussian distributed noise with standard deviation 0.2. We start with kernel based regression with a least squares loss and a Gaussian kernel. The data are shown in Figure 9.1(a) as well as the resulting fit with $\lambda = 0.001$ and $\sigma = 2$.

The first order influence function at $[5, 0.5]$ is depicted in Figure 9.1(b) as the solid line. This reflects the asymptotic change in the fit when a point would be added to the data in Figure 9.1(a) at the position $(5, 0.5)$. Obviously this influence is the largest at the x-position where we put the outlier, that is, $x = 5$. Furthermore we see that the influence is local, since it decreases as we look further away from $x = 5$. At $x = 8$ for instance the influence function is almost 0. When we change z from $[5, 0.5]$ to $[5, 1]$, the influence function changes too. It still has the same oscillating behavior, but the peaks are now higher. This reflects the non-robustness of

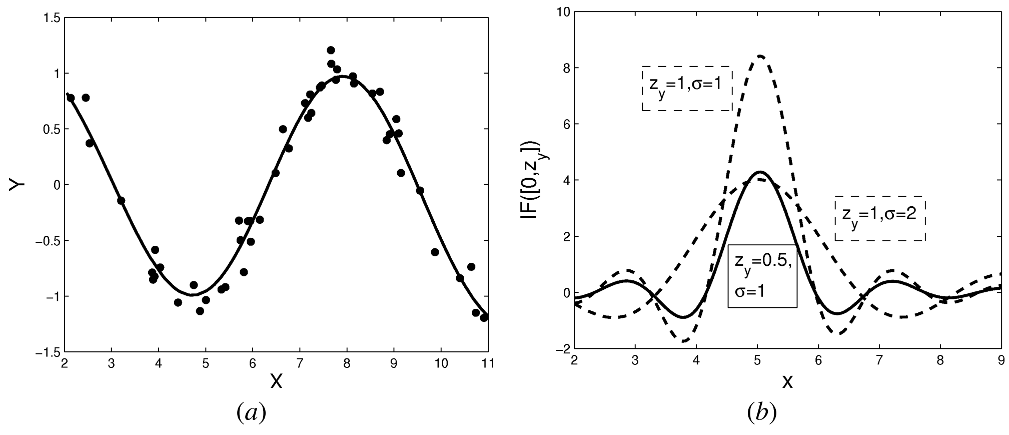


Figure 9.1: (*a*) Data and least squares fit. (*b*) Influence functions at $[5,0.5]$ with $\sigma = 1$, at $[5,1]$ with $\sigma = 1$ and $\sigma = 2$.

the least squares estimator: if we would continue raising the point z, then $IF(z; f_{\lambda,K})$ would become larger and larger, since it is an unbounded function of z. When it comes down to model selection, it is interesting to check the effect of the hyperparameters in play. When we change the bandwidth σ from 1 to 2, the peaks in the resulting influence function in Figure 9.1 are less sharp and less high. This reflects the loss in stability when small bandwidths are chosen: then the fit is more sensitive to small changes in the data and thus less stable.

Consider now the approximation of the leave-one-out error using the influence functions. We still use the same data as in the previous paragraph. The dashed lines in Figure 9.2(*a*) show the approximations using (9.11), that is simply cutting off the expansion after a number of steps, at fixed $\lambda = 0.001$ as a function of the bandwidth σ. We observe convergence from the training error towards the leave-one-out error as the number of terms included is increased. Unfortunately the convergence rate depends on the value of σ: convergence is quite slow at small values of σ. This is no surprise looking at (9.12). There we approximated the remainder term by a quantity depending on $(1 - H_{i,i})^{-1}$. When σ is small, the diagonal elements of H become close to 1. In that case the deleted remainder term can indeed be quite large. Nevertheless, this approach can still be useful if some care is taken not to consider values of λ and σ that are too small. However, the criterion C^5_{IF} from Definition 9.8 using the approximation in (9.12) is clearly superior. We see that the remainder term is now adequately estimated and a good approximation is obtained at any σ. The resulting curve is undistinguishable from the exact leave-one-out error. The mean absolute difference is $3.2\ 10^{-5}$, the maximal difference is $1.8\ 10^{-4}$. The CV approximation also yields a good result being indistinguishable from the exact leave-one-out error on the plot as well. The mean absolute difference is $4.1\ 10^{-4}$ and the maximal difference equals $1.8\ 10^{-3}$. Thus C^5_{IF} is closer to the true leave-one-out error than CV, although the difference is irrelevant when it comes down to selecting a good σ.

Figure 9.2 also shows plots for the leave-one-out error and its various approximations at (*b*) $\lambda = 0.005$ as a function of σ, (*c*) $\sigma = 1$ as a function of λ, (*d*) $\sigma = 2$ as a function of λ. In these cases as well it is observed that the cutoff strategy yields decent results if a sufficient number of terms is taken into account and if one does not look at values of λ and σ that are extremely small. The best strategy is to take the remainder term into account using the criterion C^k_{IF} from Definition 9.8.

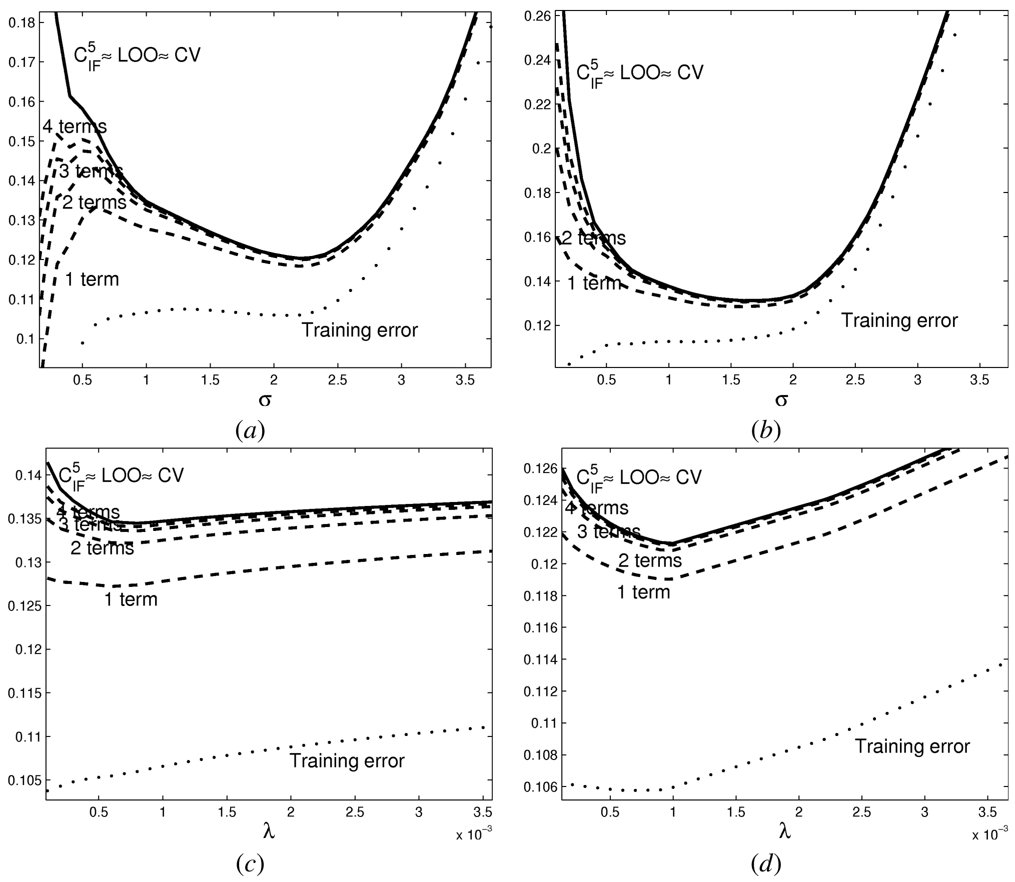


Figure 9.2: Comparison of training error (dotted line), approximations using (9.11) (dashed lines), the proposed criterion C_{IF}^k with $k = 5$ (solid line), the exact leave-one-out error and the CV approximation (both collapsing with C_{IF}^k on these plots). Situation (a): as a function of σ at $\lambda = 0.001$, (b) as a function of σ at $\lambda = 0.005$, (c) as a function of λ at $\sigma = 1$, (d) as a function of λ at $\sigma = 2$.

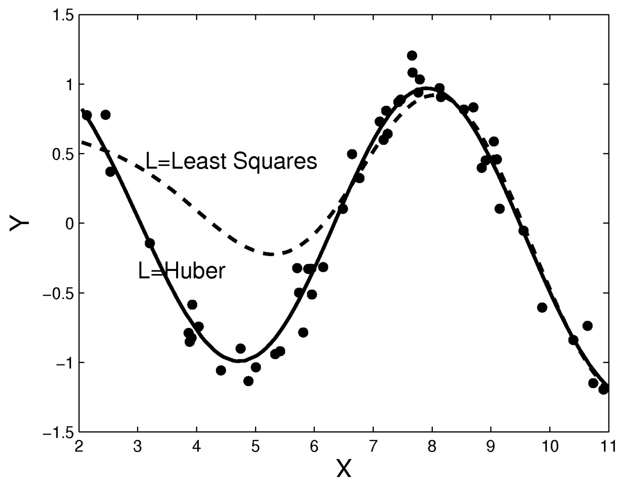


Figure 9.3: Data with outlier at $(4,5)$. The parameters $\lambda = 0.001$ and $\sigma = 2$ are fixed. Dashed: KBR with least squares loss function. Solid: KBR with Huber loss function ($b = 0.2$).

In Figure 9.3 we illustrate robustness. An (extreme) outlier was added at position $(4,5)$ (not visible on the plot). This outlier leads to a bad fit when LS-KBR is used with $\lambda = 0.001$ and $\sigma = 2$ (dashed line). When a Huber loss function is used with $b = 0.2$ a better fit is obtained that still nicely predicts the majority of observations. This behavior can be explained by Proposition 9.5. The least squares loss has an unbounded first derivative and thus the influence of outliers can be arbitrary large. The Huber loss has a bounded first derivative and thus the influence of outliers is bounded as well. However, note that in this example as well as in Proposition 9.5 the hyperparameters λ and σ are assumed to have fixed values. In practice one wants to choose these parameters in a data driven way.

Figure 9.4(*a*) shows the optimization of σ at $\lambda = 0.001$ for KBR with L the Huber loss with $b = 0.2$. In the upper panel the least squares loss is used for V in the model selection criteria. Both exact leave-one-out and C^5_{IF} indicate that a value of $\sigma \approx 3.6$ should be optimal. This results in the dashed fit in Figure 9.4(*b*). In the lower panel of Figure 9.4 the L_1 loss is used for V in the model selection criteria. Both exact leave-one-out and C^5_{IF} indicate that a value of $\sigma \approx 2.3$ should be optimal. This results in the solid fit in Figure 9.4(*b*). We clearly see that, although in both cases a robust estimation procedure is used (Huber loss for L), the outlier can still be quite influential through the model selection. To obtain full protection against outliers, both the estimation and the model selection step require robustness, for example by selecting both L and V in a robust way.

Finally let us investigate the role of the parameter b used in the Huber loss function. We now use C^5_{IF} with V the L_1 loss.

When we apply C^5_{IF} to the clean data without the outlier, we observe in Figure 9.5(*a*) that the choice of b does not play an important role. This is quite expected: since there are no outliers, there is no reason why least squares ($b = \infty$) would not perform well. On the contrary, if we use a small b such as $b = 0.1$ we get a slightly worse result. Again this is not a surprise, since with small b we will downweight a lot of points that are actually perfectly ok.

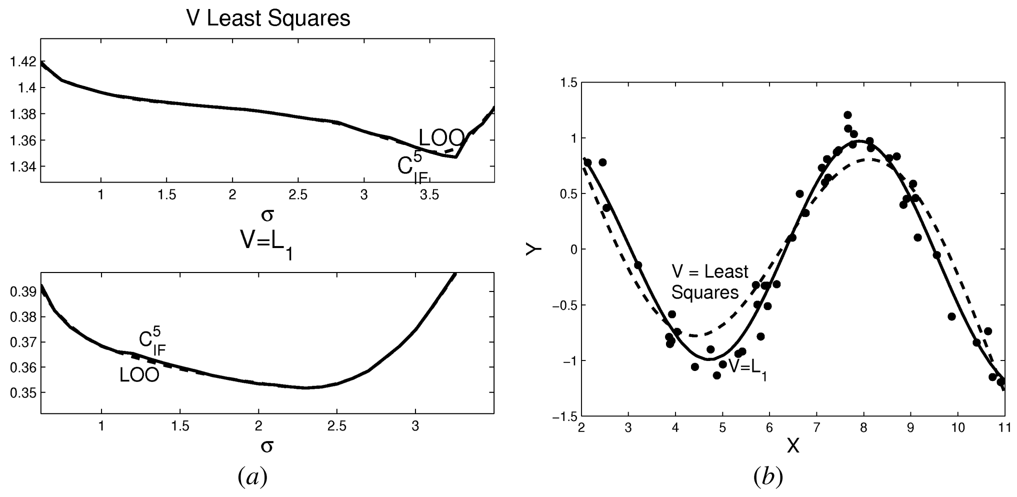


Figure 9.4: (*a*) Optimization of σ at $\lambda = 0.001$. Upper: using least squares loss V in the model selection. Lower: using L_1 loss V in the model selection. For the estimation the loss function L is always the Huber loss with $b = 0.2$. (*b*) Resulting fits. Dashed line: $\sigma = 3.6$ (optimal choice using V least squares). Solid line: $\sigma = 2.3$ (optimal choice using L_1 loss for V.)

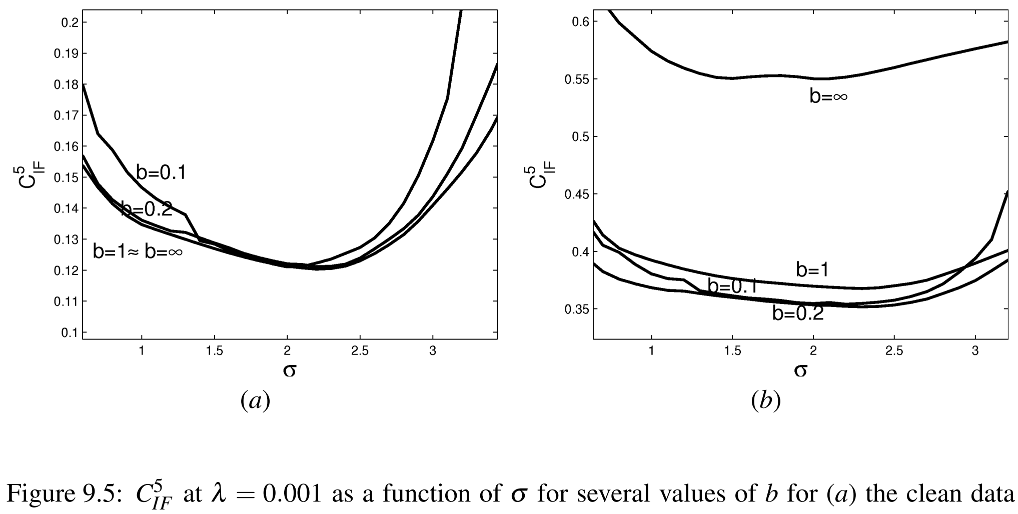


Figure 9.5: C^5_{IF} at $\lambda = 0.001$ as a function of σ for several values of b for (*a*) the clean data without the outlier, (*b*) the data with the outlier.

The same plot for the data containing the outlier yields a different view in Figure 9.5(*b*). The values of C^5_{IF} are much higher for least squares than for the Huber loss with smaller b. Thus it is automatically detected that a least squares loss is not the appropriate choice, which is a correct assessment since the outlier will have a large effect (cf. the dashed line in Figure 9.3). The criterion C^5_{IF} indicates a choice $b = 0.2$, which leads to a better result indeed (cf. the solid line in Figure 9.3)

9.6.2. Other Examples

This part presents the results of a small simulation study. We consider some well known settings.

- Friedman 1 ($d = 10$): $y(x) = 10\sin(\pi x_1 x_2) + 20(x_3 - 1/2)^2 + 10x_4 + 5x_5 + \sum_{i=6}^{10} 0.x_i$. The covariates are generated uniformly in the hypercube in $\mathbb{R}^{10}$.

- Friedman 2 ($d = 4$): $y(x) = \frac{1}{3000}(x_1^2 + (x_2x_3 - (x_2x_4)^{-2}))^{1/2}$, with $0 < x_1 < 100$, $20 < x_2/(2\pi) < 280$, $0 < x_3 < 1$, $1 < x_4 < 11$.

- Friedman 3 ($d = 4$): $y(x) = \tan^{-1}(\frac{x_2x_3-(x_2x_4)^{-2}}{x_1})$, with the same range for the covariates as in Friedman 2. For each of the Friedman data sets 100 observations were generated with Gaussian noise and 200 noise free test data were generated.

- Boston Housing Data from the UCI machine learning depository with 506 instances and 13 covariates. Each split 450 observations were used for training and the remaining 56 for testing.

- Ozone data from `ftp://ftp.stat.berkeley.edu/pub/users/breiman/` with 202 instances and 12 covariates. Each split 150 observations were used for training and the remaining 52 for testing.

- Servo data from the UCI machine learning depository with 167 instances and 4 covariates. Each split 140 observations were used for training and the remaining 27 for testing.

For the real data sets (Boston, Ozone and Servo), new contaminated data set were constructed as well by adding large noise to 10 training points, making these 10 points outliers.

The hyperparameters λ and σ are optimized over the following grid of hyperparametervalues:

- $\lambda \in \{50, 10, 5, 3, 1, 0.8, 0.5, 0.3, 0.1, 0.08, 0.05, 0.01, 0.005\} \times 10^{-3}$.

- For each data set 500 distances were calculated between two randomly chosen observations. Let $d_{(i)}$ be the ith largest distance. Then the following grid of values for σ is considered:
$\sigma \in \{\frac{1}{2}d_{(1)}, d_{(1)}, d_{(50)}, d_{(100)}, d_{(150)}, d_{(200)}, d_{(250)}, d_{(300)}, d_{(350)}, d_{(400)}, d_{(450)}, d_{(500)}, 2d_{(500)}\}$.

In each replicate the Mean Squared Error of the test data is computed. For every data set the average MSE over 20 replicates is shown in Table 9.1 (upper table). A two-sided paired Wilcoxon rank test is used to check statistical significance: values in italic are significantly different from the smallest value at significance level 0.05. If underlined significance holds even at significance level 10^{-4}. Standard errors are shown as well (lower table). First we consider the least squares loss for L with the criterion $C^5_{IF}(\lambda, \sigma, \infty)$ (Definition 9.8), with exact leave-one-out (9.17) and with CV (9.22). These are the first 3 columns in Table 9.1. We see that the difference between these 3 criteria is very small. This means that both CV and C_{IF}^5 provide good approximations of the leave-one-out error.

Secondly, we considered each time the residuals of the least squares fit with optimal λ and σ according to $C_{IF}^5(\lambda, K, \infty)$. An estimate $\hat{\sigma}_{err}$ of the scale of the residuals is computed as the MAD of these residuals (9.18). Then we applied KBR with a Huber loss and parameter $b = 3\hat{\sigma}_{err}$. The resulting MSE with this loss and λ and σ minimizing $C_{IF}^5(\lambda, \sigma, 3\hat{\sigma}_{err})$ is given in column 4 in Table 9.1. Similar results are obtained for $b = 2\hat{\sigma}_{err}$ in column 5 and with $b = \hat{\sigma}_{err}$ in column 6. For the data sets without contamination we see that using a Huber loss instead of least squares gives similar results except for the Boston housing data, Friedman 1 and especially Friedman 2. For those data sets a small value of b is inappropriate. This might be explained by the relationship between the loss function and the error distribution. For a Gaussian error distribution least squares is often an optimal choice (cf. maximum likelihood theory). Since the errors in the Friedman data are explicitly generated as Gaussian, this might explain why least squares outperforms the Huber loss. For real data sets, the errors might not be exactly Gaussian, and thus other loss function can perform at least equally well as least squares. For the data sets containing the outliers the situation changes of course. Now least squares is not a good option because of its lack of robustness. Clearly the outliers have a large and bad effect on the quality of the predictions. This is not the case when the Huber loss function is chosen. Then the effect of the outliers is reduced. Choosing $b = 3\hat{\sigma}_{err}$ already leads to a large improvement. Decreasing b leads to even better results (note that the p-values are smaller than 10^{-4} for any significant pairwise comparison).

Finally we also consider optimizing b. We apply the algorithm outlined in Section 9.5.2. Corresponding MSE's are given in the last column of Table 9.1. For the Friedman 1 and Friedman 2 data sets for instance this procedure indeed detects that least squares is an appropriate loss function and automatically avoids choosing b too small. For the contaminated data sets the procedure detects that least squares is not appropriate and that changing to a Huber loss with a small b is beneficial, which is indeed a correct choice yielding smaller MSE's. In fact, only for the Friedman 2 data, the automatic choice of b is significantly worse than the optimal choice (p-value=0.03), whereas the benefits at the contaminated data are large (all p-values $< 10^{-4}$).

9.7. Conclusion

Heuristic links between the concept of the influence function and concepts as leave-one-out cross validation and stability were considered in Section 9.2, indicating some interesting applications of the influence function and the leave-one-out error in previous literature. New results include the calculation of higher order influence functions and a recursive relation between subsequent terms. It is shown that these theoretical results can be applied in practice to approximate the leave-one-out estimator. Experiments indicate that the quality of this approximation is quite good. The approximation is used in a model selection criterion to select the regularization and kernel parameters.

We discussed the importance of robustness in the model selection step. A specific procedure is suggested using an L_1 loss in the model selection criterion and a Huber loss in the estimation. Due to an iterative reweighting algorithm to compute such a Huber loss estimator and due to the fast approximation of the leave-one-out error, everything can be computed fast starting from the least squares framework. With an a priori choice of the parameter b in the Huber loss this leads to better robustness if b is chosen small enough. If b is chosen too small on the other hand this might result in worse predictions. However, this parameter can be selected in a data driven way as well. Experiments suggest that this often yields a good trade-off between the robustness of choosing a small b and the sometimes better predictive capacity of least squares.

Table 9.1: Simulation results. Upper: Mean Squared Errors. Lower: standard errors. Friedman 1 (F1), Friedman 2 (F2), Friedman 3 (F3), Boston Housing (B), Ozone (O), Servo (S), Boston Housing with outliers (B+o), Ozone with outliers (O+o) and Servo with outliers (S+o). Italic values are significantly different from the smallest value in the row with p-value in between 0.05 and 0.001 using a paired Wilcoxon rank test; underlined values are significant with p-value $< 10^{-4}$.

	$b = \infty$ (=LS)			$b = 3\hat{\sigma}_{err}$	$b = 2\hat{\sigma}_{err}$	$b = \hat{\sigma}_{err}$	($b =$ optimized)
	LOO	CV	C^5_{IF}	C^5_{IF}	C^5_{IF}	C^5_{IF}	C^5_{IF}
F1	1.63	1.63	1.63	1.66	*1.70*	*1.82*	1.67
F2	1.30	1.30	1.30	*1.42*	$\underline{1.71}$	$\underline{3.02}$	*1.39*
F3	2.42	2.42	2.42	2.42	2.42	2.37	2.38
B	10.58	10.58	10.58	10.82	11.30	*12.21*	10.79
O	13.91	13.92	13.91	13.76	13.73	13.91	13.94
S	0.40	0.40	0.40	0.43	0.41	0.41	0.40
B+o	$\underline{37.54}$	$\underline{37.54}$	$\underline{37.54}$	$\underline{14.60}$	$\underline{13.73}$	12.68	12.78
O+o	$\underline{78.78}$	$\underline{78.78}$	$\underline{78.77}$	$\underline{21.20}$	$\underline{18.85}$	16.74	16.74
S+o	$\underline{1.60}$	$\underline{1.60}$	$\underline{1.60}$	$\underline{0.61}$	$\underline{0.54}$	0.46	0.46

	$b = \infty$ (=LS)			$b = 3\hat{\sigma}_{err}$	$b = 2\hat{\sigma}_{err}$	$b = \hat{\sigma}_{err}$	($b =$ optimized)
	LOO	CV	C^5_{IF}	C^5_{IF}	C^5_{IF}	C^5_{IF}	C^5_{IF}
F1	0.09	0.09	0.09	0.09	0.10	0.08	0.09
F2	0.14	0.14	0.15	0.16	0.20	0.36	0.15
F3	0.03	0.03	0.03	0.03	0.03	0.05	0.05
B	1.39	1.39	1.39	1.40	1.46	1.51	1.39
O	0.86	0.86	0.87	0.78	0.78	0.75	0.81
S	0.05	0.05	0.05	0.09	0.08	0.09	0.09
B+o	2.91	2.91	2.91	1.12	1.09	1.02	1.04
O+o	3.44	3.44	3.44	1.01	0.97	1.03	1.03
S+o	0.16	0.16	0.16	0.07	0.07	0.08	0.08

Acknowledgments

JS acknowledges support from K.U. Leuven, GOA-Ambiorics, CoE EF/05/006, FWO G.0499.04, FWO G.0211.05, FWO G.0302.07, IUAP P5/22.

MH acknowledges support from FWO G.0499.04, the GOA/07/04-project of the Research Fund KULeuven, and the IAP research network nr. P6/03 of the Federal Science Policy, Belgium.

Appendix A.

Proof of Theorem 9.6

Let P be a distribution, $z \in \mathscr{X} \times \mathscr{Y}$ and $P_{\varepsilon,z} = (1-\varepsilon)P + \varepsilon\Delta_z$ with Δ_z the Dirac distribution in z. We start from the representer theorem of DeVito et al. (2004) (a generalization of (9.13)):

$$2\lambda f_{\lambda,K,P_{\varepsilon,z}} = \mathbb{E}_{P_{\varepsilon,z}}[L'(Y - f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)].$$

By definition of $P_{\varepsilon,z}$ and since $\mathbb{E}_{\Delta_z} g(X) = g(z)$ for any function g:

$$2\lambda f_{\lambda,K,P_{\varepsilon,z}} = (1-\varepsilon)\mathbb{E}_P[L'(Y - f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)] + \varepsilon L'(z_y - f_{\lambda,K,P_{\varepsilon,z}}(z_x))\Phi(z_x).$$

Taking the first derivative on both sides with respect to ε yields

$$\begin{aligned}2\lambda \frac{\partial}{\partial\varepsilon} f_{\lambda,K,P_{\varepsilon,z}} =&(1-\varepsilon)\mathbb{E}_P[-\frac{\partial}{\partial\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(X) L''(Y - f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)]\\ &-\mathbb{E}_P[L'(Y - f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)] + L'(z_y - f_{\lambda,K,P_{\varepsilon,z}}(z_x))\Phi(z_x)\\ &-\varepsilon\frac{\partial}{\partial\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(z_x)) L''(z_y - f_{\lambda,K,P_{\varepsilon,z}}(z_x))\Phi(z_x).\end{aligned}$$

The second derivative equals

$$\begin{aligned}2\lambda \frac{\partial}{\partial^2\varepsilon} f_{\lambda,K,P_{\varepsilon,z}} =& -\mathbb{E}_P[-\frac{\partial}{\partial\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(X) L''(Y - f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)]\\ &+(1-\varepsilon)\mathbb{E}_P[-\frac{\partial}{\partial^2\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(X) L''(Y - f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)]\\ &+(1-\varepsilon)\mathbb{E}_P[-\frac{\partial}{\partial\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(X) L'''(Y - f_{\lambda,K,P_{\varepsilon,z}}(X))(-\frac{\partial}{\partial\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)]\\ &-\mathbb{E}_P[L''(Y - f_{\lambda,K,P_{\varepsilon,z}}(X))(-\frac{\partial}{\partial\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)]\\ &-\frac{\partial}{\partial\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(z_x) L''(z_y - f_{\lambda,K,P_{\varepsilon,z}}(z_x))\Phi(z_x)\\ &-\varepsilon\frac{\partial}{\partial^2\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(z_x) L''(z_y - f_{\lambda,K,P_{\varepsilon,z}}(z_x))\Phi(z_x)\\ &-\varepsilon\frac{\partial}{\partial\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(z_x) L'''(z_y - f_{\lambda,K,P_{\varepsilon,z}}(z_x))(-\frac{\partial}{\partial\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(z_x))\Phi(z_x)\\ &-L''(z_y - f_{\lambda,K,P_{\varepsilon,z}}(z_x))\frac{\partial}{\partial\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(z_x)\Phi(z_x).\end{aligned}$$

Simplifying yields

$$\begin{aligned}2\lambda\frac{\partial}{\partial^2\varepsilon}f_{\lambda,K,P_{\varepsilon,z}} =&2\mathbb{E}_P[\frac{\partial}{\partial\varepsilon}f_{\lambda,K,P_{\varepsilon,z}}(X)L''(Y-f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)] \\ &-(1-\varepsilon)\mathbb{E}_P[\frac{\partial}{\partial^2\varepsilon}f_{\lambda,K,P_{\varepsilon,z}}(X)L''(Y-f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)] \\ &+(1-\varepsilon)\mathbb{E}_P[\left(\frac{\partial}{\partial\varepsilon}f_{\lambda,K,P_{\varepsilon,z}}(X)\right)^2 L'''(Y-f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)] \\ &-2\frac{\partial}{\partial\varepsilon}f_{\lambda,K,P_{\varepsilon,z}}(z_x)L''(z_y-f_{\lambda,K,P_{\varepsilon,z}}(z_x))\Phi(z_x) \\ &-\varepsilon\frac{\partial}{\partial^2\varepsilon}f_{\lambda,K,P_{\varepsilon,z}}(z_x)L''(z_y-f_{\lambda,K,P_{\varepsilon,z}}(z_x))\Phi(z_x) \\ &+\varepsilon\left(\frac{\partial}{\partial\varepsilon}f_{\lambda,K,P_{\varepsilon,z}}(z_x)\right)^2 L'''(z_y-f_{\lambda,K,P_{\varepsilon,z}}(z_x))\Phi(z_x).\end{aligned} \tag{9.23}$$

Evaluating at $\varepsilon = 0$ and bringing all terms containing $\frac{\partial}{\partial^2\varepsilon}f_{\lambda,K,P_{\varepsilon,z}}$ to the left hand side of the equation yields

$$\begin{aligned}2\lambda\frac{\partial}{\partial^2\varepsilon}f_{\lambda,K,P_{\varepsilon,z}}|_{\varepsilon=0}+\mathbb{E}_P[\frac{\partial}{\partial^2\varepsilon}f_{\lambda,K,P_{\varepsilon,z}}(X)|_{\varepsilon=0}L''(Y-f_{\lambda,K,P}(X))\Phi(X)]& \\ =2\mathbb{E}_P[\frac{\partial}{\partial\varepsilon}f_{\lambda,K,P_{\varepsilon,z}}(X)|_{\varepsilon=0}L''(Y-f_{\lambda,K,P}(X))\Phi(X)]& \\ +\mathbb{E}_P[\left(\frac{\partial}{\partial\varepsilon}f_{\lambda,K,P_{\varepsilon,z}}|_{\varepsilon=0}(X)\right)^2 L'''(Y-f_{\lambda,K,P}(X))& \\ -2\frac{\partial}{\partial\varepsilon}f_{\lambda,K,P}(z_x)|_{\varepsilon=0}L''(z_y-f_{\lambda,K,P}(z_x))\Phi(z_x).&\end{aligned}$$

Since by definition $\frac{\partial}{\partial\varepsilon}f_{\lambda,K,P_{\varepsilon,z}}|_{\varepsilon=0}$ is $IF(z;f_{\lambda,K},P)$ and $\frac{\partial}{\partial^2\varepsilon}f_{\lambda,K,P_{\varepsilon,z}}|_{\varepsilon=0}$ is $IF_2(z;f_{\lambda,K},P)$ we have that

$$\begin{aligned}S(IF_2(z;f_{\lambda,K},P)) = 2\mathbb{E}_P[IF(z;f_{\lambda,K},P)(X)L''(Y-f_{\lambda,K,P}(X))\Phi(X)]& \\ +\mathbb{E}_P[\left(IF(z;f_{\lambda,K},P)(X)\right)^2 L'''(Y-f_{\lambda,K,P}(X))& \\ -2IF(z;f_{\lambda,K},P)(z_x)L''(z_y-f_{\lambda,K,P}(z_x))\Phi(z_x)&\end{aligned}$$

with the operator S defined by $S: f \to \lambda f + \mathbb{E}_P L''(Y-f_{\lambda,K,P}(X))f(X)\Phi(X)$. Christmann and Steinwart (2007) prove that S is an invertible operator and thus Theorem 9.6 follows.

Proof of Theorem 9.7

First we proof the following for all $2 \leq k \in \mathbb{N}$:

$$\begin{aligned}2\lambda\frac{\partial}{\partial^k\varepsilon}f_{\lambda,K}(P_{\varepsilon,z}) =&(1-\varepsilon)\mathbb{E}_P[-\frac{\partial}{\partial^k\varepsilon}f_{\lambda,K,P_{\varepsilon,z}}(X)L''(Y-f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)] \\ &+k\mathbb{E}_P[\frac{\partial}{\partial^{k-1}\varepsilon}f_{\lambda,K,P_{\varepsilon,z}}(X)L''(Y-f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)] \\ &-kL''(z_y-f_{\lambda,K,P_{\varepsilon,z}}(z_x))\frac{\partial}{\partial^{k-1}\varepsilon}f_{\lambda,K,P_{\varepsilon,z}}(z_x)\Phi(z_x) \\ &-\varepsilon L''(z_y-f_{\lambda,K,P_{\varepsilon,z}}(z_x))\frac{\partial}{\partial^k\varepsilon}f_{\lambda,K,P_{\varepsilon,z}}(z_x)\Phi(z_x).\end{aligned} \tag{9.24}$$

Note that for $k = 2$ this immediately follows from (9.23). For general k we give a proof by induction. We assume that (9.24) holds for k and we then prove that it automatically holds for $k+1$ as well. Taking the derivatives of both sides in (9.24) we find

$$\begin{aligned}
\lambda \frac{\partial}{\partial^{k+1}\varepsilon} f_{\lambda,K}(P_{\varepsilon,z}) =& (1-\varepsilon)\mathbb{E}_P[-\frac{\partial}{\partial^{k+1}\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(X)L''(Y - f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)] \\
&- \mathbb{E}_P[-\frac{\partial}{\partial^{k}\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(X)L''(Y - f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)] \\
&+ k\mathbb{E}_P[\frac{\partial}{\partial^{k}\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(X)L''(Y - f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)] \\
&- k\frac{\partial}{\partial^{k}\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(z_x)L''(z_y - f_{\lambda,K,P_{\varepsilon,z}}(z_x))\Phi(z_x) \\
&- \varepsilon\frac{\partial}{\partial^{k+1}\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(z_x)L''(z_y - f_{\lambda,K,P_{\varepsilon,z}}(z_x))\Phi(z_x) \\
&- \frac{\partial}{\partial^{k}\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(z_x)L''(z_y - f_{\lambda,K,P_{\varepsilon,z}}(z_x))\Phi(z_x)
\end{aligned}$$

from which it follows that (9.24) holds for $k+1$ indeed. Evaluating this expression in $\varepsilon = 0$ yields:

$$\begin{aligned}
&\lambda \frac{\partial}{\partial^{k+1}\varepsilon} f_{\lambda,K}(P_{\varepsilon,z})|_{\varepsilon=0} + \mathbb{E}_P[\frac{\partial}{\partial^{k+1}\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(X)|_{\varepsilon=0}L''(Y - f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)] \\
&= (k+1)\mathbb{E}_P[\frac{\partial}{\partial^{k}\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}(X)|_{\varepsilon=0}L''(Y - f_{\lambda,K,P_{\varepsilon,z}}(X))\Phi(X)] \\
&- (k+1)\frac{\partial}{\partial^{k}\varepsilon} f_{\lambda,K,P_{\varepsilon,z}}|_{\varepsilon=0}(z_x)L''(z_y - f_{\lambda,K,P_{\varepsilon,z}}(z_x))\Phi(z_x).
\end{aligned}$$

Thus

$$\begin{aligned}
S(IF_{k+1}(z;f_{\lambda,K},P)) = (k+1)\Big(&\mathbb{E}_P[IF_k(z;f_{\lambda,K},P)(X)L''(Y - f_{\lambda,K}(X))\Phi(X)] \\
&- [IF_k(z;f_{\lambda,K},P)(z_x)L''(z_y - f_{\lambda,K}(z_x))\Phi(z_x)]\Big).
\end{aligned}$$

Since S is an invertible operator the result in Theorem 9.7 follows.

References

D.D. Boos and R.J. Serfling. A note on differentials and the CLT and LIL for statistical functions. *Annals of Statistics*, 8:618–624, 1980.

O. Bousquet and A. Elisseeff. Stability and generalization. *Journal of Machine Learning Research*, 2:499–526, 2001.

A. Christmann and I. Steinwart. On robust properties of convex risk minimization methods for pattern recognition. *Journal of Machine Learning Research*, 5:1007–1034, 2004.

A. Christmann and I. Steinwart. Consistency and robustness of kernel based regression. *Bernoulli*, 13:799–819, 2007.

M. Debruyne, A. Christmann, M. Hubert, and J.A.K. Suykens. Robustness and stability of reweighted kernel based regression. *Journal of Multivariate Analysis*, 101:447–463, 2010.

E. DeVito, L. Rosasco, A. Caponnetto, M. Piana, and A. Verri. Some properties of regularized kernel methods. *Journal of Machine Learning Research*, 5:1363–1390, 2004.

T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13:1–50, 2000.

L.T. Fernholz. *Von Mises Calculus for Statistical Functionals*. Lecture Notes in statistics 19, Springer, New York, 1983.

F.R. Hampel, E.M. Ronchetti, P.J. Rousseeuw, and W.A. Stahel. *Robust Statistics: The Approach Based on Influence Functions*. Wiley, New York, 1986.

P.J. Huber. *Robust Statistics*. Wiley, New York, 1981.

S. Kutin and P. Niyogi. Almost everywhere algorithmic stability and generalization error. In A. Daruich and N. Friedman, editors, *Proceedings of Uncertainty in AI*. Morgan Kaufmann, Edmonton, 2002.

T. Poggio, R. Rifkin, S. Mukherjee, and P. Niyogi. General conditions for predictivity in learning theory. *Nature*, 428:419–422, 2004.

J.A.K. Suykens, J. De Brabanter, L. Lukas, and J. Vandewalle. Weighted least squares support vector machines : Robustness and sparse approximation. *Neurocomputing*, 48:85–105, 2002a.

J.A.K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines*. World Scientific, Singapore, 2002b.

A.N. Tikhonov and V.Y. Arsenin. *Solutions of Ill Posed Problems*. W.H. Winston, Washington D.C., 1977.

G. Wahba. *Spline Models for Observational Data*. CBMS-NSF Regional Conference Series in Applied Mathematics, SIAM, 1990.

Part IV

Ensemble Methods

Overview

Ensemble methods, to a large extent, circumvent the problem of hyperparameter selection, by averaging the predictions of many models. In recent challenges we have organized, **ensemble methods always demonstrated excellent performances.** The overall winner of both the performance prediction challenge and the ALvsPK challenge (agnostic track), **Roman Lutz, uses a serial ensemble called Logitboost** (see Fact sheet in Appendix B), a gradient-based shallow tree-classifier "boosting" method, performing logistic regression. **Boosting methods are serial ensembles** incorporating progressively classifiers, which reduce the residual error by focusing on remaining misclassified examples. In contrast, **"bagging" methods yield parallel ensembles,** all members of the ensemble being trained independently on a bootstrap sample of the training data, drawn at random with replacement. In Chapter 10, Corine Dahinden reports results obtained with Random Forest, an parallel ensemble method based on "bagging" tree classifiers, which obtained fourth place overall in the performance prediction challenge. In Chapter 11, **Eugene Tuv, Alexander Borisov, George Runger, and Kari Torkkola propose a combination of parallel and serial ensembles** based on gradient tree boosting and on bagging feature selection. The authors obtained eighth place in the performance prediction challenge and second place in the ALvsPK challenge (agnostic track). In Chapter 12, **Vladimir Nikulin explores the methods of boosting and bagging with other predictors than decision trees,** including least-square kernel methods and naive Bayes and performs a rather extensive method comparison. His methods allowed him to win first place in the ALvsPK challenge (prior knowledge track).

Chapter 10

An Improved Random Forests Approach with Application to the Performance Prediction Challenge Datasets

Corinne Dahinden DAHINDEN@STAT.MATH.ETHZ.CH
Seminar für Statistik
CH-8092 Zürich, Switzerland

Editor: Isabelle Guyon

Abstract

Random Forests is a popular ensemble technique developed by Breiman (2001) which yields exceptional performance. These excellent results are achieved with little need to fine-tune parameters. The method is computationally effective, does not overfit, is robust to noise and can also be applied when the number of variables is much larger than the number of samples. We propose a slightly modified Random Forests scheme, with cross-validation as a means for tuning parameters and estimating error-rates. This simple and computationally very efficient approach was found to yield better predictive performance on the WCCI 2006 Performance Prediction Challenge datasets than many algorithms of much higher complexity.

Keywords: Random Forests, Ensemble Methods

10.1. Introduction

During the last couple of years, an overwhelming variety of new classification algorithms have been developed. It was, and still is a rapidly evolving field, as the constant influx of new applications creates a need for novel prediction approaches. Many of these methods focus on special cases, as for example in Bioinformatics, where the small m - large n phenomenon, i.e. many features but few samples, drew a lot of attention. Just for this particular application, dozens of algorithms are praised to perform well, but the existence of so many possible classifiers also makes it hard to keep track and choose the best one. That said, neither is there a widely accepted consensus on a method that performs well on a wide range of problems from different applications. Of course we cannot expect a single algorithm to work optimally on all conceivable applications and moreover, tuning and tailoring classifiers for a special task at hand is difficult, and mostly requires laborious human interaction. Nevertheless, we regard it as desirable to have an algorithm with relatively few hyperparameters, a minimal requirement of human input, good predictive performance on a wide array of datasets from different fields and low computational cost.

As far as predictive performance is concerned, in recent years, a number of works have reported that ensembles of base learners exhibit substantial performance improvement over single base learners. The resulting classifiers, referred to as ensemble classifiers, are the aggregation of classifiers whose individual decisions are combined by weighted or unweighted voting to classify new samples. Bagging (Breiman, 1996) and Boosting (Freund and Schapire, 1996) are well-known and popular representatives of this methodology.

While examining the datasets for the WCCI 2006 Performance Prediction Challenge, we tried a number of different algorithms and observed that ensemble techniques outperformed other methods by far.

It might be surprising that our final decision fell on a relatively simple algorithm, namely Random Forests (Breiman, 2001) with small adaptations. This method performs very well, if not best, on all five datasets of the Challenge which were our test datasets. Random Forests has shown its success on many applications, and was a very strong competitor in the NIPS 2003 Challenge as well (Saffari, 2006).

10.2. Algorithm

We put our main focus on finding a classifier with good predictive power in the first instance and estimated the prediction performance simply by cross-validation.

Predictive power is measured in terms of the balanced error rate (BER) which is the average of the error rates in each class. The datasets under consideration have binary response variables $y_k \in \{-1,1\}, k \in \{1,\dots,m\}$ and the response vector is denoted by Y. The input variables are stored in a matrix $X \in \mathbb{R}^{m\times n}$. The rows consist of the m samples $\boldsymbol{x}_k \in \mathbb{R}^n$ and the columns of the n different variables. A new observation to be classified is denoted by $\boldsymbol{x} \in \mathbb{R}^n$. We apply Random Forests, an ensemble method, that combines the output from multiple base learners to improve the performance by using their weighted outcome to predict new data. It can easily handle large numbers of input variables and provides an importance measure for each variable, making it also suitable for variable selection.

10.2.1. Classification and Regression Trees – CART

Random Forests are made up of an ensemble of decision trees. A prominent method for fitting trees is *CART*, an acronym for *Classification and Regression Tree* (Breiman et al., 1984). Trees are iteratively built by recursively partitioning the data into regions $R_1,\dots,R_M$, so-called nodes. The procedure is graphically displayed on the basis of a two-dimensional feature space in Figures 10.1 and 10.2 below. The decision which node R_t is further split at which variable j depends on the value of a specific splitting criteria, a so-called impurity measure. For classification, CART uses the Gini-index as impurity measure. For binary response variables, the Gini-index at a node R_t is calculated by the following expression:

$$I_G(t) = \sum_{i=1}^{2} \hat{p}_{ti}(1-\hat{p}_{ti}),$$

where $\hat{p}_{ti}$ is the probability of class i estimated from the samples in node t, e.g. if the node R_t represents a region with N_t observations, $\hat{p}_{ti} = \frac{1}{N_t}\sum_{\boldsymbol{x}_i \in R_t} I(y_i = i)$ is the estimation of this probability. At each step in the algorithm, the node R_t is split at the variable j and split point s into a pair of half-planes, so-called subnodes $R_{tL} = \{\boldsymbol{x} \in R_t | x_j \leq s\}$ and $R_{tR} = \{\boldsymbol{x} \in R_t | x_j \geq s\}$. The variable j and the split point s are chosen to maximize the decrease in Gini-index. The decrease in the Gini-index due to a split on variable j and node t is

$$\Delta I_G(x_j,t) = I_G(t) - \hat{p}_{tL} I_G(tL) - \hat{p}_{tR} I_G(tR).$$

To prevent the decision tree from overfitting the data, the growth of the tree has to be halted at a predefined stopping criterion or alternatively, the tree can be grown to maximum depth and then pruned. To illustrate the procedure, we assume a two-dimensional feature space. First, the feature space is divided into two rectangles R_1 and R_2 at the split point s_1 (see left side of

Figure 10.1). The feature space is then recursively partitioned into a set of rectangles $R_1 \ldots R_5$ (see right side of Figure 10.1). The final model can be graphically represented in a tree representation such as can be seen in Figure 10.2.

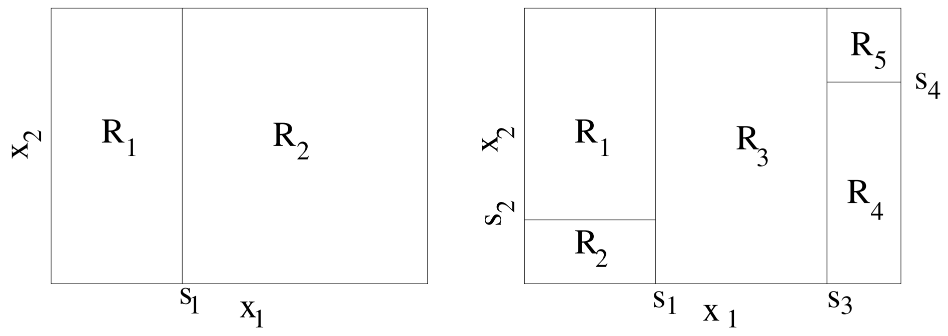


Figure 10.1: Left: The first split is performed at variable x_1 at split point s_1 and the feature space is divided into two rectangles R_1 and R_2. Right: Recursive partition of feature space in a set of rectangles.

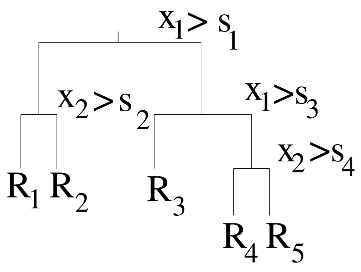


Figure 10.2: Tree representation of the final CART model.

For classifying a new observation, the input vector is put down the tree. This means that we check to which rectangle R_j the new observation belongs to (see e.g. Figure 10.2). Within that rectangle the classification is performed by majority voting. This means that the most frequent label within that rectangle is assigned to the new observation. A more comprehensive description on how to build classification trees is given in Breiman et al. (1984).

10.2.2. Random Forests

Random Forests grows a number of such classification trees. Each tree is grown as follows:

1. A tree of maximal depth is grown on a bootstrap sample of size s of the training set. This means that we sample s observations with replacement from the training set and fit a tree using this so-called *bootstrap sample*. No pruning is performed.

2. A number s which is much smaller than the total number of variables $s \ll n$ (typically $\sqrt{n}$) is specified, such that at each node, s variables are sampled at random out of the n. The best split on these variables is used to split the node into two subnodes.

The final classification is given by majority voting of the ensemble of trees in the forest. In contrast to bagging, an additional layer of randomness is included in step 2 of the algorithm above. Instead of just constructing trees of different bootstrap samples, step 2 changes the way the individual trees are constructed, namely at each node the splitting variable is not chosen among *all* variables but the best possible split among a random *subset of variables* is performed.

For our calculations we used the statistical software R (see R Development Core Team, 2007) and the package *randomForest* by Liaw and Wiener (2002). There are basically two tuning parameters to adjust for in the Random Forests function in R: The number of trees to grow and the number of possible splitting variables which are sampled at each node. However, as is mentioned in Liaw and Wiener (2002), we have also observed that the sensitivity to those is minimal and the default values are a good choice.

10.2.3. Adaptation of Random Forests

Our experience based on the application of Random Forests to the Performance Prediction Challenge datasets is that the method performs very well using the R implementation with default values. It has also been shown on other datasets that Random Forests is superior to compared classifiers (see for example Breiman, 2001). However, especially for very unbalanced datasets, there is some potential for improvement, as we noticed while working on the Challenge datasets and was also observed by others (see for example Dudoit and Fridlyand, 2002). Although Random Forests has options to balance the error rates in unbalanced settings, that is either the introduction of additional class weighting parameters or sampling techniques such as down-sampling the majority class or over-sampling the minority class, the corresponding outputs did not fully satisfy us. To improve the performance, we use a slight adaptation of the above algorithm described below.

Random Forests assigns probabilities $\hat{p}_i$ to observations. These are the probabilities that the observation belongs the class i. These probabilities are calculated by the fraction of votes for the corresponding class in the training data. Since the BER is used as performance measure, erroneous prediction in the minority class is penalized harder than a misclassification in the majority class. Therefore the cutoff is lower than 0.5, so that doubtful samples are more likely assigned to the class with fewer observations. Theoretically, for unbiased probability estimates, it is optimal to set the cutoff equal to the proportion of samples in the minority class, e.g. for the Ada dataset, a sample is classified as +1 if $\hat{p}_1 > 0.248$, where 0.248 is the fraction of samples belonging to the minority class. For reasonably balanced datasets, like Ada with a fraction of 1/4 belonging to the minority class, this works very well with standard Random Forests. However, our experience is that for very unbalanced settings, it proves beneficial to assign an observation to the minority class already at a lower cutoff than this proportion. We optimize this cutoff by cross-validation (see Figure 10.3). This adaptation improves the cross-validated balanced error rate for unbalanced datasets considerably, while the impact for balanced datasets is low (see Figure 10.4 and Tables 10.1 and 10.2). The reason why this cutoff optimization is necessary might be that Random Forests leads to biased probability estimates in very unbalanced settings. To estimate the accuracy, 10-fold cross-validation is performed. One could also use the out-of-bag samples (those which are left out while fitting the tree) to perform the cutoff-optimization and then do cross-validation using the optimized cutoff. To estimate the performance prediction, which was also part of the Challenge's scope, we used this cross-validated estimation of the error rate.

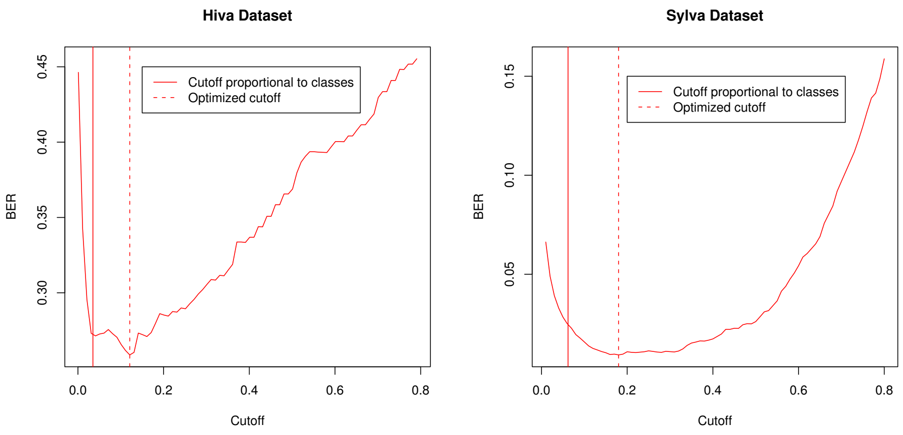


Figure 10.3: Cross-Validated BER in dependence of the cutoff value for imbalanced datasets.

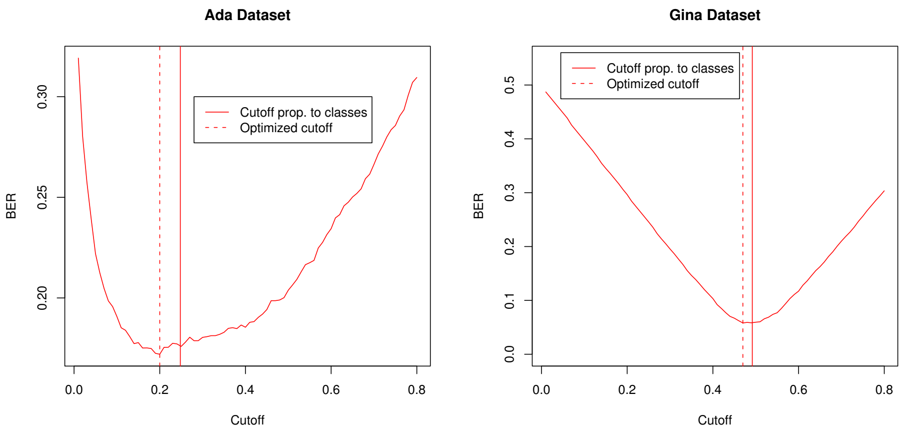


Figure 10.4: Cross-Validated BER in dependence of the cutoff value for balanced datasets.

10.3. Results

The results of the application of the standard Random Forests procedure without cutoff-adaptation are summarized in Table 10.1. Table 10.2 yields the results with cutoff-adaptation. *Cutoff* stands for the estimated optimal cutoff, calculated by cross-validation. In a second cross-validation step on the training set, the estimation for the error rate *CV BER* is calculated with the cutoff delivered by the first cross-validation. In addition, for the Nova dataset which originally includes 16969 features and 1754 samples, we reduced the feature dimensionality to 400 by just using the first 400 principal components. For the other datasets, no preprocessing was needed. We used 4000 trees for all datasets. As far as computing time is concerned, it took approximately 2 minutes to fit the Random Forests model to the Ada dataset with 48 features and 4147 observations and 2 hours to fit the forest to the Hiva dataset with 1617 features and 3845 observations. The calculations were made on a dual core AMD Opteron 2.6 GHz with 32 GB RAM.

Table 10.1: Results without cutoff-adaptation.

Dataset	CV BER	BER on test set
Ada	0.174	0.191
Gina	0.056	0.049
Hiva	0.272	0.291
Nova	0.083	0.084
Sylva	0.0191	0.0250

Table 10.2: Results with cutoff-adaptation.

Dataset	Theoretical Cutoff	Estimated Cutoff	CV BER	BER on test set
Ada	0.248	0.20	0.165	0.180
Gina	0.492	0.46	0.049	0.041
Hiva	0.035	0.13	0.270	0.299
Nova	0.285	0.34	0.053	0.053
Sylva	0.062	0.19	0.0065	0.0054

One can clearly see the improvement of the cross-validated error rate, if the cutoff is optimized. In addition, the balanced error rate on the test set is listed. The number of samples in the test set is approximately ten times the sample size of the training set.

In addition, in Table 10.3 our results are directly compared to the Challenge's best entries. We clearly see that the adapted Random Forests procedure works fairly well on *all* dataset. Even though it has no top ranking entry, its performance seems to be good for a wide range of problems without further individual dataset adaptations.

10.4. Conclusions

Our research in the course of the WCCI 2006 Performance Prediction Challenge, left us with the experience that Random Forests seems to keep up even with the most sophisticated algorithms, as far as the predictive performance is concerned. Applying plain standard Random Forests to the five WCCI 2006 Performance Prediction Challenge datasets leads to very competitive prediction results. However, we suggest a novel extension where the cutoff parameter is tuned by cross-validation, instead of just using a fixed threshold that is proportional to the fraction of samples in the minority class. The tuning is done by cross-validation and can be fully automated. We have shown that this leads to a considerable performance improvement on the Challenge datasets, especially for very unbalanced data. In addition, Random Forests only requires small computing efforts, can deal with many input variables and only needs a minimum of human interaction in the fitting process to perform well.

Table 10.3: Comparison of our best entries with the Challenge's best entries.

Dataset	Our best entry				
	Test AUC	Test BER	BER guess	guess error	Test score rank
Ada	0.8200	0.1800	0.1650	0.0150	0.1950 (16)
Gina	0.9587	0.0413	0.0490	0.0077	0.0490 (17)
Hiva	0.7009	0.2994	0.2700	0.0294	0.3284 (32)
Nova	0.9470	0.0530	0.0530	0.0000	0.0530 (15)
Sylva	0.9946	0.0054	0.0065	0.0011	0.0065 (3)
Overall	0.8842	0.1158	0.1087	0.0106	16.6

Dataset	The Challenge's best entry				
	Test AUC	Test BER	BER guess	guess error	Test score rank
Ada	0.9149	0.1723	0.1650	0.0073	0.1793 (1)
Gina	0.9712	0.0288	0.0305	0.0017	0.0302(1)
Hiva	0.7671	0.2757	0.2692	0.0065	0.2797 (1)
Nova	0.9914	0.0445	0.0436	0.0009	0.0448 (1)
Sylva	0.9991	0.0061	0.0060	0.0001	0.0062 (1)
Overall	0.8910	0.1090	0.1040	0.0079	6.2

Acknowledgments

I would like to thank Professor Peter Bühlmann for the encouragement and the helpful comments.

References

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Statistics/Probability Series. Wadsworth Publishing Company, Belmont, California, U.S.A., 1984.

Sandrine Dudoit and Jane Fridlyand. *Statistical Analysis of Gene Expression Microarray Data*, chapter Classification in microarray experiments. CRC Press, 2002.

Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3): 18–22, 2002. URL http://CRAN.R-project.org/doc/Rnews/.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. URL http://www.R-project.org. ISBN 3-900051-07-0.

Amir Saffari. *Variable Selection using Correlation and Single Variable Classifier Methods: Applications*, chapter Feature Extraction: Foundations and Applications, pages 343–358. Springer-Verlag, 2006.

Journal of Machine Learning Research 10(Jul):1341–1366, 2009 Submitted 5/08; Revised 3/09; Published 6/09

Chapter 11

Feature Selection with Ensembles, Artificial Variables, and Redundancy Elimination

Eugene Tuv EUGENE.TUV@INTEL.COM
Intel, Logic Technology Development
Chandler, AZ, USA

Alexander Borisov ALEXANDER.BORISOV@INTEL.COM
Intel, Logic Technology Development
N.Novgorod, Russia

George Runger RUNGER@ASU.EDU
Arizona State University
Tempe, AZ, USA

Kari Torkkola KARITO@AMAZON.COM
Amazon.com
Seattle, WA, USA

Editor: Isabelle Guyon and Amir Reza Saffari

Abstract

Predictive models benefit from a compact, non-redundant subset of features that improves interpretability and generalization. Modern data sets are wide, dirty, mixed with both numerical and categorical predictors, and may contain interactive effects that require complex models. This is a challenge for filters, wrappers, and embedded feature selection methods. We describe details of an algorithm using tree-based ensembles to generate a compact subset of non-redundant features. Parallel and serial ensembles of trees are combined into a mixed method that can uncover masking and detect features of secondary effect. Simulated and actual examples illustrate the effectiveness of the approach.

Keywords: trees, resampling, importance, masking, residuals

11.1. Introduction

Large data sets are becoming the norm and traditional methods designed for data sets with a modest number of features will struggle in the new environment. This problem area was described by Guyon and Elisseeff (2003) along with other publications in the same issue, and it has increased in importance since then. Additional comments and examples have been provided by Liu and Yu (2005) in a recent survey article.

11.1.1. Feature Selection

There are three major categories of feature selection methods. Filter methods score variables, typically individually, and eliminate some before a model is constructed. The filter needs to be generated carefully to relate well to the requirements of the modeling task. In particular, the filter may not consider the value of one variable in the presence of others. For example, the widely-used value difference metric (VDM) (Stanfill and Waltz, 1986) and its modified version

(MVDM) (Cost and Salzberg, 1993) consider the conditional probability distribution of the response at a predictor value. Such a measure is not sensitive to the effects of some predictors in a model with others present even though interactions among predictors might be critical for an effective subset. A sequential, subset search is sometimes implemented to improve the performance when interactions are important, although a greedy search also has disadvantages in the presence of interactions. Several common filter methods such as ReliefF (Robnik-Sikonja and Kononenko, 2003), CFS (Hall, 2000), and FOCUS (Almuallin and Dietterich, 1994) were modified with sequential search and compared by Yu and Liu (2004).

Wrapper methods form a second group of feature selection methods. The prediction accuracy (or the change in accuracy) of a model directly measures the value of a feature set. Although effective, the exponential number of possible subsets places computational limits for the wide data sets that are the focus of this work.

Embedded methods form a third group for feature selection. These methods use all the variables to generate a model and then analyze the model to infer the importance of the variables. Consequently, they directly link variable importance to the learner used to model the relationship.

11.1.2. Subset Feature Selection

Fundamentally, the goal of feature selection is to model a target response (or output) variable y, with a subset of the (important) predictor variables (inputs). This is a general goal and several more specific objectives can be identified. Each can lead to different strategies and algorithms. In *filtering* the interest is to remove irrelevant variables. Another objective is *variable ranking* where the interest is in obtaining relative relevance for all input variables with respect to the target. Finally, we might be interested in a compact, yet effective model, where the goal is to identify the smallest subset of independent features with the most predictive power, although a few alternative groups might be reasonable. An important concept here is *the masking relationships* among the predictor variables. Masking occurs when one variable can effectively represent others in a model. Along with the related issue of masking, this paper focuses on the subset selection.

11.1.3. Contributions of this Paper

Existing tree ensembles such as random forest (Breiman, 2001) or gradient boosting trees (Friedman, 1999) were developed primarily for predictive modeling. In addition, they can provide an importance ranking of the features, but this information has been considered an ad hoc benefit. Random forest (RF) is a random subspace method, and is capable of efficiently ranking features for large data sets. We exploit this property of RF, augment the original data with *artificial contrast variables* constructed independently from the target, and use their ranking for removal of irrelevant variables from the original set. The tree construction method is also modified to produce a more reliable variable ranking in the presence of high cardinality variables. A variable masking measure is then introduced that incorporates surrogate variable scores from ensembles of trees. This forms the basis for *redundancy elimination*. Residual effects are calculated to enable the method to detect variables of secondary importance. These elements are integrated into an efficient algorithm for subset selection called ACE (artificial contrasts with ensembles).

The structure of this paper is as follows. In Section 11.2 we describe previous work and outline directions taken in this paper. Section 11.3 describes variable importance measures defined through tree ensembles and explains how they could be used to remove irrelevant features using random, artificial features. Next, we introduce a masking measure and use it for redundancy elimination. Section 11.4 describes the details of the ACE algorithm to generate the selected subset, and compares ACE with its closest competitors in detail. Section 11.5 provides results from experiments. Section 11.6 provides conclusions.

11.2. Background

This section defines the problem of finding the best subset of features, discusses previous approaches, and outlines our solution.

11.2.1. Markov Boundaries

Let F be a full set of features. A feature selection solution can be described in terms of a Markov blanket (Koller and Sahami, 1996). Given a target feature Y, let $M \subset F$ and $Y \notin M$. M is said to be a Markov blanket for Y if $Y \perp (F - M) | M$. That is, Y is conditionally independent of other features given M. A minimal Markov blanket is referred to as Markov boundary (MB) and such a subset might be considered a feature selection solution. However, an important issue is that a MB need not be unique. Redundant features can replace others in a feature subset. Usually feature redundancy is defined in terms of feature correlation (Hall, 2000). For example, two features are redundant to each other if their values are completely correlated. In reality, it is not so straightforward to determine feature redundancy if a feature is partially correlated to a set of features.

Our goal is to focus on the important case with redundant features and obtain at least one MB. In most real-life problems exactly determining the MB or measuring feature relevance is very difficult because of a limited sample size, high time complexity, and noise in the data. Furthermore, evaluation of the distribution of the input variables and the response always relies on some model (linear, support vector machine, frequency tables, trees, etc.). In practice, most algorithms just try to remove irrelevant features and then apply some heuristics that remove “possibly" redundant variables.

11.2.2. Existing Approaches in Feature Selection

The nature of real life data sets provides strong restrictions for model fitting and feature selection methods. First, data sets may be very large both in terms of the number of predictors and in the number of samples (tens of thousands $\times$ tens of millions). Second, the predictors and the response can be of mixed type (both numeric and categoric), and can contain missing values. Lastly and also very importantly, dependency of the response on predictors can be highly non-linear, noisy and multivariate.

This leaves most existing methods out of scope for such problems. For example, wrapper methods (forward selection or backward elimination) are simply computationally unfeasible when dealing with thousands of predictors. Filter methods are also useless for the minimal subset selection problem, as they do not deal with the notion of redundancy and most of them are inherently univariate. However, there are filters that use a “local” feature importance measure (like RELIEF) that can be considered multivariate (Kira and Rendell, 1992), but still they do not deal with redundancy giving just a ranked list of features instead of a selected minimal subset.

Subset evaluation filter methods such as CFS (Hall, 2000) are neither suitable because they do not deal explicitly with redundancy, and have to iterate over many feature subsets incurring a high time complexity. For example, the time complexity of the CFS is at least quadratic in the number of features and linear in number of samples. Also CFS is highly sensitive to outliers as it uses correlations between features.

Many embedded methods that use a built-in feature relevance measurement, such as SVM-RFE (Guyon et al., 2002) and linear regression with backward feature elimination are heavily dependent on the model (linear or SVM), that can fail to fit the data well. These methods have at least quadratic complexity in the number of samples for fitting an SVM and at least cubic complexity in the number of features ($O(nm^2+m^3)$, where m is the number of features, and n is number of samples) for fitting a regression model. Data sets with tens of thousands of features or samples become very time consuming and impractical to handle. For example, SVM-RFE involves retraining the SVM after features with smallest relevance are removed, thus incurring at least cubic complexity in number of samples ($O(max(m,n)n^2)$).

An issue that discourages using regression methods and methods that rely on some kind of distance measure between observations (linear regression, SVM, Kernel-based methods, RELIEF) is the difficulty of dealing with outliers in the input (predictor) space. Also, selection of important model parameters (kernel width and type, feature relevance thresholds, etc) is non-obvious, and the results of feature selection depend heavily on them.

Most methods return just a ranked list of features instead of an optimal subset. These methods include RELIEF, Koller's Markov blanket based backward elimination (referred to here as MBBE) (Koller and Sahami, 1996), and SVM-RFE. Some methods such as FCBS use a relevance threshold that is not clear how to adjust (Yu and Liu, 2004). In reality, the user also obtains a number of feature subsets corresponding to different values of parameters without a hint of how to choose the best subset.

Many methods work with frequency tables. They can thus deal well with categorical inputs only. For numerical inputs, they require discretization. Such methods are not always able to deal with interacting variables and have great difficulties with multivariate dependencies on numerical inputs. Examples of such methods are FCBS and MBBE. These two algorithms need discretization because they use an entropy measure computed on frequency tables. If the number of categories is large, or if we use frequency tables with more than two inputs, the tables can be sparse and may not represent the data distribution well. Another issue for MBBE is computational complexity. Considering all feature pairs incurs a quadratic complexity on the number of features.

Hence we see that most methods at hand are either not applicable at all to the best subset selection problem, or have some major problems. The most useful methods in such a setting (that appeared to be applicable to the examples of large "real-life" data in the challenge data sets discussed in Section 11.5.3) are methods based on backward feature elimination using an approximate Markov blanket concept (Koller and Sahami, 1996; Yu and Liu, 2004). Our method approximates the optimal Markov blanket redundancy elimination procedure, but without most of the drawbacks of previous methods.

11.2.3. Towards Efficient and Approximately Optimal Feature Selection

We propose a method that uses an idea similar to those proposed by Koller and Sahami (1996) and Yu and Liu (2004) that tries to overcome their limitations. It does not have quadratic time complexity in the number of features, can deal with thousands of predictors, uses a model (ensembles of trees) that can be applied to mixed variable types, thus eliminating need for discretization of numeric inputs, does not require imputation of missing values, captures local information (like RELIEF), is invariant to a monotone transformation of inputs, thus not very sensitive to noise and outliers, and deals well with multivariate dependencies.

It is well known that trees and especially ensembles of trees can provide robust and accurate models in "real-life" data settings. They handle mixed and noisy data, and are scale insensitive.

Ensembles of trees have high predictive power and are resistant to over-fitting (Breiman, 2001). Our approach relies heavily on ensembles of trees.

First, we find irrelevant features that are conditionally independent of the response given the rest of the features. It is accomplished by comparing the relevance of the original variables with the relevance of random, artificial features (appended to the original data) constructed from the same distribution, but independently from the response. These features are referred to as artificial contrasts. We measure feature relevance as variable importance in random forests with a modified robust splitting criteria. We assume that if an original variable had a relevance score not statistically higher than that of an artificial probe (independent from the target by construction) then it is also independent from the target, irrelevant, and should be removed. Note that we try to remove irrelevant features by directly assessing conditional independence without searching for a MB, the existence of which is a much stronger requirement. Although the idea of artificial contrasts was already used by other researchers in simple filter methods with success (Stoppiglia et al., 2003), its application to tree ensembles is novel and promising. Actually, our approach can be considered as non-parametric because all parameters in our algorithm can be assigned reasonable default values that work well for wide range of problems.

Then the redundant feature elimination step is performed. Redundancy between features is measured using surrogate scores. The variable with the largest impurity reduction score at a node is the primary splitter. If surrogate variables (ones that partition the node in same way as the primary variable) are present, these surrogate variables are considered as “masked”. Masking scores between all pairs of important variables are computed and evaluated using a statistical test, and variables masked by more important variables (“approximately redundant”) are removed iteratively.

Finally, after a set of non-redundant relevant features has been found, our method removes the influence of the found subset with an ensemble and proceeds. Because redundancy elimination is approximate in nature this iterative approach is another advantage of our method. It allows one to recover variables with small importance and to reduce the chance to lose important variables during redundancy elimination.

11.3. Tree Ensembles for Feature Selection

For our embedded method, we focus on ensembles of decision trees for the following reasons. Trees can be applied in ubiquitous scenarios so that they provide a good entry point for feature selection for interdisciplinary, wide data sets. They apply to either a numerical or a categorical response. They are nonlinear, simple and fast learners that handle also both numerical and categorical predictors well. They are scale invariant and robust to missing values. A simple decision tree also provides an embedded measure of variable importance that can be obtained from the number and the quality of splits that are generated from a predictor variable. However, a single tree is produced by a greedy algorithm that generates an unstable model. A small change to the data can result in a very different model. Consequently, ensemble methods have been used to counteract the instability of a single tree.

Supervised ensemble methods construct a set of simple models, called base learners, and use their weighted outcome (or vote) to predict new data. That is, ensemble methods combine outputs from multiple base learners to form a committee with improved performance. Numerous empirical studies confirm that ensemble methods often outperform any single base learner (Freund and Schapire, 1996; Bauer and Kohavi, 1999; Dietterich, 2000a). The improvement can be dramatic when a base algorithm is unstable. More recently, a series of theoretical developments (Bousquet and Elisseeff, 2001; Poggio et al., 2002; Mukherjee et al., 2006; Poggio et al., 2004) also confirmed the fundamental role of stability for the generalization of a learning algorithm. More comprehensive overviews of ensemble methods were presented by Dietterich (2000b) and Valentini and Masulli (2002). There are two primary approaches to ensemble construction: parallel and serial.

A parallel ensemble combines independently constructed and diverse base learners. That is, different base learners should make different errors on new data. An ensemble of such base learners can outperform any single one of its components since diverse errors cancel out (Hansen and Salamon, 1990; Amit and Geman, 1997). Parallel ensembles are variance-reduction techniques, and in most cases, they are applied to unstable, high-variance algorithms (such as trees). Also, Valentini and Dietterich (2003) showed that ensembles of low-bias support vector machines (SVMs) often outperformed a single, best-tuned, canonical SVM (Boser et al., 1992).

Random forest (RF) is an exemplar for parallel ensembles (Breiman, 2001). It is an improved bagging method (Breiman, 1996) that extends the "random subspace" method (Ho, 1998). It grows a forest of random decision trees on bagged samples showing excellent results comparable with the best known classifiers. A RF can be summarized as follows: (1) Grow each tree on a bootstrap sample of the training set to maximum depth, (2) Given M predictors, select at random $m < M$ variables at each node, and (3) Use the best split selected from the possible splits on these m variables. Note that for every tree grown in RF, about one-third of the cases are out-of-bag (out of the bootstrap sample). The out-of-bag (OOB) samples can serve as a test set for the tree grown on the non-OOB data. We discuss later how OOB samples can be used for feature selection.

In serial ensembles, every new learner relies on previously built learners so that the weighted combination forms an accurate model. A serial ensemble algorithm is often more complex. It is targeted to reduce both bias and variance. A serial ensemble results in an additive model built by a forward-stagewise algorithm. The *adaboost* algorithm was introduced by Freund and Schapire (1996). At every step of ensemble construction the boosting scheme adds a new base learner that is forced (by iteratively reweighting the training data) to concentrate on the training observations that are misclassified by the previous sequence. Boosting showed dramatic improvement in accuracy even with very weak base learners (like decision stumps, single split trees). Breiman (1998) and Friedman et al. (2000) showed that the adaboost algorithm is a form of gradient optimization in functional space, and is equivalent to a forward-stagewise, additive algorithm with the exponential loss function $\Psi(y, F(\mathbf{x})) = \exp(-yF(\mathbf{x}))$ referred to as a gradient boosted tree (GBT).

11.3.1. Relative Variable Importance Metrics

A single decision tree partitions the input space into a set of disjoint regions, and assigns a response value to each corresponding region. It uses a greedy, top-down recursive partitioning strategy. At every step an exhaustive search is used to test all variables and split points to achieve the maximum reduction in impurity. Therefore, the tree constructing process itself can be considered as a type of variable selection (a kind of forward selection, embedded algorithm), and the impurity reduction due to a split on a specific variable indicates the relative importance of that variable to the tree model (Breiman et al., 1984). For ensembles, the metric is averaged over the collection of base learners. Note, that this relative importance automatically incorporates variable interaction effects thus being very different from the relevance measured by a univariate filter method.

For a single decision tree the measure of variable importance is

$$VI(X_i, T) = \sum_{t \in T} \Delta I(X_i, t), \tag{11.1}$$

where $\Delta I(X_i,t)$ is the decrease in impurity due to an actual (or potential) split on variable X_i at a node t of the optimally pruned tree T (Breiman et al., 1984). Node impurity $I(t)$ for regression is defined as $\sum_{i\in t}(y_i - \bar{y})^2/N(t)$ where the sum and mean are taken over all observations i in node t, and $N(t)$ is the number of observations in node t. For classification $I(t) = Gini(t)$ where $Gini(t)$ is the Gini index of node t defined as

$$Gini(t) = \sum_{i\neq j} p_i^t p_j^t,$$

and p_i^t is the proportion of observations in t whose response label equals i $(y = i)$ and i, j run through all response class numbers. The Gini index is in the same family of functions as *cross-entropy* $= -\sum_i p_i^t log(p_i^t)$, and measures node impurity. It is zero when t has observations only from one class, and is maximum when classes are perfectly mixed. The decrease $\Delta I(X_i,t)$ computes the impurity at the node t and the weighted average of impurities at each child node of t. The weights are proportional to the number of observations that are assigned to each child from the split at node t so that $\Delta I(X_i,t) = I(t) - p_L I(t_L) - p_R I(t_R)$.

For an ensemble of M trees this importance measure is easily generalized. It is simply averaged over the trees

$$E(X_i) = \frac{1}{M}\sum_{m=1}^{M} VI(X_i, T_m). \tag{11.2}$$

The averaging makes this measure more reliable.

This split weight measure $\Delta I(X_i,t)$ in Equation (11.1) can be improved if OOB samples are used. The split value for a variable is calculated using the training data as usual. However, the variable selected as the primary splitter uses only the OOB samples. Also, the variable importance measure is calculated from only the OOB samples. This provides a more accurate and unbiased estimate of variable importance in each tree and improves the filtering of noise variables.

Breiman (2001) also proposed a *sensitivity* based measure of variable relevance evaluated by a RF. For a classification problem it is summarized as follows: (1) Classify the OOB cases and count the number of votes cast for the correct class in every tree grown in the forest, (2) randomly permute the values of variable m in the OOB cases and classify these cases down the tree, (3) Subtract the number of votes for the correct class in the variable-m-permuted OOB data from the original OOB data, and (4) Average this number over all trees in the forest to obtain the raw importance score for variable m. Similar ideas were presented by Parmanto et al. (1996) and a similar resampling strategy was successfully used in a more traditional model by Wisnowski et al. (2003). The sensitivity measure is computationally expensive. Furthermore, it does not account for masking, nor does it consider an iterative process with residuals (that we describe in Section 11.4.2). Experiments by Tuv (2006) demonstrated that weaker but independent predictors can rank higher than stronger, but related predictors. Also, related predictors can all be identified as important. Neither of these results are desirable for a best subset model and a more effective algorithm is described in Section 11.4.

With the importance measure (11.2) we can thus merely rank the variables. The following two subsections discuss how to amend the ranking so that irrelevant variables can be reliably detected, and how the redundancies among the remaining relevant variables can then be handled.

11.3.2. Removing Irrelevant Features by Artificial Contrasts

Although an ensemble can be used to calculate a relative feature ranking from the variable importance score in (11.2) the metric does not separate relevant features from irrelevant. Only a list of importance values is produced without a clear indication which variables to include, and which to discard. Also, trees tend to split on variables with more distinct values. This effect is more pronounced for categorical predictors with many levels. It often makes a less relevant (or completely irrelevant) input variable more "attractive" for a split only because it has high cardinality.

The variable importance score in (11.2) is based on the relevance of an input variable to the target. Consequently, any stable feature ranking method should favor a relevant input X_i over an artificially generated variable with the same distribution as X_i but generated to be irrelevant to the target. That is, a higher variable importance score is expected from a true relevant variable than from an artificially generated contrast variable. With sufficient replicates in an analysis one can select important variables from those that have statistically significantly higher variable importance scores than the contrast variables (Tuv et al., 2006). Here, these contrast variables are integrated into a subset algorithm. We discuss this in detail in Section 4.

Also, artificial contrasts can be applied to masking discussed in the next subsection. Given a selected subset of relevant variables, one computes the masking scores of all variables by elements of this subset, and the masking of contrast variables by this subset. Masking scores statistically higher than the contrast variables are considered to be real masking. Variables that are masked are dropped from the relevant subset list over a sequence of iterations of the algorithm.

11.3.3. Masking Measures

An important issue for variable importance in tree-based models is how to evaluate or rank variables that were masked by others with slightly higher splitting scores, but could provide as accurate a model if used instead. One early approach in the CART methodology used surrogate splits (Breiman et al., 1984). The predictive association of a surrogate variable X^s for the best splitter X^* at a tree node T is defined through the probability that X^s predicts the action of X^* correctly and this is estimated as

$$p(X^s, X^*) = p_L(X^s, X^*) + p_R(X^s, X^*),$$

where $p_L(X^s, X^*)$ and $p_R(X^s, X^*)$ define the estimated probabilities that both X^s and X^* send a case in T left (right). The predictive measure of association $\lambda(X^*|X^s)$ between split X^s and primary split X^* is defined as

$$\lambda(X^*|X^s) = \frac{\min(\pi_L, \pi_R) - [1 - p(X^s, X^*)]}{\min(\pi_L, \pi_R)},$$

where π_L, π_R are the proportions of cases sent to the left(or right) by X^*. It measures the relative reduction in error $(1 - p(X^s, X^*))$ due to using X^s to predict X^* as compared with the "naive" rule that matches the action with $\max(\pi_L, \pi_R)$ (with error $\min(\pi_L, \pi_R)$). If $\lambda(X^*|X^s) < 0$ then X^s is disregarded as a surrogate for X^*. Sometimes a small, nonnegative threshold is used instead. The variable importance sum in Equation (11.1) is taken over all internal tree nodes where X_i is a primary splitter or a surrogate variable ($\lambda(X^*|X_i) > 0$ for a primary splitter X^*). Often a variable that does not appear as a primary splitter in a tree is still ranked high on the variable importance list constructed using surrogate variables.

We extend the surrogate concept to define a masking score as follows. Variable i is said to mask variable j in a tree, if there is a split in variable i in a tree with a surrogate on variable j. We define the masking measure for a pair of variables i, j in tree T as

$$M_{ij}(T) = \sum_{\{t \in T | \text{split on } X_i\}} w(X_i, t) \lambda(X_i | X_j),$$

where $w(X_i,t) = \Delta I(X_i,t)$ is the decrease in impurity from the primary split on variable X_i, and summation is done over the nodes where primary split was made on variable X_i. Here we take into account both the similarity between variables X_i, X_j at the node, and the contribution of the actual split of variable X_i to the model. For an ensemble the masking measure is simply averaged over the trees. Note that in general the measure is not symmetric in the variables. One variable may mask several others, but for a single selected masked variable the reverse may not be true.

11.4. Algorithm: Ensemble-Based Feature Selection with Artificial Variables and Redundancy Elimination

We now integrate the previously described concepts and metrics into a subset selection algorithm. The fundamental steps outlined in Section 2.3 consist of using the advantages of a parallel ensemble to detect important variables among potentially a very large feature set, using the advantages of a serial ensemble to de-mask the important variables, and calculating residuals and repeating in order to recover variables of secondary importance.

Within the algorithm, artificial contrast variables are re-generated a number of times. Then the significance from a paired t-test over the replicates is used to identify important variables and masked variables. Essentially the t-test is used to define thresholds for selection and masking. These thresholds could also be set as tunable parameters. An advantage of the statistical test is that the significance of selected variables relative to noise can be quantified.

11.4.1. Algorithm Details

1. **Identify Important Variables:** Artificially generated noise variables are used to determine a threshold to test for statistically significant variable importance scores. The test is used to remove irrelevant variables. Details are presented in the displayed algorithms and further described as follows.

   In each replicate r, $r = 1,2,\ldots,R$ artificial variables are constructed as follows. For every real variable X_j $j = 1,2,\ldots,M$ a corresponding artificial variable Z_j is generated from a random permutation. Then in each replicate a small RF is trained and variable importance scores are computed for real and artificial variables. The scores from each replicate r are compiled into the rth row of a matrix $\mathbf{V}$ $R \times 2M$. Furthermore, the $1-\alpha$ percentile of the importance scores in replicate r is calculated from only the artificial variables. This is denoted as v_r and the vector of percentiles over the R replicates is $\mathbf{v}$ $R \times 1$. For each real variable X_j a paired t-test compares importance scores for X_j (obtained from the jth column of $\mathbf{V}$) to the vector of scores $\mathbf{v}$. A test that results in statistical significance identifies an important variable.

   Significance is evaluated through a suitably small p-value. The use of a p-value requires a feature to consistently score higher than the artificial variables over multiple replicates. Furthermore, this statistical testing framework also allows any method to control false selections to be applied. We routinely use the Bonferroni adjustment, but a false discovery rate approach is also reasonable. Each replicate uses a RF with $L =$ 20-50 trees to score the importance of the original and artificial noise variables. Also, the split weight calculation for variable importance in (11.2) only uses OOB samples as described previously.

2. **Calculate Masking Scores:** A masking matrix is computed from independent replicates in order to evaluate the statistical significance of masking results. Suppose there are m important variables from step 1. For similar reasons as in the previous step, replicates

and noise variables are used to detect masking among the relevant variables. These are currently the same replicates that are used for variable importance. A set of R independent GBT models are generated each with $L =$ 10-50 trees. Note that all variables are tested in each node in each tree in a serial ensemble. Therefore, richer, more effective masking information is obtained from a serial ensemble than from a random subspace method like RF. In these calculations, the surrogate scores and the split weights are calculated from the OOB samples as in the previous step. Let $M^r_{i,j}$ denote the masking score for variables X_i and X_j from the ensemble in replicate r, for $r = 1,2,\ldots,R$. Also, let $M^r_{i,\alpha}$ denote the $(1-\alpha)$-percentile of the masking score in replicate r from the distribution of scores between variable X_i and the noise variables. That is, $M^r_{i,\alpha}$ denotes the $(1-\alpha)$-percentile of $M^r_{i,j}$ for $j = m+1,\ldots,2m$. Similar to the check for variable importance, a paired t-test compares the masking score between variables (X_i, X_j) with masking score $M^r_{i,\alpha}$ computed from the noise variables. There is a significant masking between variables (X_i, X_j) if the paired t-test is significant. Variable X_j is masked by variable X_i if the test is significant.

3. **Eliminate Masked Variables:** Masked variables are removed from the list of important variables as follows. Given a list of important variables upon entry to this step, the variables are sorted by the importance score calculated in step 2. The most important variable is added to an exit list, and dropped from the entry list. Assume this is variable X_i. All variables that are masked by X_i are dropped from the entry list. This is repeated until the entry list is empty. The exit list represents the unmasked important variables.

4. **Generate Residuals for Incremental Adjustment:** An iteration is used to enhance the ability of the algorithm to detect variables that are important, but possibly weaker than a primary set. Given a current subset of important variables, only this subset is used to predict the target. Residuals are calculated and form a new target. For a numerical target the residuals are simply the actual minus the predicted values. For a classification problem residuals are calculated from a multiclass logistic regression procedure (Friedman et al., 2000). We predict the log-odds of class probabilities for each class (typically GBT is used), and then take pseudo residuals as summarized in the following multi-class logistic regression algorithm. The algorithms are described using the notation in Table 11.1.

   The iterations are similar to those used in forward selection. See, for example, Stoppiglia et al. (2003). The Gram-Schmidt procedure first selects the variable with highest correlation with the target. To remove the information from this variable the remaining predictors and the target are orthogonalized with respect to the selected variable. This provides residuals from the fit of the target to the first selected variable. In the feature selection method here we do not require orthogonal predictors, but we adjust the target for the variables already selected through residuals. We also can select more than a single variable in each iteration. The method also uses a conservative selection criterion (Bonferroni adjustment) and the residuals allow a variable to enter on another iteration. There are similar procedures used elsewhere in regression model building. Least angle regression (Efron et al., 2004) and projection pursuit methods (Friedman et al., 1981) are well known examples that use residuals in forward-stagewise modeling.

The algorithm returns to step 1 and continues until no variables with statistically significant importance scores remain. The current subset of important variables is used for the prediction model. Whenever step 1 is calculated, all variables are used to build the ensembles—not only the currently important ones. This approach allows the algorithm to recover partially masked variables that still contribute predictive power to the model. This can occur after the effect of a masking variable is completely removed, and the partial masking is eliminated. The algorithms for numerical (regression) and categorical (classification) targets are presented as Algorithms 11.1 and 11.2. A separate Algorithm 11.3 describes the variable masking calculations.

Algorithm 11.1: Ensemble-Based Feature Selection, Regression

```
1. Set $\Phi \leftarrow \{\}$; set $F \leftarrow \{X_1,\ldots,X_M\}$; set $W = 0$ ($|W| = M$).
2. for $r = 1,\ldots,R$ do
3.     $\{Z_1,\ldots,Z_M\} \leftarrow$ permute$\{X_1,\ldots,X_M\}$
4.     set $F_P \leftarrow F \cup \{Z_1,\ldots,Z_M\}$
5.     $r^{th}$ row of $\boldsymbol{V} = \boldsymbol{V}_{r.} = g_I(F_P,Y)$;
   endfor
6. $R \times 1$ vector (element wise) $\boldsymbol{v} = Percentile_{1-\alpha}(\boldsymbol{V}[\cdot,M+1,\ldots,2M])$
7. Set $\hat{\Phi}$ to those $\{X_j\}$ for which element wise $\mathbf{V}_{.j} > \boldsymbol{v}$
   with specified paired t-test significance (0.05)
8. Set $\hat{\Phi} = RemoveMasked(\hat{\Phi}, W + g_I(F_P,Y))$
9. If $\hat{\Phi}$ is empty, then quit.
10. $\Phi \leftarrow \Phi \cup \hat{\Phi}$;
11. $Y = Y - g_Y(\hat{\Phi},Y)$
12. $W(\hat{\Phi}) = W(\hat{\Phi}) + g_I(\hat{\Phi},Y)$
13. Go to 2.
```

11.4.2. Comparison to Previous Work

Two earlier methods are closely related to ACE, FCBS (Yu and Liu, 2004) and MBBE (Koller and Sahami, 1996). We compare our method in detail to these two methods. Because we use a multivariate model (tree) instead of frequency tables, our method fits in the category of embedded methods. This is unlike FCBS and MBBE that can be considered as correlation filters, although Koller works with frequency tables of 2–5 variables.

FCBS first sorts features by correlation with the response using a symmetric uncertainty, optionally removing the bottom of the list by a user-specified threshold, then

1. The feature most correlated to the response is selected.
2. All features that have correlation with the selected feature higher than it's correlation with response are considered redundant and removed. The feature is added to the minimal subset (and this is an approximate heuristic for Markov blanket filtering).
3. Return to 1).

FCBS is similar in structure to our method, with the following important differences.

Algorithm 11.2: Ensemble-Based Feature Selection, Classification

```
1.  set $\Phi \leftarrow \{\}$; $G_k(F) = 0, W_k = 0$
2.  for $k = 1, \ldots, K$ do
3.      set $V = 0$.
4.      for $r = 1, \ldots, R$ do
            $\{Z_1, \ldots, Z_M\} \leftarrow$ permute$\{X_1, \ldots, X_M\}$
            set $F \leftarrow X \cup \{Z_1, \ldots, Z_M\}$
            Compute class proportion $p_k(x) = \exp(G_k(x)) / \sum_{l=1}^{K} \exp(G_l(x))$
            Compute pseudo-residuals $Y_i^k = I(Y_i = k) - p_k(x_i)$
            $\boldsymbol{V}_{r.} = \boldsymbol{V}_{r.} + g_I(F, Y^k)$;
        endfor
5.      Element wise $\boldsymbol{v} = Percentile_{1-\alpha}(\boldsymbol{V}[\cdot, M+1, \ldots, 2M])$
6.      Set $\hat{\Phi}_k$ to those $\{X_k\}$ for which $\boldsymbol{V}_{.k} > \boldsymbol{v}$
        with specified paired t-test significance (0.05)
7.      Set $\hat{\Phi}_k = RemoveMasked(\hat{\Phi}_k, W_k + g_I(F, Y^k))$
8.      $\Phi \leftarrow \Phi \cup \hat{\Phi}_k$;
        for $k = 1, ..., K$ do
9.          $G_k(F) = G_k(F) + g_Y(\hat{\Phi}_k, Y^k)$
10.         $W_k(\hat{\Phi}_k) = W_k(\hat{\Phi}_k) + g_I(\hat{\Phi}_k, Y^k)$
        endfor
    endfor
11. If $\hat{\Phi}_k$ for all $k = 1, \ldots, K$ is empty, then quit.
12. Go to 2.
```

Algorithm 11.3: RemoveMasked(F,W)

```
1. Let $m = |F|$.
2. for $r = 1,\dots,R$ do
3.     $\{Z_1,\dots,Z_m\} \leftarrow$ permute$\{X_1,\dots,X_m\}$
4.     set $F_P \leftarrow F \cup \{Z_1,\dots,Z_m\}$
5.     Build GBT model $G_r = GBT(F_P)$.
6.     Calculate masking matrix $M^r = M(G_r)$ ($2m \times 2m$ matrix).
   endfor
7. Set $M^r_{i,\alpha_m} = Percentile_{1-\alpha_m}(M^r[i,m+1,\dots,2m]),\ r = 1,\dots,R$
8. Set $M^*_{ij} = 1$ for those $i,j = 1\dots m$ for which $M^r_{ij} > M^r_{i,\alpha_m},\ r = 1,\dots,R$
   with specified paired t-test significance (0.05), otherwise set $M^*_{ij} = 0$
9. Set $L = F, L^* = \{\}$.
10. Move $X_i \in L$ with $i = \text{argmax}_i W_i$ to $L^*$.
11. Remove all $X_j \in L$ from $L$, for which $M^*_{ij} = 1$.
12. Return to step 10 if $L \neq \{\}$.
```

1. We use tree importance instead of univariate correlation with the response. This makes ACE much more robust and accurate.

2. We use a surrogate masking measure instead of correlation. This takes the response into account, not only the correlations between inputs. No arbitrary thresholds for correlation are used.

3. We compute residuals to find smaller effects reducing the chance to drop a non-redundant feature.

Koller's MBBE works as follows:

1. For each feature X_i, find the set M_i of K features ($K = 1-4$) that are most correlated to it. (That is, which provide little information on the response when added to the selected feature in frequency table models.) Additional information is measured as KL-distance (Kullback and Liebler, 1951) $D(P(y|X_i,X_j),P(y|X_i))$. The set M_i is called the approximate Markov blanket for feature X_i. The authors state that $K = 1-2$ gives the best results.

2. For each feature compute the relevance score $\delta_i = D(P(y|M_i,X_i),P(y|M_i))$. This represents the additional information it brings when added to its approximate Markov blanket, and remove features that have the smallest relevance scores (i.e., most redundant).

3. Repeat (1,2) until all features are ranked in the order they are deleted. This method returns a ranked list of features rather than one subset.

Table 11.1: Notation in Algorithms 1–3

K	Number of classes (if classification problem)
X	set of original variables
Y	target variable
M	Number of variables
R	Number of replicates for t-test
α	quantile used for variable importance estimation
α_m	quantile used for variable masking estimation
Z	permuted versions of X
W	cumulative variable importance vector.
W_k	cumulative variable importance vector for k-th class in classification.
F	current working set of variables
Φ	set of important variables
$\boldsymbol{V}$	variable importance matrix $(R \times 2M)$
$\boldsymbol{V}_{r.}$	rth row of variable importance matrix $\boldsymbol{V}$, $r = 1 \ldots R$
$\boldsymbol{V}_{.j}$	jth column of matrix $\boldsymbol{V}$
$g_I(F,Y)$	function that trains an ensemble of L trees based on variables F and target Y, and returns a row vector of importance for each variable in F
$g_Y(F,Y)$	function that trains an ensemble based on variables F and target Y, and returns a prediction of Y
$G_k(F)$	current predictions for log-odds of k-th class
$GBT(F)$	GBT model built on variable set F
$M(G)$	Masking measure matrix calculated from model G
M^k	Masking matrix for k-th GBT ensemble G_t.
M^*	Masking flags matrix

Our ACE algorithm works more like FCBS as it uses only one feature as an approximate MB for each feature (as does the MBBE algorithm with $K = 1$). Furthermore, it filters features by relevance before computing redundancy between the features, and reports a final minimum feature subset. However, the major difference is that our redundancy measure approximates KL-distance taking the response into account and uses local information. Thus, it can deal with multivariate dependencies. MBBE for $K > 1$ will incur three (or more) dimensional frequency tables that are hard to deal with if number of categories is large.

The learner $g(.,.)$ in the ACE algorithms is an ensemble of trees. Any classifier/regressor function can be used, from which the variable importance from all variable interactions can be derived. To our knowledge, only ensembles of trees can provide this conveniently.

The computational complexity of the algorithm is of the same order as the maximal complexity of a RF on the whole feature set and a GBT model on the selected important feature subset. A GBT model is usually more complex, because all surrogate splits at every tree node are computed. However, a smaller tree depth setting for the GBT model reduces the calculations in this part of the algorithm. The complexity is proportional to

$$(Fsel + Fimpvar) * N * logN * Ntrees * Nensembles * Niter + Niter * Fimpvar^2,$$

where the variables are defined as follows: *Niter* is the number of iterations of the ACE algorithm (for example, for the challenge discussed in Section 11.5.3 this was always less than 10 and usually 3–4); *Nensembles* is the number of replicates for t-tests (equal to 20 in the challenge); *Ntrees* is the number of trees in the RF or ensemble (equal to 20–100 in the challenge); *N* is the number of samples; *Fsel* is the number of selected variables per tree split in RF (equal to the square root of the total number features or less); *Fimpvar* is the number of selected important variables (for example, for the challenge data set NOVA discussed in Section 11.5.3 this was approximately 400–800 depending on parameters). The algorithm is very fast with approximately a minute for one feature selection iteration on the challenge NOVA data set with 16K variables with 20 replicates with 70 trees on a Windows XP-based four-processor Xeon (2 x HT) 3.4GHz workstation.

11.5. Experiments

In order to evaluate the goodness of feature selection algorithms, two options have been used in the literature. The first is not to evaluate the actual feature selection performance at all, but the performance of a subsequent learner in some task. This facilitates the use of any data set in the "evaluation" but does not give much useful information at all in characterizing the actual feature selection. The second option is to directly evaluate the feature selection performance without using a subsequent proxy task. The latter dictates the need to know the ground truth behind the data, which typically means that the data must be artificially generated, either completely, or by adding some redundant and/or irrelevant features to some known data.

As the topic of the paper at hand is a method for the subset feature selection, the first evaluation method is affected by the choice of the classifier. The effects of feature selection are mixed in with how well the learner is able to handle redundant or irrelevant features. The results would thus depend on the choice of learners and on the choice of data sets. Therefore we will mainly describe experiments with two types of simulated data with known ground truth.

Experiments on data with linear relationships are presented first. Then a nonlinear data generator is used to study the sensitivity to multiple variable interactions with nonlinear relations. Further results are from the 2007 International Joint Conference on Neural Networks (IJCNN), "Agnostic learning vs. prior knowledge challenge & data representation discovery workshop". The algorithm described here had the second best performance in the agnostic track. Here we demonstrate the effect of the subset to predictor performance as compared to the full set of features. Also, an actual manufacturing data set as well as a comparison to a previous analysis of the well known Hepatitis data are also presented in terms of predictive power of the resulting feature set.

11.5.1. Generated Data with Linear Relationships

The data in this experiment has an additive structure with one numeric response variable and 203 input variables. Inputs $x_1,\ldots,x_{100}$ are highly correlated with one another, and they are all reasonably predictive of the response (regression $R^2 \sim 0.5$). But a,b, and c are independent variables that are much weaker predictors (regression $R^2 \sim 0.1$). Further $u_1,\ldots,u_{100}$ are i.i.d. $N(0,1)$ variables that are unrelated to the target. The target variable was generated as an additive model with additional noise using $y = x_1 + a + b + c + \varepsilon$, where $\varepsilon \sim N(0,1)$. This structure is chosen because it is well known that linear (oblique) relationships are not optimal for a tree representation. However, they are ideal for correlation-based methods. Thus we have here the worst possible case for ACE and the best possible case for CFS. The methods were evaluated on 50 data sets of size 400 samples.

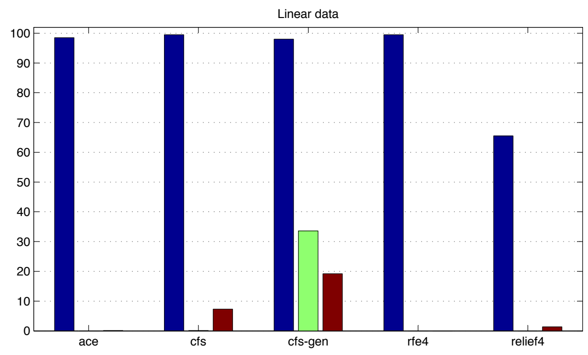


Figure 11.1: Artificial data with linear relationships. Subset discovery methods (ACE, CFS, CFS-gen) and methods finding a subset of predefined size four (RFE4, Relief4) are compared. The results for each method consist of three bars. The first is the percentage of relevant variables detected (out of four), the second is the percentage of redundant variables detected (out of 100), and the third is the percentage of irrelevant variables detected (out of 100). The results are averages over 50 data sets.

Figure 11.1 depicts the performance of ACE against methods that also discover the subsets (CFS with best-first search, CFS with genetic search), as well as against some subset ranking methods (RFE, Relief).

RFE and Relief are ranking methods. In this experiment they were given the advantage of knowing the number of relevant features beforehand, that is, their task was to "find the best possible four variable subset" (RFE4, Relief4), whereas ACE and CFS had to also find the number themselves. A further advantage was given to RFE by matching the underlying support vector regressor to the problem with a linear kernel (using the standard RBF kernel produced inferior results). This experiment demonstrates one aspect of the advantages of ACE. In a task ideal for correlation-based methods but hard for trees, we show equal performance.

11.5.2. Generated Nonlinear Data

Next, experiments were conducted using a well-known data generator (Friedman, 1999), which produces data sets with multiple non-linear interactions between input variables. The true model can be designed with relevant, redundant, and noise inputs. We selected 10 relevant inputs plus random, uniform (0, 1) noise. Also, 20 redundant inputs were used. Each was a random linear combination of three inputs plus random, uniform noise. Finally, 40 noise inputs were added, so that 70 features were available to the full model. The target function was generated as a weighted sum of 10 multidimensional Gaussians, each Gaussian at a time involving about four input variables randomly drawn from the relevant 10 variables. Thus all of the relevant 10 input variables are involved in the target, to a varying degree. The Gaussian functions also have a random mean vector and a random covariance matrix as described by Friedman (1999). Weights for the Gaussians were randomly drawn from $U[-1,1]$.

The data generator produces continuous-valued variables. Thus the data sets can be used as such for regression problems. Data sets of two different sizes were generated, 1000 and 4000 samples. In order to generate classification problems, the target variable was discretized to two levels. Mixed-type data was generated by randomly discretizing half of the variables, each to a random number of levels drawn from $U[2,32]$. There are thus eight different experiments altogether. For each experiment, 50 data sets were generated with different seeds. Figure 11.2 presents the results for each case as average percentages of features selected in each group (relevant, redundant, or noise) over the 50 generated data sets.

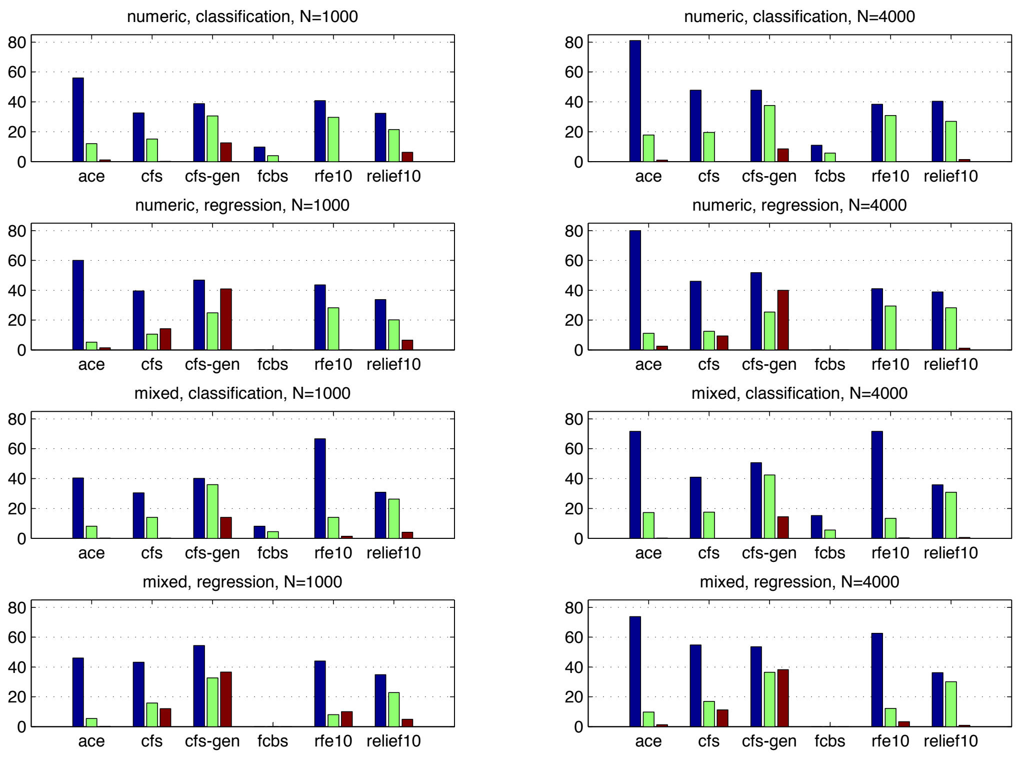


Figure 11.2: Artificial data with nonlinear relationships. Subset discovery methods (ACE, CFS, CFS-gen, FCBS) and methods finding a subset of predefined size 10 (RFE10, Relief10) are compared. FCBS works only in classification problems. The results for each method consist of three bars. The first is the percentage of relevant variables detected (out of 10), the second is the percentage of redundant variables detected (out of 20), and the third is the percentage of irrelevant variables detected (out of 40). The results are averages over 50 data sets.

RFE and Relief were again given the advantage of knowing the number of relevant features beforehand, that is, their task was to “find the best possible ten-variable subset”, whereas ACE, CFS, and FCBS had to also find the number by themselves. A further advantage was given to RFE by matching the underlying support vector classifier to the problem with an RBF kernel. Using a linear kernel produced inferior results.

The notable failure of FCBS on this data can be explained as follows. Most numerical important variables are dropped at the discretization step of FCBS, because MDL discretization works as a filter method, and it cannot deal with the multivariate dependency from Friedmans's generator. It works well with discrete variables only when the number of categories is small and the response is categorical with a small number of categories.

This experiment demonstrates another aspect of the universality of ACE. The only case where another method (RFE10) showed a superior result was a classification problem with a smaller sample size and mixed type inputs. Again RFE10 was given the advantage of knowing the number of relevant features and an appropriate kernel beforehand.

11.5.3. IJCNN 2007 Agnostic Learning vs. Prior Knowledge Challenge

In this experiment we show the effect of the selected subset within various classification tasks. The ACE feature selection algorithm was applied to the data sets in the Agnostic Learning Challenge. The number of training/validation/testing instances and the number of features are shown in the following list:

- ADA, Marketing, 4147/415/41471, 48 features
- GINA, Handwriting recognition, 3153/315/31532, 970 features
- HIVA, Drug discovery, 3845/384/38449, 1617 features
- NOVA, Text, 1754/175/17537, 16969 features
- SYLVA, Ecology, 13086/1309/130857, 216 features

For feature selection with ACE, the number of trees, importance and masking quantiles were parameters that were optimized. Next GBT with embedded feature selection (to prevent overfitting) (Borisov et al., 2006) was built on the subset. The following parameters of GBT were optimized: number of trees, tree depth, shrinkage, number of selected features per tree node, and the importance adjustment rate for embedded feature selection, stratified sampling for 0/1 class proportions, and priors. The optimization strategy (manual) was to set reasonable parameter values, and then try to adjust each parameter sequentially, so that the test error decreased. The model was trained on 60% of the training data during parameter optimization. Several passes over all the GBT parameters were used, and one for the feature selection parameters. Priors were selected using cross validation. Feature selection and GBT were used on K partitions of the data and then optimal priors were selected on the remaining part.

Table 11.2 shows the results before and after subset selection for the five challenge data sets. The CV-error was either preserved or reduced through a good subset. The overall results were the second best in the agnostic learning challenge. Redundancy elimination was applied on ADA, HIVA, SYLVA, and feature selection without redundancy elimination was used on NOVA and GINA.

Table 11.2: IJCNN 2007 Agnostic Learning vs. Prior Knowledge Challenge results.

Original	Features	CV-error from all features	Best subset size	CV-error from selected subset
Ada	47	0.1909	16	0.1855
Gina	970	0.0527	75	0.0506
Hiva	1617	0.2847	221	0.2559
Nova	12993	0.0591	400	0.0518
Sylva	212	0.0133	69	0.0129

11.5.4. TIED Data Set

A data set with multiple Markov boundaries was generated by Statnikov and Aliferis (2009). The data was obtained from a discrete Bayesian network with 1000 variables and a target variable with four classes. A training set was constructed with 750 instances simulated from the network. The network contained 72 Markov boundaries. Each boundary contained five variables (one from each of the following subsets):(1)$\{X_9\}$, (2) $\{X_4, X_8\}$, (3)$\{X_{11}, X_{12}, X_{13}\}$, (4) $\{X_{18}, X_{19}, X_{20}\}$, and (5) $\{X_1, X_2, X_3, X_{10}\}$.

The ACE feature selection method described here was used to remove irrelevant features. After three iterations of the residual calculations described previously the algorithm stopped with the important variables (and p-values from the artificial contrasts) shown in Table 11.3. The list of statistically significant variables reproduces all the variables in any of the Markov boundaries listed above, with false alarms from variables X_{14}, X_{15}, and X_{29}.

Table 11.3: Feature selection scores for the TIED data set. Variables in any Markov boundary are recovered as significant with three false alarms.

Variable	p-value	Importance Score
3	0	100.0%
2	0	98.4%
10	0	96.4%
1	1.E-10	96.4%
11	3.E-07	83.3%
12	2.E-07	83.3%
13	5.E-07	79.1%
18	3.E-09	67.5%
19	2.E-07	67.5%
15	2.E-07	41.4%
20	2.E-06	39.5%
29	2.E-06	29.8%
8	3.E-06	26.2%
14	1.E-08	11.6%
4	8.E-06	9.5%
9	6.E-06	8.3%

Although ACE recovered the variables in the Markov boundaries, there are limitations with the masking methods for a multi-class target. The GBT ensembles model each class (versus the rest) with a binary logistic function and averages variable masking scores over the binary models. Consequently, some attenuation of the importance scores are expected. Redundancy elimination did not effectively eliminate masking in the TIED data. However, we used the TIED network and TIED data for binary problems with each class versus the rest. For example, for class 1 versus the rest the TIED network generates the same collection of 72 Markov boundaries. The results from ACE without redundancy elimination for the binary target are shown in Table 11.4. The list of statistically significant variables reproduces all the variables in any of the Markov boundaries, with no false alarms.

As the importance scores are arranged in decreasing order in Table 11.4, groups of variables with similar scores become noticeable and these groups correspond to the subsets (equivalence classes) in the cross-product that defines the Markov boundaries. That is, the most important

Table 11.4: Variable importance for TIED data modified for a binary target (class 1 versus the rest). All variables in the true Markov boundaries are identified with no false alarms.

Variable	Importance Score
4	100.0%
8	100.0%
19	88.1%
18	88.1%
20	88.1%
9	64.8%
13	39.5%
12	39.5%
11	39.5%
10	21.9%
2	21.9%
3	21.9%
1	21.9%
6	0.0%

variables in Table 11.4 are those in the subset $\{X_4, X_8\}$ in the Markov boundaries and the last group matches the subset $\{X_1, X_2, X_3, X_{10}\}$. The equivalent groups are clear from their importance scores in this case.

The analysis with redundancy elimination generated the list of significantly significant variables in Table 11.5. One equivalent variable from the subset $\{X_1, X_2, X_3, X_{10}\}$ was missed in the recovery of a Markov boundary. The contribution from this subset was, however, small. The predictive performance of a tree ensemble on the recovered variables is nearly identical to a model on a true Markov boundary. In addition, the three variables $\{X_{18}, X_{20}, X_4\}$ are identified in Table 11.5 as important, but they are redundant in the true network. Although these comprise false alarms, the magnitudes of the importance scores indicate that the last three variables are much less important than the others.

Table 11.5: Variable importance for TIED data modified for a binary target (class 1 versus the rest) with redundancy elimination.

Variable	Importance Score
8	100.0%
9	61.3%
19	43.8%
1	10.2%
18	2.6%
20	0.9%
4	0.3%

Similar results (not shown here) were obtained for the binary target class 2 (versus the rest). Results without any errors were obtained for classes 0 and 3 (each versus the rest). Specifically, for class 0 the Markov boundaries from the TIED network consist of one element from $\{X_1, X_2, X_3, X_{10}\}$. In this case the ACE analysis without redundancy elimination recovered these four variables without false alarms. The analysis with redundancy elimination correctly recovered a single variable from this set. Similarly for class 3, without redundancy elimination all variables in the Markov boundaries $\{X_{12}, X_{13}, X_{14}\}$ were recovered, and only one variable from this set was recovered with redundancy elimination.

11.5.5. Manufacturing Data

In multiple real world applications collecting unnecessary variables is a cost issue and finding a suitable subset is critical in terms of cost-efficiency. As an example we present manufacturing data from a process that contained approximately 10K rows and consisted of 35 predictors that were all numerical, continuous measurements. The target was a binary response and approximately 20% of the data belonged in the rare class. Because the data is actual manufacturing data, the specific variable names are not provided. The data was analyzed extensively with traditional regression methods (the response was coded as 0 and 1) and models obtained were complex and not accurate. A list of the results from our algorithm is shown in Table 11.6. It is not unusual for manufacturing data to consist of related predictors. Without redundancy elimination, 20 variables were identified as related to the target. However, after masking scores were used to remove redundant predictors the final subset model consisted of only five predictors.

The predictive accuracy for the binary target was nearly identical using a GBT model with these 5 predictors to the full set of 35 predictors. Table 11.6 also compares other subset selection algorithms to ACE in terms of their predictive accuracy and the size of the selected feature set.

A previous analysis of this data by Berrado and Runger (2007) used association rules applied after the predictors were discretized with simple equal-frequency discretization. Only rules with consequent equal to the rare target class were considered. A total of 25 rules were detected that met the minimum support threshold. These rules contained 14 variables and 13 out of 14 are listed in the Table 11.6. Although the objectives of the association analysis were different, the relatively high proportion of important variables is consistent with the results in Table 11.6.

11.5.6. Hepatitis Data

The hepatitis data available from the UC-Irvine repository has been widely analyzed. There are 155 patients and 19 predictors and the response is a binary survival result. Breiman (2001) considered this data and cited a previous analysis from the Stanford Medical School and another analysis by Diaconis and Efron (1983). The analysis from the medical school concluded that the important variables were 6, 12, 14, 19. But Breiman (2001) concluded after a set of analyses that number 12 or 17 provided predictive power nearly equivalent to the full set of variables, and that these masked each other. A notable difficulty is the small sample size in this example.

We confirmed the strong masking between variables 12 and 17 (and vice versa) from our masking matrix. We also obtained a subset model that consists of variables 6, 17, 14, 19, and 11, similar to medical school. Variable 11 was also identified in unpublished lecture notes by Breiman. The subset selected by our algorithm has the lowest cross-validation error using logistic regression.

Table 11.6: Manufacturing data with a binary target with redundancy elimination excludes many variables. Only a smaller subset of the relevant predictors remain. We compare the extracted variables to other subset selection algorithms (selected variables are marked as ‘1’ in the table). The error rate for the full set of variables was 0.146.

Variables	ACE without redundancy elim.	ACE with redundancy elim.	CFS	CFS-gen	FCBS
V11	100.0%	72.4%	1	1	
V4	96.1%	100.0%	1	1	
V5	49.8%	49.4%	1	1	1
V12	48.6%		1	1	
V14	46.6%		1	1	
V10	43.5%		1	1	
V2	43.3%	36.4%	1	1	
V13	38.7%	21.6%			
V8	30.3%		1		
V1	27.9%			1	
V9	23.7%				
V3	23.6%			1	
V19	21.8%				
V7	21.5%				
V20	20.4%				
V26				1	
V27				1	
Errors		0.145	0.144	0.145	0.190

Table 11.7: Hepatitis data. Features selected from ACE compared to other subset selection algorithms (selected variables are marked as ‘1’ in the table). The baseline error rate for the full set of variables was 0.148.

Variables	ACE	CFS	CFS-gen	FCBS
malaise-6	1	1	1	
albumin-17	1			
bilirubin-14	1	1	1	
histology-19	1	1	1	
spiders-11	1	1	1	1
age-1		1	1	
sex-2		1	1	1
ascites-12		1	1	1
varices-13		1	1	
Errors	0.142	0.155	0.155	0.194

11.6. Conclusions

We have presented an efficient method for feature subset selection that builds upon the known strengths of the tree ensembles and is designed explicitly to discover a non-redundant, effective subset of features in large, dirty, and complex data sets.

Our method attempts to eliminate irrelevant variables using statistical comparisons with artificial contrasts to obtain a threshold for importance estimated from the parallel ensembles of trees capable of scoring very large number of variables.

It uses serial ensembles to discover significant masking effects for redundancy elimination. Furthermore we have showed that the redundancy elimination based on feature masking approximates the Markov blanket redundancy filtering. It also uses an iterative strategy to allow for weaker predictors to be identified after stronger contributors.

The superior performance of the algorithm is illustrated with a number of experiments on both artificial and real data as well as by its success in the agnostic learning challenge.

Acknowledgments

This material is partly based upon work supported by the National Science Foundation under Grant No. 0743160.

References

H. Almuallin and T. G. Dietterich. Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, 69(1-2):279–305, 1994.

Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9(7):1545–88, 1997.

E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting and variants. *Machine Learning*, 36(1/2):525–536, 1999.

A. Berrado and G.C. Runger. Using metarules to organize and group discovered association rules. *Data Mining and Knowledge Discovery*, 14(3):409–431, 2007.

A. Borisov, V. Eruhimov, and E. Tuv. Tree-based ensembles with dynamic soft feature selection. In I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, editors, *Feature Extraction Foundations and Applications: Studies in Fuzziness and Soft Computing*. Springer, 2006.

B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *5th Annual ACM Workshop on COLT, Pittsburgh, PA*, pages 144–152. ACM Press, 1992.

O. Bousquet and A. Elisseeff. Algorithmic stability and generalization performance. In *Advances in Neural Information Processing Systems*, volume 13, pages 196–202. MIT Press, 2001.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

L. Breiman. Arcing classifiers. *The Annals of Statistics*, 26(3):801–849, 1998.

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, MA, 1984.

S. Cost and S. Salzberg. A wighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10(1):57–78, 1993.

P. Diaconis and B. Efron. Computer intensive methods in statistics. *Scientific American*, (248): 116–131, 1983.

T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, 2000a.

T. G. Dietterich. Ensemble methods in machine learning. In *First International Workshop on Multiple Classifier Systems 2000, Cagliari, Italy*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2000b.

Bradley Efron, Trevor Hastie, Lain Johnstone, and Robert Tibshirani. Least angle regression. *Annals of Statistics*, 32:407–499, 2004.

Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *The 13th International Conference on Machine Learning*, pages 148–156. Morgan Kaufman, 1996.

J. Friedman. Greedy function approximation: a gradient boosting machine. *Technical report, Dept. of Statistics, Stanford University*, 1999.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 28:832–844, 2000.

Jerome H Friedman, Mark Jacobson, and Werner Stuetzle. Projection pursuit regression. *Journal of the American Statistical Association*, 76:817–823, 1981.

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, Mar 2003.

Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002.

M. A. Hall. Correlation-based feature selection for discrete and numeric class machine learning. In *Proceedings of the 17th International Conference on Machine Learning*, pages 359–366, 2000.

L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

T. K. Ho. The random subspace method for constructing decision forests. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

Kenji Kira and Larry A. Rendell. A practical approach to feature selection. In *ML92: Proceedings of the ninth international workshop on Machine learning*, pages 249–256, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc. ISBN 1-5586-247-X.

D. Koller and M. Sahami. Toward optimal feature selection. In *Proceedings of ICML-96, 13th International Conference on Machine Learning*, pages 284–292, Bari, Italy, 1996. URL citeseer.nj.nec.com/koller96toward.html.

S. Kullback and R.A. Liebler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:76–86, 1951.

H. Liu and L. Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Trans. Knowledge and Data Eng.*, 17(4):491–502, 2005.

S. Mukherjee, P. Niyogi, T. Poggio, and R. Rifkin. Learning theory: Stability is sufficient for generalization and necessary and sufficient for consistency of empirical risk minimization. *Advances in Computational Mathematics*, 25:161–193, 2006.

B. Parmanto, P. Munro, and H. Doyle. Improving committee diagnosis with resampling techniques. In D. S. Touretzky, M. C. Mozer, and M. Hesselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 882–888. Cambridge, MA: MIT Press, 1996.

T. Poggio, R. Rifkin, S. Mukherjee, and A. Rakhlin. Bagging regularizes. In *CBCL Paper 214/AI Memo 2002-003*. MIT, Cambridge, MA, 2002.

T. Poggio, R. Rifkin, S. Mukherjee, and P. Niyogi. General conditions for predictivity in learning theory. *Nature*, 428:419–422, 2004.

M. Robnik-Sikonja and I. Kononenko. Theoretical and empirical analysis of relief and relieff. *Machine Learning*, 53:23–69, 2003.

C. Stanfill and D. Waltz. Toward memory-based reasoning. *Communications of the ACM*, 29: 1213–1228, December 1986.

A. Statnikov and C.F. Aliferis. Tied: An artificially simulated dataset with multiple Markov boundaries. *Journal of Machine Learning Research Workshop Conference & Proceedings*, 2009. to appear.

H. Stoppiglia, G. Dreyfus, R. Dubois, and Y. Oussar. Ranking a random feature for variable and feature selection. *Journal of Machine Learning Research*, 3:1399–1414, March 2003.

E. Tuv. Ensemble learning and feature selection. In I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, editors, *Feature Extraction, Foundations and Applications*. Springer, 2006.

E. Tuv, A. Borisov, and K. Torkkola. Feature selection using ensemble based ranking against artificial contrasts. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2006.

G. Valentini and T. Dietterich. Low bias bagged support vector machines. In *ICML 2003*, pages 752–759, 2003.

G. Valentini and F. Masulli. Ensembles of learning machines. In M. Marinaro and R. Tagliaferri, editors, *Neural Nets WIRN Vietri-02*, Lecture Notes in Computer Science. Springer-Verlag, 2002.

J.W. Wisnowski, J.R. Simpson, D.C. Montgomery, and G.C. Runger. Resampling methods for variable selection in robust regression. *Computational Statistics and Data Analysis*, 43(3): 341–355, 2003.

L. Yu and H. Liu. Efficient feature selection via analysis of relevance and redundancy. *J. of Machine Learning Research*, 5:1205–1224, 2004.

Chapter 12

Classification with Random Sets, Boosting and Distance-based Clustering

Vladimir Nikulin VNIKULIN.UQ@GMAIL.COM
Department of Mathematics, University of Queensland, Brisbane, Australia

Editor: Isabelle Guyon

Abstract

Overfitting is commonly a significant problem in classification of high-dimensional data. Accordingly, it appears to be logical to consider a large number of component-classifiers, each of which is trained using a relatively small randomly selected subset of the available features. As an outcome, random sets approach provides an effective means of feature selection by examination of the features used by the best performing component-classifiers. The proposed method differs from Breiman's Random Forests in two respects: firstly, it has non-voting nature; secondly, the component-classifiers need not be implemented using decision trees. We also propose a novel boosting technique based on experience/innovation principles. In order to make some improvement of the training results we can increase attention to the mis-classified patterns by the random increasing of the corresponding weights (innovation). As a starting point for any iteration we can use weights, which correspond to the best past result (experience). Lastly, assuming that the data exhibit a clustered structure, and that the relationship between the target variable and the explanatory features is constant within each cluster, it is reasonable to construct an ensemble classifier using component-classifiers, each of which is trained on data drawn from the same cluster. The benefits of these innovations are demonstrated using results from the IJCNN-2007 Agnostic Learning versus Prior Knowledge challenge which were among the leading entries.

Keywords: random forests, gradient-based optimization, boosting, cross-validation, distance-based clustering

12.1. Introduction

This paper contains three new developments comparing with previous publication (Nikulin, 2006a): random sets, heuristical version of boosting and distance-based clustering approach for an ensemble classifier.

Random Forests (Breiman, 2001) grow a forest of random trees on bagged samples showing excellent results comparable with the best known classifiers (Tuv et al., 2006). Clearly, the choice of decision trees as a base model is not necessary, and as an alternative we can use quadratic minimization (QM) or Naïve Bayes model. Section 12.2.4 introduces a new general method for feature selection. This method is based on the assumption that any component-classifier, which is based on relatively small number of features, will not suffer from overfitting.

There are many ways to boost mis-classified pattern. Friedman et al. (2000) present some reasonable mathematical grounds behind AdaBoost and LogitBoost algorithms for the additive logistic regression model. Unfortunately, some of the required and essential conditions may not be fulfilled during optimization process (see Section 12.3), and, as a result, performance of the algorithm may not be monotonical or stable (Lutz, 2006). As it was noticed in Boulle

(2006), the boosting method is theoretically founded to reduce the training bias, but without guarantee against overfitting. Section 12.3.5 introduces two main types of specially defined boosting, which are based on experience-innovation (EI) principles (Nikulin and Smola, 2005). Assuming that overfitting is limited, we propose to apply heuristical boosting to some of the mis-classified patterns using best past experience as a starting point for any iteration.

Ideally, our target is to find a transformation from multi-dimensional space of features to one-dimensional Euclidean space in order to maximize the difference between different classes and minimize volatility inside classes (see, for example, Kernel Fisher Discriminant). As a next step, we can consider more advanced model using distance-based clustering technique (see Section 12.4). Note that similar ideas may be found in Kurogi et al. (2006). First, we split data into k clusters according to the given criterion. Then, we develop an ensemble system as combined complex of k classifiers, where any classifier was developed independently using data from the particular cluster.

GLiMix algorithm of the Section 12.4.1 may be useful in order to investigate uniformity of the training data in the sense of relations between the target and explanatory variables.

Model selection represents a very complex process. The main problem here is that in most cases settings for the particular model can not be optimised analytically and are very dependent on the available training datasets. At the same time, most of the models are very flexible and include many regulation parameters. Proper designed cross-validation (Section 12.2.7) may be viewed as the most important attribute for the successful performance of the whole system.

Experimental results are presented in the Section 12.5. Most of experiments were conducted during time of the IJCNN-2007 Agnostic Learning *vs.* Prior Knowledge Challenge[1] (Guyon et al., 2007). The following 5 real life datasets were used during the Challenge: ADA (marketing), GINA (handwriting), HIVA (drug discovery), NOVA (text classification) and SYLVA (ecology). The competition had two parallel tracks: "agnostic learning" where data were preprocessed in a simple feature-based representation, suitable for any data mining algorithm, and "prior knowledge" where given data were not necessarily in a form of numerical table.

Our overall results (out of 5 complete entries): 1st place in the "prior knowledge" track and 4th place in the "agnostic learning" track. Also, we can report best result in the GINA-prior track.

It is a well known fact that for various reasons it may not be possible to theoretically analyze a particular algorithm or to compute its performance in contrast to another. The results of the proper experimental evaluation are very important as these may provide the evidence that a method outperforms existing approaches.

12.2. Main Models

Let $\mathbf{X} = (\mathbf{x}_t, y_t), t = 1 \ldots n$, be a training sample of observations where $\mathbf{x}_t \in \mathbb{R}^\ell$ is ℓ-dimensional vector of features, and y_t is binary label: $y_t \in \{-1, 1\}$. Boldface letters denote vector-columns, whose components are labeled using a normal typeface. It will be more convenient for us in some cases to use indexes $\{1, 2\}$ instead of original values $\{-1, 1\}$.

In practical situation the label y_t may be hidden, and the task is to estimate it using vector of features. Let us consider the most simple linear decision function

$$u_t = u(\mathbf{x}_t) = \sum_{j=1}^{\ell} w_j \cdot x_{tj} + b \tag{12.1}$$

where w_i are weight coefficients and b is a bias term.

1. http://www.agnostic.inf.ethz.ch/

Definition 12.1 *We will call two decision functions $u(\mathbf{x})$ and $v(\mathbf{x})$ as $\mathbf{X}$-equivalent if there exist two finite constants $A \in R_+$ and $B \in R$ such that $u(\mathbf{x}) = A \cdot v(\mathbf{x}) + B, \ \forall \mathbf{x} \in \mathbf{X}$.*

We can define decision rule as a function of decision function and threshold parameter

$$f_t = f(u_t, \Delta) = \begin{cases} 1 & \text{if} \quad u_t \geq \Delta; \\ -1, & \text{otherwise.} \end{cases} \tag{12.2}$$

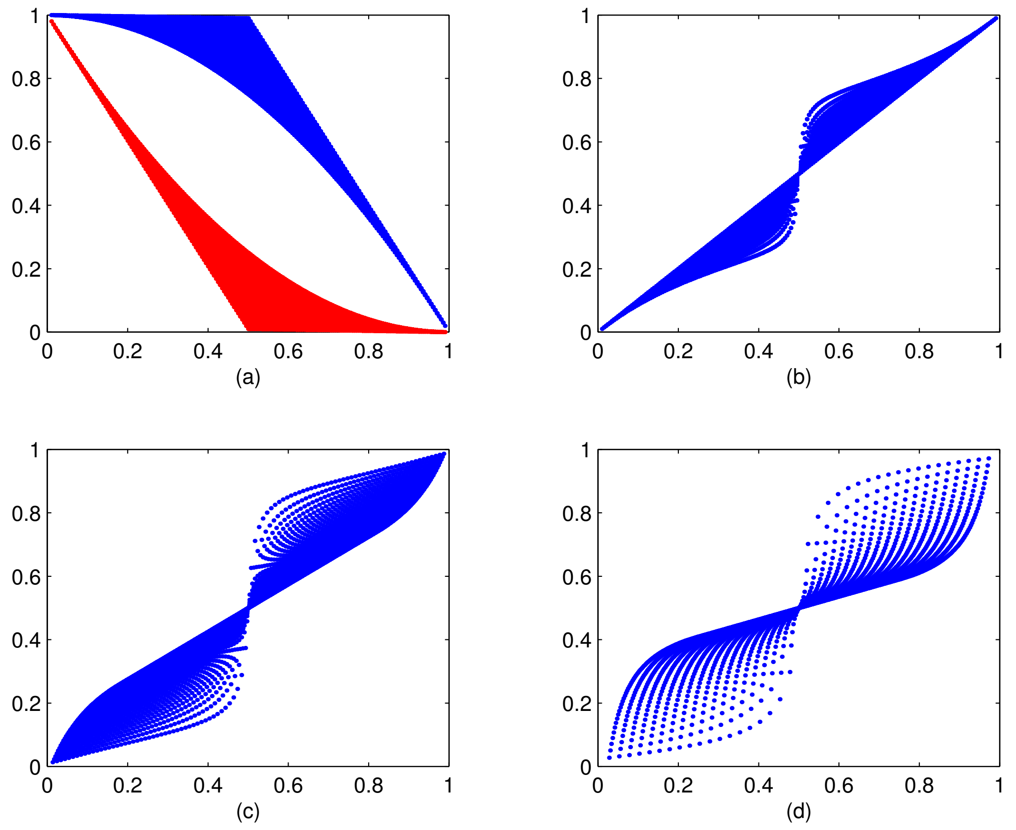


Figure 12.1: (a) $\widehat{AUC}$ (blue) and $\widetilde{AUC}$ (red) against BER; (b-d) $\widetilde{BER}$ against BER. Simulation experiments were conducted using the following parameters: (a-b) $q_1 = q_2 = 100$ - balanced case; (c) $q_1 = 50, q_2 = 150$; (d) $q_1 = 20, q_2 = 180$ - imbalanced case (see definitions in the Section 12.2.1).

The optimization criterion is to minimize the balanced error rate (BER):

$$Q(\Delta) = 0.5\left(\frac{q_{12}}{q_1} + \frac{q_{21}}{q_2}\right), \ q_1 = q_{11} + q_{12}, \ q_2 = q_{21} + q_{22}, \tag{12.3}$$

where value of q_{ij} equal to the number of j-predictions in the true cases of $i = 1 \ldots 2$. Unfortunately, the target function (12.3) can not be optimized directly. Respectively, we will consider in the following Sections 12.2.2 and 12.2.3 an alternative (differentiable) target functions assuming that the corresponding models will produce good solutions in the sense of (12.3).

12.2.1. On the relation between BER and AUC

As an alternative criterion, we used area under ROC curve (AUC). By definition, receiver operating curve (ROC) is a graphical plot of true positive rates against false positive rates.

It is interesting to clarify relations between BER and AUC using terminology of confusion matrix.

Let us denote by $\{q_{ij}\}$ set of data-entries which correspond to $q_{ij}, i, j = 1 \dots 2$.

Accordingly, we denote by $S_1 = \{q_{11}, q_{21}\}$ all data-entries which were classified as negative; $S_2 = \{q_{12}, q_{22}\}$ all data-entries which were classified as positive.

Without loss of generality we will assume that value of the decision function is -1 on S_1 and 1 on S_2. This assumption will give us a flexibility to apply an arbitrary permutation within S_1 or S_2.

Note that any permutation within S_1 or S_2 (or both, assuming that data from S_1 and S_2 are not mixed) will not affect confusion matrix. Respectively, value of BER will be strictly the same. In difference, value of AUC may vary significantly.

The upper bound for AUC (see Figure 12.1(a) – blue color)

$$\widehat{AUC} = \frac{q_{22}q_{12}}{q_1 q_2} + \frac{q_{11}}{q_1} \tag{12.4}$$

corresponds to the following sequence:

$$\{q_{11}\}\{q_{21}\}\{q_{12}\}\{q_{22}\}. \tag{12.5}$$

The low bound (see Figure 12.1(a) – red color)

$$\widetilde{AUC} = \frac{q_{11}q_{22}}{q_1 q_2} \tag{12.6}$$

corresponds to the following sequence:

$$\{q_{21}\}\{q_{11}\}\{q_{22}\}\{q_{12}\}. \tag{12.7}$$

Let us consider a marginal example. Suppose that $q_{12} = 0$ and $q_{21} = q_2$. Then, $q_{11} = q_1$, and $\widehat{AUC} = 1$. At the same time,

$$BER = \frac{q_{21}}{2q_2} = 0.5.$$

Above situation may happen if classifier ranked data correctly, but threshold parameter Δ was too big. Respectively, all data were classified as $\{-1\}$ or $S_2 = \emptyset$.

Now, let us consider more realistic second example. Suppose that $q_{11} = 90, q_{12} = 10, q_{21} = 1, q_{22} = 9$. Then, $BER = 0.1$ in both cases (12.5) or (12.7). But, $AUC = 0.99$ in the case of sequence (12.5), and $AUC = 0.81$ in the case of (12.7).

In fact, any separate re-distribution of the elements within S_1 and S_2 will not affect decision making but may change value of AUC significantly.

Let us define an alternative balanced error rate

$$\widetilde{BER} = 0.5\left(\frac{q_{12}}{q_{12}+q_{22}} + \frac{q_{21}}{q_{11}+q_{21}}\right) \tag{12.8}$$

where true and predicted labels were replaced with each other. Figures 12.1(b-d) illustrate non-symmetrical properties of the BER loss function.

Table 12.1: Regularization in conjunction with *QM* model (see Section 12.2.2) in the case of HIVA-set, used CV-100 where value of threshold parameter Δ was optimized for any particular fold (see Section 12.2.7).

N	μ	Train BER	Test BER	Mean Δ	Std Δ
1	0.001	0.0326	0.2344	-0.678	0.161
2	0.01	0.0469	0.2222	-0.722	0.147
3	0.02	0.066	**0.2188**	-0.752	0.13
4	0.03	0.0771	0.2194	-0.772	0.127
5	0.05	0.0925	0.2209	-0.789	0.12
6	0.08	0.1081	0.2235	-0.801	0.108
7	0.1	0.1156	0.224	-0.805	0.103
8	0.12	0.122	0.2253	-0.806	0.101
9	0.15	0.1302	0.2264	-0.808	0.094
10	0.18	0.1377	0.2274	-0.807	0.091
11	0.2	0.1416	0.229	-0.808	0.094
12	0.25	0.1518	0.2297	-0.819	0.081
13	0.3	0.1601	0.2316	-0.829	0.078

12.2.2. QM Model with Regularization

Let us consider the most basic quadratic minimization model with the following target function:

$$L(\mathbf{w}) = \Omega(\mu, n, \mathbf{w}) + \sum_{t=1}^{n} \xi_t \cdot (y_t - u_t)^2 \tag{12.9}$$

where $\Omega(\mu, n, \mathbf{w}) = \mu \cdot n \cdot \|\mathbf{w}\|^2$ is a regularization term with ridge parameter μ (Wichard, 2006); non-negative weight coefficients ξ are necessary in order to implement boosting algorithm in the Section 12.3.

Remark 12.2 *The target of the regularization term with parameter μ is to reduce the difference between training and test results. Value of μ may be optimized using cross-validation as it is described in the Section 12.2.7 (see, also, Table 12.1).*

Remark 12.3 *Based on observation of the Table 12.1 (last column) it is interesting to note that the regularization term Ω may be viewed as a stabilizer of the model: standard deviation of Δ is decreasing as a function of μ.*

The direction of the steepest decent is defined by the gradient vector

$$g(\mathbf{w}) = \{g_j(\mathbf{w}), j = 1..\ell\},$$

where

$$g_j(\mathbf{w}) = \frac{\partial L(\mathbf{w})}{\partial w_j} = 2\mu \cdot n \cdot w_j - 2\sum_{t=1}^{n} x_{tj}\xi_t\,(y_t - u_t)\,.$$

Initial values of the linear coefficients w_i and bias parameter b may be arbitrary. Then, we recompute the coefficients

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \delta_k \cdot g(\mathbf{w}^{(k)}), \tag{12.10a}$$

$$b^{(k+1)} = b^{(k)} + \frac{1}{n}\sum_{t=1}^{n} \xi_t \cdot (y_t - u_t) \tag{12.10b}$$

where k is a sequential number of iteration. Minimizing (12.9) we find size of the step according to the formula

$$\delta = \frac{L_1 - L_2 - \mu \cdot n \sum_{j=1}^{\ell} w_j g_j}{\sum_{t=1}^{n} \xi_t s_t^2 + \mu \cdot n \sum_{j=1}^{\ell} g_j^2} \tag{12.11}$$

where

$$L_1 = \sum_{t=1}^{n} \xi_t s_t y_t, \quad L_2 = \sum_{t=1}^{n} \xi_t s_t u_t, \quad s_t = \sum_{j=1}^{\ell} x_{tj} g_j.$$

12.2.3. Relevance Vector Machine

Good performance of pattern classifier is achieved when the number of adjustable parameters is matched to the size of the training set (Boser et al., 1992). Using above idea as a motivation we consider relevance vector machine (Tipping, 2001) with regularization

$$L(\mathbf{w}) = \Omega(\mu, n, \mathbf{w}) + \|\mathbf{y} - \Psi \mathbf{w}\|^2 \tag{12.12}$$

where

$$\Psi = \{\xi_i \cdot \xi_j \cdot \psi(\mathbf{x}_i, \mathbf{x}_j), i, j = 1..n\}$$

is a kernel matrix and $\mathbf{w}$ is vector-column of coefficients, weight coefficients ξ have the same interpretation as in the previous section. Note that bigger value of $\psi(\mathbf{x}_i, \mathbf{x}_j)$ reflects stronger similarity between patterns $\mathbf{x}_i$ and $\mathbf{x}_j$.

Value of the decision function for the pattern $\mathbf{x}$ will be computed according to the following formula

$$u(\mathbf{x}) \sim \sum_{j=1}^{n} \mathbf{w}_j \cdot \xi_j \cdot \psi(\mathbf{x}, \mathbf{x}_j),$$

which does not include bias term because we will need to optimize value of threshold parameter Δ anyway.

12.2.4. Feature Selection using Random Sets

Let us consider an illustrative example of *HIVA*-set from the IJCNN-2007 Agnostic Learning *vs.* Prior Knowledge challenge. The training sample size is 3845 where any data instance includes binary target variable and 1617 features. Pure QM model (without regularization) produced perfect training results with nearly zero value of the balanced error rate. As a next step we considered 100-folds cross-validation: test BERs were in the range from 0.35 to 0.4, which indicates very strong overfitting.

Then, we considered sequence of 10000 subsets with 70 randomly selected features (without repeats). We observed training BERs in the following range: from 0.2164 to 0.3329.

It is hardly possible to expect strong disagreement between training and test results in the case of only 70 features. The union of 25 top performing subsets includes 1074 different features (see Table 12.2).

Firstly, we note that all test results in the Table 12.2 are surprisingly good, see WCCI-2006 Performance Prediction Challenge[2]. This observation may be explained by the fact that value of the threshold parameter Δ was optimized for any particular fold (see Section 12.2.7). Also, we can see that after some point the model suffers from overfitting. According to our experiments the optimal model is based on 600–800 features out of available 1617.

2. http://clopinet.com/isabelle/Projects/modelselect/

Table 12.2: Feature selection using Random Sets (HIVA). The values in the columns “Train BER” and “Test BER” were generated using 50-folds cross validation. The second column represents number of features (without repeats) in the union of the first *k* top-performing subsets.

Number of subsets	Number of features	Train BER	Test BER	Mean Δ	Std Δ
1	70	0.2193	0.2345	-0.886	0.048
2	138	0.2091	0.2429	-0.812	0.081
3	201	0.1742	0.235	-0.806	0.07
4	261	0.1399	0.2024	-0.815	0.078
5	322	0.1146	0.21	-0.802	0.108
6	377	0.1074	0.2108	-0.789	0.085
7	434	0.098	0.2027	-0.786	0.095
8	485	0.0897	0.2064	-0.756	0.09
9	529	0.0805	**0.2001**	-0.756	0.107
10	575	0.0748	0.2081	-0.743	0.122
11	620	0.0721	0.2114	-0.738	0.13
12	656	0.0698	0.2123	-0.742	0.138
13	702	0.0635	0.2137	-0.704	0.163
14	735	0.0594	0.2082	-0.674	0.15
15	766	0.0561	0.2112	-0.689	0.155
16	797	0.0575	0.2072	-0.684	0.152
17	832	0.0565	0.2121	-0.671	0.147
18	866	0.0553	0.2138	-0.673	0.125
19	905	0.0499	0.2147	-0.677	0.13
20	942	0.0484	0.2158	-0.711	0.139
21	964	0.0466	0.2166	-0.702	0.155
22	996	0.0444	0.218	-0.701	0.142
23	1024	0.0429	0.2264	-0.713	0.16
24	1051	0.0445	0.2235	-0.676	0.139
25	1074	0.0442	0.2289	-0.713	0.148

As an alternative, we can select required number of features according to the highest numbers of repeats in the 100–200 top-performing subsets. Further, these repeats may be used in order to optimize construction of the distance-based splitter (see Section 12.4).

Remark 12.4 *According to the experimental results of the Table 12.2 we can make a conclusion that the model with smaller number of features is more stable: fluctuations of the threshold parameter are smaller.*

Clearly, randomly selected subsets of features can not claim optimality, and we can conduct further adjustment using Algorithm 12.1.

Algorithm 12.1: Feature Selection

```
1: Select leading subset, which includes all features from the 1-5 best subsets (found using RS-method).
2: Split the field of all features into 2 parts without intersection: 1) leading subset with $m$ features; 2) remaining $k = \ell - m$ features.
3: Add to the leading subset one feature (randomly selected out of $k$ remaining features).
4: Find the worst performing feature in the leading subset using Leave-One-Out principle, and remove this feature permanently.
5: $k \leftarrow k - 1$.
6: If k = 0 stop the Algorithm, otherwise, goto the Step 3.
```

Remark 12.5 *Algorithm 12.1 requires $(m+1) \cdot (\ell - m)$ optimizations, which appears to be realistic (in the sense of time), taking into account the fact that one particular optimization may be very fast assuming that m is sufficiently small.*

12.2.5. Naïve Bayes Classifier

The Naïve Bayes modelling approach is based on the assumption that the variables are independent within each output label, and simply relies on the estimation of conditional probabilities. Binary datasets (for example, NOVA or HIVA) represent an ideal case for the illustration of the concepts of Naïve Bayes approach.

We will need the following definitions

$$\pi = \mathbb{P}(y=1); \quad \theta_j = \mathbb{P}(x_j = 1|y=1); \quad \gamma_j = \mathbb{P}(x_j = 1|y=0) \tag{12.13}$$

where probabilities π, θ and γ may be estimated maximizing the following log-likelihood target function

$$\sum_{t=1}^{n} \xi_t \cdot [y_t^\star Q_{1t} + (1 - y_t^\star) Q_{2t}] \tag{12.14}$$

where label $y_t^\star = 0.5(1 + y_t)$ is taking values $\{0, 1\}$,

$$Q_{1t} = \log\{\pi\} + \sum_{j=1}^{\ell} x_{tj} \log\{\theta_j\} + \sum_{j=1}^{\ell} (1 - x_{tj}) \log\{1 - \theta_j\}, \tag{12.15a}$$

$$Q_{2t} = \log\{1 - \pi\} + \sum_{j=1}^{\ell} x_{tj} \log\{\gamma_j\} + \sum_{j=1}^{\ell} (1 - x_{tj}) \log\{1 - \gamma_j\}. \tag{12.15b}$$

Similar to (12.9), the target of the non-negative (weight) coefficients ξ is to implement boosting algorithm in the Section 12.3.

Required solution is given by the following formulas

$$\pi = \frac{\sum_{t=1}^{n} \xi_t y_t^\star}{\sum_{t=1}^{n} \xi_t}; \quad \theta_j = \frac{\sum_{t=1}^{n} \xi_t x_{tj} y_t^\star}{\sum_{t=1}^{n} \xi_t y_t^\star}; \quad \gamma_j = \frac{\sum_{t=1}^{n} \xi_t x_{tj}(1-y_t^\star)}{\sum_{t=1}^{n} \xi_t (1-y_t^\star)}. \tag{12.16}$$

Definition 12.6 *In order to avoid marginal probabilities it is proposed to use the following truncation with parameter $\phi > 0$:*

$$\theta \leftarrow \begin{cases} 1-\phi & \text{if} \quad \theta > 1-\phi; \\ \phi & \quad \text{if} \quad \theta < \phi. \end{cases} \tag{12.17}$$

Similar truncation is applicable to π and γ (used value $\phi = 0.001$).

12.2.5.1. On the Difference Between Random Forests (RF) and Random Sets (RS)

Following Boulle (2006), let us denote by $a_{ij} \in \{0,1\}$ indicator for j-feature selection in the subset i:

$$\sum_{j=1}^{\ell} a_{ij} = m \ll \ell, \; i = 1..k.$$

Next, we consider a single Naïve Bayes classifier:

$$u_{it} = \begin{cases} 1 \;\; \text{if} \;\; Q_{1t} \geq Q_{2t} + \Delta; \\ \qquad -1, \quad \text{otherwise,} \end{cases} \tag{12.18}$$

where Δ is a threshold parameter and log-likelihood conditional functions Q_{1t} and Q_{2t} are defined in (12.15a) and (12.15b).

Then, we consider RF-classifier (voting system):

$$u_t = \begin{cases} 1 \;\; \text{if} \;\; \sum_{i=1}^{k} u_{it} \geq 0; \\ -1, \quad \text{otherwise.} \end{cases} \tag{12.19}$$

In order to define RS-classifier we will evaluate long sequence of subsets $\{a_{ij}, i = 1..N, j = 1..\ell\}$ against the whole training set. The final RS-classifier (non-voting system) has the same appearance as (12.18). This classifier is based on a new subset of features $B = \{b_j \in \{0,1\}, j = 1..\ell, \sum_{j=1}^{\ell} b_j = m_1\}$ where $m_1 \geq m$. Particular configuration of the subset B may be evaluated using different methods, see for more details Section 12.2.4 and Table 12.2. We conducted experiments in the case of $HIVA$-set with $m = 70, m_1 = 200, N = 60000$.

12.2.5.2. Loglikelihood based Feature Selection Method

We can re-write target function (12.14) using different terms

$$H + \sum_{j=1}^{\ell} H_j \tag{12.20}$$

where

$$H = A_1 \log\{\pi\} + A_2 \log\{1-\pi\};$$

$$H_j = B_{1j} \log\{\theta_j\} + B_{2j} \log\{1-\theta_j\} + B_{3j} \log\{\gamma_j\} + B_{4j} \log\{1-\gamma_j\}$$

where

$$A_1 = \sum_{t=1}^{n} \xi_t y_t^\star; \quad A_2 = \sum_{t=1}^{n} \xi_t (1 - y_t^\star);$$

$$B_{1j} = \sum_{t=1}^{n} \xi_t y_t^\star x_{tj}; \quad B_{2j} = \sum_{t=1}^{n} \xi_t y_t^\star (1 - x_{tj});$$

$$B_{3j} = \sum_{t=1}^{n} \xi_t (1 - y_t^\star) x_{tj}; \quad B_{4j} = \sum_{t=1}^{n} \xi_t (1 - y_t^\star)(1 - x_{tj}).$$

We can not expect that importance of the feature j is significant if the coefficients $B_{ij}, i = 1..4$, and the difference between an alternative values θ_j and γ_j are small. Accordingly, we can measure importance of the features using ratings R_j as it is defined below.

Definition 12.7 *Subject to the important truncation (12.17), we propose to calculate likelihood-based ratings for features according to the following formula*

$$R_j = B_{1j} \log\{\frac{\theta_j}{\gamma_j}\} + B_{2j} \log\{\frac{1-\theta_j}{1-\gamma_j}\} + B_{3j} \log\{\frac{\gamma_j}{\theta_j}\} + B_{4j} \log\{\frac{1-\gamma_j}{1-\theta_j}\}$$

where bigger value indicates the higher relevance or importance (see Figure 12.2), parameters θ and γ are defined in (12.16).

12.2.6. Decision Trees

Decision Trees is a non-parametric tool of discriminant analysis, which is designed to represent decision rules in a form of so called binary trees. Binary trees split training data imposing univariate linear restrictions and represent resulting clusters hierarchically starting from root node for the whole training sample itself and ending with relatively homogenous small groups of observations. For each terminal node forecasted value is assigned, hence resulting tree structure can be interpreted as a decision rule (Breiman et al., 1984).

More specifically, we define a criterion in order to split $\mathbb{R}^\ell$-space into m *Voronoi*-regions $V_i, i = 1 \dots m$, without intersection. Let us denote by $S_i = V_i \cup \mathbf{X}$ the corresponding clusters. Assuming that clusters S_i are sufficiently large in order to ensure proper level of confidence we form decision function:

$$u_t = u(\mathbf{x}_t) = \sum_{i=1}^{m} \tilde{y}_i I_{\mathbf{x}_t \in V_i}, \quad \tilde{y}_i = \frac{\sum_{\mathbf{x}_t \in S_i} y_t}{\# S_i}$$

where I_A is an indicator of the event A.

We can apply *Gini* index in order to measure uniformity of any subset S

$$Gini(S) = \tilde{y}(1 - \tilde{y}), \quad \tilde{y} = \frac{\sum_{\mathbf{x}_t \in S} y_t}{\# S}.$$

Again, binary datasets represent an ideal cases for the illustration of the concepts of decision trees approach. Considering node S as a starting point we can continue construction of the tree deeper using feature j, which was not used previously as a splitter and must be selected in order to maximize the following difference

$$Gini(S) - p \cdot Gini(S_L) - (1-p) \cdot Gini(S_R) \geq 0, \quad p = \frac{\# S_L}{\# S},$$

where $S = S_L \cup S_R, \quad S_L \cap S_R = \emptyset$.

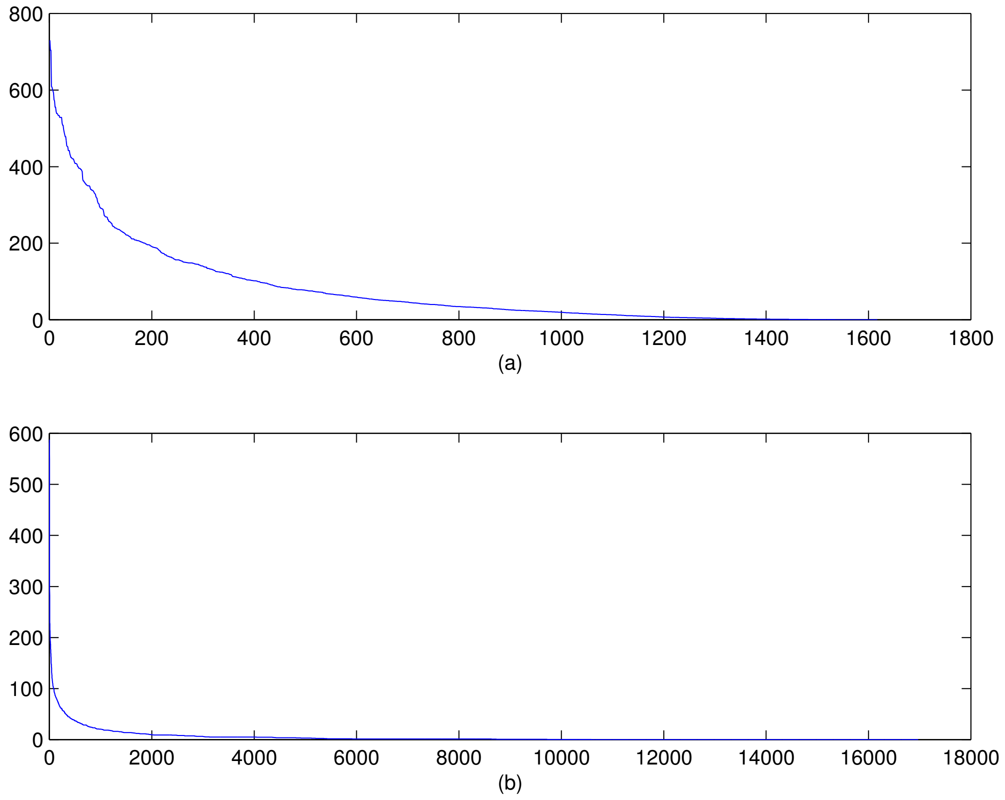


Figure 12.2: Feature selection using likelihood method (see Section 12.2.5.2): (a) HIVA and (b) NOVA-sets where rates are sorted in a decreasing order.

Remark 12.8 *In addition, we can apply such general method as threshold-based clustering with regularization in order to split data into several clusters, which are uniform in the sense of labels. Note, also, that the clustering process may include tuning of the used distance according to the given requirements (Nikulin, 2006b).*

12.2.7. Cross-Validation

CV is a very important in order to test overfitting, and it may be implemented using different methods. For example, we can split training data randomly into two subsets (with ratio 9:1) where bigger subset is to be used for training and smaller subset is to be used for testing. The most important here is selection of the threshold parameter Δ.

1: We can optimize value of Δ for any particular folder, and then use an average value in the final model, or

2: we can optimize general value of Δ for the whole experiment of 50–200 folds.

In the previous paper (Nikulin, 2006a) we employed first approach. Obviously, this approach has tendency for an optimistic *BER* prediction. Besides, as Tables 12.1 and 12.2 demonstrate fluctuation of the threshold parameter Δ may be significant. In this competition we decided to apply second strategy, which appears to be more logical (see Figure 12.4).

12.3. Boosting Algorithms

Boosting works by sequentially applying a classification algorithm to re-weighted versions of the training data, and then taking a weighted majority vote of the sequence of classifiers thus produced. For many classification algorithms, this simple strategy results in dramatic improvements in performance (Friedman, Hastie, and Tibshirani, 2000).

12.3.1. An Exponential Criterion

The motivation in support of exponential target function is very simple and clear. Let us compare squared and exponential loss functions:

$$(y_t - u_t)^2; \tag{12.21a}$$

$$\exp\{-\rho \cdot y_t \cdot u_t\}, \ \rho > 0. \tag{12.21b}$$

using two data instances $\{1,-1\}$ and $\{1,4\}$ where first and second values correspond to the label and decision function. The first example represents a mis-classification, and exponential loss function (12.21b) detects this misclassification correctly in difference to the squared loss function (12.21a):

	$\{1,-1\}$	$\{1,4\}$
squared	4	9
exponential	$e^{2\rho}$	$e^{-4\rho}$

Similar to the Logit model (Nikulin, 2006a), we can not optimize step-size in the case of exponential target function. Respectively, we will need to maintain low value of the step-size in order to ensure stability of the algorithm. As a consequence, the whole optimization process may be very slow and time-consuming. The target of the following AdaBoost Algorithm (Freund and Schapire, 1997) is to facilitate optimization process.

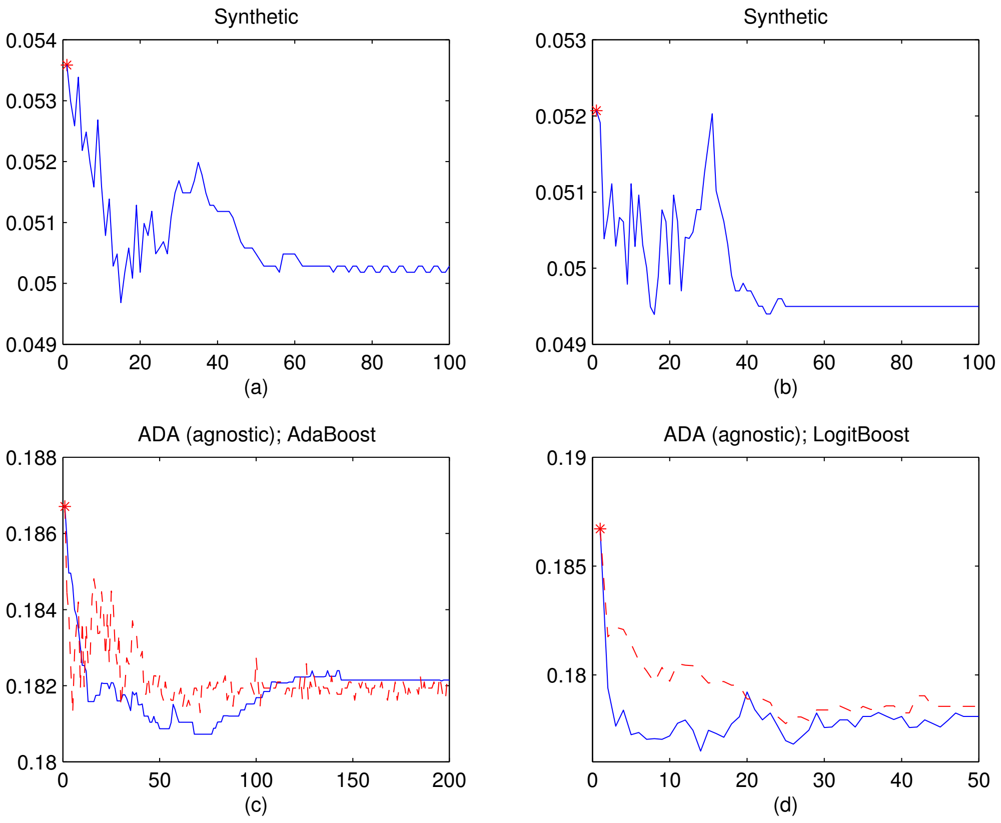


Figure 12.3: BER as a function of boosting iteration. First row illustrates application of EI-boosting to two synthetic sets, see Section 12.3.6; second row illustrates experiments against ADA-set: (c) application of AdaBoost (see Section 12.3.2) and EI-boosting (Algorithm 12.2 with $\alpha = 1.5$ and $\beta = 0.2$, red-dashed line); (d) application of LogitBoost (see Section 12.3.3) and LogitBoost2 (red-dashed line, see Section 12.3.4) algorithms.

12.3.2. AdaBoost Algorithm

Let us consider minimizing the criterion (Friedman et al., 2000)

$$\sum_{t=1}^{n} \xi(\mathbf{x}_t, y_t) \cdot e^{-y_t u(\mathbf{x}_t)} \tag{12.22}$$

where

$$\xi(\mathbf{x}_t, y_t) := \exp\{-y_t F(\mathbf{x}_t)\}. \tag{12.23}$$

We shall assume that initial values of $F(\mathbf{x}_t)$ are set to zero.

The following Taylor-approximation is valid under assumption that values of $u(\mathbf{x}_t)$ are small

$$\exp\{-y_t u(\mathbf{x}_t)\} \approx \frac{1}{2}\left[(y_t - u(\mathbf{x}_t))^2 + 1\right]. \tag{12.24}$$

Therefore, we can apply *QM*-model in order to minimize (12.22). Then, we optimize value of the threshold parameter Δ for u_t, and find corresponding decision rule $f_t \in \{-1, 1\}$.

Next, we will return to (12.22)

$$\sum_{t=1}^{n} \xi(\mathbf{x}_t, y_t) \cdot e^{-c \cdot y_t \cdot f(\mathbf{x}_t)} \tag{12.25}$$

where optimal value of the parameter c may be easily found

$$c = \frac{1}{2} \log\{\frac{A}{B}\} \tag{12.26}$$

where

$$A = \sum_{y_t = f(\mathbf{x}_t)} \xi(\mathbf{x}_t, y_t), \quad B = \sum_{y_t \neq f(\mathbf{x}_t)} \xi(\mathbf{x}_t, y_t).$$

Finally (for the current boosting iteration), we update function F:

$$F_{\text{new}}(\mathbf{x}_t) \leftarrow F(\mathbf{x}_t) + c \cdot f(\mathbf{x}_t), \tag{12.27}$$

and recompute weight coefficients ξ according to (12.23) (see Figure 12.3(c)).

Remark 12.9 *Considering test dataset (labels are not available), we will not be able to optimize value of the threshold parameter Δ. Respectively, we can use either an average (predicted) value of Δ in order to transform decision function into decision rule, or we can apply direct update:*

$$F_{\text{new}}(\mathbf{x}_t) \leftarrow F(\mathbf{x}_t) + c \cdot u(\mathbf{x}_t)$$

where value of the parameter $c \leq 1$ must be small enough in order to ensure stability of the algorithm.

12.3.3. LogitBoost Algorithm

Let us parameterize the binomial probabilities by

$$p(\mathbf{x}_t) = \frac{e^{2F(\mathbf{x}_t)}}{1 + e^{2F(\mathbf{x}_t)}}.$$

The binomial log-likelihood is

$$y_t^{\star} \log\{p(\mathbf{x}_t)\} + (1 - y_t^{\star}) \log\{1 - p(\mathbf{x}_t)\} = -\log\{1 + \exp\{-2y_t F(\mathbf{x}_t)\}\}. \tag{12.28}$$

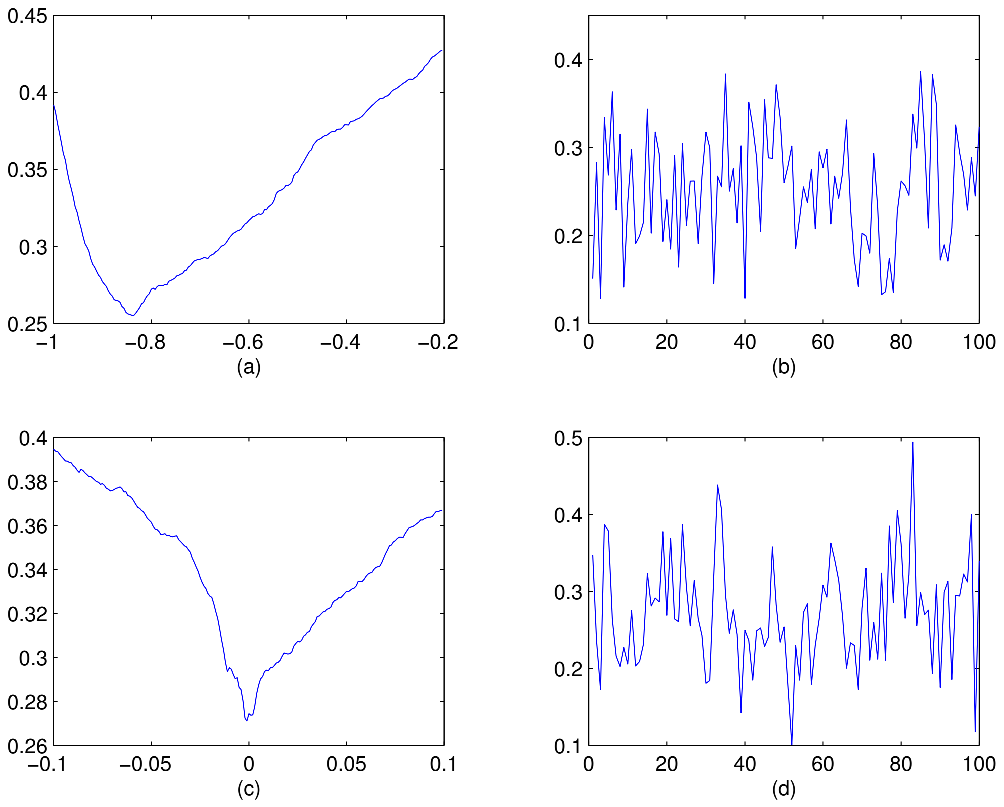


Figure 12.4: Experiments against HIVA-set. Left column: an average BER as a function of threshold parameter Δ, which was optimized for the whole CV-experiment with 100 folds (see Section 12.2.7). Right column: behavior of BERs as a function of fold-index where we used an optimal value of the threshold parameter. First row: RS+QM+reg.; second row: SVM-RBF.

The following relation is valid

$$\exp\{-2y_t F(\mathbf{x}_t)\} = \xi(\mathbf{x}_t) z_t^2 \tag{12.29}$$

where

$$z_t = \frac{y_t^\star - p(\mathbf{x}_t)}{\xi(\mathbf{x}_t)}, \quad \xi(\mathbf{x}_t) = p(\mathbf{x}_t)(1 - p(\mathbf{x}_t)).$$

We can maximize (12.28) using method with Newton's step, which is based on the matrix of second derivatives (Nikulin, 2006a). This option may be applicable in a low-dimensional case, for example, *ADA* or *SYLVA* sets. As an alternative, we can consider standard weighted *QM*-model:

$$\sum_{t=1}^{n} \xi(\mathbf{x}_t)(z_t - u_t)^2. \tag{12.30}$$

After solution $u(\mathbf{x}_t)$ was found, we update function $p(\mathbf{x}_t)$

$$p(\mathbf{x}_t) \leftarrow \begin{cases} 1 & \text{if} \quad h_t \geq 1; \\ h_t & \text{if} \quad 0 < h_t < 1; \\ 0 & \text{if} \quad h_t \leq 0 \end{cases} \tag{12.31}$$

where $h_t = p(\mathbf{x}_t) + \xi(\mathbf{x}_t)u(\mathbf{x}_t)$. Then, we recompute weight coefficients ξ, and return to the minimization criterion (12.30).

Let us consider update of function F assuming that $0 < h_t < 1$. By definition,

$$F_{\text{new}}(\mathbf{x}_t) = \frac{1}{2}\log\{\frac{h_t}{1-h_t}\} = \frac{1}{2}\log\{\frac{p(\mathbf{x}_t)}{1-p(\mathbf{x}_t)}\} + \frac{1}{2}\log\{1 + \frac{u(\mathbf{x}_t)}{1-p(\mathbf{x}_t)u(\mathbf{x}_t)}\}$$

$$\approx F(\mathbf{x}_t) + \nu \cdot u(\mathbf{x}_t), \quad \nu = 0.5. \tag{12.32}$$

Remark 12.10 *Boosting trick (similar to the well-known kernel trick): as an alternative to QM-solution, we can apply in (12.27) or (12.32) decision function, which was produced by another method, for example, Naïve Bayes or Decision Trees (Lutz, 2006).*

Remark 12.11 *Approximation (12.32) coincides with update formula of Friedman et al. (2000), and is valid under condition that value of $u(\mathbf{x}_t)$ is small enough. Lutz (2006) suggests careful approach with the following range $0.1 \leq \nu \leq 0.3$ depending on the particular dataset. Also, it appears to be reasonable (Friedman et al., 2000) to restrict values of z_t in (12.30). However, experiments against ADA-set (see Figure 12.3(d)) were conducted strictly according to the above formulas (12.31–12.32) with $\nu = 0.5$.*

12.3.4. LogitBoost2 Algorithm

Let us consider logit target function

$$L(\mathbf{w}) = \sum_{t=1}^{n} (y_t - \phi(u_t))^2, \quad \phi(u) = \tanh(u). \tag{12.33}$$

Above target function appears to be more natural comparing with squared loss, but we can not find an optimal value of the step-size in analytical form. Respectively, we will need to maintain low value of the step-size in order to ensure stability of the algorithm.

The following simple boosting procedure may be efficient in order to facilitate optimization process (see Figure 12.3(d)). Essentially, the procedure includes 2 steps (NN2-3):

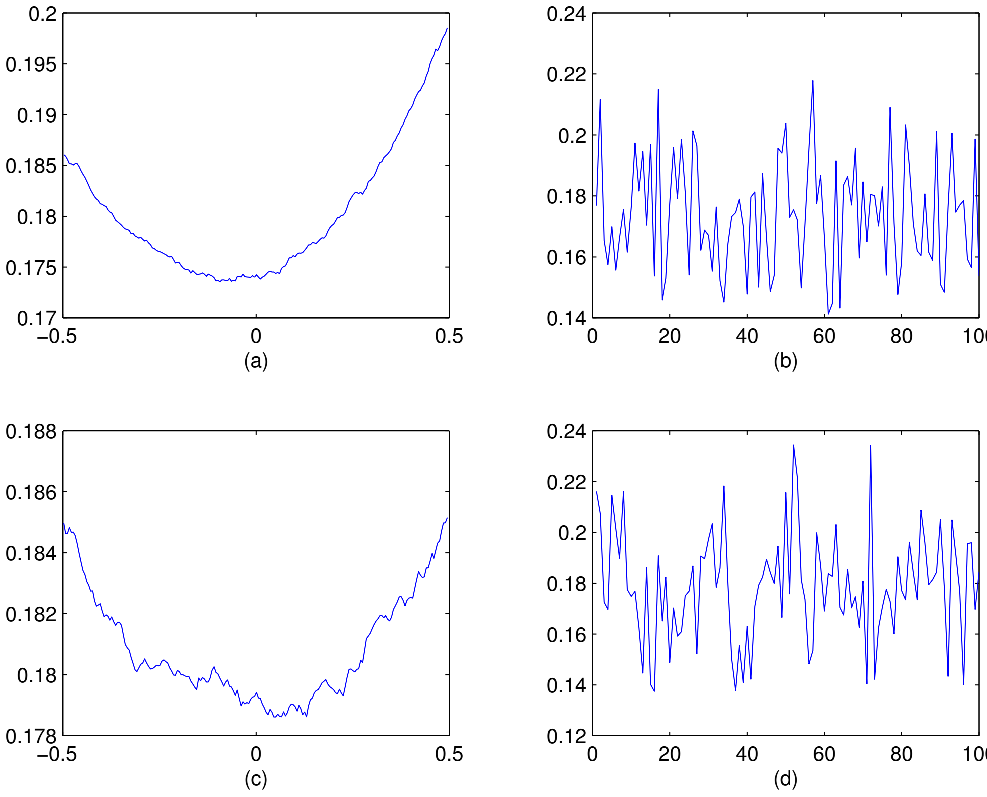


Figure 12.5: Experiments against ADA-set (used LogitBoost). Left column: an average BER as a function of threshold parameter Δ, which was optimized for the whole CV-experiment with 100 folds (see Section 12.2.7). Right column: behavior of BERs as a function of fold-index where we used an optimal value of the threshold parameter. First row: ADA-agnostic; second row: ADA-prior (see Section 12.5).

```
1: Set initial values: $j = 0$, $z_t^{(j)} = y_t$ and $p^{(j)}(\mathbf{x}_t) = 0, t = 1..n$;
2: find solution of the standard $QM$-problem
       $$L(\mathbf{w}) = \sum_{t=1}^{n} \left(z_t^{(j)} - u_t^{(j)}\right)^2 \qquad (12.34)$$
   where $j$ is a sequential number of iteration;
3: re-compute the target function
       $$z_t^{(j+1)} = y_t - p^{(j+1)}(\mathbf{x}_t),$$
       $$p^{(j+1)}(\mathbf{x}_t) \leftarrow \begin{cases} 1 & \text{if } p^{(j)}(\mathbf{x}_t) + u_t^{(j)} \geq 1; \\ p^{(j)}(\mathbf{x}_t) + u_t^{(j)} & \text{if } -1 < p^{(j)}(\mathbf{x}_t) + u_t^{(j)} < 1; \\ -1 & \text{if } p^{(j)}(\mathbf{x}_t) + u_t^{(j)} \leq -1. \end{cases} \qquad (12.35)$$
4: $j \leftarrow j + 1$, and goto step 2.
 Repeat $K$ times above steps 2–4 and use $p^{(K)}(\mathbf{x}_t)$ as a decision function.
```

Algorithm 12.2: EI-Boosting for the weight coefficients (see Section 12.3.5)

```
1: Let us consider model (12.9) of the Section 12.2.2.
2: Set initial weights $\xi^{(0)} = \xi$ as uniform.
3: Set initial value of optimal BER $Q_0 = 1$, and
4: select values of parameters $\alpha > 1$ and $0 < \beta < 1$.
5: Repeat for $k = 1..K$ the following steps 6-8:
6: Evaluate $QM$ model and compute the corresponding value of $BER\ Q$.
7: Make update $Q_0 = Q, \xi^{(0)} = \xi$ if $Q < Q_0$.
8: Boost mis-classified patterns with probability $\beta$
       $$\xi_t := \xi_t^{(0)} \cdot \alpha \quad \text{if} \quad f_t \cdot y_t = -1.$$
9: Based on the above experiment select the optimal value of $K$.
```

Remark 12.12 *Further, we can extend above model (combined with backpropagation algorithm, see Abid et al. (2006)) to the case of arbitrary neural networks with several hidden layers where computational speed may be very important.*

Remark 12.13 *Bootstrap Aggregation was introduced by Breiman (1996) as a means for improving accuracy of estimators, and it appears to be logical to combine bagging and boosting (Pfahringer, 2000).*

12.3.5. Experience-Innovation Approach

The motivation for *EI*-approach is very simple: after some standard experiments against *ADA* or *SYLVA*-sets we can make a conclusion that overfitting is very limited. Respectively, we would be interested to improve training results under expectation that the corresponding test results will follow. This target may be pursued by the natural approach: we propose to increase attention to the mis-classified patterns, and we can employ here two main methods: 1) increase weights (Algorithm 12.2), or 2) increase absolute values of the corresponding target functions (Algorithm 12.3).

Algorithm 12.3: EI-Boosting for the target function (see Section 12.3.5)

```
1: Let us consider model (12.9) of the Section 12.2.2.
2: Set initial values of target function $z^{(0)} = z_t = y_t, t = 1..n$.
3: Set initial value of optimal BER $Q_0 = 1$, and
4: select values of parameters $\alpha > 1$ and $0 < \beta < 1$.
5: Repeat for $k = 1..K$ the following steps 6-8:
6: Evaluate $QM$ model and compute the corresponding value of $BER\ Q$.
7: Make update $Q_0 = Q, z^{(0)} = z$ if $Q < Q_0$.
8: Boost mis-classified patterns with probability $\beta$
       $z_t := z_t^{(0)} \cdot \alpha \quad \text{if} \quad f_t \cdot y_t = -1.$
9: Based on the above experiment select the optimal value of $K$.
```

Remark 12.14 *There may be nearly identical vectors of features with opposite labels (see example of the Section 12.3.6). Respectively, it appears to be not a good idea to boost all misclassified patterns identically.*

12.3.6. Synthetic Set

The structure of this example (with non-linear dependence between explanatory variables x_i and target variable y) was motivated by ADA-set: it is very understandable that people with the same demographical characteristics may or may not have an income of $50,000 per year.

We define target variable according to the following rule:

$$y := \begin{cases} 1 \ \text{ if } \ \dfrac{h}{1+h} \geq \tau; \\ -1, \ \text{ alternatively,} \end{cases} \tag{12.36}$$

where

$$h = \exp\{z \cdot (x_1 - x_2 + 0.3 \cdot x_3)\}, \tag{12.37}$$

$$z := \begin{cases} 1 \ \text{ if } \ \theta \geq 0.8; \\ 1 - 2 \cdot \exp\{-\lambda\psi\}, \ \text{ alternatively,} \end{cases} \tag{12.38}$$

where $\lambda = 3; \tau = 0.45; \theta, \psi \sim R_{[0,1]}; x_i \sim \mathcal{N}(1,1), i = 1..\ell; \ell = 3$.

Remark 12.15 *Role of the regulation parameter z is very essential in (12.37): it is positive in most cases, but may be, also, negative.*

According to the above algorithm we simulated a sample of $n = 10000$ observations, which were used for testing of the Algorithms 12.2 and 12.3 (see Figure 12.3).

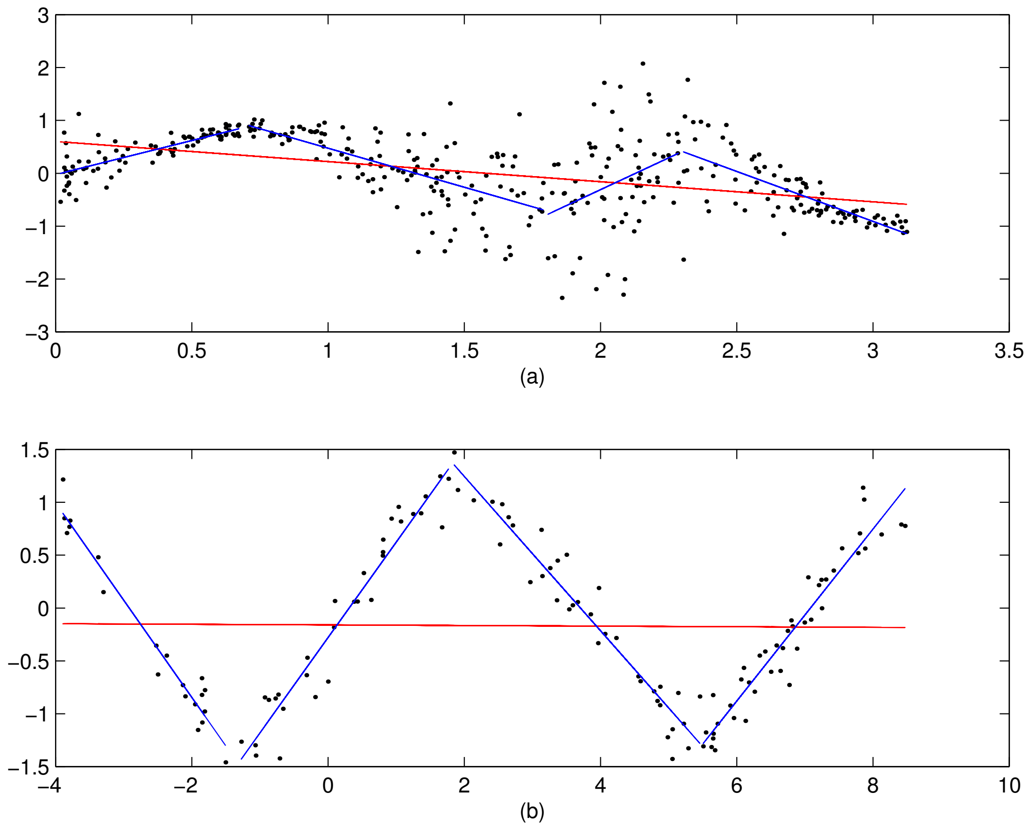


Figure 12.6: Distance-based clustering (see Section 12.4).

12.4. Distance-based Clustering

Let us consider synthetic example (see Figure 12.6(a)) with 384 observations from the Predictive Uncertainty in Environmental Modelling Challenge[3] (Gawley et al., 2006).

It is common to start any case study of a regression problem with linear modelling (Bagnall et al., 2006). The *QM* model produced $MSE = 0.4096$ (see red line in Figure 12.6(a)). Then, we decided to split the data (based on the visual consideration) into 4 clusters: 1) $x \leq 0.7$; 2) $x > 0.7$ and $x \leq 1.8$; 3) $x > 1.8$ and $x \leq 2.3$; 4) $x > 2.3$. As a next step we computed regression coefficients specifically for the particular clusters. Consequently, we created a non-linear classifier with $MSE = 0.2828$ (see blue line in Figure 12.6(a)).

Remark 12.16 *Further improvement may be achieved using specially selected transformation of the target variable. As a result, we can expect to reduce non-uniform variance or heteroscedasticity.*

3. http://theoval.cmp.uea.ac.uk/~gcc/competition/

Figure 12.6(b) illustrates the UDON data-set from the NIPS-2006 data-mining competition[4] "Learning when the test and training inputs have different distributions". Here we used also 4 clusters: 1) $x \leq -1.5$; 2) $x > -1.5$ and $x \leq 1.8$; 3) $x > 1.8$ and $x \leq 5.5$; 4) $x > 5.5$ with improvement of MSE from 0.6174 to 0.0384.

In the case of real high dimensional data (where visual consideration may not be possible) we can use k-means algorithm with Euclidean or Manhatten distance and special weight coefficients (12.9). For example, weight coefficients may be computed according to the RS method (see Section 12.2.4).

We can employ k-means algorithm in order to split available training dataset into several subsets/clusters where any cluster is represented by centroid. Then, we can compute specific vector of regression coefficients for any particular cluster.

The combination of two matrices 1) centroids and 2) coefficients may be used as an indirect non-linear kNN classifier, which may be described as a two steps procedure: for any data instance 1) find nearest centroid, and 2) compute decision function according to the corresponding vector of linear coefficients.

Remark 12.17 *Using prior information (see, for example, Table 12.3), we can split data according to one or several features as it is described in the Section 12.2.6. But, in difference to the Section 12.2.6 uniformity of the clusters in the sense of labels is not an issue here.*

Remark 12.18 *Also, we can use likelihood-based splitter, see Nikulin and Smola (2005).*

12.4.1. Generalized Linear Mixture Model (GLiMix)

The *GLiMix* is an essentially different comparing with the popular *EM*-algorithm (Dempster et al., 1977). Using *GLiMix* we can investigate uniformity of training data in terms of relations between the target and explanatory variables. Then, we apply these insights in order to optimize construction of the distance-based splitter.

The definition of the algorithm is based on the following log-likelihood function:

$$L(m) = \sum_{t=1}^{n} \sum_{c=1}^{m} p_{tc} \cdot \log\{\pi_c \cdot f_{\sigma_c}(y_t - u_{tc})\}, \; u_{tc} = \sum_{j=1}^{\ell} w_{cj} \cdot x_{tj}, \tag{12.43}$$

where p_{tc} are probabilities of memberships of data-instance $\mathbf{x}_t$ within cluster c; $\pi_c, c = 1 \dots m$, are prior probabilities; f_σ is a density of normal distribution:

$$f_\sigma(u) \sim \frac{1}{\sigma} \exp\{-\frac{u^2}{2\sigma^2}\}.$$

In order to simplify notations it is assumed that the constant is one of the features.

Remark 12.19 *Note that hard-clustering analogue of the above algorithm GLiC may be found in Nikulin and Smola (2005).*

12.5. Experiments

We considered only three sets in the prior track: ADA, GINA and SYLVA where GINA and SYLVA have the same format as the corresponding sets from the agnostic track. The data related to ADA require some preprocessing, which was conducted according to the following Table 12.3:

Algorithm 12.4: GLiMix

```
1: Select number of segments $m$.
2: Select initial values of $\pi$, $\sigma$ and $w$.
3: Compute probabilities of membership according to Bayesian formula:
   $$p_{tc} = \frac{\pi_c \cdot f_{\sigma_c}(y_t - u_{tc})}{\sum_{j=1}^{m} \pi_j \cdot f_{\sigma_j}(y_t - u_{tj})}, \ t = 1..n, c = 1..m. \qquad (12.39)$$
4: Re-compute prior probabilities:
   $$\pi_c = \frac{1}{n} \sum_{t=1}^{n} p_{tc}. \qquad (12.40)$$
5: Re-compute linear coefficients:
   $$B_{cv} = \sum_{j=1}^{\ell} A_{cvj} \cdot w_{cj}, \ v = 1..\ell, \qquad (12.41)$$
   where
   $$B_{cv} = \sum_{t=1}^{n} p_{tc} \cdot y_t \cdot x_{tv}; \ A_{cvj} = \sum_{t=1}^{n} p_{tc} \cdot x_{tv} \cdot x_{tj}.$$
6: Re-compute standard deviations:
   $$\sigma_c^2 = \frac{\sum_{t=1}^{n} p_{tc} \cdot (y_t - u_{tc})^2}{\sum_{t=1}^{n} p_{tc}}. \qquad (12.42)$$
7: Repeat above steps 3-6 until convergence.
```

Table 12.3: Preprocessing of ADA-prior information into numerical matrix with 127 columns or features.

1	Age	categorical with 32 dummy variables (step size is 2 years);
2	Employment	categorical with 6 dummy variables;
3	fnlwgt	continuous, used transformation: $\log\{\frac{x}{9999}+1\}$;
4	Education	categorical with 15 dummy variables;
5	Education level	continuous, used transformation: $\log\{x+1\}$;
6	Marital status	categorical with 6 dummy variables;
7	Profession	categorical with 13 dummy variables;
8	Family	categorical with 5 dummy variables;
9	Race	categorical with 4 dummy variables;
10	Sex	dummy variables $\{0,1\}$;
11	Capital-gain	continuous, used transformation: $\log\{\frac{x}{100}+1\}$;
12	Capital-loss	continuous, used transformation: $\log\{\frac{x}{100}+1\}$;
13	Hours per week	categorical with 40 dummy variables (step size is 2 hours);
14	Native Country	dummy variables $\{0,1\}$ where 1 was used for USA.

Table 12.4: Winning results of the IJCNN-2007 Agnostic Learning *vs.* Prior Knowledge Challenge (used only 5 last complete entries).

Agnostic track				
Data	Entrant name	Entry name	Test-BER	AUC
ADA	Roman Lutz	LogitBoost with trees	0.166	0.9168
GINA	Roman Lutz	LogitBoost/Doubleboost	0.0339	0.9668
HIVA	Vojtech Franc	RBF SVM	0.2827	0.7707
NOVA	Mehreen Saeed	Submit E final	0.0456	0.9552
SYLVA	Roman Lutz	LogitBoost with trees	0.0062	0.9938
Overall	Roman Lutz	LogitBoost with trees	0.1117	0.8892
Prior track				
ADA	Marc Boulle	Data Grid	0.1756	0.8464
GINA	Vladimir Nikulin	vn2	0.0226	0.9777
HIVA	Chloe Azencott	SVM	0.2693	0.7643
NOVA	Jorge Sueiras	Boost mix	0.0659	0.9712
SYLVA	Roman Lutz	Doubleboost	0.0043	0.9957
Overall	Vladimir Nikulin	vn3	0.1095	0.8949

Tables 12.4 and 12.5 illustrate some of the test results. It appears that none of the models may be regarded as a perfect for all datasets. Based on our experimental results, which are presented in the Table 12.5, we can recommend the following selections: LogitBoost for ADA and SYLVA; RBF-SVM for GINA ($\gamma = 0.013$); LinearSVM for NOVA and regularized linear model for HIVA with ridge parameter $\mu = 0.07$. Note that the entry "SVM+GbO+trees" dated *15th November 2006* with very competitive overall result 0.1139 was produced using only specially developed original software written in C (means without any involvement of R-packages, CLOP, TreeNet or similar). Guyon et al. (2007) noted that the best performing complete entries do not necessarily include best individual entries. For example, if we will optimise the structure of the complete entries using information of the above Table 12.5, the results will be 0.1125 for agnostic, and 0.1068 for prior tracks. According to our experimental results, an advantage of SYLVA-prior was not significant in difference to GINA-prior. We spent a lot of time for the preprocessing of ADA-prior information, and, surprisingly, did not achieved any improvement. Moreover, the performance of ADA-prior was slightly worse comparing with ADA-agnostic.

We used an opportunity of the challenge to test CLOP Version 1.1 – October, 2006. The most basic (and sufficient) instructions may be found on the last page of Guyon et al. (2007). The package is a quite efficient and can produce competitive results in application to any dataset of the Challenge. It is very easy to arrange suitable cross validations with required number of folds in order to evaluate any particular model, and there is a wide range of choices. For example, we can recommend the settings in Table 12.6.

All necessary details in relation to the above models may be found in the file "model_examples.m" in the directory "../CLOP/sample_code".

In order to check the quality of the model we used cross-validation as it is described in the Section 12.2.7 with number of folds from 10 to 100 depending on the complexity of the model.

4. `http://different.kyb.tuebingen.mpg.de/`

Table 12.5: Some selected experimental results where abbreviation "FS-RS" means feature selection with random sets; "ADA" means package in R, "RF" means randomForest package in R.

Data	Method	FS-RS	Test-BER	AUC
ADA	ADA($l = 100, \nu = 0.3$)	38	0.1751	0.8331
ADA	TreeNet (Salford Systems)	38	0.1786	0.8306
ADA	CLOP-gentleboost (neural)	38	0.183	0.8213
ADA	LogitBoost	38	0.1838	0.8038
ADA	Exponential	38	0.1847	0.8066
ADA-prior	ADA($l = 50, \nu = 0.3$)	108	0.1788	0.8225
ADA-prior	TreeNet (Salford Systems)	108	0.1817	0.805
ADA-prior	Exponential	108	0.186	0.8189
ADA-prior	QM	108	0.1875	0.8172
ADA-prior	CM2+QM	108	0.1886	0.7892
ADA-prior	CLOP-gentleboost (neural)	108	0.195	0.7928
GINA-prior	CLOP-svc	All	0.0226	0.9777
GINA-prior	SVM-RBF	540	0.0266	0.975
GINA	CLOP-svc	All	0.0503	0.9507
GINA	SVM-RBF	720	0.0535	0.9464
GINA	RVM	720	0.0546	0.95
HIVA	SVM-RBF ($\gamma = 0.05$)	797	0.282	0.7104
HIVA	QM+reg. ($\mu = 0.07$)	797	0.2833	0.7322
HIVA	RVM ($\mu = 0.1$)	797	0.2916	0.7296
HIVA	RF ($200, 30, 10$)	797	0.2929	0.742
HIVA	CLOP-gentleboost (kridge)	All	0.297	0.7105
HIVA	Naïve Bayes ($\phi = 0.006, m = 200$)	400	0.3025	0.6908
NOVA	CLOP-gentleboost (neural)	All	0.0471	0.9456
NOVA	LinearSVM	3200	0.0589	0.9345
NOVA	CLOP-gentleboost (kridge)	All	0.0601	0.9289
NOVA	RVM+reg.	3200	0.0696	0.9585
SYLVA-prior	ADA($l = 50, \nu = 0.3$)	36	0.0071	0.9959
SYLVA-prior	CLOP-gentleboost (neural)	All	0.0075	0.9918
SYLVA-prior	TreeNet (Salford Systems)	36	0.0076	0.994
SYLVA-prior	AdaBoost+trees	36	0.0084	0.9924
SYLVA	TreeNet (Salford Systems)	74	0.0082	0.9939
SYLVA	RF($600, 10, 10$)	74	0.0084	0.9917
SYLVA	AdaBoost+trees	74	0.0087	0.9966
SYLVA	ADA($l = 50, \nu = 0.3$)	74	0.0096	0.9933
SYLVA	CLOP-gentleboost (neural)	All	0.0115	0.9858

Table 12.6: CLOP settings for GINA and NOVA.

Data	Model	CLOP-specifications
GINA	base_model	svc('coef0=0.1', 'degree=7', 'gamma=0.005', 'shrinkage=0.01')
	my_model	chain(normalize , base_model , bias)
NOVA	base_model	neural('units=1', 'shrinkage=0.2', 'balance=1', 'maxiter=50')
	my_model	chain(normalize , gentleboost(base_model , 'units=5') , bias)

12.5.1. Small and Low-Dimensional Datasets

In the case of small datasets we can optimize exponential or logit losses directly using gradient-based optimization (Nikulin, 2006a). Table 12.7 shows some results, which were obtained using data of the data-mining competition “Learning when the test and training inputs have different distributions”.

Table 12.7: Values in the first column “Data” indicate sizes of the samples for training, testing and validation. The best results are given in bold style. Full convergence of the algorithms was achieved after less than 30000 iterations with step size 0.0001 (about 1min time).

		Exponential model				Logit model			
Data	ℓ	Train	Test	Valid	ρ	Train	Test	Valid	ϕ
Barley: 400-1000-50	5	0.2625	0.1402	0.1336	0.5	0.2201	**0.1229**	0.1093	0.0001
Wheat: 400-1000-50	5	0.2288	**0.2674**	0.3064	0.5	0.2321	0.2694	0.2920	0.005
Schitzel: 280-185-40	3	0.5964	**0.6732**	0.6879	1	0.5489	0.7439	0.7540	0.1

The competition criterion, an average negative log estimated predictive probability of the true labels,

$$-\frac{1}{n}\left[\sum_{t:y_t=1}\log\{\hat{p}(\mathbf{x}_t)\}+\sum_{t:y_t=-1}\log\{1-\hat{p}(\mathbf{x}_t)\}\right] \tag{12.44}$$

may be computed using values of the decision functions $u(\mathbf{x}_t)$. We can estimate required values of probabilities in (12.44) according to the standard method

$$\hat{p}(\mathbf{x}_t)=\left[\frac{\exp\{u(\mathbf{x}_t)+a\}}{1+\exp\{u(\mathbf{x}_t)+a\}}\right]^b. \tag{12.45}$$

In order to avoid ∞-result we must apply truncation according to the formula (12.17) where suitable values of ϕ may be found in the last column of the Table 12.7.

Remark 12.20 *Parameters a and b in (12.45) represent an analogues of the threshold parameter* Δ. *These parameters may be very effective in application to imbalanced datasets. For example, we used 1)* $a=-1.07, b=2.51$ *in the case of Exponential model, and 2)* $a=-1.51, b=2.01$ *in the case of Logit model (Barley, see Table 12.7).*

12.6. Concluding Remarks

The main idea behind *RS*-method may be implemented in conjunction with different base models: for example, decision trees, quadratic-minimization or Naïve Bayes classifiers. The proposed method is an essentially different comparing with Random Forests where the final classification is given by the majority of voting on the ensemble of trees in the forest (Dahinden, 2006). Assuming that overfitting is limited we can estimate importance of the single subset of features according to the training simulation result. Then, based on CV-evaluation, we can select an optimal number of best performing features and recompute the final classifier.

The motivation for *EI*-boosting is very simple: heuristical innovation of the best past experience. Generally, boosting may be efficient in the case when overfitting is limited. Respectively, it appears to be natural to boost any single *RS*-classifier.

The current experimental results with Distance-based Clustering are promising and may be improved further. For example, *RS*-method or *GLiMix* algorithm may be efficient in order to optimize selection of the distance-based splitter.

Finally, we considered in this paper several methods which may be used independently or in conjunction. We can not expect that any of the methods may demonstrate an absolute superiority against the others. Therefore, performance of the particular method depends on the dataset, and the main strength of our approach rests on flexibility.

Acknowledgments

This paper passed very long but useful review process and I would like to thank all involved reviewers. In particular, I am grateful to Isabelle Guyon and Gavin Cawley.

References

S. Abid, A. Mouelhi, and F. Fnaiech. Accelerating the multilayer perceptron learning with the Davidon Fletcher Powell algorithm. In *International Joint Conference on Neural Networks, Vancouver, BC, Canada, July 16-21*, pages 6421–6426. IEEE, 2006.

A. Bagnall, I. Whittley, M. Studley, M. Pettipher, F. Tekiner, and L. Bull. Variance stabilizing regression ensembles for environmental models. In *International Joint Conference on Neural Networks, Vancouver, BC, Canada, July 16-21*, pages 11004–11110. IEEE, 2006.

B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Fifth COLT, Pittsburgh, USA*, pages 144–152, 1992.

M. Boulle. Regularization and averaging of the selective Naive Bayes classifier. In *International Joint Conference on Neural Networks, Vancouver, BC, Canada, July 16-21*, pages 2989–2997. IEEE, 2006.

L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and regression trees. 1984.

C. Dahinden. Classification with tree-based ensembles applied to the WCCI-2006 performance prediction challenge datasets. In *International Joint Conference on Neural Networks, Vancouver, BC, Canada, July 16-21*, pages 2978–2981. IEEE, 2006.

A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39:1–38, 1977.

Y. Freund and R. Schapire. A decision-theoretic generalization of online learning and an application to boosting. *J. Comput. System Sciences*, 55:119–139, 1997.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:337–374, 2000.

G. Gawley, M. Haylock, and S. Dorling. Predictive uncertainty in environmental modelling. In *International Joint Conference on Neural Networks, Vancouver, BC, Canada, July 16-21*, pages 11096–11103. IEEE, 2006.

I. Guyon, A. Saffari, G. Dror, and G. Gawley. Agnostic learning vs. prior knowledge challenge. In *International Joint Conference on Neural Networks, Orlando, Florida, August 12-17*. IEEE, 2007.

S. Kurogi, D. Kuwahara, and S. Tanaka. Ensemble of competitive associative nets and multiple k-fold cross-validation for estimating predictive uncertainty in environmental modelling. In *International Joint Conference on Neural Networks, Vancouver, BC, Canada, July 16-21*, pages 11111–11115. IEEE, 2006.

R. Lutz. LogitBoost with trees applied to the WCCI 2006 performance prediction challenge datasets. In *International Joint Conference on Neural Networks, Vancouver, BC, Canada, July 16-21*, pages 2966–2969. IEEE, 2006.

V. Nikulin. Learning with mean-variance filtering, SVM and gradient-based optimization. In *International Joint Conference on Neural Networks, Vancouver, BC, Canada, July 16-21*, pages 4195–4202. IEEE, 2006a.

V. Nikulin. Weighted threshold-based clustering for intrusion detection systems. *International Journal of Computational Intelligence and Applications*, 6(1):1–19, 2006b.

V. Nikulin and A. Smola. Parametric model-based clustering. In B. Dasarathy, editor, *Data Mining, Intrusion Detection, Information Assurance, and Data Network Security, 28-29 March 2005, Orlando, Florida, USA*, volume 5812, pages 190–201. SPIE, 2005.

B. Pfahringer. Winning the KDD99 classification cup: Bagged Boosting. In *SIGKDD Explorations*, volume 1 of *2*, pages 65–66. ACM SIGKDD, 2000.

M. Tipping. Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

E. Tuv, A. Borisov, and K. Torkkola. Feature selection using ensemble based ranking against artificial contrasts. In *International Joint Conference on Neural Networks, Vancouver, BC, Canada, July 16-21*, pages 4183–4187. IEEE, 2006.

J. Wichard. Model selection in an ensemble framework. In *International Joint Conference on Neural Networks, Vancouver, BC, Canada, July 16-21*, pages 4189–4194. IEEE, 2006.

Part V

Multi-level Inference

Overview

While ensemble methods presented in Part IV are attractive because of their ease of use and excellent performance, they require significant computer resources and end up with a complex and difficult to interpret classifier. In this part, the authors present advanced techniques to search for a single optimum model, whose performances can approach or match those of an ensemble. Robust parameter estimation methods (Part III) addressed overfitting avoidance at the first level of inference (parameter estimation). **Multi-level inference methods address the problem of overfitting avoidance at the second level of inference: model selection.** In Chapter 13, **Gavin C. Cawley and Nicola L. C. Talbot present a method for regularizing the second level of inference** based on Bayesian priors. They use a least-square kernel classifier (LSSVM) with feature scaling factors (so-called ARD prior). The hyperparameters of the first level of inference (the so-called "ridge", aka the Lagrange multiplier for the 2-norm regularizer, and the kernel parameters) are adjusted on the basis of the leave-one-out (least-square) error, as given by the PRESS statistic. This criterion is itself regularized by the 2-norm of the vector of scaling factors. Using the Bayesian prior methodology, the authors obtain a surprisingly simple and intuitive result: the resulting two-part cost function at the second level of inference weighs PRESS with the number of training examples and the regularizer with the number of features. In Chapter 14, **H. Jair Escalante, Manuel Montes and Enrique Sucar devote their attention to the problem of full model selection,** namely the search for an optimum model in a large combinatorial space offered by a machine learning toolbox. They use the CLOP package provided by the challenge organizers (See Appendix C) and an efficient search method, "particle swarm optimization", which is a biologically inspired search algorithm. The authors ranked fourth overall in the ALvsPK challenge (agnostic track) and Juha Reunanen, using also an extensive search strategy (see Appendix A) obtained second place, also with CLOP models. In Chapter 15, **Gautam Kunapuli, Jong-Shi Pang, and Kristin P. Bennett use bilevel optimization** to simultaneously optimize parameters and hyperparameters, and thus have the two-levels of inference talk together in the optimization process. All these pioneering works point to a new direction of research in which, via a proper overfitting prevention at the second level of inference, it is possible to do full model selection is the large model spaces offered by machine learning toolboxes.

Journal of Machine Learning Research 8(Apr):841–861, 2007 Submitted 11/06; Published 4/07

Chapter 13

Preventing Over-Fitting during Model Selection via Bayesian Regularisation of the Hyper-Parameters

Gavin C. Cawley GCC@CMP.UEA.AC.UK
Nicola L. C. Talbot NLCT@CMP.UEA.AC.UK
School of Computing Sciences
University of East Anglia
Norwich, United Kingdom NR4 7TJ

Editor: Isabelle Guyon

Abstract

While the model parameters of a kernel machine are typically given by the solution of a convex optimisation problem, with a single global optimum, the selection of good values for the regularisation and kernel parameters is much less straightforward. Fortunately the leave-one-out cross-validation procedure can be performed or a least approximated very efficiently in closed form for a wide variety of kernel learning methods, providing a convenient means for model selection. Leave-one-out cross-validation based estimates of performance, however, generally exhibit a relatively high variance and are therefore prone to over-fitting. In this paper, we investigate the novel use of Bayesian regularisation at the second level of inference, adding a regularisation term to the model selection criterion corresponding to a prior over the hyper-parameter values, where the additional regularisation parameters are integrated out analytically. Results obtained on a suite of thirteen real-world and synthetic benchmark datasets clearly demonstrate the benefit of this approach.

13.1. Introduction

Leave-one-out cross-validation (Lachenbruch and Mickey, 1968; Luntz and Brailovsky, 1969; Stone, 1974) provides the basis for computationally efficient model selection strategies for a variety of kernel learning methods, including the Support Vector Machine (SVM) (Cortes and Vapnik, 1995; Chapelle et al., 2002), Gaussian Process (GP) (Rasmussen and Williams, 2006; Sundararajan and Keerthi, 2001), Least-Squares Support Vector Machine (LS-SVM) (Suykens and Vandewalle, 1999; Cawley and Talbot, 2004), Kernel Fisher Discriminant (KFD) analysis (Mika et al., 1999; Cawley and Talbot, 2003; Saadi et al., 2004; Bo et al., 2006) and Kernel Logistic Regression (KLR) (Keerthi et al., 2005; Cawley and Talbot, 2007). These methods have proved highly successful for kernel machines having only a small number of hyper-parameters to optimise, as demonstrated by the set of models achieving the best average score in the WCCI-2006 performance prediction challenge[1] (Cawley, 2006; Guyon et al., 2006). Unfortunately, while leave-one-out cross-validation estimators have been shown to be almost unbiased (Luntz and Brailovsky, 1969), they are known to exhibit a relatively high variance (e.g. Kohavi, 1995). A kernel with many hyper-parameters, for instance those used in Automatic Relevance Determination (ARD) (e.g. Rasmussen and Williams, 2006) or feature scaling methods (Chapelle et al., 2002; Bo et al., 2006), may provide sufficient degrees of freedom to over-fit leave-one-out

1. http://www.modelselect.inf.ethz.ch/index.php

cross-validation based model selection criteria, resulting in performance inferior to that obtained using a less flexible kernel function. In this paper, we investigate the novel use of regularisation (Tikhonov and Arsenin, 1977) of the hyper-parameters in model selection in order to ameliorate the effects of the high variance of leave-one-out cross-validation based selection criteria, and so improve predictive performance. The regularisation term corresponds to a zero-mean Gaussian prior over the values of the kernel parameters, representing a preference for smooth kernel functions, and hence a relatively simple classifier. The regularisation parameters introduced in this step are integrated out analytically in the style of Buntine and Weigend (1991), to provide a Bayesian model selection criterion that can be optimised in a straightforward manner via, e.g., scaled conjugate gradient descent (Williams, 1991).

The paper is structured as follows: The remainder of this section provides a brief overview of the least-squares support vector machine, including the use of leave-one-out cross-validation based model selection procedures, given in sufficient detail to ensure the reproducibility of the results. Section 13.2 describes the use of Bayesian regularisation to prevent over-fitting at the second level of inference, i.e. model selection. Section 13.3 presents results obtained over a suite of thirteen benchmark datasets, which demonstrate the utility of this approach. Section 13.4 provides discussion of the results and suggests directions for further research. Finally, the work is summarised and directions for further work are outlined in Section 13.5.

13.1.1. Least Squares Support Vector Machine

In the remainder of this section, we provide a brief overview of the least-squares support vector machine (Suykens and Vandewalle, 1999) used as the testbed for the investigation of the role of regularisation in the model selection process described in this study. Given training data,

$$\mathscr{D} = \{(x_i,\, y_i)\}_{i=1}^{\ell}, \quad \text{where} \quad x_i \in \mathscr{X} \subset \mathbb{R}^d \quad \text{and} \quad y_i \in \{-1, +1\},$$

we seek to construct a linear discriminant, $f(x) = \phi(x) \cdot w + b$, in a *feature* space, $\mathscr{F}$, defined by a fixed transformation of the input space, $\phi : \mathscr{X} \to \mathscr{F}$. The parameters of the linear discriminant, $(w,\, b)$, are given by the minimiser of a *regularised* (Tikhonov and Arsenin, 1977) least-squares training criterion,

$$L = \frac{1}{2}\|w\|^2 + \frac{1}{2\mu}\sum_{i=1}^{\ell}\left[y_i - \phi(x_i)\cdot w - b\right]^2, \tag{13.1}$$

where μ is a regularisation parameter controlling the bias-variance trade-off (Geman et al., 1992). Rather than specify the feature space directly, it is instead induced by a kernel function, $\mathscr{K} : \mathscr{X} \times \mathscr{X} \to \mathbb{R}$, which evaluates the inner-product between the projections of the data onto the feature space, $\mathscr{F}$, i.e. $\mathscr{K}(x, x') = \phi(x)\cdot\phi(x')$. The interpretation of an inner-product in a fixed feature space is valid for any Mercer kernel (Mercer, 1909), for which the Gram matrix, $K = [k_{ij} = \mathscr{K}(x_i, x_j)]_{i,j=1}^{\ell}$ is positive semi-definite, i.e.

$$a^T K a \geq 0, \qquad \forall\, a \in \mathbb{R}^{\ell}, \qquad a \neq 0.$$

The Gram matrix effectively encodes the spatial relationships between the projections of the data in the feature space, $\mathscr{F}$. A linear model can thus be implicitly constructed in the feature space using only information contained in the Gram matrix, without explicitly evaluating the positions of the data in the feature space via the transformation $\phi(\cdot)$. Indeed, the *representer* theorem (Kimeldorf and Wahba, 1971) shows that the solution of the optimisation problem (13.1) can be written as an expansion over the training patterns,

$$w = \sum_{i=1}^{\ell} \alpha_i \phi(x_i) \qquad \Longrightarrow \qquad f(x) = \sum_{i=1}^{\ell} \alpha_i \mathscr{K}(x_i, x) + b.$$

The advantage of the "kernel trick" then becomes apparent; a linear model can be constructed in an extremely rich, high- (possibly infinite-) dimensional feature space, using only finite-dimensional quantities, such as the Gram matrix, K. The "kernel trick" also allows the construction of statistical models that operate directly on structured data, for instance strings, trees and graphs, leading to the current interest in kernel learning methods in computational biology (Schölkopf et al., 2004) and text-processing (Joachims, 2002). The Radial Basis Function (RBF) kernel,

$$\mathscr{K}(x,x') = \exp\left\{-\eta\|x-x'\|^2\right\} \tag{13.2}$$

is commonly encountered in practical applications of kernel learning methods, here η is a *kernel parameter*, controlling the sensitivity of the kernel function. The feature space for the radial basis function kernel consists of the positive orthant of an infinite-dimensional unit hyper-sphere (e.g. Shawe-Taylor and Cristianini, 2004). The Gram matrix for the radial basis function kernel is thus of full rank (Micchelli, 1986), and so the kernel model is able to form an arbitrary shattering of the data.

13.1.1.1. A DUAL TRAINING ALGORITHM

The basic training algorithm for the least-squares support vector machine (Suykens and Vandewalle, 1999) views the regularised loss function (13.1) as a constrained minimisation problem:

$$\min_{w,b,\varepsilon_i} \frac{1}{2}\|w\|^2 + \frac{1}{2\mu}\sum_{i=1}^{\ell}\varepsilon_i^2 \qquad \text{subject to} \qquad \varepsilon_i = y_i - w\cdot\phi(x_i) - b.$$

The primal Lagrangian for this constrained optimisation problem gives the unconstrained minimisation problem defined by the following regularised loss function,

$$\mathscr{L} = \frac{1}{2}\|w\|^2 + \frac{1}{2\mu}\sum_{i=1}^{\ell}\varepsilon_i^2 - \sum_{i=1}^{\ell}\alpha_i\left\{w\cdot\phi(x_i) + b + \varepsilon_i - y_i\right\}, \tag{13.3}$$

where $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_\ell) \in \mathbb{R}^\ell$ is a vector of Lagrange multipliers. The optimality conditions for this problem can be expressed as follows:

$$\frac{\partial\mathscr{L}}{\partial w} = 0 \implies w = \sum_{i=1}^{\ell}\alpha_i\phi(x_i) \tag{13.4}$$

$$\frac{\partial\mathscr{L}}{\partial b} = 0 \implies \sum_{i=1}^{\ell}\alpha_i = 0 \tag{13.5}$$

$$\frac{\partial\mathscr{L}}{\partial \varepsilon_i} = 0 \implies \alpha_i = \frac{\varepsilon_i}{\mu}, \quad \forall_i \in \{1,2,\ldots,\ell\} \tag{13.6}$$

$$\frac{\partial\mathscr{L}}{\partial \alpha_i} = 0 \implies w\cdot\phi(x_i) + b + \varepsilon_i - y_i = 0, \quad \forall\, i \in \{1,2,\ldots,\ell\}. \tag{13.7}$$

Using (13.4) and (13.6) to eliminate w and $\varepsilon = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_\ell)$, from (13.7), we find that

$$\sum_{j=1}^{\ell}\alpha_j\phi(x_j)\cdot\phi(x_i) + b + \mu\alpha_i = y_i \qquad \forall\, i \in \{1,2,\ldots,\ell\}. \tag{13.8}$$

Noting that $\mathscr{K}(x,x') = \phi(x)\cdot\phi(x')$, the system of linear equations, (13.8) and (13.5), can be written more concisely in matrix form as

$$\begin{bmatrix} K+\mu I & 1 \\ 1^T & 0 \end{bmatrix}\begin{bmatrix} \alpha \\ b \end{bmatrix} = \begin{bmatrix} y \\ 0 \end{bmatrix},$$

where $K = [k_{ij} = \mathscr{K}(x_i, x_j)]_{i,j=1}^{\ell}$, I is the $\ell \times \ell$ identity matrix and 1 is a column vector of ℓ ones. The optimal parameters for the model of the conditional mean can then be obtained with a computational complexity of $\mathscr{O}(\ell^3)$ operations, using direct methods, such as Cholesky decomposition (Golub and Van Loan, 1996).

13.1.1.2. Efficient Implementation Via Cholesky Decomposition

A more efficient training algorithm can be obtained, taking advantage of the special structure of the system of linear equations (Suykens et al., 2002). The system of linear equations to be solved in fitting a least-squares support vector machine is given by,

$$\begin{bmatrix} M & 1 \\ 1^T & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ b \end{bmatrix} = \begin{bmatrix} y \\ 0 \end{bmatrix}, \tag{13.9}$$

where $M = K + \mu I$. Unfortunately the matrix on the left-hand side is not positive definite, and so we cannot solve this system of linear equations directly using the Cholesky decomposition. However, the first row of (13.9) can be re-written as

$$M\left(\alpha + M^{-1}1b\right) = y \tag{13.10}$$

Rearranging (13.10), we see that $\alpha = M^{-1}(y - 1b)$, using this result to eliminate α, the second row of (13.9) can be written as,

$$1^T M^{-1} 1 b = 1^T M^{-1} y \tag{13.11}$$

The system of linear equations can then be re-written as

$$\begin{bmatrix} M & 0 \\ 0^T & 1^T M^{-1} 1 \end{bmatrix} \begin{bmatrix} \alpha + M^{-1} 1 b \\ b \end{bmatrix} = \begin{bmatrix} y \\ 1^T M^{-1} y \end{bmatrix}. \tag{13.12}$$

In this case, the matrix on the left hand side is positive-definite, as $M = K + \lambda I$ is positive-definite and $1^T M^{-1} 1$ is positive since the inverse of a positive definite matrix is also positive definite. The revised system of linear equations (13.12) can be solved as follows: First solve

$$M\rho = 1 \qquad \text{and} \qquad M\nu = y, \tag{13.13}$$

which may be performed efficiently using the Cholesky factorisation of M. The model parameters of the least-squares support vector machine are then given by

$$b = \frac{1^T \nu}{1^T \rho} \qquad \text{and} \qquad \alpha = \nu - \rho b.$$

The two systems of linear equations (13.13) can be solved efficiently using the Cholesky decomposition of $M = R^T R$, where R is the upper triangular Cholesky factor of M.

13.1.2. Leave-One-Out Cross-Validation

Cross-validation (Stone, 1974) is commonly used to obtain a reliable estimate of the test error for performance estimation or for use as a model selection criterion. The most common form, k-fold cross-validation, partitions the available data into k disjoint subsets. In each iteration a classifier is trained on a different combination of $k - 1$ subsets and the unused subset is used to estimate the test error rate. The k-fold cross-validation estimate of the test error rate is then simply the average of the test error rate observed in each of the k iterations, or folds. The most extreme form of cross-validation, where $k = \ell$ such that the test partition in each fold consists of only a single pattern, is known as leave-one-out cross-validation (Lachenbruch and Mickey, 1968) and has been shown to provide an almost unbiased estimate of the test error rate (Luntz and Brailovsky, 1969). Leave-one-out cross-validation is however computationally expensive, in the case of the least-squares support vector machine a naïve implementation having a complexity of $\mathscr{O}(\ell^4)$ operations. Leave-one-out cross-validation is therefore normally only used in circumstances where the available data are extremely scarce such that the computational expense is no longer prohibitive. In this case the inherently high variance of the leave-one-out estimator (Kohavi, 1995) is offset by the minimal decrease in the size of the training set in each fold, and so may provide a more reliable estimate of generalisation performance than conventional k-fold cross-validation. Fortunately leave-one-out cross-validation of least-squares support vector machines can be performed in closed form with a computational complexity of only $\mathscr{O}(\ell^3)$ operations Cawley and Talbot (2004). Leave-one-out cross-validation can then be used in medium to large scale applications, where there may be a few thousand data-points, although the relatively high variance of this estimator remains potentially problematic.

13.1.2.1. VIRTUAL LEAVE-ONE-OUT CROSS-VALIDATION

The optimal values of the parameters of a Least-Squares Support Vector Machine are given by the solution of a system of linear equations:

$$\begin{bmatrix} K+\mu I & 1 \\ 1^T & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ b \end{bmatrix} = \begin{bmatrix} y \\ 0 \end{bmatrix}. \tag{13.14}$$

The matrix on the left-hand side of (13.14) can be decomposed into block-matrix representation, as follows:

$$\begin{bmatrix} K+\mu I & 1 \\ 1^T & 0 \end{bmatrix} = \begin{bmatrix} c_{11} & c_1^T \\ c_1 & C_1 \end{bmatrix} = C.$$

Let $[\alpha^{(-i)}; b^{(-i)}]$ represent the parameters of the least-squares support vector machine during the ith iteration of the leave-one-out cross-validation procedure, then in the first iteration, in which the first training pattern is excluded,

$$\begin{bmatrix} \alpha^{(-1)} \\ b^{(-1)} \end{bmatrix} = C_1^{-1} \left[y_2, \ldots, y_\ell, 0\right]^T.$$

The leave-one-out prediction for the first training pattern is then given by,

$$\hat{y}_1^{(-1)} = c_1^T \begin{bmatrix} \alpha^{(-1)} \\ b^{(-1)} \end{bmatrix} = c_1^T C_1^{-1} \left[y_2, \ldots, y_\ell, 0\right]^T$$

Considering the last ℓ equations in the system of linear equations (13.14), it is clear that $[c_1\ C_1]\left[\alpha_1, \ldots, \alpha_\ell, b\right]^T = \left[y_2, \ldots, y_\ell, 0\right]^T$, and so

$$\hat{y}_1^{(-1)} = c_1^T C_1^{-1} [c_1\ C_1] \left[\alpha^T, b\right]^T = c_1^T C_1^{-1} c_1 \alpha_1 + c_1 \left[\alpha_2, \ldots, \alpha_\ell, b\right]^T.$$

Noting, from the first equation in the system of linear equations (13.14), that $y_1 = c_{11}\alpha_1 + c_1^T \left[\alpha_2, \ldots, \alpha_\ell, b\right]^T$, thus

$$\hat{y}_1^{(-1)} = y_1 - \alpha_1 \left(c_{11} - c_1^T C_1^{-1} c_1\right)$$

Finally, via the block matrix inversion lemma,

$$\begin{bmatrix} c_{11} & c_1^T \\ c_1 & C_1 \end{bmatrix}^{-1} = \begin{bmatrix} \kappa^{-1} & -\kappa^{-1} c_1 C_1^{-1} \\ C_1^{-1} + \kappa^{-1} C_1^{-1} c_1^T c_1 C_1^{-1} & -\kappa^{-1} C_1^{-1} c_1^T \end{bmatrix},$$

where $\kappa = c_{11} - c_1^T C_1^{-1} c$, and noting that the system of linear equations (13.14) is insensitive to permutations of the ordering of the equations and of the unknowns, we have that,

$$y_i - \hat{y}_i^{(-i)} = \frac{\alpha_i}{C_{ii}^{-1}}. \tag{13.15}$$

This means that, assuming the system of linear equations (13.14) is solved via explicit inversion of C, a leave-one-out cross-validation estimate of an appropriate model selection criterion can be evaluated using information already available a by-product of training the least-squares support vector machine on the entire dataset (c.f. Sundararajan and Keerthi, 2001).

13.1.2.2. EFFICIENT IMPLEMENTATION VIA CHOLESKY FACTORISATION

The leave-one-out cross-validation behaviour of the least-squares support vector machine is described by (13.15). The coefficients of the kernel expansion, α, can be found efficiently, via Cholesky factorisation, as described in Section 13.1.1.2. However we must also determine the diagonal elements of C^{-1} in an efficient manner. Using the block matrix inversion formula, we obtain

$$C^{-1} = \begin{bmatrix} M & 1 \\ 1^T & 0 \end{bmatrix}^{-1} = \begin{bmatrix} M^{-1} + M^{-1} 1 S_M^{-1} 1^T M^{-1} & -M^{-1} 1 S_M^{-1} \\ -S_M^{-1} 1^T M^{-1} & S_M^{-1} \end{bmatrix}$$

where $M = K + \mu I$ and $S_M = -1^T M^{-1} 1 = -1^T \eta$ is the Schur complement of M. The inverse of the positive definite matrix, M, can be computed efficiently from its Cholesky factorisation, via the `SYMINV` algorithm (Seaks, 1972), for example using the LAPACK routine `DTRTRI` (Anderson et al., 1999). Let $R = [r_{ij}]_{i,j=1}^{\ell}$ be the lower triangular Cholesky factor of the positive definite matrix M, such that $M = RR^T$. Furthermore, let

$$S = [s_{ij}]_{i,j=1}^{\ell} = R^{-1}, \qquad \text{where} \qquad s_{ii} = \frac{1}{r_{ii}} \qquad \text{and} \qquad s_{ij} = -s_{ii} \sum_{k=1}^{i-1} r_{ik} s_{kj},$$

represent the (lower triangular) inverse of the Cholesky factor. The inverse of M is then given by $M^{-1} = S^T S$. In the case of efficient leave-one-out cross-validation of least-squares support vector machines, we are principally concerned only with the diagonal elements of M^{-1}, given by

$$M_{ii}^{-1} = \sum_{j=1}^{i} s_{ij}^2 \qquad \Longrightarrow \qquad C_{ii}^{-1} = \sum_{j=1}^{i} s_{ij}^2 + \frac{\rho_i^2}{S_M} \qquad \forall\, i \in \{1, 2, \ldots, \ell\}.$$

The computational complexity of the basic training algorithm is $\mathcal{O}(\ell^3)$ operations, being dominated by the evaluation of the Cholesky factor. However, the computational complexity of the analytic leave-one-out cross-validation procedure, when performed as a by-product of the training algorithm, is only $\mathcal{O}(\ell)$ operations. The computational expense of the leave-one-out cross-validation procedure therefore becomes increasingly negligible as the training set becomes larger.

13.1.3. Model Selection

The virtual leave-one-out cross-validation procedure described in the previous section provides the basis for a simple automated model selection strategy for the least-squares support vector machine. Perhaps the most basic model selection criterion is provided by the Predicted REsidual

Sum of Squares (PRESS) criterion (Allen, 1974), which is simply the leave-one-out estimate of the sum-of-squares error,

$$Q(\theta) = \frac{1}{2}\sum_{i=1}^{\ell}\left[y_i - \hat{y}_i^{(-i)}\right]^2 \tag{13.16}$$

A minimum of the model selection criterion is often found via a simple grid-search procedure in the majority of practical applications of kernel learning methods. However, this is rarely necessary and often highly inefficient as a grid-search spends a large amount of time investigating hyper-parameter values outside the neighbourhood of the global optimum. A more efficient approach uses the Nelder-Mead simplex algorithm (Nelder and Mead, 1965), as implemented by the `fminsearch` function of the MATLAB optimisation toolbox. An alternative easily implemented approach uses conjugate gradient methods, with the required gradient information estimated by the method of finite differences, and implemented by the `fminunc` function from the MATLAB optimisation toolbox. In this study however, we use scaled conjugate gradient descent (Williams, 1991), with the required gradient information evaluated analytically, as this is approximately twice as efficient.

13.1.3.1. Partial Derivatives of the PRESS Model Selection Criterion

Let $\theta = \{\theta_1,\ldots,\theta_n\} = \{\lambda,\eta_1,\ldots,\eta_d\}$ represent the vector of hyper-parameters for a least-squares support vector machine, where $\eta_1,\ldots,\eta_d$ represent the kernel parameters. The PRESS statistic Allen (1974) can be written as

$$Q(\theta) = \frac{1}{2}\sum_{i=1}^{\ell}\left[r_i^{(-i)}\right]^2, \qquad \text{where} \qquad r_i^{(-i)} = y_i - \hat{y}_i^{(-i)} = \frac{\alpha_i}{C_{ii}^{-1}}. \tag{13.17}$$

Using the chain rule, the partial derivative of the PRESS statistic, with respect to an individual hyper-parameter, θ_j, is given by,

$$\frac{\partial Q(\theta)}{\partial \theta_j} = \sum_{i=1}^{\ell}\frac{\partial Q(\theta)}{\partial r_i^{(-i)}}\frac{\partial r_i^{(-i)}}{\partial \theta_j},$$

where

$$\frac{\partial Q(\theta)}{\partial r_i^{(-i)}} = r_i^{(-i)} = \frac{\alpha_i}{C_{ii}^{-1}} \qquad \text{and} \qquad \frac{\partial r_i^{(-i)}}{\partial \theta_j} = \frac{\partial \alpha_i}{\partial \theta_j}\frac{1}{C_{ii}^{-1}} - \frac{\alpha_i}{\left[C_{ii}^{-1}\right]^2}\frac{\partial C_{ii}^{-1}}{\partial \theta_j},$$

such that

$$\frac{\partial Q(\theta)}{\partial \theta_j} = \sum_{i=1}^{\ell}\frac{\alpha_i}{C_{ii}^{-1}}\left\{\frac{\partial \alpha_i}{\partial \theta_j}\frac{1}{C_{ii}^{-1}} - \frac{\alpha_i}{\left[C_{ii}^{-1}\right]^2}\frac{\partial C_{ii}^{-1}}{\partial \theta_j}\right\}. \tag{13.18}$$

We begin by deriving the partial derivatives of the model parameters, $\left[\alpha^T\ b\right]^T$, with respect to the hyper-parameter θ_j. The model parameters are given by the solution of a system of linear equations, such that

$$\left[\alpha^T\ b\right]^T = C^{-1}\left[y^T\ 0\right]^T.$$

Using the following identity for the partial derivatives of the inverse of a matrix,

$$\frac{\partial C^{-1}}{\partial \theta_j} = -C^{-1}\frac{\partial C}{\partial \theta_j}C^{-1}, \tag{13.19}$$

we obtain,

$$\frac{\partial \left[\alpha^T \; b\right]^T}{\partial \theta_j} = -C^{-1}\frac{\partial C}{\partial \theta_j}C^{-1}\left[y^T \; 0\right] = -C^{-1}\frac{\partial C}{\partial \theta_j}\left[\alpha^T \; b\right]^T.$$

Note the computational complexity of evaluating the partial derivatives of the model parameters is $\mathscr{O}(\ell^2)$, as only two successive matrix-vector products are required. The partial derivatives of the diagonal elements of C^{-1} can be found using the inverse matrix derivative identity (13.19). For a kernel parameter, $\partial C/\partial \eta_j$ will generally be fully dense, and so the computational complexity of evaluating the diagonal elements of $\partial C^{-1}/\partial \eta_j$ will be $\mathscr{O}(\ell^3)$ operations. If, on the other hand, we consider the regularisation parameter, μ, we have that

$$\frac{\partial C}{\partial \mu} = \left[\begin{array}{cc} I & 0 \\ 0^T & 0 \end{array}\right],$$

and so the computation of the partial derivatives of the model parameters, with respect to the regularisation parameter, is slightly simplified,

$$\frac{\partial \left[\alpha^T \; b\right]^T}{\partial \mu} = -C^{-1}\left[\alpha^T \; b\right]^T.$$

More importantly, as $\partial C/\partial \mu$ is diagonal, the diagonal elements of (13.19) can be evaluated with a computational complexity of only $\mathscr{O}(\ell^2)$ operations. This suggests that it may be more efficient to adopt different strategies for optimising the regularisation parameter, μ, and the vector of kernel parameters, η, (c.f. Saadi et al., 2004). For a kernel parameter, η_j, the partial derivatives of C with respect to η_j are given by the partial derivatives of the kernel matrix, i.e.

$$\frac{\partial C}{\partial \eta_j} = \left[\begin{array}{cc} \partial K/\partial \eta_j & 0 \\ 0^T & 0 \end{array}\right].$$

For the spherical radial basis function kernel, used in this study, the partial derivative with respect to the kernel parameter is given by

$$\frac{\partial \mathscr{K}(x,x')}{\partial \eta} = -\mathscr{K}(x,x')\|x - x'\|^2. \tag{13.20}$$

Finally, since the regularisation parameter, μ, and the scale parameter of the radial basis function kernel are strictly positive quantities, in order to permit the use of an unconstrained optimisation procedure, we adopt the parameterisation $\widetilde{\theta}_j = \log_2 \theta_j$, such that

$$\frac{\partial Q(\theta)}{\partial \widetilde{\theta}_j} = \frac{\partial Q(\theta)}{\partial \theta_j}\frac{\partial \theta_j}{\partial \widetilde{\theta}_j} \quad \text{where} \quad \frac{\partial \theta_j}{\partial \widetilde{\theta}_j} = \theta_j \log 2. \tag{13.21}$$

13.1.3.2. AUTOMATIC RELEVANCE DETERMINATION

Automatic Relevance Determination (ARD) (e.g. Rasmussen and Williams, 2006), also known as feature scaling (Chapelle et al., 2002; Bo et al., 2006), aims to identify informative input features as a natural consequence of optimising the model selection criterion. This can be most easily achieved using an *elliptical* radial basis function kernel,

$$\mathscr{K}(x,x') = \exp\left\{-\sum_{i=1}^{d}\eta_i[x_i - x_i']^2\right\}, \tag{13.22}$$

that incorporates individual scaling factors for each input dimension. The partial derivatives with respect to the kernel parameters are then given by,

$$\frac{\partial \mathscr{K}(x,x')}{\partial \eta_i} = -\mathscr{K}(x,x')[x_i - x_i']^2. \tag{13.23}$$

Generalisation performance is likely to be enhanced if irrelevant features are down-weighted. It is therefore hoped that minimising the model selection criterion will lead to very small values for the scaling factors associated with redundant input features, allowing them to be identified and pruned from the model.

13.2. Bayesian Regularisation in Model Selection

In order to overcome the observed over-fitting in model selection using leave-one-out cross-validation based methods, we propose to add a regularisation term (Tikhonov and Arsenin, 1977) to the model selection criterion, which penalises solutions where the kernel parameters take on unduly large values. The regularised model selection criterion is then given by

$$M(\theta) = \zeta Q(\theta) + \xi \Omega(\theta), \tag{13.24}$$

where ξ and ζ are additional regularisation parameters, $Q(\theta)$ is the model selection criterion, in this case the PRESS statistic and $\Omega(\theta)$ is a regularisation term,

$$Q(\theta) = \frac{1}{2}\sum_{i=1}^{\ell}\left[y_i - \hat{y}_i^{(-i)}\right]^2 \qquad \text{and} \qquad \Omega(\theta) = \frac{1}{2}\sum_{i=1}^{d}\eta_i^2.$$

In this study we have left the regularisation parameter, μ, unregularised. However, we have now introduced two further regularisation parameters ξ and ζ for which good values must also be found. This problem may be solved by taking a Bayesian approach and adopting an ignorance prior and integrating out the additional regularisation parameters analytically in the style of Buntine and Weigend (1991). Adapting the approach taken by Williams (1995), the regularised model selection criterion (13.24) can be interpreted as the posterior density in the space of the hyper-parameters,

$$P(\theta|\mathscr{D}) \propto P(\mathscr{D}|\theta)P(\theta),$$

by taking the negative logarithm and neglecting additive constants. Here $P(\mathscr{D}|\theta)$ represents the *likelihood* with respect to the hyper-parameters and $P(\theta)$ represents our prior beliefs regarding the hyper-parameters, in this case that they should have a small magnitude, corresponding to a relatively simple model. These quantities can be expressed as

$$P(\mathscr{D}|\theta) = Z_Q^{-1}\exp\{-\zeta Q(\theta)\} \qquad \text{and} \qquad P(\theta) = Z_\Omega^{-1}\exp\{-\xi\Omega(\theta)\}$$

where Z_Q and Z_Ω are the appropriate normalising constants. Assuming the data represent an i.i.d. sample, the *likelihood* in this case is Gaussian,

$$P(\mathscr{D}|\theta) = \prod_{i=1}^{\ell}\frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{\left[y_i - \hat{y}_i^{(-i)}\right]^2}{2\sigma^2}\right\} \quad \text{where} \quad \zeta = \frac{1}{\sigma^2} \implies Z_Q = \left(\frac{2\pi}{\zeta}\right)^{\ell/2}$$

Likewise, the *prior* is a Gaussian, centred on the origin,

$$P(\theta) = \prod_{i=1}^{d}\frac{1}{\sqrt{2\pi/\xi}}\exp\left\{-\frac{\xi}{2}\eta_i^2\right\} \qquad \text{such that} \qquad Z_\Omega = \left(\frac{2\pi}{\xi}\right)^{d/2}$$

Minimising (13.24) is thus equivalent to maximising the posterior density with respect to the hyper-parameters. Note that the use of a prior over the hyper-parameters is in accordance with normal Bayesian practice and has been investigated in the case of Gaussian Process classifiers by Williams and Barber (1998). The combination of frequentist and Bayesian approaches at the first and second levels of inference is however somewhat unusual. The marginal likelihood is dependent on the assumptions of the model, which may not be completely appropriate. Cross-validation based procedures may therefore be more robust in the case of model mis-specification (Wahba, 1990). It seems reasonable for the model to be less sensitive to assumptions at the second level of inference than the first, and so the proposed approach represents a pragmatic combination of techniques.

13.2.1. Elimination of Second Level Regularisation Parameters ξ and ζ

Under the *evidence framework* proposed by MacKay (1992a,b,c) the hyper-parameters ξ and ζ are determined by maximising the marginal likelihood, also known as the Bayesian *evidence* for the model. In this study, however we opt to integrate out the hyper-parameters analytically, extending the work by Buntine and Weigend (1991) and Williams (1995) to consider Bayesian regularisation at the second level of inference, namely the selection of good values for the hyper-parameters. We begin with the prior over the hyper-parameters, which depends on ξ,

$$P(\theta|\xi) = Z_\Omega(\xi)^{-1}\exp\{-\xi\Omega\}.$$

The regularisation parameter ξ may then be integrated out analytically using a suitable prior, $P(\xi)$,

$$P(\theta) = \int P(\theta|\xi)P(\xi)d\xi.$$

The improper Jeffreys' prior, $P(\xi) \propto 1/\xi$ is an appropriate ignorance prior in this case as ξ is a scale parameter, noting that ξ is strictly positive,

$$p(\theta) = \frac{1}{(2\pi)^{d/2}}\int_0^\infty \xi^{d/2-1}\exp\{-\xi\Omega\}\,d\xi$$

Using the Gamma integral $\int_0^\infty x^{\nu-1}e^{-\mu x}dx = \Gamma(\nu)/\mu^\nu$ (Gradshteyn and Ryzhic, 1994, equation 3.384), we obtain

$$p(\theta) = \frac{1}{(2\pi)^{d/2}}\frac{\Gamma(d/2)}{\Omega^{d/2}} \implies -\log p(\theta) \propto \frac{d}{2}\log\Omega.$$

Finally, adopting a similar procedure to eliminate ζ, we obtain a revised model selection criterion with Bayesian regularisation,

$$L = \frac{\ell}{2}\log Q(\theta) + \frac{d}{2}\log\Omega(\theta). \tag{13.25}$$

in which the regularisation parameters have been eliminated. As before, this criterion can be optimised via standard methods, such as the Nelder-Mead simplex algorithm (Nelder and Mead, 1965) or scaled conjugate gradient descent (Williams, 1991). The partial derivatives of the proposed Bayesian model selection criterion are given by

$$\frac{\partial L}{\partial\theta_i} = \frac{\ell}{2Q(\theta)}\frac{\partial Q(\theta)}{\partial\theta_i} + \frac{d}{2\Omega(\theta)}\frac{\partial\Omega(\theta)}{\partial\theta_i} \qquad \text{and} \qquad \frac{\partial\Omega(\theta)}{\partial\eta_i} = \eta_i.$$

The additional computational expense involved in Bayesian regularisation of the model selection criterion is only $\mathscr{O}(d)$ operations, and is extremely small in comparison with the $\mathscr{O}(\ell^3)$ operations involved in obtaining the leave-one-out error (including the cost of training the model on the entire dataset). Per iteration of the model selection process, the cost of the Bayesian regularisation is therefore minimal. There seems little reason to suppose that the regularisation will have an adverse effect on convergence, and this seems to be the case in practice.

13.2.2. Relationship with the Evidence Framework

Under the evidence framework of MacKay (1992a,b,c) the regularisation parameters, ξ and ζ, are selected so as to maximise the marginal likelihood, also known as the Bayesian evidence, for the model. The log-evidence is given by

$$\log P(\mathscr{D}) = -\xi\Omega(\theta) - \zeta Q(\theta) - \frac{1}{2}\log|A| + \frac{d}{2}\log\xi + \frac{\ell}{2}\log\zeta - \frac{\ell}{2}\log\{2\pi\},$$

where A is the Hessian of the regularised model selection criterion (13.24) with respect to the hyper-parameters, θ. Setting the partial derivatives of the log evidence with respect to the regularisation parameters, ξ and ζ, equal to zero, we obtain the familiar update formulae,

$$\xi^{\text{new}} = \frac{\gamma}{2\Omega(\theta)} \qquad \text{and} \qquad \zeta^{\text{new}} = \frac{\ell-\gamma}{2Q(\theta)},$$

where γ is the number of well defined hyper-parameters, i.e. the hyper-parameters for which the optimal value is primarily determined by the log-likelihood term, $Q(\theta)$ rather than by the regulariser, $\Omega(\theta)$. In the case of the L2 regularisation term, corresponding to a Gaussian prior, the number of well determined hyper-parameters is given by

$$\gamma = \sum_{j=1}^{n} \frac{\lambda_j}{\lambda_j + \xi}$$

where, $\lambda_i, \ldots, \lambda_n$ represent the eigenvalues of the Hessian of the unregularised model selection criterion, $Q(\theta)$ with respect to the kernel parameters. Comparing the partial derivatives of the regularised model selection criterion (13.24) with those of the Bayesian criterion (13.25), reveals that the Bayesian regularisation scheme is equivalent to optimising the regularised model selection criterion (13.24) assuming that the regularisation parameters, ξ and ζ, are continuously updated according to the following update rules,

$$\xi^{\text{eff}} = \frac{d}{2\Omega(\theta)} \qquad \text{and} \qquad \zeta^{\text{eff}} = \frac{\ell}{2Q(\theta)}.$$

This exactly corresponds to the "cheap and cheerful" approximation of the evidence framework suggested by MacKay (1994), which assumes that all of the hyper-parameters are well-determined and that the number of hyper-parameters is small in relation to the number of training patterns. Since $\gamma \leq d$, it seems self evident that the proposed Bayesian regularisation scheme will be prone to a degree of under-fitting, especially in the case of a feature scaling kernel with many redundant features. The theoretical and practical pros and cons of the integrate-out approach and the evidence framework are discussed in some detail by MacKay (1994) and Bishop (1995) and references therein. However, the integrate-out approach does not require the evaluation of the Hessian matrix of the original selection criterion, $Q(\theta)$, which is likely to prove computationally prohibitive.

13.3. Results

In this section, we present experimental results demonstrating the benefits of the proposed model selection strategy incorporating Bayesian regularisation to overcome the inherent high variance of leave-one-out cross-validation based selection criteria. Table 13.2 shows a comparison of the error rates of least-squares support vector machines, using model selection procedures with, and without, Bayesian regularisation, (LS-SVM and LS-SVM-BR respectively) over the suite of thirteen public domain benchmark datasets used in the study by Mika et al. (2000). Results obtained using a Gaussian process classifier (Rasmussen and Williams, 2006), based on the expectation propagation method, are also provided for comparison (EP-GPC). The same set of 100 random partitions of the data (20 in the case of the larger `image` and `splice` benchmarks) to form training and test sets used in that study are also used here. In each case, model selection is performed independently for each realisation of the dataset, such that the standard errors reflect the variability of both the training algorithm and the model selection procedure with changes in the sampling of the data. Both conventional spherical and elliptical radial basis kernels are used for all kernel learning methods, so that the effectiveness of each algorithm for automatic relevance determination can be assessed. The use of multiple training/test partitions allows an estimate of the statistical significance of differences in performance between algorithms to be computed. Let $\hat{x}$ and $\hat{y}$ represent the means of the performance statistic for a pair of competing algorithms, and e_x and e_y the corresponding standard errors, then the z statistic is computed as

$$z = \frac{\hat{y} - \hat{x}}{\sqrt{e_x^2 + e_y^2}}.$$

The z-score can then be converted to a significance level via the normal cumulative distribution function, such that $z = 1.64$ corresponds to a 95% significance level. All statements of statistical significance in the remainder of this section refer to a 95% level of significance.

Table 13.1: Details of datasets used in empirical comparison.

Dataset	Training Patterns	Testing Patterns	Number of Replications	Input Features
Banana	400	4900	100	2
Breast cancer	200	77	100	9
Diabetis	468	300	100	8
Flare solar	666	400	100	9
German	700	300	100	20
Heart	170	100	100	13
Image	1300	1010	20	18
Ringnorm	400	7000	100	20
Splice	1000	2175	20	60
Thyroid	140	75	100	5
Titanic	150	2051	100	3
Twonorm	400	7000	100	20
Waveform	400	4600	100	21

Table 13.2: Error rates of least-squares support vector machine, with and without Bayesian regularisation of the model selection criterion, in this case the PRESS statistic (Allen, 1974), and Gaussian process classifiers over thirteen benchmark datasets (Rätsch et al., 2001), using both standard radial basis function and automatic relevance determination kernels. The results for the EP-GPC were obtained using the MATLAB software accompanying the book by Rasmussen and Williams (2006). The results for each method are presented in the form of the mean error rate over test data for 100 realisations of each dataset (20 in the case of the image and splice datasets), along with the associated standard error. The best results are shown in boldface and the second best in italics (without implication of statistical significance).

Dataset	**Radial Basis Function**			**Automatic Relevance Determination**		
	LSSVM	**LSSVM-BR**	**EP-GPC**	**LSSVM**	**LSSVM-BR**	**EP-GPC**
Banana	10.60 ± 0.052	10.59 ± 0.050	$\mathbf{10.41 \pm 0.046}$	10.79 ± 0.072	10.73 ± 0.070	$\mathit{10.46 \pm 0.049}$
Breast cancer	$\mathit{26.73 \pm 0.466}$	27.08 ± 0.494	$\mathbf{26.52 \pm 0.489}$	29.08 ± 0.415	27.81 ± 0.432	27.97 ± 0.493
Diabetes	23.34 ± 0.166	$\mathbf{23.14 \pm 0.166}$	$\mathit{23.28 \pm 0.182}$	24.35 ± 0.194	23.42 ± 0.177	23.86 ± 0.193
Flare solar	34.22 ± 0.169	34.07 ± 0.171	34.20 ± 0.175	34.39 ± 0.194	$\mathit{33.61 \pm 0.151}$	$\mathbf{33.58 \pm 0.182}$
German	$\mathit{23.55 \pm 0.216}$	23.59 ± 0.216	$\mathbf{23.36 \pm 0.211}$	26.10 ± 0.261	23.88 ± 0.217	23.77 ± 0.221
Heart	$\mathit{16.64 \pm 0.358}$	$\mathbf{16.19 \pm 0.348}$	16.65 ± 0.287	23.65 ± 0.355	17.68 ± 0.623	19.68 ± 0.366
Image	3.00 ± 0.158	2.90 ± 0.154	2.80 ± 0.123	$\mathbf{1.96 \pm 0.115}$	$\mathit{2.00 \pm 0.113}$	2.16 ± 0.068
Ringnorm	$\mathbf{1.61 \pm 0.015}$	$\mathbf{1.61 \pm 0.015}$	$\mathit{4.41 \pm 0.064}$	2.11 ± 0.040	1.98 ± 0.026	8.58 ± 0.096
Splice	10.97 ± 0.158	10.91 ± 0.154	11.61 ± 0.181	$\mathit{5.86 \pm 0.179}$	$\mathbf{5.14 \pm 0.145}$	7.07 ± 0.765
Thyroid	4.68 ± 0.232	4.63 ± 0.218	$\mathit{4.36 \pm 0.217}$	4.68 ± 0.199	4.71 ± 0.214	$\mathbf{4.24 \pm 0.218}$
Titanic	$\mathbf{22.47 \pm 0.085}$	22.59 ± 0.120	22.64 ± 0.134	$\mathit{22.58 \pm 0.108}$	22.86 ± 0.199	22.73 ± 0.134
Twonorm	$\mathbf{2.84 \pm 0.021}$	$\mathbf{2.84 \pm 0.021}$	$\mathit{3.06 \pm 0.034}$	5.18 ± 0.072	4.53 ± 0.077	4.02 ± 0.068
Waveform	$\mathit{9.79 \pm 0.045}$	$\mathbf{9.78 \pm 0.044}$	10.10 ± 0.047	13.56 ± 0.141	11.48 ± 0.177	11.34 ± 0.195

13.3.1. Performance of Models Based on the Spherical RBF Kernel

The results shown in the first three data columns of Table 13.2 show the performance of LS-SVM, LS-SVM-BR and EP-ARD models based on the spherical Gaussian kernel. The performance of LS-SVM models with and without Bayesian regularisation are very similar, with neither model proving significantly better than the other on any of the datasets. This seems reasonable given that only two hyper-parameters are optimised during model selection and so there is little scope for over-fitting the PRESS model selection criterion and the regularisation term will have little effect. The LS-SVM model with Bayesian regularisation is significantly out-performed by the Gaussian Process classifier on one benchmark `banana`, but performs significantly better on a further four (`ringnorm`, `splice`, `twonorm`, `waveform`). Demšar (2006) recommends the use of the Wilcoxon signed rank test for assessing the statistical significance of differences in performance over multiple datasets. According to this test, neither the LSSVM-BR nor the EP-PGC is statistically superior at the 95% level of significance.

13.3.2. Performance of Models Based on the Elliptical RBF Kernel

The performances of LS-SVM, LS-SVM-BR and EP-GPC models based on the elliptical Gaussian kernel, which includes a separate scale parameter for each input feature, are shown in the last three columns of Table 13.2. Before evaluating the effects of Bayesian regularisation in model selection it is worth noting that the use of elliptical RBF kernels does not generally improve performance. For the LS-SVM, the elliptical kernel produces significantly better results on only two benchmarks (`image` and `splice`) and significantly worse results on a further eight (`banana`, `breast cancer`, `diabetis`, `german`, `heart`, `ringnorm`, `twonorm`, `waveform`), with the degradation in performance being very large in some cases (e.g. `heart`). This seems likely to be a result of the additional degrees of freedom involved in the model selection process, allowing over-fitting of the PRESS model selection criterion as a result of its inherently high variance. Note that fully Bayesian approaches, such as the Gaussian Process Classifier, are also unable to reliably select kernel parameters for the elliptical RBF kernel. The elliptical kernel is significantly better on only three benchmarks (`flare solar`, `image` and `splice`), while being significantly worse on six (`breast cancer`, `diabetis`, `heart`, `ringnorm`, `twonorm` and `waveform`).

In the case of the elliptical RBF kernel, the use of Bayesian regularisation in model selection has a dramatic effect on the performance of LS-SVM models, with the LS-SVM-BR model proving significantly better than the conventional LS-SVM on nine of the thirteen benchmarks (`breast cancer`, `diabetis`, `flare solar`, `german`, `heart`, `ringnorm`, `splice`, `twonorm` and `waveform`) without being significantly worse on any of the remaining four datasets. This demonstrates that over-fitting in model selection, due to the larger number of kernel parameters, is likely to be the significant factor causing the relatively poor performance of models with the elliptical RBF kernel. Again, the Gaussian Process classifier is significantly better than the LS-SVM with Bayesian regularisation on the `banana` and `twonorm` datasets, but is significantly worse on four of the remaining eleven (`diabetis`, `heart`, `ringnorm` and `splice`). Again, according to the Wilcoxon signed rank test, neither the LSSVM-BR nor the EP-PGC is statistically superior at the 95% level of significance. However the magnitude of the difference in performance between LSSVM-BR and EP-GPC approaches tends to be greatest when the LSSVM-BR out-performs EP-GPC, most notably on the `heart`, `splice` and `ringnorm` data sets. This provides some support for the observation of Wahba (1990) that cross-validation based model selection procedures should be more robust against model mis-specification (see also Rasmussen and Williams, 2006).

13.4. Discussion

The experimental evaluation presented in the previous section demonstrates that over-fitting can occur in model selection, due to the variance of the model selection criterion. In many cases the minimum of the selection criterion using the elliptical RBF kernel is lower than that achievable using the spherical RBF kernel, however this results in a degradation in generalisation performance. Using the PRESS statistic, the over-fitting is likely to be most severe in cases with a small number of training patterns, as the variance of the leave-one-out estimator decreases as the sample size becomes larger. Using the standard LSSVM, the elliptical RBF kernel only out-performs the spherical RBF kernel on two of the thirteen datasets, `image` and `splice`, which also happen to be the two largest datasets in terms of the number of training patterns. The greatest degradation in performance is obtained on the `heart` benchmark, the third smallest. The `heart` dataset also has a relatively large number of input features (13). A large number of input features introduces a many additional degrees of freedom to optimise the model selection criterion, and so will generally tend to encourage over-fitting. However, there may be a compact subset of highly relevant features with the remainder being almost entirely uninformative. In this case the advantage of suppressing the noisy inputs is so great that it overcomes the predisposition towards over-fitting, and so results in improved generalisation (as observed in the case of the `image` and `splice` benchmarks). Whether the use of an elliptical RBF kernel will improve or degrade generalisation largely depends on such characteristics of the data that are not known *a-priori*, and so it seems prudent to consider a range of kernel functions and select the best via cross-validation.

The experimental results indicate that Bayesian regularisation of the hyper-parameters is generally beneficial, without at this stage providing a complete solution to the problem of over-fitting the model selection criterion. The effectiveness of the Bayesian regularisation scheme is to a large extent dependent on the appropriateness of the prior imposed on the hyper-parameters. There is no reason to assume that the simple Gaussian prior used here is in any sense optimal, and this is an issue where further research is necessary (see Section 13.4.2). The comparison of the integrate-out approach and the evidence framework highlights a deficiency of the simple Gaussian prior. It suggests that the integrate-out approach is likely to result in mild over-regularisation of the hyper-parameters in the presence of a large number of irrelevant features, as the corresponding hyper-parameters will be ill-determined.

The LSSVM with Bayesian regularisation of the hyper-parameters does not significantly out-perform the expectation propagation based Gaussian process classifier over the suite of thirteen benchmark datasets considered. This is not wholly surprising as the EP-GPC is at least very close to the state-of-the-art, indeed it is interesting that the EP-GPC does not out-perform such a comparatively simple model. The EP-GPC uses the marginal likelihood as the model selection criterion, which gives the probability of the data, *given the assumptions of the model* (Rasmussen and Williams, 2006). Cross-validation based approaches, on the other hand, provide an estimate of generalisation performance that does not depend on the model assumptions, and so may be more robust against model mis-specification Wahba (1990). The no free lunch theorems suggest that, at least in terms of generalisation performance, there is a lack of inherent superiority of one classification method over another, in the absence of *a-priori* assumptions regarding the data. This implies that if one classifier performs better than another on a particular dataset it is because the inductive biases of that classifier provide a better fit to the particular pattern recognition task, rather than to its superiority in a more general sense. A model with strong inductive biases is likely to benefit when these biases are well suited to the data, but will perform badly when they do not. While a model with weak inductive biases will be more robust, it is less likely to perform conspicuously well on any given dataset. This means there are complementary advantages and disadvantages to both approaches.

13.4.1. Relationship to Existing Work

The use of a prior over the hyper-parameters is in accordance with normal Bayesian practice and has been used in Gaussian Process classification (Williams and Barber, 1998). The problem of over-fitting in model selection has also been addressed by Qi et al. (2004), in the case of selecting informative features for a logistic regression model using an Automatic Relevance Determination (ARD) prior (c.f. Tipping, 2000). In this case, the Expectation Propagation method (Minka, 2001) is used to obtain a deterministic approximation of the posterior, and also (as a by-product) a leave-one-out performance estimate. The latter is then used to implement a form of early-stopping (e.g. Sarle, 1995) to prevent over-fitting resulting from the direct optimization of the marginal likelihood until convergence. It seems likely that this approach would be also be beneficial in the case of tuning the hyper-parameters of the covariance function of Gaussian process model, using either the leave-one-out estimate arising from the EP approximation, or an approximate leave-one-out estimate from the Laplace approximation (c.f. Cawley and Talbot, 2007).

13.4.2. Directions for Further Research

In this paper, the regularisation term corresponds to a simple spherical Gaussian prior over the kernel parameters. One direction of research would be to investigate alternative regularisation terms. The first possibility would be to use a regularisation term corresponding to a separable Laplace prior,

$$\Omega(\theta) = \frac{1}{2}\sum_{i=1}^{d}|\eta_i| \qquad \Longrightarrow \qquad p(\theta) = \prod_{i=1}^{d}\frac{\xi}{2}\exp\{-\xi|\eta_i|\}$$

Setting the derivative of the regularised model selection criterion (13.24) to zero, we obtain

$$\left|\frac{\partial Q}{\partial \eta_i}\right| = \frac{\xi}{\zeta} \quad \text{if } |\eta_i| > 0 \qquad \text{and} \qquad \left|\frac{\partial Q}{\partial \eta_i}\right| < \frac{\xi}{\zeta} \quad \text{if } |\eta_i| = 0,$$

which implies that if the sensitivity of the leave-one-out error, $Q(\theta)$, falls below ξ/ζ, the value of the hyper-parameter, η_i will be set exactly to zero, effectively pruning that input from the model. In this way explicit feature selection may be obtained as a consequence of (regularised) model selection. The model selection criterion with Bayesian regularisation then becomes

$$L = \frac{\ell}{2}\log Q(\theta) + N\log\Omega(\theta)$$

where N is the number of input features with non-zero scale factors. This potentially overcomes the propensity towards under-fitting the data that might be expected when using the Gaussian prior, as the pruning action of the Laplace prior means that the values of all remaining hyper-parameters are well-determined by the data. In the case of the Laplace prior, the integrate-out approach is exactly equivalent to continuous updates of the hyper-parameters according to the update formulae under the evidence framework (Williams, 1995). Alternatively, defining a prior over the *function* of a model seems more in accordance with Bayesian ideals than choosing a prior over the parameters of the model. The use of a prior over the hyper-parameters based on the smoothness of the resulting model also provides a potential direction for future research. In this case, the regularisation term might take the form,

$$\Omega(\theta) = \frac{1}{2\ell} \sum_{i=1}^{\ell} \sum_{j=1}^{d} \left[\frac{\partial^2 \hat{y}_i}{\partial x_{ij}^2} \right]^2,$$

directly penalising models with excess curvature. This regularisation term corresponds to curvature driven smoothing in multi-layer perceptron networks (Bishop, 1993), except that the model output $\hat{y}_i$ is viewed as a function of the hyper-parameters, rather than of the weights. A penalty term based on the first-order partial derivatives is also feasible (c.f. Drucker and Le Cun, 1992).

13.5. Conclusion

Leave-one-out cross-validation has proved to be an effective means of model selection for a variety of kernel learning methods, provided the number of hyper-parameters to be tuned is relatively small. The use of kernel functions with large numbers of parameters often provides sufficient degrees of freedom to over-fit the model selection criterion, leading to poor generalisation. In this paper, we have proposed the use of regularisation at the second level of inference, i.e. model selection. The use of Bayesian regularisation is shown to be effective in reducing over-fitting, by ensuring the values of the kernel parameters remain small, giving a smoother kernel and hence a less complex classifier. This is achieved with only a minimal computational expense as the additional regularisation parameters are integrated out analytically using a reference prior. While a fully Bayesian model selection strategy is conceptually more elegant, it may also be less robust to model mis-specification. The use of leave-one-out cross-validation in model selection and Bayesian methods at the next level seems to be a pragmatic compromise. The effectiveness of this approach is clearly demonstrated in the experimental evaluation where, on average, the LS-SVM with Bayesian regularisation out-performs the expectation-propagation based Gaussian process classifier, using both spherical and elliptical RBF kernels.

Acknowledgments

The authors would like to thank the organisers of the WCCI model selection workshop and performance prediction challenge and the NIPS multi-level inference workshop and model selection game, and fellow participants for the stimulating discussions that have helped to shape this work. We also thank Carl Rasmussen and Chris Williams for their advice regarding the EP-GPC and the anonymous reviewers for their detailed and constructive comments that have significantly improved this paper.

References

D. M. Allen. The relationship between variable selection and prediction. *Technometrics*, 16: 125–127, 1974.

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorenson. *LAPACK Users' Guide*. SIAM Press, third edition, 1999.

C. M. Bishop. Curvature-driven smoothing: a learning algorithm for feedforward networks. *IEEE Transactions on Neural Networks*, 4(5):882–884, September 1993.

C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

L. Bo, L. Wang, and L. Jiao. Feature scaling for kernel Fisher discriminant analysis using leave-one-out cross-validation. *Neural Computation*, 18:961–978, April 2006.

W. L. Buntine and A. S. Weigend. Bayesian back-propagation. *Complex Systems*, 5:603–643, 1991.

G. C. Cawley. Leave-one-out cross-validation based model selection criteria for weighted LS-SVMs. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN-2006)*, pages 2970–2977, Vancouver, BC, Canada, July 16–21 2006.

G. C. Cawley and N. L. C. Talbot. Efficient leave-one-out cross-validation of kernel Fisher discriminant classifiers. *Pattern Recognition*, 36(11):2585–2592, November 2003.

G. C. Cawley and N. L. C. Talbot. Fast leave-one-out cross-validation of sparse least-squares support vector machines. *Neural Networks*, 17(10):1467–1475, December 2004.

G. C. Cawley and N. L. C. Talbot. Approximate leave-one-out cross-validation for kernel logistic regression. *Machine Learning* (submitted), 2007.

C. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1):131–159, 2002.

C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

H. Drucker and Y. Le Cun. Improving generalization performance using double back-propagation. *IEEE Transactions on Neural Networks*, 3(6):991–997, 1992.

S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilema. *Neural Computation*, 4(1):1–58, 1992.

G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, third edition edition, 1996.

I. S. Gradshteyn and I. M. Ryzhic. *Table of Integrals, Series and Products*. Academic Press, fifth edition, 1994.

I. Guyon, A. R. Saffari Azar Alamdari, G. Dror, and J. Buhmann. Performance prediction challenge. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN-2006)*, pages 1649–1656, Vancouver, BC, Canada, July 16–21 2006.

T. Joachims. *Learning to Classify Text using Support Vector Machines - Methods, Theory and Algorithms*. Kluwer Academic Publishers, 2002.

S. S. Keerthi, K. B. Duan, S. K. Shevade, and A. N. Poo. A fast dual algorithm for kernel logistic regression. *Machine Learning*, 61(1–3):151–165, November 2005.

G. S. Kimeldorf and G. Wahba. Some results on Tchebycheffian spline functions. *J. Math. Anal. Applic.*, 33:82–95, 1971.

R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence (IJCAI)*, pages 1137–1143, San Mateo, CA, 1995. Morgan Kaufmann.

P. A. Lachenbruch and M. R. Mickey. Estimation of error rates in discriminant analysis. *Technometrics*, 10(1):1–11, February 1968.

A. Luntz and V. Brailovsky. On estimation of characters obtained in statistical procedure of recognition (in Russian). *Techicheskaya Kibernetica*, 3, 1969.

D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1992a.

D. J. C. MacKay. A practical Bayesian framework for backprop networks. *Neural Computation*, 4(3):448–472, 1992b.

D. J. C. MacKay. The evidence framework applied to classification networks. *Neural Computation*, 4(5):720–736, 1992c.

D. J. C. MacKay. Hyperparameters : optimise or integrate out? In G. Heidbreder, editor, *Maximum Entropy and Bayesian Methods*. Kluwer, 1994.

J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London, A*, 209:415–446, 1909.

C. A. Micchelli. Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2:11–22, 1986.

S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K.-R. Müller. Fisher discriminant analysis with kernels. In *Neural Networks for Signal Processing*, volume IX, pages 41–48. IEEE Press, New York, 1999.

S. Mika, G. Rätsch, J. Weston, B. Schölkopf, A. J. Smola, and K.-R. Müller. Invariant feature extraction and classification in feature spaces. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12, pages 526–532. MIT Press, 2000.

T. P. Minka. Expectation propagation for approximate Bayesian inference. In *Proceedings of the 17th Annual Conference on Uncertainty in Artificial Intelligence*, pages 362–369. Morgan Kauffmann, 2001.

J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7: 308–313, 1965.

Y. Qi, T. P. Minka, R. W. Picard, and Z. Ghahramani. Predictive automatic relevance determination by expectation propagation. In *Proceedings of the 21st International Conference on Machine Learning*, pages 671–678, Banff, Alberta, Canada, July 4–8 2004.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, 2006.

G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for AdaBoost. *Machine Learning*, 42(3): 287–320, March 2001.

K. Saadi, N. L. C. Talbot, and G. C. Cawley. Optimally regularised kernel Fisher discriminant analysis. In *Proceedings of the 17th International Conference on Pattern Recognition (ICPR-2004)*, volume 2, pages 427–430, Cambridge, United Kingdom, August 23–26 2004.

W. S. Sarle. Stopped training and other remidies for overfitting. In *Proceedings of the 27*th *Symposium on the Interface of Computer Science and Statistics*, pages 352–360, Pittsburgh, PA, USA, June 21–24 1995.

B. Schölkopf, K. Tsuda, and J.-P. Vert. *Kernel Methods in Computational Biology*. MIT Press, 2004.

T. Seaks. SYMINV : An algorithm for the inversion of a positive definite matrix by the Cholesky decomposition. *Econometrica*, 40(5):961–962, September 1972.

J. Shawe-Taylor and N. Cristianini. *Kernel methods for Pattern Analysis*. Cambridge University Press, 2004.

M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, B 36(1):111–147, 1974.

S. Sundararajan and S. S. Keerthi. Predictive approaches for choosing hyperparameters in Gaussian processes. *Neural Computation*, 13(5):1103–1118, May 2001.

J. A. K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, June 1999.

J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines*. World Scientific, 2002.

A. N. Tikhonov and V. Y. Arsenin. *Solutions of ill-posed problems*. John Wiley, New York, 1977.

M. E. Tipping. Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, June 2000.

G. Wahba. *Spline models for observational data*. SIAM Press, Philadelphia, PA, 1990.

C. K. I. Williams and D. Barber. Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351, December 1998.

P. M. Williams. A Marquardt algorithm for choosing the step size in backpropagation learning with conjugate gradients. Technical Report CSRP-229, University of Sussex, February 1991.

P. M. Williams. Bayesian regularization and pruning using a Laplace prior. *Neural Computation*, 7(1):117–143, 1995.

Journal of Machine Learning Research 10(Feb):405–440, 2009 Submitted 01/07; Published 02/09

Chapter 14

Particle Swarm Model Selection

Hugo Jair Escalante HUGO.JAIR@GMAIL.COM
Manuel Montes MMONTESG@INAOEP.MX
Luis Enrique Sucar ESUCAR@INAOEP.MX
Department of Computational Sciences
National Institute of Astrophysics, Optics and Electronics
Puebla, México, 72840

Editor: Isabelle Guyon and Amir Saffari

Abstract

This paper proposes the application of particle swarm optimization (*PSO*) to the problem of *full model selection, FMS,* for classification tasks. *FMS* is defined as follows: given a pool of preprocessing methods, feature selection and learning algorithms, to select the combination of these that obtains the lowest classification error for a given data set; the task also includes the selection of hyperparameters for the considered methods. This problem generates a vast search space to be explored, well suited for stochastic optimization techniques. *FMS* can be applied to any classification domain as it does not require domain knowledge. Different model types and a variety of algorithms can be considered under this formulation. Furthermore, competitive yet simple models can be obtained with *FMS*. We adopt *PSO* for the search because of its proven performance in different problems and because of its simplicity, since neither expensive computations nor complicated operations are needed. Interestingly, the way the search is guided allows *PSO* to avoid overfitting to some extend. Experimental results on benchmark data sets give evidence that the proposed approach is very effective, despite its simplicity. Furthermore, results obtained in the framework of a model selection challenge show the competitiveness of the models selected with *PSO*, compared to models selected with other techniques that focus on a single algorithm and that use domain knowledge.

Keywords: Full model selection, machine learning challenge, particle swarm optimization, experimentation, cross validation

14.1. Introduction

Model selection is the task of picking the model that best describes a data set (Hastie et al., 2001). Since the phrase *describing a data set* can be interpreted in several different ways, the model selection task can denote diverse related problems, including: variable and feature selection (Bengio and Chapados, 2003; Guyon et al., 2006a; Guyon and Elisseeff, 2003), system identification (Voss and Feng, 2002; Nelles, 2001), parameter-hyperparameter optimization (Guyon et al., 2006b; Kim et al., 2002; Hastie et al., 2001; Cawley and Talbot, 2007b; Escalante et al., 2007), and discretization (Boullé, 2007; Hue and Boullé, 2007). In this paper we give a broader interpretation to this task and call it *full model selection* (*FMS*). The *FMS* problem consists on the following: given a pool of preprocessing methods, feature selection and learning algorithms, select the combination of these that obtains the lowest classification error for a given data set. This task also includes the selection of hyperparameters for the considered methods, resulting in a vast search space that is well suited for stochastic optimization techniques.

Adopting a broader interpretation to the model selection problem allows us to consider different model types and a variety of methods, in contrast to techniques that consider a single model type (i.e. either learning algorithm or feature selection method, but not both) and a single method (e.g. neural networks). Also, since neither prior domain knowledge nor machine learning knowledge is required, *FMS* can be applied to any classification problem without modification. This is a clear advantage over ad-hoc model selection methods that perform well on a single domain or that work for a fixed algorithm. This will help users with limited machine learning knowledge, since *FMS* can be seen as a black-box tool for model selection. Machine learning experts can also benefit from this approach. For example, several authors make use of search strategies for the selection of *candidate models* (Lutz, 2006; Boullé, 2007; Reunanen, 2007; Wichard, 2007), the *FMS* approach can be adopted for obtaining such candidate models.

One could expect a considerable loss of accuracy by gaining generality. However, this is not the case of the proposed approach since in international competitions it showed comparable performance to other techniques that were designed for a single algorithm (i.e. doing hyperparameter optimization) and to methods that took into account domain knowledge (Guyon et al., 2008). The main drawback is the computational cost to explore the vast search space, particularly for large data sets. But, we can gain efficiency without a significant loss in accuracy, by adopting a random subsampling strategy, see Section 14.4.3. The difficult interpretability of the selected models is another limitation of the proposed approach. However, naive users may accept to trade interpretably for ease-of-use, while expert users may gain insight in the problem at hand by analyzing the structure of the selected model (type of preprocessing chosen, number of features selected, linearity or non-linearity of the predictor).

In this paper, we propose to use particle swarm optimization (*PSO*) for exploring the full-models search space. *PSO* is a bio-inspired search technique that has shown comparable performance to that of evolutionary algorithms (Angeline, 1998; Reyes and Coello, 2006). Like evolutionary algorithms, *PSO* is useful when other techniques such as gradient descend or direct analytical discovery are not applicable. Combinatoric and real-valued optimization problems in which the optimization surface possesses many locally optimal solutions, are well suited for swarm optimization. In *FMS* it must be found the best combination of methods (for preprocessing, feature selection and learning) and simultaneously optimizing real valued functions (finding pseudo-optimal parameters for the considered methods), in consequence, the application of *PSO* is straightforward.

The methodological differences between swarm optimization and evolutionary algorithms have been highlighted by several authors (Angeline, 1998; Kennedy and Eberhart, 1995, 2001). However a difference in performance has not been demonstrated in favor of either method. Such demonstration would be a difficult task because no black-box stochastic optimization algorithm can outperform another over all optimization problems, not even over random search (Wolpert and Macready, 1997; van den Bergh, 2001). We selected *PSO* instead of evolutionary algorithms because of its simplicity and generality as no important modification was made for applying it to *FMS*. *PSO* is easier to implement than evolutionary algorithms because it only involves a single operator for updating solutions. In contrast, evolutionary algorithms require a particular representation and specific methods for cross-over, mutation, speciation and selection. Furthermore, *PSO* has been found to be very effective in a wide variety of applications, being able to produce good solutions at a very low computational cost (Gudise and Venayagamoorthy, 2003; Hernández et al., 2004; Xiaohui et al., 2003; Yoshida et al., 2001; Robinson, 2004; Kennedy and Eberhart, 2001; Reyes and Coello, 2006).

PSO is compared to pattern search (*PS*) in order to evaluate the added value of using the swarm strategy instead of another intensive search method. We consider *PS* among other search techniques because of its simplicity and proved performance in model selection (Momma and

Bennett, 2002; Bi et al., 2003; Dennis and Torczon, 1994). Cross validation (*CV*) is used in both techniques for assessing the *goodness* of models. Experimental results in benchmark data give evidence that both *PSO* and *PS* are effective strategies for *FMS*. However, it was found that *PSO* outperforms *PS*, showing better convergence behavior and being less prone to overfitting. Furthermore, the proposed method was evaluated in the context of a model selection competition in which several specialized and prior-knowledge based methods for model selection were used. Models selected with *PSO* were always among the top ranking models through the different stages of the challenge (Guyon et al., 2006c, 2007, 2008; Escalante et al., 2007). During the challenge, our best entry was ranked 8th over all ranked participants, 5th among the methods that did not use domain knowledge and 2nd among the methods that used the software provided by the organizers (Guyon et al., 2006c, 2007, 2008). In this paper we outperform the latter entry while reducing the computational burden by using a subsampling strategy; our best entry is currently the top-ranked one among models that do not use prior domain knowledge and 2nd over all entries, see Section 14.4.3.

PSO has been widely used for parameter selection in supervised learning (Kennedy and Eberhart, 1995, 2001; Salerno, 1997; Gudise and Venayagamoorthy, 2003). However, parameter selection is related with the first level of inference in which, given a learning algorithm, the task is to find parameters for such algorithm in order to describe the data. For example, in neural networks the adjustment of weights between units according to some training data is a parameter selection problem. Hyperparameter optimization, on the other hand, is related with the second level of inference, that is, finding parameters for the methods that in turn should determine parameters for describing the data. In the neural network example, selecting the optimal number of units, the learning rate, and the number of epochs for training the network is a hyperparameter optimization problem. *FMS* is capable of operating across several levels of inference by simultaneously performing feature selection, preprocessing and classifier selection, and hyperparameter optimization for the selected methods. *PSO* has been used for hyperparameter optimization by Voss et al. (Voss and Feng, 2002), however they restricted the problem to linear systems for univariate data sets, considering one hundred data observations. In this paper we are going several steps further: we applied *PSO* for *FMS* considering non-linear models in multivariate data sets with a large number of observations.

Parallel to this work, Gorissen et al. used genetic algorithms for meta-model selection in regression tasks (Gorissen, 2007; Gorissen et al., 2008), a similar approach that emerged totally independently to our proposal. One should note, however, that this method has been used for a different task in low dimensional data sets; most results are reported for 2D data. Even with this dimensionality the method has been run for only a few iterations with a small population size. Their use of a genetic algorithm required the definition of specific operators *for each of the considered models*. Gorissen et al. considered seven different models (including neural networks and kernel methods) that required of 18 different genetic operators for creation, mutation and cross-over (Gorissen, 2007; Gorissen et al., 2008). Additionally, general operators for speciation and selection were also defined. In the present work a single operator was used for updating solutions, regardless of the considered models. This clearly illustrates the main advantage of *PSO* over genetic algorithms, namely generality and simplicity.

The main contribution of this work is experimental: we provide empirical evidence indicating that by using *PSO* we were able to perform intensive search over a huge space and succeeded in selecting competitive models without significantly overfitting. This is due to the way the search is guided in *PSO*: performing a broad search around promising solutions but not overdoing in terms of really fine optimization. This sort of search is known to help avoiding overfitting by *undercomputing* (Dietterich, 1995). Experimental results supported by some a posteriori analysis give evidence of the validity of our approach. The way we approached the model selection problem and the use of a stochastic-search strategy are also contributions. To the best of our knowledge there are no similar works that consider the *FMS* problem for classification tasks.

The rest of this paper is organized as follows. In the next section we describe the general *PSO* algorithm. In Section 14.3, we describe the application of *PSO* to *FMS*. Section 14.4 presents experimental results in benchmark data; comparing the performance of *PSO* to that of *PS* in *FMS* and analyzing the performance of *PSO* under different parameter settings; also, are described the results obtained in the framework of a model selection competition. In Section 14.5, we analyze mechanisms in *PSMS* that allow to select competitive models without overfitting the data. Finally, in Section 14.6, we present the conclusions and outline future research directions.

14.2. Particle swarm optimization (*PSO*)

PSO is a population-based search algorithm inspired by the behavior of biological communities that exhibit both individual and social behavior; examples of these communities are flocks of birds, schools of fishes and swarms of bees. Members of such societies share common goals (e.g. finding food) that are realized by exploring its environment while interacting among them. Proposed by Kennedy et al. (Kennedy and Eberhart, 1995), *PSO* has become an established optimization algorithm with applications ranging from neural network training (Kennedy and Eberhart, 1995; Salerno, 1997; Kennedy and Eberhart, 2001; Gudise and Venayagamoorthy, 2003; Engelbrecht, 2006) to control and engineering design (Hernández et al., 2004; Xiaohui et al., 2003; Yoshida et al., 2001; Robinson, 2004). The popularity of *PSO* is due in part to the simplicity of the algorithm (Kennedy and Eberhart, 1995; Reyes and Coello, 2006; Engelbrecht, 2006), but mainly to its effectiveness for producing good results at a very low computational cost (Gudise and Venayagamoorthy, 2003; Kennedy and Eberhart, 2001; Reyes and Coello, 2006). Like evolutionary algorithms, *PSO* is appropriate for problems with immense search spaces that present many local minima.

In *PSO* each solution to the problem at hand is called a particle. At each time t, each particle, i, has a position $\mathbf{x}_i^t =< x_{i,1}^t, x_{i,2}^t, \dots, x_{i,d}^t >$ in the search space; where d is the dimensionality of the solutions. A set of particles $\mathbf{S} = \{\mathbf{x}_1^t, \mathbf{x}_2^t, \dots, \mathbf{x}_m^t\}$ is called a swarm. Particles have an associated velocity value that they use for *flying* (exploring) through the search space. The velocity of particle i at time t is given by $\mathbf{v}_i^t =< v_{i,1}^t, v_{i,2}^t, \dots, v_{i,d}^t >$, where $v_{i,k}^t$ is the velocity for dimension k of particle i at time t. Particles adjust their flight trajectories by using the following updating equations:

$$v_{i,j}^{t+1} = W \times v_{i,j}^t + c_1 \times r_1 \times (p_{i,j} - x_{i,j}^t) + c_2 \times r_2 \times (p_{g,j} - x_{i,j}^t) \tag{14.1}$$

$$x_{i,j}^{t+1} = x_{i,j}^t + v_{i,j}^{t+1} \tag{14.2}$$

where $p_{i,j}$ is the value in dimension j of the best solution found so far by particle i; $\mathbf{p}_i =< p_{i,1}, \dots, p_{i,d} >$ is called personal best. $p_{g,j}$ is the value in dimension j of the best particle found so far in the swarm ($\mathbf{S}$); $\mathbf{p}_g =< p_{g,1}, \dots, p_{g,d} >$ is considered the leader particle. Note that through $\mathbf{p}_i$ and $\mathbf{p}_g$ each particle i takes into account individual (local) and social (global) information for updating its velocity and position. In that respect, $c_1, c_2 \in \mathbb{R}$ are constants weighting the influence of local and global best solutions, respectively. $r_1, r_2 \sim U[0,1]$ are values that introduce randomness into the search process. W is the so called inertia weight, whose goal is to control the impact of the past velocity of a particle over the current one, influencing the local and global exploration abilities of the algorithm. This is one of the most used improvements of *PSO* for enhancing the rate of convergence of the algorithm (Shi and Eberhart, 1998, 1999; van den Bergh, 2001). For this work we considered an adaptive inertia weight specified by a triplet $\mathbf{W} = (w_{start}, w_f, w_{end})$; where w_{start} and w_{end} are the initial and final values for W, respectively, and w_f indicates the fraction of iterations in which W is decreased. Under this setting W is decreased by $W = W - wdec$ from iteration $t = 1$ (where $W = w_{start}$) up to iteration $t = I \times w_f$ (after which $W = w_{end}$); where $w_{dec} = \frac{w_{start} - w_{end}}{I \times w_f}$ and I is the maximum number of iterations. This setting allows us to explore a large area at the start of the optimization, when W is large, and to slightly refine the search later by using a smaller inertia weight (Shi and Eberhart, 1998, 1999; van den Bergh, 2001).

An adaptive W can be likened to the temperature parameter in simulated annealing (Kirkpatrick et al., 1983); this is because, in essence, both parameters influence the global and local exploration abilities of their respective algorithms, although in different ways. A constant W is analogous to the momentum parameter p in gradient descend with momentum term (Qian, 1999), where weights are updated by considering both the current gradient and the weight change of the previous step (weighed by p). Interestingly, the inertia weight is also similar to the weight-decay constant (γ) used in machine learning to prevent overfitting. In neural networks the weights are decreased by $(1 - \gamma)$ in each learning step, which is equivalent to add a penalty term into the error function that encourages the magnitude of the weights to decay towards zero (Bishop, 2006; Hastie et al., 2001); the latter penalizes complex models and can be used to obtain sparse solutions (Bishop, 2006).

The pseudo code of the *PSO* algorithm considered in this work is shown in Algorithm 14.1; default recommended values for the *FMS* problem are shown as well (these values are based on the analysis of Section 14.2.1 and experimental results from Section 14.4.2). The swarm is randomly initialized, considering restrictions on the values that each dimension can take. Next, the *goodness* of each particle is evaluated and $\mathbf{p}_g$, $\mathbf{p}_{1,\dots,m}$ are initialized. Then, the iterative *PSO* process starts, in each iteration: *i*) the velocities and positions of each particle in every dimension are updated according to Equations (14.1) and (14.2); *ii*) the *goodness* of each particle is evaluated; *iii*) $\mathbf{p}_g$ and $\mathbf{p}_{1,\dots,m}$ are updated, if needed; and *iv*) the inertia weight is decreased. This process is repeated until either a maximum number of iterations is reached or a minimum fitness value is obtained by a particle in the swarm (we used the first criterion for *FMS*); eventually, an (locally) optimal solution is found.

A fitness function is used to evaluate the aptitude (*goodness*) of candidate solutions. The definition of a specific fitness function depends on the problem at hand; in general it must reflect the proximity of the solutions to the optima. A fitness function $F : \Psi \to \mathbb{R}$, where Ψ is the space of particles positions, should return a scalar $f_{\mathbf{x}_i}$ for each particle position $\mathbf{x}_i$, indicating how far particle i is from the optimal solution to the problem at hand. For *FMS* the goal is to improve classification accuracy of full models. Therefore, any function F, which takes as input a model and returns an estimate of classification performance, is suitable (see Section 14.3.3).

Note that in Equation (14.1) every particle in the swarm knows the best position found so far by any other particle within the swarm, that is $\mathbf{p}_g$. Under this formulation a fully-connected swarm-topology is considered in which every member knows the leader particle. This topology has shown to converge faster than any other topology (Kennedy and Mendes, 2002; Reyes and Coello, 2006; Kennedy and Eberhart, 2001; Engelbrecht, 2006). With this topology, however, the swarm is prone to converge to local minima. We tried to overcome this limitation by using the adaptive inertia weight, W.

14.2.1. PSO Parameters

Selecting the best parameters (W, c_1, c_2, m, I) for *PSO* is another model selection task. In the application of *PSO* for *FMS* we are dealing with a very complex problem lying in the third level

Algorithm 14.1: Particle swarm optimization.

```
Require: [Default recommended values for FMS]
– $\mathbf{c}_1, \mathbf{c}_2$: individual/social behavior weights; $[\mathbf{c}_1 = \mathbf{c}_2 = 2]$
– $\mathbf{m}$: swarm size; $[\mathbf{m} = 5]$
– $\mathbf{I}$: number of iterations; $[\mathbf{I} = 50]$
– $\mathbf{F}(\Psi \to \mathbb{R})$: fitness function; $[\mathbf{F}(\Psi \to \mathbb{R}) = 2-$fold $CV\ BER$ ]
– $\mathbf{W}$: Inertia weight $\mathbf{W}$ = (1.2, 0.5, 0.4)
Set decrement factor for $W$ ($w_{dec} = \frac{w_{start} - w_{end}}{I \times w_f}$)
Initialize swarm ($\mathbf{S} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$)
Compute $f_{\mathbf{x}_{1,\ldots,m}} = F(\mathbf{x}_{1,\ldots,m})$ (Section 14.3.3)
Locate leader ($\mathbf{p}_g$) and set personal bests ($\mathbf{p}_{1,\ldots,m} = \mathbf{x}_{1,\ldots,m}$)
$t = 1$
while $t < I$ do
  for all $\mathbf{x}_i \in \mathbf{S}$ do
    Calculate velocity $\mathbf{v}_i$ for $\mathbf{x}_i$ (Equation (14.1))
    Update position of $\mathbf{x}_i$ (Equation (14.2))
    Compute $f_{\mathbf{x}_i} = F(\mathbf{x}_i)$
    Update $\mathbf{p}_i$ (if necessary)
  end for
  Update $\mathbf{p}_g$ (if necessary)
  if $t < \lfloor I \times w_f \rfloor$ then
    $W = W - w_{dec}$
  end if
  $t^{++}$
end while
return $\mathbf{p}_g$
```

of inference. Fortunately, several empirical and theoretical studies have been performed about the parameters of *PSO* from which useful information can be obtained (Shi and Eberhart, 1998, 1999; Kennedy and Mendes, 2002; Reyes and Coello, 2006; Ozcan and Mohan, 1998; Clerc and Kennedy, 2002; van den Bergh, 2001). In the rest of this section the *PSO* parameters are analyzed in order to select appropriate values for *FMS*. Later on, in Section 14.4.2, results of experiments with *PSO* for *FMS* under different parameter settings are presented.

We will start analyzing $c_1, c_2 \in \mathbb{R}$, the weighting factors for individual and social behavior. It has been shown that convergence is guaranteed[1] for *PSO* under certain values of c_1, c_2 (van den Bergh, 2001; Reyes and Coello, 2006; Ozcan and Mohan, 1998; Clerc and Kennedy, 2002). Most convergence studies have simplified the problem to a single one-dimensional particle, setting $\phi = c_1 \times r_1 + c_2 \times r_2$, and $\mathbf{p}_g$ and $\mathbf{p}_i$ constant (van den Bergh, 2001; Reyes and Coello, 2006; Ozcan and Mohan, 1998; Clerc and Kennedy, 2002). A complete study including the inertia weight is carried out by Van Den Bergh (van den Bergh, 2001). In agreement with this study the value $\phi < 3.8$ guarantees eventual convergence of the algorithm when the inertia weight W is close to 1.0. For the experiments in this work it was fixed $c_1 = c_2 = 2$, with these values we have $\phi < 3.8$ with high probability $P(\phi < 3.8) \approx 0.9$, given the uniformly distributed numbers r_1, r_2 (van den Bergh, 2001). The configuration $c_1 = c_2 = 2$ also has been proven, empirically, to be an effective choice for these parameters (Kennedy and Eberhart, 1995; Shi and Eberhart, 1998, 1999; Kennedy and Mendes, 2002).

Note, however, the restriction that W remains close to 1.0. In our experiments we considered a value of $W = 1.2$ at the beginning of the search, decreasing it during 50% (i.e. $w_f = 0.5$) of the *PSO* iterations up to the value $W = 0.4$. In consequence, it is possible that at the end of the search process the swarm may show divergent behavior. In practice, however, the latter configuration for W resulted very useful for *FMS*, see Section 14.4. This selection of W should not be surprising since even the configuration $\mathbf{W} = (1, 1, 0)$ has obtained better results than using a fixed W in empirical studies (Shi and Eberhart, 1998). Experimental results with different configurations of $\mathbf{W}$ for *FMS* give evidence that the configuration we selected can provide better models than using constant $\mathbf{W}$, see Section 14.4.2; although further experiments need to be performed in order to select the best $\mathbf{W}$ configuration for *FMS*.

With respect to m, the size of the swarm, experimental results suggest that the size of the population does not damage the performance of *PSO* (Shi and Eberhart, 1999), although slightly better results have been obtained with a large value of m, (Shi and Eberhart, 1999). Our experimental results in Section 14.4.2 confirm that the selection of m is not crucial, although (contrary to previous results) slightly better models are obtained by using a small swarm size. The latter is an important result for *FMS* because using a small number of particles reduces the number of fitness function evaluations, and in consequence the computational cost of the search.

Regarding the number of iterations, to the best of our knowledge, there is no work on the subject. This is mainly due to the fact that this issue depends mostly on the complexity of the problem at hand and therefore a general rule cannot be derived. For *FMS* the number of iterations should not be large to avoid oversearching (Dietterich, 1995). For most experiments in this paper we fixed $I = 100$. However, experimental results in Section 14.4.2 show that by running *PSO* for a smaller number of iterations is enough to obtain models of the same, and even better, generalization performance. This gives evidence that early stopping can be an useful mechanism to avoid overfitting, see Section 14.5.

1. Note that in *PSO* we say that the swarm converges iff $\lim_{t \to \inf} \mathbf{p}_{gt} = \mathbf{p}$, where $\mathbf{p}$ is an arbitrary position in the search space and t indexes iterations of *PSO*. Since $\mathbf{p}$ refers to an arbitrary position, this definition does not mean either convergence to local or global optimum, but convergence to the global best position in the swarm (van den Bergh, 2001; Reyes and Coello, 2006; Engelbrecht, 2006).

14.3. Particle swarm model selection

Since one of the strong advantages of *PSO* is its simplicity, its application to *FMS* is almost direct. Particle swarm full model selection (hereafter *PSMS*, that is the application of *PSO* to *FMS*) is described by the pseudocode in Algorithm 14.1, in this section are presented additional details about *PSMS*: first we describe the pool of methods considered in this work; then, we describe the representation of particles and the fitness function used; finally, we briefly discuss complexity issues. The code of *PSMS* is publicly available from the following website `http://ccc.inaoep.mx/~hugojair/code/psms/`.

14.3.1. The challenge learning object package

In order to implement *PSMS* we need to define the models search space. For this purpose we consider the set of methods in a machine learning toolbox from which full models can be generated. Currently, there are several machine learning toolboxes, some of them publicly available (Franc and Hlavac, 2004; van der Heijden et al., 2004; Wichard and Merkwirth, 2007; Witten and Frank, 2005; Saffari and Guyon, 2006; Weston et al., 2005); even there is a track of this journal (*JMLR*) dedicated to machine learning software. This is due to the increasing interest from the machine learning community in the dissemination and popularization of this research field (Sonnenburg, 2006). The *Challenge Learning Object Package*[2] (*CLOP*) is one of such development kits distributed under the GNU license (Saffari and Guyon, 2006; Guyon et al., 2006c, 2007, 2008). *CLOP* is a *Matlab*® toolbox with implementations of feature-variable selection methods and machine learning algorithms (*CLOP* also includes the *PSMS* implementation used in this work). The list of available preprocessing, feature selection and postprocessing methods in the *CLOP* toolbox is shown in Table 14.1; a description of the learning algorithms available in *CLOP* is presented in Table 14.2. One should note that this version of *CLOP* includes the methods that best performed in a model selection competition (Guyon et al., 2008; Cawley and Talbot, 2007a; Lutz, 2006).

In consequence, for *PSMS* the pool[3] of methods to select from are those methods described in Tables 14.1 and 14.2. In *CLOP* a typical model consists of the *chain*, which is a grouping object that allows us to perform serial concatenation of different methods. A chain may include combinations of (several/none) feature selection algorithm(s) followed by (several/none) preprocessing method(s), in turn followed by a learning algorithm and finally (several/none) postprocessing algorithm(s). For example, the model given by:
chain*{gs($fmax = 8$),standardize(center=1),neural(units=10,s=0.5,balance=1,iter=10)}*
uses *gs* for feature selection, *standardization* of data and a balanced *neural network* classifier with 10 hidden units, learning rate of 0.5, and trained for 10 iterations. In this work chain objects that include methods for preprocessing, feature selection and classification are considered full-models. Specifically, we consider models with at most one feature selection method, but allowing to perform preprocessing before feature selection and viceversa, see Section 14.3.2. The bias method was used as postprocessing in every model tested to set an optimal threshold in the output of the models in order to minimize their error. The search space in *FMS* is given by all the possible combinations of methods and hyperparameters; an infinite search space due to the real valued parameters.

2. `http://clopinet.com/CLOP/`
3. Notice that the *CLOP* package includes also the spider package (Weston et al., 2005) which in turn includes other implementations of learning algorithms and preprocessing methods. However, in this work we only used *CLOP* objects.

Table 14.1: Feature selection (*FS*), preprocessing (*Pre*) and postprocessing (*Pos*) objects available in *CLOP*. A brief description of the methods and their hyperparameters is presented.

ID	Object name	F	Hyperparameters	Description
1	*Ftest*	*FS*	$f_{max}, w_{min}, p_{val}, fdr_{max}$	Feature ranking according the F-statistic
2	*Ttest*	*FS*	$f_{max}, w_{min}, p_{val}, fdr_{max}$	Feature ranking according the T-statistic
3	*aucfs*	*FS*	$f_{max}, w_{min}, p_{val}, fdr_{max}$	Feature ranking according to the AUC criterion
4	*odds-ratio*	*FS*	$f_{max}, w_{min}, p_{val}, fdr_{max}$	Feature ranking according to the odds ratio statistic
5	*relief*	*FS*	$f_{max}, w_{min}, k_{num}$	Relief ranking criterion
6	*rffs*	*FS*	f_{max}, w_{min}	Random forest used as feature selection filter
7	*svcrfe*	*FS*	f_{max}	Recursive feature elimination filter using svc
8	*Pearson*	*FS*	$f_{max}, w_{min}, p_{val}, fdr_{max}$	Feature ranking according to the Pearson correlation coef.
9	*ZFilter*	*FS*	f_{max}, w_{min}	Feature ranking according to a heuristic filter
10	*gs*	*FS*	f_{max}	Forward feature selection with Gram-Schmidt orth.
11	*s2n*	*FS*	f_{max}, w_{min}	Signal-to-noise ratio for feature ranking
12	$pc - extract$	*FS*	f_{max}	Extraction of features with *PCA*
1	*normalize*	*Pre*	center	Normalization of the lines of the data matrix
2	*standardize*	*Pre*	center	Standardization of the features
3	$shift - scale$	*Pre*	$take_{log}$	Shifts and scale data
1	*bias*	*Pos*	none	Finds the best threshold for the output of the classifiers

Table 14.2: Available learning objects with their respective hyperparameters in the CLOP package.

ID	Object name	Hyperparameters	Description
1	*zarbi*	none	Linear classifier
2	*naive*	none	Naïve Bayes
3	*klogistic*	none	Kernel logistic regression
4	*gkridke*	none	Generalized kridge (VLOO)
5	*logitboost*	units number, shrinkage, depth	Boosting with trees (R)
6	*neural*	units number, shrinkage, maxiter, balance	Neural network (Netlab)
7	*svc*	shrinkage, kernel parameters (coef0, degree, gamma)	SVM classifier
8	*kridge*	shrinkage, kernel parameters (coef0, degree, gamma)	Kernel ridge regression
9	*rf*	units number, balance, mtry	Random forest (R)
10	*lssvm*	shrinkage, kernel parameters (coef0, degree, gamma), balance	Kernel ridge regression

14.3.2. Representation

In *PSO* each potential solution to the problem at hand is considered a particle. Particles are represented by their position, which is nothing but a $d-$dimensional numerical vector (d being the dimensionality of the solution). In *FMS* potential solutions are full-models, in consequence, for *PSMS* we need a way to codify a full-model by using a vector of numbers. For this purpose we propose the representation described in Equation (14.3), the dependence on time (t) is omitted for clarity.

$$\mathbf{x}_i =< x_{i,pre}, y_{i,1\dots N_{pre}}, x_{i,fs}, y_{i,1,\dots N_{fs}}, x_{i,sel}, x_{i,class}, y_{i,1,\dots N_{class}} > \tag{14.3}$$

Where $x_{i,pre} \in \{1,\dots,8\}$ represents a combination of preprocessing methods. Each combination is represented by a binary vector of size 3 (i.e. the number of preprocessing methods considered), there are $2^3 = 8$ possible combinations. Each element of the binary vector represents a single preprocessing method; if the value of the kth element is set to 1 then the preprocessing method with *ID* = k is used (see Table 14.1). For example, the first combination $< 0,0,0 >$ means *no preprocessing*; while the seventh $< 1,1,0 >$ means that this model ($\mathbf{x}_i$) uses *normalization* and *standardization* as preprocessing. $y_{i,1\dots N_{pre}}$ codify the hyperparameters for the selected combination of preprocessing methods, $N_{pre} = 3$ because each preprocessing method has a single hyperparameter; note that the order of the preprocessing methods is fixed (i.e. *standardization* can never be performed before *normalization*), in the future we will relax this constraint. $x_{i,fs} \in \{0,\dots,12\}$ represents the *ID* of the feature selection method used by the model (see Table 14.1), and $y_{i,1\dots N_{fs}}$ its respective hyperparameters; N_{fs} is set to the maximum number of hyperparameters that any feature selection method can take. $x_{i,sel}$ is a binary variable that indicates whether preprocessing should be performed before feature selection or viceversa. $x_{i,class} \in \{1,\dots,10\}$ represents the classifier selected and $y_{i,1,\dots N_{class}}$ its respective hyperparameters; N_{class} is the maximum number of hyperparameters that a classifier can take. This numerical codification must be decoded and used with the *chain* grouping object for obtaining a full-model from a particle position $\mathbf{x}_i$. Note that the dimensionality of each particle is $d = 1 + N_{pre} + 1 + N_{fs} + 1 + 1 + N_{class} = 16$.

14.3.3. Fitness function

In *FMS* it is of interest to select models that minimize classification errors on unseen data (i.e. maximizing generalization performance). Therefore, the fitness function (F) should relate a model with an estimate of its classification performance in unseen data. The simplicity of *PSMS* allows us to use any classification performance measure as F, because the method does not require derivatives. Thus, valid options for F include mean absolute error, balanced error rate, squared root error, recall, precision, area under the *ROC* curve, etcetera. For this work it was used the balanced error rate (*BER*) as F. *BER* takes into account misclassification rates in both classes, which prevents *PSMS* of selecting biased models (favoring the majority class) in imbalanced data sets. Furthermore, *BER* has been used in machine learning challenges as leading error measure for ranking participants (Guyon et al., 2007, 2008). The *BER* of model ψ is the average of the misclassifications obtained by ψ over the classes in a data set, as described in Equation (14.4):

$$BER(\psi) = \frac{E_+ + E_-}{2} \tag{14.4}$$

where E_+ and E_- are the misclassifications rates for the positive and negative classes, respectively.

The selection of the fitness function is a crucial aspect in *PSO*. However, for *PSMS*, the critical part lies in the way an estimate of generalization performance of a model (given F and training data) is obtained, and not in the fitness function itself. This is the main challenge of model selection, since error estimates using training data are very optimistic about the behavior of models on unseen data, this phenomenon is known as overfitting (Bishop, 2006; Hastie et al., 2001; Nelles, 2001). In order to overcome overfitting the *BER* was calculated using a $k-$fold cross validation *(k-fold CV)* approach (Note that the *BER* is still obtained from training data). This is the only explicit mechanism of *PSMS* to avoid overfitting. *k-fold CV* is the most used hold-out approach for model selection (Nelles, 2001; Hastie et al., 2001; Cawley and Talbot, 2007b). Using a high value for k, say $k = N$ where N is the training set size (i.e. *leave one out - CV*), the error estimate is almost unbiased, but can have high variance because the N training sets are very similar to each other (Nelles, 2001; Cawley and Talbot, 2007b); furthermore, computation time could be a serious problem (Hastie et al., 2001; Nelles, 2001). With a low value for k, the estimate can have low variance but the bias could be a problem. The selection of an optimal k is still an open problem in the model selection field. For this work were performed experiments with $k \in \{2, 5, 10\}$, but no statistically-significant difference, among these values, was found, see Section 14.4.2.

14.3.4. Computational complexity

As we have seen, the search space in the *FMS* problem is composed by all the possible models that can be built given the considered methods and their hyperparameters. This is an infinite search space even with the restriction imposed to the values that models can take; this is the main drawback of *FMS*. However, the use of particle swarm optimization (PSO) allows us to harness the complexity of this problem. Most algorithms used for *FMS* cannot handle very big search spaces. But PSO is well suited to large search spaces: it converges fast and has a manageable computational complexity (Kennedy and Eberhart, 2001; Reyes and Coello, 2006). As we can see from Algorithm 14.1, *PSO* is very simple and does not involve expensive operations.

The computational expensiveness of *PSMS* is due to the fitness function we used. For each selected model the fitness function should train and evaluate such model $k-$times. Depending on the model complexity this process can be performed on linear, quadratic or higher order times. Clearly, computing the fitness function using the entire training set, as opposed to *k-fold CV*, could reduce *PSMS* complexity, although we could easily overfit the data. For a single run of *PSMS* the fitness function should be evaluated $\rho = m \times (I + 1)$ times, with m being the swarm size and I the number of iterations. Suppose the computational complexity of model λ is bounded by λ_O then the computational complexity of *PSMS* will be bounded by $\rho \times k \times \lambda_O$. Because λ_O is related to the computational complexity of model λ (which depends on the size and dimensionality of the data set) this value may vary dramatically. For instance, computing the fitness function for a naïve Bayes model in a high dimensional data set takes around two seconds[4], whereas computing the same fitness function for the same data set could take several minutes if a support vector classifier is used.

In order to reduce the computational cost of *PSMS* we could introduce a complexity penalty-term into the fitness function (this is current work). A simpler alternative solution is calculating the fitness function for each model using only a small subset of the available data; randomly selected and different each time. This approach can also be useful for avoiding local minima. The subsampling heuristic was used for high-dimensional data sets and for data sets with a large number of examples. Experimental results show an important reduction of processing time, without a significant loss of accuracy, see Section 14.4.3. We emphasize that complexity is due to the nature of the *FMS* problem. With this approach, however, users will be able to

4. Most of the experiments were carried out on a workstation with *Pentium*^{TM} 4 processor at 2.5 GHz, and 1 gigabyte in RAM.

obtain models for their data without requiring knowledge on the data or on machine learning techniques.

14.4. Experimental results

In this section results of experiments with *PSMS* using benchmark data from two different sources are presented. First, we present results on a suite of benchmark machine learning data sets[5] used by several authors (Mika et al., 2000; Rätsch et al., 2001; Cawley and Talbot, 2007b); such data sets are described in Table 14.3, ten replications (i.e. random splits of training and testing data) of each data set were considered. These data sets were used to compare *PSO* to *PS* in the *FMS* problem (Section 14.4.1) and to study the performance of *PSMS* under different settings (Section 14.4.2).

Table 14.3: Data sets used in the comparison of *PSO* and *PS*, ten replications (i.e. random splits of training and testing sets) for each data set were considered.

ID	Data set	Training Patterns	Testing Patterns	Input Features
1	Breast cancer	200	77	9
2	Diabetes	468	300	8
3	Flare solar	666	400	9
4	German	700	300	20
5	Heart	170	100	13
6	Image	1300	1010	20
7	Splice	1000	2175	60
8	Thyroid	140	75	5
9	Titanic	150	2051	3

Next we applied *PSMS* to the data sets used in a model selection competition (Section 14.4.3). The goal of the latter experiments is to compare the performance of *PSMS* against other model selection strategies.

14.4.1. A comparison of *PSO* and *PS*

In the first set of experiments we compared the performance of *PSO* to that of another search strategy in the *FMS* problem. The goal was to evaluate the advantages of using the swarm strategy over another intensive search method. Among the available search techniques we selected *PS* because of its simplicity and proved performance in model selection (Dennis and Torczon, 1994; Momma and Bennett, 2002; Bi et al., 2003). *PS* is a direct search method that samples points in the search space in a fixed pattern about the best solution found so far (the center of the pattern). Fitness values are calculated for the sampled points trying to find a minimizer; if a new minimum is find then the center of the pattern is changed, otherwise the search step is reduced by half; this process is iterated until a stop criteria is met. The considered *PS* algorithm described in Algorithm 14.2 is an adaptation of that proposed by Momma et al. for hyperparameter optimization of support vector regression (Momma and Bennett, 2002; Bi et al., 2003).

The input to *PS* is the pattern P and the search step Δ. Intuitively, P specifies the direction of the neighboring solutions that will be explored; while Δ specifies the distance to such neighboring solutions. There are several ways to generate P; for this work we used the nearest

5. http://ida.first.fraunhofer.de/projects/bench/benchmarks.htm.

Algorithm 14.2: Pattern search. p_i is the i^{th} column of P, N_c the number of columns in P.

```
Require:
– $I_{ps}$: number of iterations
– $\mathbf{F}(\Psi \rightarrow \mathbb{R})$: fitness function
– $\mathbf{P}$: pattern
– $\Delta$: search step
Initialize solution $\mathbf{q}_i$ $(i = 1)$
Compute $f_{\mathbf{q}_i} = F(\mathbf{q}_i)$ (Section 14.3.3)
Set $\mathbf{q}_g = \mathbf{q}_i$; $f_{min} = f_{\mathbf{q}_i}$
while $i < I_{ps}$ do
  for all $p_j \in P_{1,\dots,N_c-1}$ do
    $s_j = \Delta.p_j$
    $\mathbf{q}_j = \mathbf{q}_g + s_j$
    Compute $f_{\mathbf{q}_j} = F(\mathbf{q}_j)$
    if $f_{\mathbf{q}_j} < f_{min}$ then
      Update $\mathbf{q}_g$ $(\mathbf{q}_g = \mathbf{q}_j, f_{min} = f_{\mathbf{q}_j})$
    end if
    $i^{++}$
  end for
  $\Delta = \Delta/2$
end while
return $\mathbf{q}_g$
```

neighbor sampling pattern (Momma and Bennett, 2002). Such pattern is given numerically by $P = [\mathbf{I}_d \; -\mathbf{I}_d \; \mathbf{0}^T_{1\times d}]$, where $\mathbf{I}_d$ is the identity matrix of size d; $\mathbf{0}_{1\times d}$ is a vector of size $1_\times d$ with all zero entries; d is the dimensionality of the problem. Δ, a vector of size $1 \times d$, is the search step by which the space of solutions is explored. We defined $\Delta = \frac{\mathbf{q}_{maxvals} - \mathbf{q}_{minvals}}{2}$, where $\mathbf{q}_{maxvals}$ and $\mathbf{q}_{minvals}$ are the maximum and minimum values that the solutions can take, respectively. Each iteration of *PS* involves the evaluation of $N_c - 1$ solutions (where N_c is the number of columns of P). Solutions are updated by adding s_j $(j \in 1,\dots,N_c - 1)$ to the current-best solution $\mathbf{q}_g$; where each s_j is obtained by multiplying (element-by-element) the search step vector Δ and the j^{th} column of P. $\mathbf{q}_g$ is replaced by a new solution $\mathbf{q}_j$ only if $f_{\mathbf{q}_j} < f_{min} = f_{\mathbf{q}_g}$. The output of *PS* is $\mathbf{q}_g$, the solution with the lowest fitness value; we encourage the reader to follow the references for further details (Momma and Bennett, 2002; Dennis and Torczon, 1994).

For applying *PS* to the *FMS* problem (hereafter *PATSMS*) the solution $\mathbf{q}_g$ was initialized in the same way that each particle in the swarm was initialized for *PSMS* (i.e. randomly). The same representation, fitness function and restrictions on the values that each dimension can take were considered for both methods, see Section 14.3. Under these settings *PS* is a very competitive baseline for *PSO*.

In each experiment described below, we let *PS* and *PSO* perform the same number of fitness function evaluations, using exactly the same settings and data sets, guaranteeing a fair comparison. Since both methods use the same fitness function and perform the same number of fitness function evaluations, the difference in performance indicates the gain we have by guiding the search according to Equations (14.1) and (14.2)). As recommended by Demsar, we used the Wilcoxon signed-rank test for the comparison of the resultant models (Demsar, 2006). In the following we will refer to this statistical test with 95% of confidence when mentioning statistical significance.

We compared the *FMS* ability of *PSMS* and *PATSMS* by evaluating the accuracy of the selected models at the end of the search process; that is, we compared the performance of models $\mathbf{p}_g$ and $\mathbf{q}_g$ in Algorithms 14.1 and 14.2, respectively. We also compared the performance of the solutions tried through the search. For each trial we fixed the number of iterations for *PSMS* to $I = 100$ with a swarm size of $m = 10$. In consequence, both search strategies performed $m \times (I+1) = 1010$ evaluations of the fitness function in each run. Because more than $180,000$ models were tested we used $2-$fold *CV* for computing the fitness function. In each trial the training set was used for *FMS* and the resultant model was evaluated in the test set. This process was repeated for each replica of each data set described in Table 14.3. Averaged results of this experiment are shown in Table 14.4. We show the average *CV* error obtained through the search process (*CV-BER*) and the error obtained by the selected model, at the end of the search, in the test set (*test-BER*).

Table 14.4: Average and variance of *test-BER* and *CV-BER* obtained by models selected with *PSMS* and *PATSMS*, the best results are shown in **bold**. *test-BER* is the *BER* obtained by the selected model using the test set (averaged over 10 replications of each data set). *CV-BER* is the average of *CV BER* obtained by each of the candidate solutions through the search process (averaging performed over all particles and iterations and over 10 replications of each data set) .

ID	**Data set**	**PATSMS test-BER**	**PSMS test-BER**	**PATSMS CV-BER**	**PSMS CV-BER**
1	Breast-cancer	36.98$\pm$0.08	**33.59$\pm$0.12**	**32.64$\pm$0.06**	32.96$\pm$0.01
2	Diabetes	26.07$\pm$0.03	**25.37$\pm$0.02**	**25.39$\pm$0.02**	26.48$\pm$0.05
3	Flare-solar	32.87$\pm$0.02	**32.65$\pm$0.01**	**32.69$\pm$0.01**	33.13$\pm$0.01
4	German	28.65$\pm$0.02	**28.28$\pm$0.02**	**31.00$\pm$0.00**	31.02$\pm$0.00
5	Heart	19.50$\pm$0.19	**17.35$\pm$0.06**	**16.96$\pm$0.07**	19.93$\pm$0.03
6	Image	3.58$\pm$0.01	**2.50$\pm$0.01**	**11.54$\pm$0.10**	15.88$\pm$0.04
7	Splice	13.94$\pm$0.99	**9.46$\pm$0.25**	**18.01$\pm$0.05**	19.15$\pm$0.07
8	Thyroid	10.84$\pm$0.39	**5.98$\pm$0.06**	**11.15$\pm$0.20**	15.49$\pm$0.12
9	Titanic	29.94$\pm$0.00	**29.60$\pm$0.00**	**27.19$\pm$0.13**	27.32$\pm$0.13

The performance of both search strategies is similar. *PSMS* outperformed *PATSMS* through all of the data sets at the end of the search process (Columns 3 and 4 in Table 14.4). The *test-BER* differences are statistically significant for all but the *flare-solar* and *german* data sets. In the latter data sets the hypothesis that models selected with *PATSMS* and *PSMS* perform equally cannot be rejected. However, note that for these data sets the models selected with *PSMS* outperformed those selected with *PATSMS* in 6 out of 10 replications. Globally, we noticed that from the 90 trials (9$-$data sets, $\times$ 10$-$replications for each, see Table 14.3), 68.9% of the models selected with *PSMS* outperformed those obtained with *PATSMS*. While only in 22.2% of the runs *PATSMS* outperformed *PSMS*, in the rest both methods tied in performance. A statistical test over these 90 results was performed and a statistically-significant difference, favoring *PSMS*, was found. Despite *PATSMS* being a strong baseline, these results give evidence that *PSMS* outperforms *PATSMS* at the end of the search process.

Columns five and six in Table 14.4 show the average *BER* obtained with each strategy through the 1010 evaluations for each data set (averaged over replications). *CV-BER* reflects the behavior of the search strategies through the search process. *PATSMS* slightly outperformed *PSMS* in this aspect, though the difference is statistically significant only for the *Heart*, *image* and *thyroid* data sets. The slight superior performance of *PATSMS* in *CV-BER* is due to the pattern we used and the search procedure itself. *PATSMS* performs a finer grained search over a restricted search area around the initial solution $\mathbf{q}_g$. The latter results in a lower *CV-BER* because *PATSMS* always moves the pattern towards the local minimum nearest to the initial solution, $\mathbf{q}_g$. *PSMS*, on the other hand, explores a much larger search space because the search is not only guided by the best solution found so far ($\mathbf{p}_g$), but also by the individual best solutions of each particle ($\mathbf{p}_{1,\dots,m}$). The latter produces a higher *CV-BER* in average because, even when the *CV-BER* of the final models is low, many models of varied *CV-BER* performance are tried through the search.

In Figure 14.1 we show the performance of the solutions tried through the search by each method for a single replication of the *Heart* data set (for clarity in the plots we used $m = 5$ and $I = 50$ for this experiment). We show the *CV* and test *BER* for every model tried through the search. It can be seen that the *CV-BER* of *PATSMS* is lower than that of *PSMS*, showing an apparent better convergence behavior (left plot in Figure 14.1). However, by looking the test *BER* of the models tried, it becomes evident that *PATSMS* is trapped into a local minimum since the very first iterations (right plot in Figure 14.1). *PSMS*, on the other hand, obtains a higher *CV-BER* through the search, though it is less prone to follow a local minimum. This is because with *PSMS* the search is guided by one global and m local solutions, which prevents *PSMS* from performing a pure local search; the latter in turn prevents *PSMS* of overfitting the data. This result gives evidence of the better convergence behavior of *PSMS*.

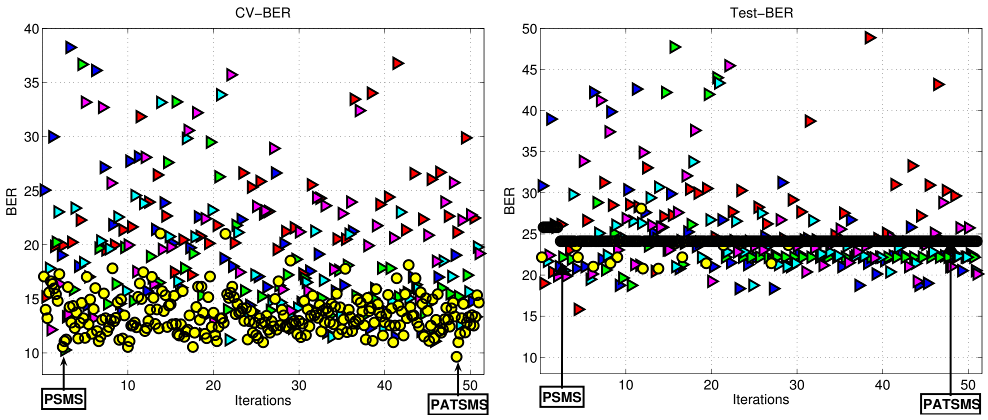


Figure 14.1: Performance of *PATSMS* (circles) and *PSMS* (triangles) as a function of number of iterations for an experiment with one replication of the *Heart* data set. For *PSMS* we show the performance of each particle with a different color. Left: Cross-validation Balanced Error Rate (*BER*). Right: Test set *BER*. In both plots we indicate with an arrow the model selected by each search strategy.

The model selected with *PSMS* obtained a lower *test-BER* than that selected with *PATSMS* (see right plot in Figure 14.1). In fact all of the solutions in the final swarm outperformed the solution obtained with *PATSMS* in *test-BER*. With *PSMS* the model of lowest test *BER* was obtained after 3 iterations of *PSMS*, giving evidence of the fast convergence of *PSO*. One should note that the *test-BER* of the worst *PSMS* solutions is higher than that of the best *PATSMS* solution. However, near the end of the search, the possibilities that *PSMS* can select a worse solution than *PATSMS* are small.

We have seen that *PATSMS* is prone to get trapped into a local minimum because it performs a fine grained search over a small area. This causes that only a few methods are considered through the search with this method. *PSMS*, on the other hand, tries a wide variety of classifiers and feature selection methods, exploring a larger search space and being able to escape from local minima. In order to analyze the diversity of the methods considered by *PSMS*, in Figure 14.2 we show the normalized frequency of methods preferred by *PSMS*[6] through the search. The results shown in this figure are normalized over the $90,900$ models tried for obtaining the results from Table 14.4. From this figure, we can see that most of the classifiers and feature selection methods were considered for creating solutions. No classifier, feature selection method or combination of preprocessing methods was used for more than 27% of the models. This reflects the fact that different methods are required for different data sets and that some are equivalent. There were, however, some methods that were slightly more used than others.

The preferred classifier was the *zarbi CLOP*-object, that was used for about 23% of the models. This is a surprising result because *zarbi* is a very simple linear classifier that separates the classes by using information from the mean and variance of training examples (Golub et al., 1999). However, in 97.35% of the times that *zarbi* was used it was combined with a feature selection method. *Gkridge, svc, neural* and *logitboost* were all equally selected after *zarbi*. $Ftest$ was the most used feature selection method, though most of the feature selection strategies were used. Note that *Pearson, Zfilter, gs* and *s2n* were considered only for a small number of models. The combination *standardize* + *shift-scale* was mostly used for preprocessing, although the combination *normalize* + *standardize* + *shift-scale* was also highly used. Interestingly, in 70.1% of the time preprocessing was performed before feature selection. These plots illustrate the diversity of classifiers considered by *PSMS* through the search process, showing that *PSMS* is not biased towards models that tend to perform very well individually (e.g. *logitboost, rf* or *gkridge*). Instead, *PSMS* attempts to find the best full model for each data set.

14.4.2. Parameter selection for *PSMS*

In this section we analyze the performance of *PSMS* under different settings. The goal is to identify mechanisms in *PSMS* that allow us obtaining competitive models and, to some extent, avoiding overfitting. For the experiments described in this section we consider a single replication for each data set. As before, we show the average *CV* error of the solutions considered during the search as well as the error of the selected model in the test set. We also show the average test set error of all solutions tried through the search *test-BER** (averaging performed over all particles and all iterations), providing information about the generalization performance of the models considered throughout the search.

VALUE OF K IN K-FOLD CV

First we analyze the behavior of *PSMS* for different values of k in the *CV* for computing the fitness function. We consider the values $k = [2, 5, 10]$. Average results of this experiment are shown in Table 14.5.

6. We do not show those preferred by *PATSMS* because the methods selected with this strategy are, most of the times, those considered in the initial solution. In our experiments we found that for each replication *PATSMS* used the same classifier and feature selection method for about 95% of the search iterations. The selection of these methods depended on the initial solution instead of the way the search space is explored or the individual performance of the methods. Therefore, no useful information can be obtained about why some methods are preferred over others, even when in average (over the 90,900 solutions tried) the histograms may look similar to those shown in Figure 14.2.

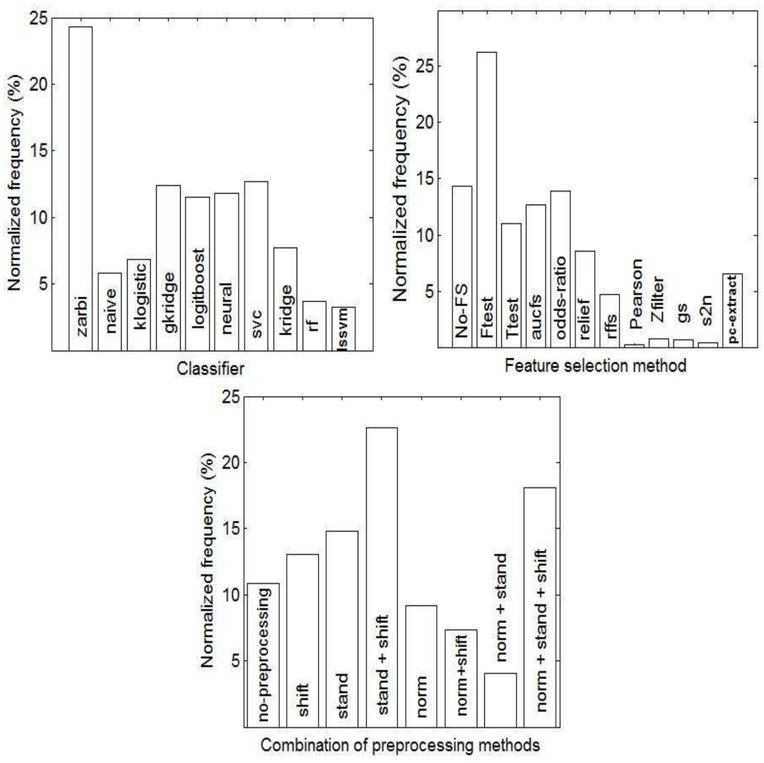


Figure 14.2: Normalized frequency of classifiers (left), feature selection method (middle) and combination of preprocessing methods (right) preferred by *PSMS* through the search process. In the right plot, the abbreviations *shift*, *stand* and *norm* stand for *shift-scale*, *standardization* and *normalization*, respectively. Results are normalized over the 90,900 models tried for obtaining the results from Table 14.4.

Table 14.5: Average *CV-BER* (average CV result over all particles and iterations), *test-BER* (test result corresponding to the best CV result), *test-BER** (average test result over all particles and iterations) and processing time (in seconds) for different values of k in the $k-$fold *CV*. Results are averaged over a single replication of each data set described in Table 14.3.

k	CV-BER	test-BER	test-BER*	Time
2	25.36$\pm$0.56	22.99$\pm$1.18	25.79$\pm$0.59	4166.85
5	25.18$\pm$0.59	20.50$\pm$1.53	26.30$\pm$0.46	6062.56
10	24.54$\pm$0.63	20.82$\pm$1.74	26.03$\pm$0.51	7737.11

From this table, we can see that the performance is similar for the different values of k considered. The best results at the end of the search are obtained with $k = 5$ (column 3 in Table 14.5), while the best generalization performance is obtained with $k = 2$ (column 4 in Table 14.5). However, these differences are not statistically significant. Therefore, the null hypothesis that models selected with $k = [2,5,10]$ perform equally in *test-BER* and *test-BER** cannot be rejected. The latter is an important result because by using $k = 2$ the processing time of *PSMS* is considerably reduced. Note that for models of quadratic (or higher order) complexity, computing the fitness function with $2-$fold *CV* is even more efficient than computing the fitness function using the full training set. It is not surprising that processing time increases as we increase k (column 5 in Table 14.5). Although it is worth mentioning that the variance in processing time was very large (e.g. for $k = 2$ it took 7 minutes applying *PSMS* to the *titanic* data set and about five hours for the *image* data set).

NUMBER I OF ITERATIONS

Next we performed experiments varying the number of iterations (I), using $2 - fold\ CV$ for computing the fitness function and a swarm size of $m = 10$. We considered the values $I = [10,25,50,100]$. Averaged results for this experiment are shown in Table 14.6 (rows 2–5).

Table 14.6: Average *CV-BER*, *test-BER* and *test-BER** for different settings of m, I and $\mathbf{W}$. The best results are shown in **bold**. Results are averaged over a single replication of each data set described in Table 14.3.

Setting	CV-BER	test-BER	test-BER*
I=10	25.33$\pm$0.33	22.17$\pm$1.81	27.64$\pm$0.52
I=25	25.29$\pm$0.33	21.88$\pm$1.68	27.59$\pm$0.49
I=50	**24.02$\pm$0.38**	**21.12$\pm$1.74**	**26.72$\pm$0.65**
I=100	24.57$\pm$0.37	22.81$\pm$1.44	27.27$\pm$0.48
m=5	**24.27$\pm$0.49**	**20.81$\pm$1.50**	**25.01$\pm$0.85**
m=10	25.07$\pm$0.34	21.64$\pm$2.04	25.99$\pm$0.74
m=20	25.09$\pm$0.34	21.76$\pm$1.84	26.00$\pm$0.64
m=40	24.82$\pm$0.43	21.45$\pm$2.13	25.96$\pm$0.78
m=50	25.32$\pm$0.44	22.54$\pm$1.65	26.11$\pm$0.77
W=(0,0,0)	**23.86$\pm$0.71**	20.40$\pm$1.71	**22.46$\pm$1.32**
W=(1.2,0.5,0.5)	24.22$\pm$0.76	**19.41$\pm$1.37**	23.38$\pm$1.34
W=(1,1,1)	27.62$\pm$0.30	21.88$\pm$1.68	27.13$\pm$0.53

As we can see the best results were obtained by running *PSMS* for 50 iterations. Interestingly, models selected by running *PSMS* for 10 iterations outperformed those selected after 100 iterations in *test-BER*, though the difference was not statistically significant. The difference in performance between models selected after 50 and 100 iterations was statistically significant. This result shows the fast convergence property of *PSMS* and that early stopping could be an useful mechanism to avoid overfitting, see Section 14.5.

SWARM SIZE M

In the next experiment we fixed the number of iterations to $I = 50$ and varied the swarm size as follows, $m = [5,10,20,40,50]$. Results of this experiment are shown in rows 6–10 in Table 14.6. This time the best result was obtained by using a swarm size of 5, however there is a statistically-significant difference only between $m = 5$ and $m = 50$. Therefore, models of comparable performance can be obtained by using $m = [5, 10, 20, 40]$. This is another interesting result because using a small swarm size reduces the number of fitness function evaluations for *PSMS* and therefore makes it more practical. An interesting result is that by using 50 iterations with any swarm size the *test-BER* is very close to the *CV-BER* estimate. Again, this provides evidence that early stopping can improve the average generalization performance of the models.

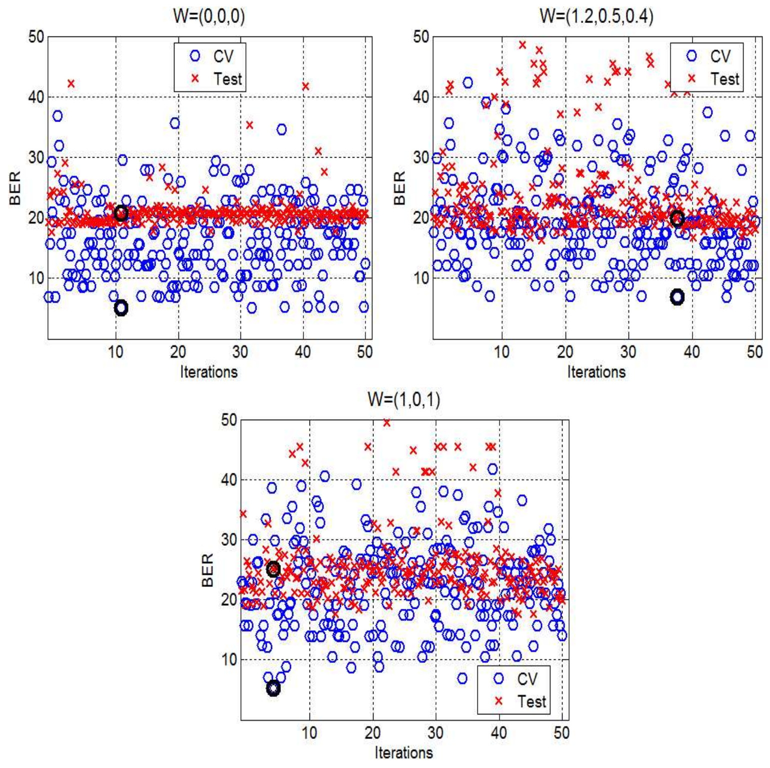


Figure 14.3: Algorithm performance as a function of number of iterations for different configurations of **W**. *CV-BER* (circles) and *test-BER** (crosses) for the *Heart* data set. We are displaying the test and CV BER values for each particle at every time step. Since each time step involves m=5 particles, then for each iteration are displayed m=5 crosses and m=5 circles. We consider the following configurations: $\mathbf{W} = (0, 0, 0)$ (left), $\mathbf{W} = (1.2, 0.5, 0.4)$ (middle) and $\mathbf{W} = (1, 0, 1)$ (right). The *CV-BER* and *test-BER* of the best solution found by *PSMS* are enclosed within a bold circle.

INERTIA WEIGHT W

We also performed experiments with different configurations for **W**, the adaptive inertia weight. Each configuration is defined by a triplet $\mathbf{W} = (w_{start}, w_f, w_{end})$, whose elements indicate the starting value for W, the proportion of iterations to vary it and its final value, respectively, see Section 14.2. Three configurations were tried with the goal of evaluating the advantages of using an adaptive inertia weight instead of constant values. Results of this experiment are shown in rows 11–13 in Table 14.6. It can be seen that the best results in *CV-BER* and *Test-BER** were obtained with $\mathbf{W} = (0,0,0)$; the differences with the other results are statistically-significant. Under this configuration *PSMS* is not taking into account information from past velocities for updating solutions; which causes *PSMS* to converge quickly into a local minimum and refining the search around this point. The best result at the end of the search (column 3 in Table 14.6) was obtained with $\mathbf{W} = (1.2,0.5,0.4)$, the difference with the other results is statistically-significant. Under this configuration both global and local search is performed during the *PSMS* iterations; which caused higher *CV-BER* and *test-BER** than that of the first configuration, however, the generalization performance of the final model was better. The configuration $\mathbf{W} = (1,0,1)$ obtained the worst results in all of the measures; this is because under this configuration the search is never refined, since *PSMS* always takes into account the past velocity for updating solutions. The latter configuration could be a better choice for *FMS* because this way *PSMS* does not over-search around any solution; however, local search is also an important component of any search algorithm. In Figure 14.3 we show the *CV* and test *BER* of solutions tried during the search for the *Heart* data set under the different configurations tried. From this figure we can appreciate the fact that using constant values for W results in more local (when $\mathbf{W} = (0,0,0)$) or global search (when $\mathbf{W} = (1,0,1)$). An adaptive inertia weight, on the other hand, aims to control the tradeoff between global and local search, which results in a model with lower variance for this example. Therefore, an adaptive inertia weight seems to be a better option for *FMS*; this is because it prevents, to some extend, *PSMS* to overfit the data. However, further experiments need to be performed in order to select the best configuration for $\mathbf{W}$.

INDIVIDUAL (c_1) AND GLOBAL (c_2) WEIGHTS

We now analyze the performance of *PSMS* under different settings of the individual (c_1) and global (c_2) weights. We considered three configurations for c_1 and c_2; in the first one both weights have the same influence in the search (i.e. $c_1 = 2; c_2 = 2$), this was the setting used for all of the experiments reported in Section 14.4. In the second setting the local weight has no influence in the search (i.e. $c_1 = 0; c_2 = 2$), while in the third configuration the global weight is not considered in the search (i.e. $c_1 = 2; c_2 = 0$). We ran *PSMS* for $I = 50$ iterations with a swarm size of $m = 5$, using a single replication for each data set described in Table 14.3; for the three configurations considered it was used the same adaptive inertia weight $\mathbf{W} = (1.2,0.5,0.4)$; averaged results of this experiment are shown in Table 14.7.

Table 14.7: Average *CV-BER*, *test-BER* and *test-BER** for different settings of c_1 and c_2 in Equation (14.1). The best results are shown in **bold**. Results are averaged over a single replication of each data set described in Table 14.3.

ID	**Setting**	**CV-BER**	**test-BER**	**test-BER***
1	$c_1 = 2; c_2 = 2$	**23.69±0.68**	**19.72±1.45**	**23.92±1.16**
2	$c_1 = 0; c_2 = 2$	26.87±0.43	22.42±1.32	27.13±0.55
3	$c_1 = 2; c_2 = 0$	24.99±0.41	21.73±1.42	25.59±0.66

From this table we can see that the best performance is obtained by assigning equal weights to both factors. The difference in performance is statistically-significant over all measures with respect to the other two configurations. Therefore, by using the first configuration we can obtain solutions of better performance through and at the end of the search. More importantly, solutions of better generalization performance can be obtained with this configuration as well. The difference in performance is higher with the second configuration, where the individual-best solutions have no influence in the search; therefore, *PSMS* is searching locally around the global best solution. In the third configuration the global-best solution has no influence in the search; in consequence, the search is guided according the $m-$individual-best solutions. For illustration in Figures 14.4 and 14.5 we show the performance of *PSMS* as a function of the number of iterations for a single replication of the *Heart* data set. In Figure 14.4 we show the performance of *PSMS* for $I = 25$ iterations and in Figure 14.5 for $I = 100$ iterations.

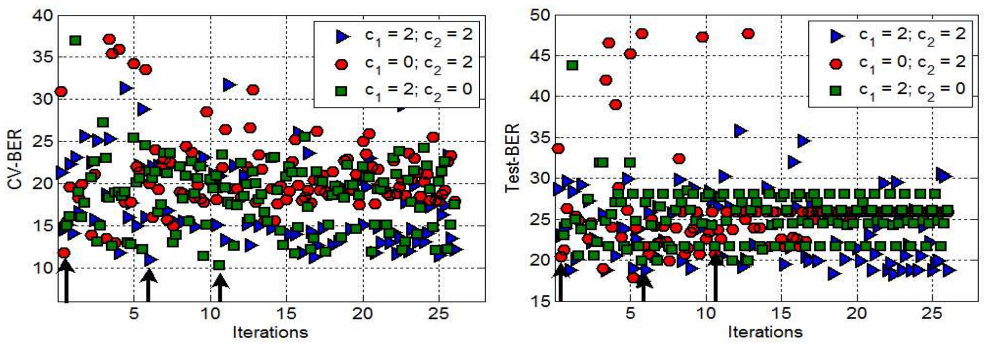


Figure 14.4: Performance of *PSMS* as a function of number of iterations using different settings for c_1 and c_2, see Table 14.7. We show the *CV-BER* (left) and *Test-BER* (right) for a single replication of the *Heart* data set. *PSMS* was ran for $I = 25$ iterations in this experiment. The models selected with each configuration are indicated with arrows.

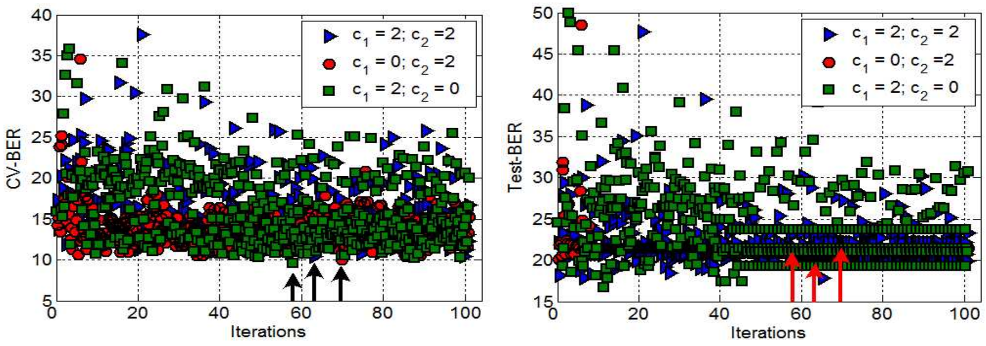


Figure 14.5: Performance of *PSMS* as a function of number of iterations using different settings for c_1 and c_2, see Table 14.7. We show the *CV-BER* (left) and *Test-BER* (right) for a single replication of the *Heart* data set. *PSMS* was ran for $I = 100$ iterations in this experiment. The models selected with each configuration are indicated with arrows.

From these figures we can see that the *CV* estimate is very similar for the different settings we considered (left plots in Figures 14.4 and 14.5). However, by looking at the performance of the solutions in the test set (right plots in Figures 14.4 and 14.5), we can appreciate that configurations 2 and 3 overfit the data (red circles and green squares). With $c_1 = 0$ we have that the $m = 5$ particles converge to single local minima, performing a fine grained search over this solution (red circles). With $c_2 = 0$ each of the $m-$particles converge to different local minima, overdoing the search over each of the m solutions (green squares). On the other hand, with the configuration $c_1 = 2$, $c_2 = 2$ *PSMS* is not trapped into a local minimum (blue triangles); searching around promising solutions, but without doing a fine grained search over any of them.

Better models (indicated by arrows) are selected by *PSMS* with the first configuration, even when their *CV* is higher than that of the models selected with the other configurations. This result confirms that *PSMS* is overfitting the data with the configurations 2 and 3. Note that with $I = 25$ iterations (Figure 14.4) the first configuration is not converging to a local minimum yet; while with $I = 100$ iterations (Figure 14.5) it looks like *PSMS* starts searching locally at the last iterations. This result illustrates why early stopping can be useful for obtaining better models with *PSMS*.

In order to better appreciate the generalization performance for the different configurations, in Figure 14.6 we plot the *CV-BER* as a function of *test-BER* for the run of *PSMS* with $I = 25$, we plot each particle with a different color.

From this figure we can see that the best model is obtained with the first configuration; for the configurations 2 and 3 the particles obtain the same *test-BER* for different solutions (middle and right plots in Figure 14.6). Despite the *CV* estimate is being minimized for these configurations, the *test-BER* performance of models does not improve. It is clear from the right plot in Figure 14.6 that with $c_2 = 0$ each particle is trapped in different local minima, doing a fine grained search over them that causes *PSMS* to overfit the data. It also can be seen from the middle plot that with $c_1 = 0$ the search is biased towards a single global-best solution (magenta circle), again, causing *PSMS* to overfit the data. On the other hand, results with the first configuration (left plot in Figure 14.6) show that particles do not oversearch at any solution.

14.4.3. Results on the model selection challenge

In this section we describe experimental results of *PSMS* in the framework of a model selection competition called *agnostic learning vs. prior knowledge challenge* (*ALvsPK*) (Guyon et al., 2007, 2008). The goal of these experiments is to compare the performance of *PSMS* against other model selection strategies that work for a single algorithm or that use domain knowledge for this task. Through its different stages, the *ALvsPK* competition evaluated novel strategies for model selection as well as the added value of using prior knowledge for improving classification accuracy (Guyon et al., 2008). This sort of competitions are very useful because through them the real effectiveness of methods can be evaluated; motivating further research in the field and collaborations among participants.

CHALLENGE PROTOCOL AND CLOP

The rules of the challenge were quite simple, the organizers provided five data sets for binary classification together with the *CLOP* toolbox (Saffari and Guyon, 2006). The task was to obtain the model with the lowest *BER* over the five data sets on unseen data. Participants were free to elect using *CLOP* or their own learning machine implementations. The challenge is over now, although the challenge website[7] still remains open, allowing the evaluation of

7. http://www.agnostic.inf.ethz.ch/

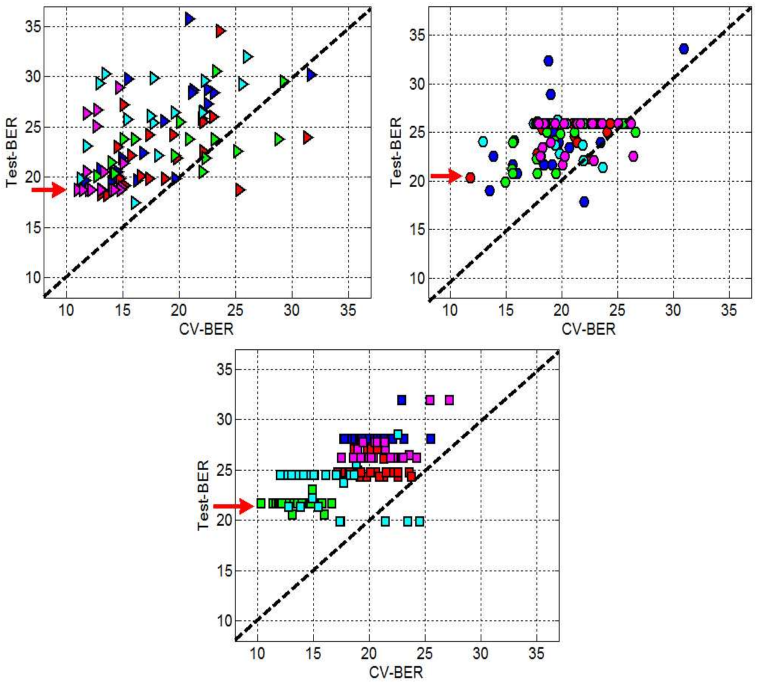


Figure 14.6: *Test-BER* as a function of *CV-BER* for a run of *PSMS* for $I = 25$ iterations in the *Heart* data set. Results with different configurations for c_1 and c_2 are shown. Left: $c_1 = 2$, $c_2 = 2$. Middle: $c_1 = 0$, $c_2 = 2$. Right: $c_1 = 2$, $c_2 = 0$. In each plot each particle is shown with a different color. The selected model with each configuration is indicated with an arrow.

learning techniques and model selection methods. A complete description of the challenge and a comprehensive analysis of the results are described by Guyon et al. (Guyon et al., 2006c, 2007, 2008).

The competition was divided into two stages. The first stage, called *the model selection game* (Guyon et al., 2006c), was focused on the evaluation of pure model selection strategies. In the second stage, the goal was to evaluate the gain we can have by introducing prior knowledge into the model selection process (Guyon et al., 2007, 2008). In the latter stage participants could introduce knowledge of the data domain into the model selection process (*prior knowledge track*). Also, participants could use agnostic methods in which no domain knowledge is considered in the selection process (*agnostic track*).

The data sets used in the agnostic track of the *ALvsPK* challenge are described in Table 14.8, these data sets come from real domains. Data sets used for the agnostic and prior knowledge tracks were different. For the agnostic track the data were preprocessed and dummy features were introduced, while for the prior knowledge track raw data were used, together with a description of the domain. We should emphasize that, although all of the approaches evaluated in the *ALvsPK* competition faced the same problem (that of choosing a model that obtains the lowest classification error for the data), such methods did not adopt the *FMS* interpretation. Most of the proposed approaches focused on a fixed machine learning technique like tree-based classifiers (Lutz, 2006), or kernel-based methods (Cawley, 2006; Pranckeviciene et al., 2007; Guyon et al., 2008), and did not take into account feature selection methods. Participants in the prior knowledge track could introduce domain knowledge. Furthermore, most of participants used their own implementations, instead of the *CLOP* toolbox.

After the challenge, the CLOP toolkit was augmented with methods, which performed well in the challenge (Cawley, 2006; Cawley and Talbot, 2007a; Lutz, 2006). These include Logitboost (Friedman et al., 2000), LSSVM (Suykens and Vandewalle, 1999), and kernel ridge regression (Saunders et al., 1998; Hastie et al., 2001).

COMPETITIVENESS OF PSMS

In both stages of the competition we evaluated models obtained with *PSMS* under different settings. Models obtained by *PSMS* were ranked high in the participants list, showing the competitiveness of *PSMS* for model selection (Guyon et al., 2006c, 2007, 2008; Escalante et al., 2007). Furthermore, the difference with methods that used prior knowledge was relatively small, showing that *FMS* can be a viable solution for the model selection problem without the need of investing time in introducing domain knowledge, and by considering a wide variety of methods.

The results of *PSMS* in the *ALvsPK* challenge have been partially analyzed and discussed elsewhere (Escalante et al., 2007; Guyon et al., 2007, 2008). During the challenge, our best entry (called *Corrida-final*) was ranked[8] 8th over all ranked participants, 5th among the methods that did not use domain knowledge and 2nd among the methods that used the software provided by the organizers (Guyon et al., 2006c, 2007, 2008). For *Corrida-final* we used $k = 5$ and the full training set for computing the fitness function; we ran *PSMS* for 500 iterations for the *Ada* data set and 100 iterations for *Hiva*, *Gina* and *Sylva*. We did not applied *PSMS* to the *Nova* data set in that entry, instead we selected a model for *Nova* by trial and error. For such entry we used a version of *CLOP* where only there were available the following classifiers *zarbi, naive, neural* and *svc* (Escalante et al., 2007); also, only four feature selection methods were considered.

8. `http://www.clopinet.com/isabelle/Projects/agnostic/Results.html`

Post-challenge experiments

In the rest of this section we present results of *PSMS* using the augmented toolkit, including all methods described in Tables 14.1 and 14.2. In these tables we consider implementations of *logitboost, lssvm and gkridge,* which are the classifiers that won the *ALvsPK* challenge (Cawley, 2006; Cawley and Talbot, 2007a; Lutz, 2006) and were added to CLOP after the end of the challenge.

Table 14.8: Benchmark data sets used for the model selection challenges (Guyon et al., 2006c, 2007, 2008).

Data set	Domain	Type	Features	Training	Validation	Testing
Ada	Marketing	Dense	48	4174	415	41471
Gina	Digits	Dense	970	3153	315	31532
Hiva	Drug discovery	Dense	1617	3845	384	38449
Nova	Text classification	Sparse binary	16969	1754	175	17537
Sylva	Ecology	Dense	216	13086	1309	130857

In order to efficiently apply *PSMS* to the challenge data sets we adopted a subsample strategy in which, instead of using the full training set, small subsamples of the training data were used to compute the fitness function. Each time the fitness function is computed we obtain a different random sample of size $S_{sub} = \frac{N}{SF}$, where N is the number of instances and SF is a constant that specifies the proportion of samples to be used. Subsamples are only used for the search process. At the end of the search the selected model is trained using the full training set (for the experiments reported in this paper we considered as training set the union of the training and validation data sets, see Table 14.8). Due to the dimensionality of the *Nova* data set we applied principal component analysis to this data set. Then we used the first 400 components for applying *PSMS*. We fixed $k = 2$, $I = 50$ and $m = 5$ for our experiments based on the results from previous sections. Then we ran *PSMS* for all of the data sets under the above described settings using different values for *SF*. The predictions of the resultant models were uploaded to the challenge website in order to evaluate them. Our best ranked entry in the *ALvsPK* challenge website (called *psmsx_jmlr_run_I*) is described in Table 14.9, and a comparison of it with the currently best-ranked entries is shown in Table 14.10.

Table 14.9: Models selected with *PSMS* for the data sets of the *ALvsPK* challenge. For each data set we show the subsampling factor used (**SF**), the selected model (**Model**, some hyperparameters are omitted for clarity), the processing **Time** in minutes and the **test-BER** obtained. *sns* is for *shift-scale*, *std* is for *standardize* and *norm* is for *normalize*. See Tables 14.1 and 14.2 for a description of methods and their hyperparameters.

Data	SF	Model	Time (m)	Test-BER
Ada	1	chain({logitboost(units=469,shrinkage=0.4,depth=1),bias}	368.12	16.86
Gina	2	chain({sns(1),relief(fmax=487),gkridge,bias}	482.23	2.41
Hiva	3	chain({norm(1),rffs(fmax=1001),lssvm(gamma=0.096),bias}	124.54	28.01
Nova	1	chain({rffs(fmax=338),norm(1),std(1),sns(1),gkridge,bias}	82.12	5.27
Sylva	10	chain({sns(1),odds-ratio(fmax=60),gkridge,bias}	787.58	0.62

We can see from Table 14.9 that very different models were selected by *PSMS* for each data set. This is the main advantage the *FMS* because it allows us selecting an ad-hoc model for each data set by considering different model types and a wide diversity of methods. With exception of *Ada*, the selected models included a feature selection method; this result shows the importance of feature selection methods and that some of them are more compatible than others with some classifiers; note that different numbers of features were selected for each data set. In all but the *Nova* data set preprocessing was performed before feature selection. This can be due to the fact that for *Nova* we used principal components instead of the original features. For *Ada* it was selected a *logitboost* classifier, while for *Hiva* it was selected a *lssvm* classifier with gaussian kernel. For *Gina, Nova* and *Sylva* it was selected the *gkridge* classifier; this classifier performs virtual leave-one out model selection each time it is trained (Cawley et al., 2007). Note that both *gkridge* and *logitboost* were the classifiers that best performed during the challenge (Guyon et al., 2007, 2008); this result gives evidence that *PSMS* can obtain similar and even better models, without spending time on ad-hoc modifications for each data set and without using domain knowledge.

The use of the subsampling strategy allowed to efficiently apply *PSMS* to all of the data sets. About six hours were required to obtain a competitive model for *Ada*, while about only two for the *Hiva* data set. Note that applying *PSMS* for *Hiva* using the entire training set (*Corrida-final*) took about 80 hours in the same machine (Escalante et al., 2007). During our experiments we found that the larger the subsamples the better the performance of the selected model. However, the selection of *SF* also depends on the available computer resources and time restrictions. One should note that we can increase speed of *PSMS* by caching intermediate results that can be reused (e.g. preprocessing the data sets off-line).

Despite the use of the subsampling technique the models selected with *PSMS* resulted very competitive as shown in Table 14.10. Currently, the *PSMS* run is the top-ranked agnostic entry in the challenge website; furthermore, the performance of this entry is superior than all but one prior-knowledge entry: *Interim-all-prior*; which is the best ranked entry overall, using "prior knowledge" or "domain knowledge". Note that the latter entry is formed of models that were designed ad-hoc for each data set, requiring much more time, effort and knowledge than *PSMS*. In average *PSMS* outperforms the best results presented in the *ALvsPK* challenge: *IJCNN07AL*, row 4 in Table 14.10 (Guyon et al., 2007); and the best-ranked agnostic entry (after *PSMS*): *Logitboost-with-trees*, row 5 in Table 14.10 (Lutz, 2006).

The performance of *PSMS* is very close to that of *IJCNN07AL*, tieing in *Sylva* and outperforming one to the other in two data sets. *PSMS* outperforms *Logitboost-with-trees* in three out of the five data sets and achieves very close performance for the other two data sets. The latter entry is ranked 10th in the challenge website. It is very interesting that for the *Ada* data set the best model so far is a *logitboost* classifier with about 1000 trees (Lutz, 2006); while with *PSMS* we were able to achieve almost the same performance by using a half that number of trees, see row 2 in Table 14.9. This is important because simpler models of similar performance can be obtained with *PSMS*.

PSMS clearly outperformed our best-ranked entry during the challenge (row 6 in Table 14.10); this result gives evidence that we obtained better results by using more and better classifiers; also, the use of subsamples instead of the entire training set (when computing the fitness function), does not damage the performance of *PSMS*, although the reduction in processing time is very important. Note that for *Nova* the *PSMS* entry obtained a slightly worse result than that of *Corrida-final*; however, the model for *Nova* in *Corrida-final* was selected by trial and error which required of much more effort and time.

Results reported in this section show the efficacy of *PSMS* for model selection. Despite its simplicity it has shown comparable and even superior performance to those obtained by

Table 14.10: Comparison of models selected with *PSMS* and the best entries in the *ALvsPK* challenge data sets. We show, for reference, the best prior-knowledge entry (*Interim-all-prior*); the entry formed by the models described in Table 14.9 (*psmx_jmlr_run_I*); the best individual entries for each data set in the *ALvsPK* challenge (*IJCNN07AL*) (Guyon et al., 2007, 2008); the second-best entry of the agnostic track (*Logitboost-with-trees*) and our best ranked entry evaluated during the challenge *Corrida-final*. The best results in the agnostic track are shown in **bold**.

Entry	Description	Ada	Gina	Hiva	Nova	Sylva	Overall	Rank
Interim-all-prior	Best-PK	17.0	2.33	27.1	4.71	0.59	10.35	1th
psmsx_jmlr_run_I	PSMS	16.86	**2.41**	**28.01**	5.27	**0.62**	10.63	2nd
IJCNN07AL	Best-AL	**16.60**	3.39	28.27	**4.56**	**0.62**	10.68	4th
Logitboost-with-trees	Best-AL	**16.60**	3.53	30.18	4.69	0.78	11.15	10th
Corrida-final	Best-PSMS-ALvsPK	18.27	6.14	28.54	5.11	1.22	11.86	42th

other model selection strategies that focused on a single learning algorithm and to methods that used prior domain knowledge for guiding the model selection process (Lutz, 2006; Reunanen, 2007; Boullé, 2007; Pranckeviciene et al., 2007; Wichard, 2007). Models selected with *PSMS* are simple and yet very competitive; furthermore, with *PSMS* no knowledge is needed on the methods to choose from, nor on the domain. In consequence, it is very easy to obtain classifiers that can achieve state-of-the-art performance without spending time on designing, developing and optimizing an ad-hoc model. Even though *PSMS* can be applied to any binary classification problem, we are not claiming it will obtain satisfactory results in every domain; however, it can be considered as a first option when dealing with binary classification tasks. It is expected that this will further improve the performance of models selected with *PSMS* if domain knowledge is used.

14.5. Discussion

In this section, we discuss the advantages and disadvantages of *PSMS* and perform a synthesis of our experiments aiming at better understanding how *PSMS* performs intensive search in hyperparameter space without overfitting the data.

14.5.1. Robust and computationally tractable intensive search

In Section 14.4 we reported experimental results that give evidence of the validity of the *PSMS* approach, demonstrating in particular that *PSMS* can outperform *PATSMS* through and at the end of the search, showing better convergence behavior and generalization performance. This is obtained at the expense of moderate additional computational complexity. This claim is supported by the theoretical analysis of the computational complexity (Section 14.3.4), indicating that computations are dominated by the number of learning machine trainings, and by the experiments (Section 14.4.2), indicating as few as 5 particles (learning machines) and 10 iterations (*i*.e., 50 trainings) are needed to attain the best performance. The efficiency of *PSMS* can be improved, for instance by preferably exploring learning machines, which have a lower computational cost of training. We explored successfully other heuristics, including subsampling training data, which reduced computations, at the expense of no performance degradation.

An analysis of the diversity of models tried by *PSMS* shows that this method is not biased towards models that tend to perform well individually. Investigating the operation of *PSMS* under different settings we found that the performance of *PSMS* is not significantly affected by modifying its hyperparameters. However, experimental results indicate that the use of an adaptive inertia weight may be helpful to explore the search space better.We also observed that certain parameter configurations allow the selection of competitive models, while reducing processing time. Section 14.5.4 provides a final set of practical recommendations.

Results of international competitions suggest that *PSMS* is competitive with model selection strategies specific to single algorithms and to strategies using prior domain knowledge. The latter is an important result because it shows that we can obtain competitive models without the need of an expert on the domain, a careful analysis to the data, or even machine learning knowledge.

14.5.2. Intensive search without overfitting

The findings summarized above provide empirical evidence suggesting *PSMS* is a reliable strategy for agnostic model selection. However, it is not obvious why *PSMS* succeeds in selecting competitive models without significantly overfitting data. Our hypothesis is that *PSMS* is able to avoid overfitting because of the way the search is guided: *PSMS* performs a broad search around promising solutions without focusing on reaching local minima. Solutions in *PSMS* are updated by taking into account information from both: the global-best solution ($\mathbf{p}_g$, weighted by c_2) and the individual-best solutions of each particle ($\mathbf{p}_{1,\ldots,m}$, weighted by c_1), see Equations (14.1) and (14.2)). The latter combined with an adaptive inertia weight and early stopping cause do not exaggerate the search at any local minima. Methods like *PATSMS*, on the contrary, update solutions by moving the pattern towards the local minimum nearest the initial solution, see Algorithm 14.2. Reaching exactly a local minimum causes *PATSMS* to learn peculiarities in the data set and does not necessarily result in obtaining a better predictive model.

The experiments we performed in Section 14.4.2 support the above conjecture. The results from Table 14.7 and Figures 14.4, 14.5 and 14.6 indicate that *PSMS* is able to avoid overfitting (to some extend) because of the way the search is guided. *PSMS* searches around good solutions without overdoing in terms of really fine-grained optimization. This sort of search can be considered as suboptimal in Dietterich's sense: *"In machine learning it is optimal to be suboptimal!"* (Dietterich, 1995). The latter statement makes reference to the well known fact that oversearching (i.e. trying to be optimal) in model selection can lead to select models that fit very well the peculiarities of the considered data set without deriving a general predictive rule (Dietterich, 1995; Jensen and Cohen, 2000; Quinlan and Cameron-Jones, 1995; Hastie et al., 2001; Loughrey and Cunningham, 2005). On the contrary, *PSMS* is able to *undercompute* because for updating solutions it considers local and global knowledge, information from past solutions (weighted by the inertia term) and randomness; note that the latter holds when a reasonable small number of iterations is performed. Furthermore, our experimental results provide empirical evidence that agrees with recent, yet traditional, explanations about why and how *PSO* works (Kennedy, 2008; Kennedy and Eberhart, 2001). Kennedy has argued the success of *PSO* is due to the fact that it performs a *"collaborative trial and error"* search (Kennedy, 2008). That is, *PSO* obtains good results mainly because the search is directed according both individual and social knowledge; the same conclusion derived from experimental results in this section. It is not surprising that distributed and collaborative computing can improve results of centralized (individualized) methods. For *FMS*, however, it is very interesting that updating solutions in a heterogeneous way allows us to avoid oversearching and, in consequence, overfitting; even when the *FMS* search space is infinite and has many local minima solutions.

14.5.3. Comparison with related work

A variety of approaches have been proposed to select parameters of specific methods for regression, classification, feature selection, discretization etcetera (Hastie et al., 2001; Bishop, 2006; Voss and Feng, 2002; Nelles, 2001; Guyon et al., 2006b; Kim et al., 2002; Hastie et al., 2001; Cawley and Talbot, 2007b; Boullé, 2007; Hue and Boullé, 2007). However, despite the potential advantages of *FMS* (namely generality, simplicity and competitiveness), this problem has been little studied because of the huge search space involved and because intensive search methods are prone to overfit the data (Gorissen et al., 2008; Escalante et al., 2007). Nevertheless, in the rest of this section we outline techniques used to avoid oversearching/overfitting in search that are applicable/related to *FMS*. One should note that traditional model selection techniques just like Akaike and Bayesian information criteria, the minimum description length principle and the *VC*-dimension are not directly applicable to *FMS* and therefore they are excluded of analysis.

Grid search (with *CV*) is the widely used search-approach to model selection in practical applications (Momma and Bennett, 2002; Hsu et al., 2003). This method consists of defining a uniform grid over the search space where each point in the grid defines a solution; every point in the grid is evaluated and the point of lowest error is selected. The granularity of the grid determines both the performance of the selected solution and the efficiency of the search. A fine-grained grid may be inefficient and can lead to oversearching; while a sparse grid will result in low performance models. Note that the heterogeneousness of models and the variety of ranges for the models parameters make very difficult the application of grid search to the *FMS* problem; furthermore, the choice of an adequate granularity can be a serious problem. Other methods already used for parameter optimization that can be applied to *FMS* include: greedy search (Dietterich, 1995), random search (e.g. the bumping technique) (Hastie et al., 2001), *PS* (Bi et al., 2003; Momma and Bennett, 2002), evolutionary computation approaches (Engelbrecht, 2006; Gorissen et al., 2008; Angeline, 1998), and other swarm-optimization techniques (Kennedy and Eberhart, 2001; Engelbrecht, 2006). Note that despite one may think that exhaustive search is the best search option in model selection, this approach is impractical for most real world problems and when applicable it suffers from the oversearching phenomenon (Dietterich, 1995).

Early stopping has been widely used to prevent overfitting in search methods (Hastie et al., 2001; Engelbrecht, 2006; Loughrey and Cunningham, 2005). The goal of this heuristic is to stop searching/learning when the model starts overfitting the data. There are several variants to stop the search: after a small number of iterations, when no improvement is found after a number of iterations, when a solution of acceptable performance has been found, etcetera. A problem with early stopping is that premature stopping the algorithm would lead to selecting a solution that has not converged yet, while a late stopping of the search will cause severely overfitting the data because of oversearching. For *PSMS* we found that a small number of iterations can be enough to obtain satisfactory results in benchmark data, although the number of iterations is problem dependant; therefore, we can adopt other stopping criteria for *FMS* in the future.

Randomness has bring into play in machine learning in order avoid overfitting and to escape from local minima in search (Hastie et al., 2001; Bishop, 2006; Kirkpatrick et al., 1983; Kennedy and Eberhart, 2001). In learning algorithms, it has been successfully used to prevent overfitting and to obtain better predictors; learning methods that use randomness include bagging classifiers (Hastie et al., 2001), neural and deep belief networks (Hastie et al., 2001; Hinton et al., 2006) and randomized decision-tree algorithms (Breiman, 2001; Geurts et al., 2006). Bootstrapping is a technique (used in bagging and random forest classifiers) based on random sampling that has been widely used to estimate the generalization performance of methods as an alternative to *CV* (Hastie et al., 2001). In *PSMS* randomness played an important role because it introduces diversity into the search process and allows *PSMS* to avoid local minima. Furthermore, the subsampling strategy we used to increase the speed of *PSMS* is related to bootstrapping; in future work on *PSMS* we will explicitly consider different subsampling estimations for the selection of the final model.

As the adaptive inertia weight in *PSO*, see Section 14.2, there are parameters in other algorithms that aim to avoid overfitting by exploring the search space both globally and locally; examples are the temperature parameter in simulated annealing (Kirkpatrick et al., 1983) and the momentum term in on-line gradient descend and backpropagation (Qian, 1999). The ridge in ridge-regression and weight decay in neural networks training are also related to the inertia weight. Model averaging and the use of ensembles have proved to be helpful to improve predictions and avoid overfitting; this is because different models have different biases that in average result in improved performance (Hastie et al., 2001; Bishop, 2006). Future work includes in *PSMS* consists of combining particles in order to improve the performance of the swarm strategy. Finally, adding noise to the training data is another overfitting avoidance mechanism in model selection that also can be used with *PSMS*.

14.5.4. A practical guide to PSMS

In this section we describe the way *PSMS* can be put in practice in any binary classification problem. Due to the simplicity and generality of the approach below we describe a practical guide to use the *Matlab*® implementation of *PSMS* (included in the *CLOP* toolbox). It is assumed that the user has available a data set (in *Matlab*® format) with N samples for binary classification: a matrix $\mathbf{X}_{N\times d}$ contains the N training samples of dimensionality d and a vector $\mathbf{Y}_{N\times 1}$ their respective labels ($y_i \in [-1,1]$). After downloading and installing *CLOP* (Saffari and Guyon, 2006), *PSMS* can be applied to any data set by typing the following *Matlab*® code:

```
%% load your data into the Matlab® workspace
1: >> load train_data.mat;
%%  Create A Clop Data-Object
2: >> D = data(X,Y);
%% Create A CLOP PSMS-Object with default parameters
3: >> P = psmsx;
%% Perform PSMS
4: >> [Dat, Res] = train(P, D);
%% Train the selected model with the full training set
5: >> [Odat, TrM] = train(Res.Best_Model,D);
%% Create a test data set, note that Y_t can be empty
6: >> load test_data; Dt = data(X_t,Y_t);
%% Test the selected and trained model on unseen data Dt
7: >> [Pred] = test(TrM, Dt);
%% Estimate the model's performance on unseen data Dt, if Y_t is available
8: >> [BER] = balanced_errate(Pred.X, Pred.Y);
%% Analyze the ROC performance of the selected model
9: >> roc(Pred);
```

Note that steps 1–2 and 5–9 are associated with loading the data and the evaluation of the selected model, respectively; which are operations not attained to *PSMS*. Steps 3 and 4 will create the *PSMS* object and will start the search, respectively. Besides the selected model, the output of the search (**Res**, line 4), is a structure with useful information about the search process, see the *PSMS* documentation (Escalante, In preparation, 2009).

In Section 14.4.2 were presented experimental results that suggest that there is not significant difference in performance by modifying most of the *PSMS* hyperparameters. Therefore, one can choose parameter settings for *PSMS* that make practical its application without a significant decrement of performance. Below are shown recommended values for the *PSMS* hyperparameters. These parameters and other options of the current implementation can be modified very simply (Escalante, In preparation, 2009).

Recommended PSMS parameters:

Weight for individual best solution	$c_1 = 2$
Weight for global best solution	$c_2 = 2$
Adaptive inertia weight	$\mathbf{W} = (1.2, 0.5, 0.4)$
Number of iterations	$I = 50$
Swarm size	$m = 5$
Folds in *CV*	$k = 2$
Subsampling factor	$SF = 1$ (as small as possible in large data sets)

14.6. Conclusions

In this paper we proposed Particle Swarm Model Selection (*PSMS*), that is, the application of Particle Swarm Optimization (*PSO*) to the problem of Full Model Selection (*FMS*). Given a data set, the *FMS* problem consists of selecting the combination of preprocessing, feature selection and learning methods that obtains the lowest classification error. *FMS* also includes hyperparameter optimization for the selected methods. This approach to the model selection problem has the following advantages. First, the generality of the approach allows us to consider different model types (for preprocessing, feature selection and learning) and a variety of methods. Second, *PSMS* can be applied to any data set, since neither domain knowledge nor machine learning knowledge is required, therefore it can be considered a black-box model selection method. Third, and most importantly, competitive and yet simple models can be obtained with *PSMS*. We provide empirical evidence that shows that *PSMS* can be applied efficiently, without a significant loss of accuracy, by using a subsampling heuristic and parameter settings that reduce the computational cost.

The simplicity of *PSO* and its proven performance, comparable to that of evolutionary algorithms, make this search algorithm well suited for *FMS*. However, the application of any other stochastic optimization strategy is also possible. The main advantage of *PSO* is that a single equation for updating solutions is needed, as opposed to evolutionary algorithms where methods for representation, mutation, cross-over, speciation and selection have to be considered. Interestingly, the way the search is guided in *PSMS* allows it obtaining competitive models without significantly overfitting. Experimental results in benchmark data show superior performance of *PSMS* when compared to Pattern Search Model Selection (*PATSMS*), a direct search method that constitutes a competitive baseline.

Results obtained by models selected with *PSMS* in the framework of a model selection challenge show that it is a very competitive model selection method, despite its simplicity and generality. In such competitions, models selected with *PSMS* were always among the top ranking models, together with methods performing solely hyperparameter selection in a given model family and methods relying on prior knowledge. This demonstrates that, via the use of *PSO*, *FMS* is a viable strategy for model selection. This is remarkable because we noted in previous competitions (Guyon et al., 2005; Guyon et al., 2006b) that each data set had a different best performing method, yet researchers performing *FMS* (in an effort to find the model family best suited to a given problem) were not successful. The participants, which obtained the best results on average over all data sets restricted themselves to hyperparameter selection in one given model family. In contrast, in this paper we demonstrated the viability of *FMS* using the *PSO* search strategy. Our work paves the way to the use of intensive search techniques to perform *FMS* in the entire model space of machine learning toolkits. With the increasing availability of diverse and sophisticated machine learning toolkits and the improvements in computing power, we foresee that *FMS* will become an effective methodology.

Current work includes the use of *PSMS* for the selection of model-members for ensembles and the hierarchical application of *PSO* for *FMS* and hyperparameter optimization. *PSMS* is currently being applied to different tasks, including galaxy classification, automatic image annotation, object recognition and text classification. Future work includes the introduction of a penalty-term into the fitness function; such that (computationally) inexpensive models be favored by PSMS. The extension of *PSMS* to the multi-class classification and regression problems is another future work direction.

Acknowledgments

We would like to thank the organizers and participants of the NIPS multi-level inference workshop and model selection game, and of the *IJCNN ALvsPK* challenge. The first author thanks the UNIPEN foundation and the US National Science Foundation for travel support. We also thank editors and anonymous reviewers for their useful comments that have help us to improve this paper. We thank Dr. Eduardo Morales for his useful suggestions about the content of the paper. Last, but not least, we thank Dr. Aurelio López and INAOE for the provided support.

References

P. J. Angeline. Evolutionary optimization vs particle swarm optimization: Philosophy and performance differences. In *Proceedings of the 7th Conference on Evolutionary Programming*, volume 1447 of *LNCS*, pages 601–610, San Diego, CA, March 1998. Springer.

Y. Bengio and N. Chapados. Extensions to metric-based model selection. *Journal of Machine Learning Research*, 3:1209–1227, 2003.

J. Bi, M. Embrechts K. P. Bennett, C. M. Breneman, and M. Song. Dimensionality reduction via sparse support vector machines. *Journal of Machine Learning Research*, Mar(3):1229–1243, Mar 2003.

C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

M. Boullé. Report on preliminary experiments with data grid models in the agnostic learning vs prior knowledge challgenge. In *Proceedings of the 20th International Joint Conference on Neural Networks*, pages 1802–1808, 2007.

L. Breiman. Random forest. *Machine Learning*, 45(1):5–32, 2001.

G. Cawley. Leave-one-out cross-validation based model selection criteria for weighted LS-SVMs. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2006)*, pages 2970–2977, Vancouver, Canada, July 2006.

G. Cawley and N. L. C. Talbot. Agnostic learning vs prior knowledge in the design of kernel machines. In *Proceedings of the 20th International Joint Conference on Neural Networks*, pages 1444–1450, Orlando, Florida, 2007a.

G. Cawley, G. Janacek, and N. L. C. Talbot. Generalised kernel machines. In *Proceedings of the 20th International Joint Conference on Neural Networks*, pages 1439–1445, Orlando, Florida, 2007.

G. C. Cawley and N. L. C. Talbot. Preventing over-fitting during model selection via bayesian regularisation of the hyper-parameters. *Journal of Machine Learning Research*, 8:841–861, April 2007b.

M. Clerc and J. Kennedy. The particle swarm: Explosion, stability and convergenge in a multi-dimensional complex space. *IEEE Transactions on on Evolutionary Computation*, 6(1):58–73, February 2002.

J. Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, January 2006.

J. E. Dennis and V. J. Torczon. Derivative-free pattern search methods for multidisciplinary design problems. In *Proceedings of the AIAA / USAF / NASA / ISSMO Symposium on Multi-disciplinary Analysis and Optimizatino*, pages 922–932, 1994.

T. Dietterich. Overfitting and undercomputing in machine learning. *ACM Comput. Surv.*, 27(3): 326–327, 1995. ISSN 0360-0300.

A. P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. Wiley, 2006.

H. J. Escalante. Particle swarm optimization for classifier selection: A practical guide to PSMS. http://ccc.inaoep.mx/~hugojair/psms/psms_doc.pdf, In preparation, 2009.

H. J. Escalante, M. Montes, and E. Sucar. PSMS for neural networks on the IJCNN 2007 agnostic vs prior knowledge challenge. In *Proceedings of the 20th International Joint Conference on Neural Networks*, pages 1191–1197, Orlando, FL, USA., 2007.

V. Franc and V. Hlavac. The statistical pattern recognition toolbox. http://cmp.felk.cvut.cz/cmp/software/stprtool/index.html, 2004.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28(2):337–407, 2000.

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1): 3–42, 2006. ISSN 0885-6125.

T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, 286:531–537, October 1999.

Dirk Gorissen. Heterogeneous evolution of surrogate models. Master's thesis, Katholieke Universiteit Leuven, Belgium, June 2007.

Dirk Gorissen, Luciano De Tommasi, Jeroen Croon, and Tom Dhaene. Automatic model type selection with heterogeneous evolution: An application to RF circuit block modeling. In *IEEE Proceedings of WCCI 2008*, pages 989–996, 2008.

V.G. Gudise and G.K. Venayagamoorthy. Comparison of particle swarm optimization and back-propagation as training algorithms for neural networks. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium, 2003. (SIS03)*, pages 110–117, 2003.

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3(Mar):1157–1182, 2003.

I. Guyon, S. Gunn, A. Ben-Hur, and G. Dror. Result analysis of the NIPS 2003 feature selection challenge. In *Advances in Neural Information Processing Systems 17*, pages 545–552. MIT Press, Cambridge, MA, 2005.

I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, editors. *Feature Extraction, Foundations and Applications*. Series Studies in Fuzziness and Soft Computing. Springer, 2006a.

I. Guyon, A. Saffari, G. Dror, and J. M. Buhmann. Performance prediction challenge. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2006)*, pages 2958–2965, Vancouver, Canada, July 2006b.

I. Guyon, A. Saffari, G. Dror, G. Cawley, and O. Guyon. Benchmark datasets and game result summary. In *NIPS Workshop on Multi-level Inference and the Model Selection Game*, Whistler, Canada, December 2006c.

I. Guyon, A. Saffari, G. Dror, and G. Cawley. Agnostic learning vs prior knowledge challenge. In *Proceedings of the 20th International Joint Conference on Neural Networks*, pages 1232–1238, Orlando, Florida, 2007.

I. Guyon, A. Saffari, G. Dror, and Gavin Cawley. Analysis of the IJCNN 2007 competition agnostic learning vs. prior knowledge. *Neural Networks*, 21(2–3):544–550, 2008.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Verlag, New York, 2001.

E. Hernández, C. Coello, and A. Hernández. On the use of a population-based particle swarm optimizer to design combinational logic circuits. In *Evolvable Hardware*, pages 183–190, 2004.

G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006. ISSN 0899-7667.

C. W. Hsu, C. C. Chang, and C. J. Lin. A practical guide to support vector classification. Technical report, Taipei, 2003. URL http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf.

C. Hue and M. Boullé. A new probabilistic approach in rank regression with optimal bayesian partitioning. *Journal of Machine Learning Research*, 8:2727–2754, December 2007.

D. Jensen and P Cohen. Multiple comparisons in induction algorithms. *Machine Learning*, 38 (3):309–338, 2000. ISSN 0885-6125.

J. Kennedy. How it works: Collaborative trial and error. *International Journal of Computational Intelligence Research*, 4(2):71–78, 2008.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the International Conference on Neural Networks*, volume IV, pages 1942–1948. IEEE, 1995.

J. Kennedy and R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, 2001.

J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002)*, volume 2, pages 1671–1676, 2002.

Y. Kim, N. Street, and F. Menczer. Evolutionary model selection in unsupervised learning. *Intelligent Data Analysis*, 6:531–556, 2002.

S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220 (4598):671–680, 1983.

J. Loughrey and P. Cunningham. Overfitting in wrapper-based feature subset selection: The harder you try the worse it gets. In F. Coenen M. Bramer and T. Allen, editors, *Proceedings of AI-2004, the Twenty-fourth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, Research and Development in Intelligent Systems XXI, pages 33–43, 2005.

R. Lutz. Logitboost with trees applied to the WCCI 2006 performance prediction challenge datasets. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2006)*, pages 1657– 1660, Vancouver, Canada, July 2006.

S. Mika, G. Rätsch, J. Weston, B. Schölkopf, A. J. Smola, and K.-R. Müller. Invariant feature extraction and classification in kernel spaces. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 526–532, Cambridge, MA, 2000. MIT Press.

M. Momma and K. Bennett. A pattern search method for model selection of support vector regression. In *Proceedings of SIAM Conference on Data Mining*, 2002.

O. Nelles. *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models*. Springer, 2001.

E. Ozcan and C. K. Mohan. Analysis of a simple particle swarm optimization system. In *Intelligent Engineering Systems Through Artificial Neural Networks*, pages 253–258, 1998.

E. Pranckeviciene, R. Somorjai, and M. N. Tran. Feature/model selection by the linear programming SVM combined with state-of-art classifiers: What can we learn about the data. In *Proceedings of the 20th International Joint Conference on Neural Networks*, pages 1422–1428, 2007.

N. Qian. On the momentum term in gradient descent learning algorithms. *Neural Netw.*, 12(1): 145–151, 1999. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/S0893-6080(98)00116-6.

J. R. Quinlan and R. M. Cameron-Jones. Oversearching and layered search in empirical learning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1019–1024, 1995.

G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for adaboost. *Mach. Learn.*, 42(3):287–320, 2001. ISSN 0885-6125. doi: http://dx.doi.org/10.1023/A:1007618119488.

J. Reunanen. Model selection and assessment using cross-indexing. In *Proceedings of the 20th International Joint Conference on Neural Networks*, pages 1674–1679, 2007.

M. Reyes and C. Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International Journal of Computational Intelligence Research*, 3(2):287–308, 2006.

Y. Robinson, J. Rahmat-Samii. Particle swarm optimization in electromagnetics. *IEEE Transactions on Antennas and Propagation*, 52(2):397– 407, February 2004.

A. Saffari and I. Guyon. Quickstart guide for clop. Technical report, Graz University of Technology and Clopinet, May 2006. http://www.ymer.org/research/files/clop/QuickStartV1.0.pdf.

J. Salerno. Using the particle swarm optimization technique to train a recurrent neural model. In *Proceedings of the Ninth International Conference on Tools with Artificial Intelligence*, pages 45–49, 1997.

C. Saunders, A. Gammerman, and V. Vovk. Ridge regression learning algorithm in dual variables. In Jude W. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning*, pages 515–521, Madison, WI, USA, 1998.

Y. Shi and R. C. Eberhart. Parameter selection in particle swarm optimization. In *Evolutionary Programming VII*, pages 591–600, New York, 1998. Springer-Verlag.

Y. Shi and R. C. Eberhart. Emprirical study of particle swarm optimization. In *Proceedings of the Congress on Evolutionary Computation*, pages 1945–1949, Piscataway, NJ, USA, 1999. IEEE.

S. Sonnenburg. NIPS workshop on machine learning open source software. http://www2.fml.tuebingen.mpg.de/raetsch/workshops/MLOSS06/, December 2006.

J.A.K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(1):293–300, 1999.

F. van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, University of Pretoria, Sudafrica, November 2001.

F. van der Heijden, R. P.W. Duin, D. de Ridder, and D. M.J. Tax. PRTools: a Matlab based toolbox for pattern recognition. http://www.prtools.org/, 2004.

M. Voss and X. Feng. Arma model selection using particle swarm optimization and aic criteria. In *Proceedings of the 15th IFAC World Congress on Automatic Control*, 2002.

J. Weston, A. Elisseeff, G. BakIr, and F. Sinz. The spider machine learning toolbox. http://www.kyb.tuebingen.mpg.de/bs/people/spider/, 2005.

J. Wichard. Agnostic learning with ensembles of classifiers. In *Proceedings of the 20th International Joint Conference on Neural Networks*, pages 1753–1759, 2007.

J. Wichard and C. Merkwirth. ENTOOL – a Matlab toolbox for ensemble modeling. http://www.j-wichard.de/entool/, 2007.

I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 4:67–82, 1997.

H. Xiaohui, R. Eberhart, and Y. Shi. Engineering optimization with particle swarm. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium, 2003. (SIS03)*, pages 53–57, 2003.

H. Yoshida, K. Kawata, Y. Fukuyama, S. Takayama, and Y. Nakanishi. A particle swarm optimization for reactive power and voltage control considering voltage security assessment. *IEEE Transactions on Power Systems*, 15(4):1232–1239, Jan 2001.

Chapter 15

Bilevel Cross-validation-based Model Selection

Gautam Kunapuli[1] KUNAPG@RPI.EDU
Jong-Shi Pang[2] JSPANG@UIUC.EDU
Kristin P. Bennett[1] BENNEK@RPI.EDU
[1] *Department of Mathematical Sciences,*
Rensselaer Polytechnic Institute, 110 8th Street, Troy NY 12180
[2] *Department of Industrial and Enterprise Systems Engineering,*
University of Illinois, Urbana-Champaign, 104 S. Mathews Avenue, Urbana IL 61801

Editor: Isabelle Guyon, Gavin Cawley, Gideon Dror and Amir Saffari

Abstract

A key step in many statistical learning methods is solving a convex optimization problem containing one or more user-selected hyper-parameters. While cross validation is widely employed for selecting these parameters, its discretized grid-search implementation in parameter space effectively limits the number of hyper-parameters that can be handled, due to the combinatorial explosion of grid points. Instead, cross validation is formulated as a continuous bilevel program which is a mathematical program whose constraints are functions of optimal solutions of another mathematical program. The resultant bilevel cross validation problem is transformed to an equivalent mathematical program with equilibrium constraints (MPEC). Two approaches are considered to optimize the MPEC and provide for a systematic search of the hyper-parameters. In the first approach, the equilibrium constraints of the MPEC are relaxed to form a nonlinear program (NLP) with linear objective and non-convex quadratic inequality constraints, which is then solved using a general-purpose NLP solver. In the second approach, the equilibrium constraints are moved to the objective via a quadratic penalty and the resulting non-convex objective with linear constraints is solved using a successive linearization algorithm. While the focus of this paper is cross validation for support vector regression, the proposed methods are readily extended to other model selection problems.

15.1. Introduction

Many supervised machine learning tasks such as classification and regression are usually formulated as convex machine learning problems. This is possible because the loss functions and regularizers considered for the purposes of modeling are generally continuous and either (piecewise) linear or quadratic. One well-studied and widely used example of such problems are *support vector machines* (SVM) (Boser et al., 1992; Cortes and Vapnik, 1995; Vapnik, 2000). SVMs are based on statistical- learning-based approaches and have been applied with great success owing to their generalizability, modularity and extensibility into nonlinear domains via the kernel trick (Shawe-Taylor and Cristianini, 2004).

However, the use of these methods entails solving several other related problems—such as parameter and feature selection—in conjunction with the central machine learning problem. For instance, consider the case of linear support vector regression where the problem is to construct a linear function that minimizes an L_2-regularized ε-insensitive loss (Smola and Schölkopf, 1998). The formulation contains two parameters: the regularization parameter C, that controls the trade-off between complexity and loss and the tube parameter ε, that determines, via the width of the insensitivity tube, the amount of loss tolerated and ultimately the sparsity of the resulting support vector machine. This problem is commonly viewed as a convex, quadratic minimization problem and indeed it is, if the hyper- parameters C and ε are fixed.

Far from being fixed, these parameters are, in fact, continuous variables whose "optimal" values need to be determined judiciously as they affect the generalization behavior of the SVM. In fact, parameter selection is a crucial step in order to obtain learners that generalize well. Conventional approaches tend to either gloss over the issue or treat parameter selection as a separate problem when it is clear that it is inextricably linked to the learning task at hand. When viewed from this perspective: that the hyper-parameters are also variables in the model, the machine learning task becomes *non-convex*. This view can also be extended to the nonlinear case where, in addition to the model hyper-parameters, the kernel parameters for a class of kernels also needs to be selected, further complicating the problem.

Another step that is usually performed prior to solving the learning problem is dimensionality reduction. This task, also known as feature selection, is also an integral part of the learning process. Conventional approaches treat this too as a stand-alone pre-processing step that is performed on the data prior to learning. This suggests that the conventional approach to solving machine learning problems involves a multi-step, sequential parameter and feature selection (collectively known as model selection) process that is necessitated by the myth of convexity in machine learning.

In this chapter, we examine a new paradigm, based on bilevel optimization, which provides a unified framework within which the non-convex model selection problem can be formulated, studied and solved. Recent algorithmic advances in bilevel programming allow for the systematic treatment of model hyper-parameters as continuous variables in the learning problem. This is achieved by embedding cross validation into the bilevel setting in order to perform model selection. The flexibility of the approach is such that tasks such as regression, classification, semi-supervised learning, learning from missing data, and multitask learning among others can be formulated under this paradigm.

15.1.1. Challenges in Model Selection

As mentioned before, we are interested in solving the non-convex model selection problem in order to simultaneously determine the hyper-parameters and possibly relevant features. We focus initially on parameter selection. While there have been many interesting attempts to use bounds or other techniques to pick these hyper-parameters (Chapelle et al., 2002; Duan et al., 2003), the most commonly used and widely accepted method is *cross validation* (CV) (Kohavi, 1995). In cross validation, the hyper-parameters are selected to minimize some estimate of the out-of-sample generalization error. As test data are not available during training, cross validation trains on subsets of data called the training sets and simulates out-of-sample testing on other subsets of the training data called validation sets.

Typically, a grid is defined over the hyper-parameters of interest and k-fold cross validation is performed at each grid point (Stone, 1974), that is k training problems are solved and validated at each grid point (see Figure 15.1). The inefficiencies and expense of such a grid-search effectively limit the number of hyper-parameters in a model due to the combinatorial explosion of grid points in a high dimensional hyper-parameter space. Problems with many parameters are ubiquitous in data analysis; they arise frequently in feature selection (Bi et al., 2003; Guyon et al., 2002), kernel construction (Lanckriet et al., 2004; Ong et al., 2005), and multitask learning (Caruana, 1997; Evgeniou and Pontil, 2004). For such high-dimensional problems, such as in feature selection, greedy strategies such as stepwise regression, backward elimination, filter

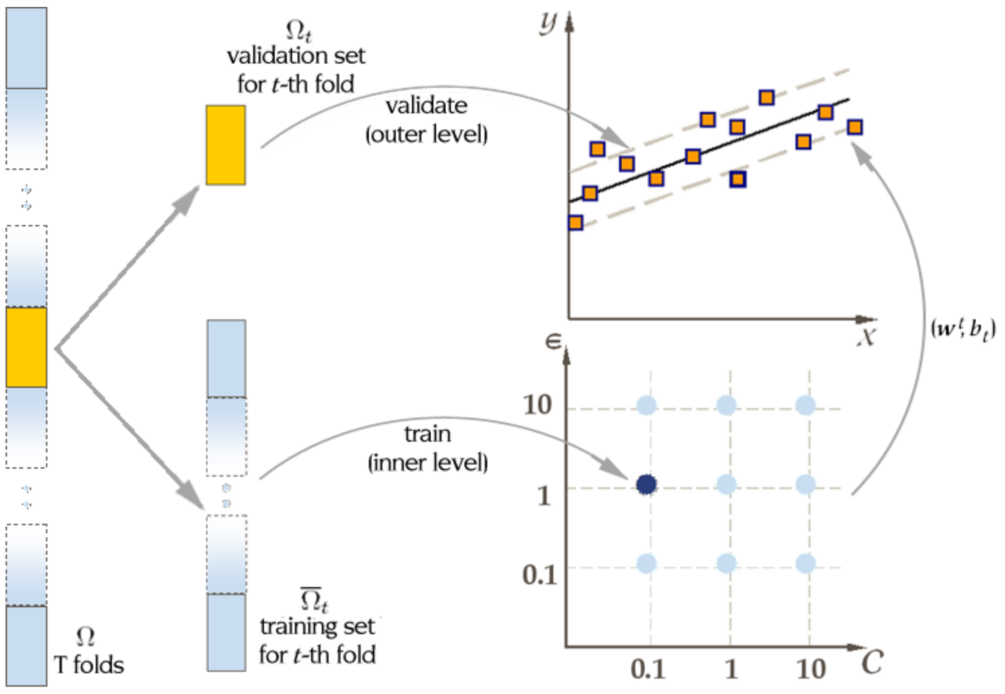


Figure 15.1: Schematic of T-fold cross validation for support vector regression via a classical grid search procedure. A base-10 logarithmic grid is defined over the hyper-parameters, C and ε and T problems are solved and validated for each grid point. The pair (C^*, ε^*) that gives the smallest averaged validation error is selected as the "optimal" set of parameters.

methods, or genetic algorithms have been used. Yet, these heuristic methods, including grid search, have a fundamental deficiency in addition to their practical inefficiency; namely, they are incapable of assuring the overall quality of the produced "solution".

Another drawback in grid search is that its discretization of the hyper-parameter space fails to take into account the fact that the model parameters are continuous. Recent work on determining the full regularization path of support vector machines underscores the fact that the regularization parameter is continuous. In particular, the paper by Hastie et al. (2004) argues that the choice of the single regularization parameter C is critical and shows that it is quite tractable to compute the SVM solution for all possible values of the regularization parameter C. However, this parametric programming approach for a single parameter is not extendable to models with multiple parameters and certainly is not possible for models with a large number of parameters. Bayesian methods can treat model parameters as random variables but then the challenge becomes the choice of appropriate priors.

In the end, out-of-sample testing is still the gold standard for selecting parameters values. From the standpoint of "optimizing model selection" using out-of-sample estimates, there is an urgent need for improved methodologies that combine sound theoretical foundation and robust computational efficiency. This paper proposes one such methodology that is based on the methods of *bilevel optimization*.

15.1.2. A New Methodology

Since bilevel optimization is a relatively novel tool in the machine learning community, we introduce this methodology by way of a brief historical perspective. In a nutshell, bilevel optimization problems are a class of constrained optimization problems whose constraints contain a *inner-level* optimization problem that is parameterized by a multi-dimensional design variable. Thus, in a bilevel program the goal is to optimize the *outer-level* objective subject to constraints which themselves are functions of stationarity points of other mathematical programs called inner-level programs.

In operations research literature, the class of bilevel optimization problems was introduced in the early 1970s by Bracken and McGill (1973). Problem (15.1) below, is a general bilevel formulation:

$$\begin{array}{lll} \max\limits_{x \in X, y} & F(x,y) & \text{outer level} \\ \text{s.t.} & G(x,y) \leq 0, & \\ & y \in \left\{ \begin{array}{ll} \arg\max\limits_{y \in Y} & f(x,y) \\ \text{s.t.} & g(x,y) \leq 0 \end{array} \right\}. & \text{inner level} \end{array} \tag{15.1}$$

These problems are closely related to the economic problem of the Stackelberg game, whose origin predates the work of Bracken and McGill. In a Stackelberg game, T players called *followers* try to chose an optimal market strategy based only on knowledge of the *leader's* strategy. The leader, on the other hand, is one level up in the hierarchy and is aware of the strategies of the followers.

The contribution of this research is to directly tackle model selection using cross-validation-based out-of-sample testing as a bilevel program. As in done in cross validation, the data is partitioned or bootstrapped into training and validation sets. The idea is that the cross-validation estimate is optimized in the outer level subject to some optimal choice of hyper-parameters which become outer-level variables. While validation takes place in the outer level, training takes place, for T-fold cross-validation, in T inner levels such that when the optimal training problem is solved for each training set, the loss over the validation sets is minimized (Figure 15.2).

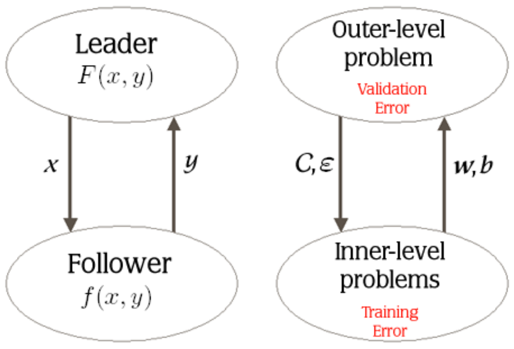


Figure 15.2: The Stackelberg game (left), showing the hierarchy between the leader and the follower; Cross validation modeled as a bilevel program (right), showing the interaction between the parameters, which are optimized in the outer level and the models which are trained in the inner level.

Prior bilevel approaches have been developed and successfully used for inner-level problems with closed form solutions and a single parameter, e.g. the generalized cross validation method for selecting the ridge parameter in ridge regression (Golub et al., 1979). Again, however, these approaches are limited to a single hyper-parameter and lower-level function with a closed-form solution. Path-following methods such as by Hastie et al. (2004) and Wang et al. (2006) are quite efficient for optimizing one or two hyper-parameters trained on a single training set and evaluated on a single validation set, but they are not applicable for cross-validation and models with many hyper-parameters.

The proposed method, bilevel model selection, is motivated by the need to search a continuous, high-dimensional hyper-parameter space for "optimal" model parameters. In addition, this methodology offers several fundamental advantages over prior approaches. First, recent advances in bilevel programming in the optimization community permit the systematic treatment of models based on popular loss functions used for SVM and kernel methods with many hyper-parameters. Our proposed bilevel programming approach places this parameter selection problem on well-studied ground, enabling the employment of the state-of-the-art nonlinear programming (NLP) methodology, including many highly effective algorithmic solvers that are freely available on the NEOS website[1]. In addition to the ability to simultaneously optimize many hyper-parameters, the bilevel programming approach offers a broad framework in which novel regularization methods can be developed, valid bounds on the test set errors can be obtained, and most significantly, improved model selection can be performed.

In the late 1980s, bilevel programming was given renewed study in the extended framework of a *mathematical program with equilibrium constraints* (MPEC) (Luo et al., 1996), which is an extension of a bilevel program with the optimization constraint replaced by a finite-dimensional variational inequality (Facchinei and Pang, 2003). The systematic study of the bilevel optimization problem and its MPEC extension attracted the intense attention of mathematical programmers about a decade ago with the publication of a focused monograph by Luo et al. (1996), which was followed by two related monographs by Outrata et al. (1998) and Dempe (2002). During the past decade, there has been an explosion of research on these optimization problems.

1. http://www-neos.mcs.anl.gov/neos/solvers/

See the annotated bibliography by Dempe (2003), which contains many references. In general, bilevel programs/MPECs provide a novel paradigm and a powerful computational framework for dealing with parameter identification problems in an optimization setting. Instead of describing a bilevel optimization problem in its full generality, we focus our discussion on its application to CV for model selection.

15.2. Model Selection as a Bilevel Program

We consider the machine learning problem of finding optimal model parameters, $\boldsymbol{\lambda} \in \Lambda$, that pick the best hypothesis function $f^\star : \mathscr{X} \to \mathbb{R}$. Here, $\mathscr{X} \subset \mathbb{R}^n \times \mathbb{R}$ is the labeled training set and $f^\star \in \mathscr{F}$, the class of candidate hypothesis functions. The best hypothesis function will be the one that has the least generalization error. A general machine learning problem, with $\boldsymbol{\lambda}$ fixed is

$$f^\star \in \underset{f \in \mathscr{F}}{\arg\min} \left\{ \mathscr{P}(f;\boldsymbol{\lambda}) + \sum_{(\mathbf{x}^j, y_j) \in \mathscr{X}} \mathscr{L}\left(y_j, f(\mathbf{x}^j); \boldsymbol{\lambda}\right) \right\}, \tag{15.2}$$

where $\mathscr{P}$ is the regularization operator and $\mathscr{L}$ is the loss function. It is immediately apparent that this generalized formulation admits many common loss functions and regularizers, and consequently many well-known machine learning problems such as misclassification minimization, ridge regression, one- and two-norm SVMs for classification and regression, elastic-net SVMs and many more. When the parameters, $\boldsymbol{\lambda}$, are fixed, the general machine learning problem is usually convex. However, we wish to determine the optimal parameters, $\boldsymbol{\lambda}^\star$ that will yield a model that generalizes best; now, $\boldsymbol{\lambda}$ is no longer fixed in the general problem making it non-convex.

We will solve this parameter selection problem within the general framework of cross validation formulated as a continuous bilevel program. Let $\Omega := \{(\mathbf{x}^1, y_1), \ldots, (\mathbf{x}^\ell, y_\ell)\}$ in the Euclidean space $\mathscr{X}$, for some positive integers ℓ and n, denote the given labeled data set. As in classical cross validation, we partition the ℓ data points into T disjoint partitions, Ω_t for $t = 1, \ldots, T$, such that $\bigcup_{t=1}^T \Omega_t = \Omega$. Let $\overline{\Omega}_t \equiv \Omega \setminus \Omega_t$ be the subset of the data other than those in group Ω_t. The sets $\overline{\Omega}_t$ are called *training sets* while the sets Ω_t are called the *validation sets*. The index sets for the validation and training sets are $\mathscr{N}_t$ and $\overline{\mathscr{N}}_t$ respectively. The loss functions, $\mathscr{L}_{val}$ and $\mathscr{L}_{trn}$, are functions of f (typically convex for fixed $\boldsymbol{\lambda}$) that quantify the the error between the desired and predicted function values for each point. With these definitions, T-fold cross validation for some general machine learning program can be formulated as a bilevel program:

$$\begin{array}{lll}
\underset{\boldsymbol{\lambda}}{\text{minimize}} & \displaystyle\frac{1}{T}\sum_{t=1}^{T}\frac{1}{|\Omega_t|}\sum_{(\mathbf{x}^j, y_j)\in\Omega_t}\mathscr{L}_{val}\left(y_j, f(\mathbf{x}^j); \boldsymbol{\lambda}\right) & \text{(outer-level problem)} \\
\text{subject to} & \boldsymbol{\lambda} \in \Lambda, & \\
& \text{and for } t = 1, \ldots, T, & \\
& f^t \in \underset{f\in\mathscr{F}}{\arg\min}\left\{\mathscr{P}(f;\boldsymbol{\lambda}) + \displaystyle\sum_{(\mathbf{x}^j, y_j)\in\overline{\Omega}_t}\mathscr{L}_{trn}\left(y_j, f(\mathbf{x}^j); \boldsymbol{\lambda}\right)\right\}. & \text{(inner-level problems)}
\end{array} \tag{15.3}$$

In the problem above, there are T *inner-level problems* where, for an optimal choice of $\boldsymbol{\lambda}$, T problems are solved on the training sets $\overline{\Omega}_t$ to find models, f^t. There is one *outer-level problem* where the model parameters $\boldsymbol{\lambda}$ are determined such that the generalization error, which we denote $\Theta(f^1,\ldots,f^T;\boldsymbol{\lambda})$, is minimized for the models f^t evaluated over the validation sets Ω_t using loss $\mathscr{L}_{val}$.This is the *generalized bilevel cross validation* model.

The general definition (15.3) admits as many variations and well-known machine learning problems as does the formulation (15.2). We will apply the general model to the well-known problem of statistical learning theory of finding a function $f^\star : \mathscr{X} \to \mathbb{R}$ among a given class $\mathscr{F}$ that minimizes the regularized risk functional

$$R[f] \equiv \mathscr{P}[f] + \lambda \sum_{i=1}^{\ell} \mathscr{L}(y_i, f(\mathbf{x}^i)),$$

where C is the regularization parameter. Considering the class of linear functions: $f(\mathbf{x}) = \mathbf{w}'\mathbf{x} - b = \sum_{i=1}^{n} w_i x_i - b$, where $\mathbf{w} \in \mathbb{R}^n$. The following are some examples of the types of machine learning problems that can be cast into this framework:

- ridge regression: $\mathscr{P}[f] \equiv \frac{1}{2}\|\mathbf{w}\|_2^2$; squared loss, $\mathscr{L}(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$
- SV classification: $\mathscr{P}[f] \equiv \frac{1}{2}\|\mathbf{w}\|_2^2$; hinge loss, $\mathscr{L}(y, f(\mathbf{x})) = \max(1 - yf(\mathbf{x}), 0)$
- SV regression: $\mathscr{P}[f] \equiv \frac{1}{2}\|\mathbf{w}\|_2^2$; ε-insensitive loss, $\mathscr{L}_\varepsilon(y, f(\mathbf{x})) = \max(|y - f(\mathbf{x})| - \varepsilon, 0)$
- elastic net SVMs: $\mathscr{P}[f] \equiv \lambda_1\|\mathbf{w}\|_1 + \lambda_2\|\mathbf{w}\|_2^2$; squared loss, $\mathscr{L}(y, f(\mathbf{x})) = (y - f(x))^2$
- other regularization functions: $\mathscr{P}[f] \equiv \|\mathbf{w}\|_1$, $\mathscr{P}[f] \equiv \|\mathbf{w}\|_\infty$

The outer-level cross validation error function, $\mathscr{L}_{val}$ can be chosen appropriate to the machine learning task being cross-validated. For example, classical cross validation uses the misclassification loss, $\mathscr{L}_{val}(y, f(\mathbf{x})) = \text{step}(-yf(\mathbf{x}))$, for classification; and mean average deviation, $\mathscr{L}_{val}(y, f(\mathbf{x})) = \frac{1}{\ell}\sum |y - f(\mathbf{x})|$ or mean squared error, $\mathscr{L}_{val} = \frac{1}{\ell}\sum(y - f(\mathbf{x}))^2$ for regression.

The work presented in this chapter non-trivially extends the initial proof-of-concept ideas presented in Bennett et al. (2006) for support vector regression. The method has also been applied to SV classification; for a detailed discussion of bilevel model selection for support vector classification solved using off-the-shelf NLP solvers such as FILTER and SNOPT on NEOS, see Kunapuli et al. (2008b). The work has also been extended to learning with missing values in data for support vector regression; for more information see (Kunapuli, 2008). In addition to the above models, the methodology can be extended to other machine learning problems including semi-supervised learning for classification and regression, kernel methods and multi-task learning; see Kunapuli et al. (2008a). The models presented above are restricted to linear function spaces. However, we show that (see Section 15.3.2) the "kernel trick" can be applied to bilevel cross validation in order to enable it to handle nonlinear data sets. We now continue this discussion while focussed solely on applying bilevel cross validation to perform model selection for SV regression.

15.3. Bilevel Cross Validation for Support Vector Regression

We will apply the general model to the well-known problem of support vector regression of finding a function $f^\star : \mathscr{X} \to \mathbb{R}$ among a given class $\mathscr{F}$ that minimizes the regularized risk functional

$$R[f] \equiv \frac{1}{2}\|\mathbf{w}\|_2^2 + C \sum_{i=1}^{\ell} \max\{|y - f(\mathbf{x})| - \varepsilon, 0\},$$

where C is the regularization parameter. Usually the ε-insensitive loss is used in SVR, where $\varepsilon > 0$ is the *tube parameter*, which could be difficult to select as one does not know beforehand how accurately the function will fit the data. We consider linear functions: $f(\mathbf{x}) = \mathbf{w}'\mathbf{x} - b = \sum_{i=1}^{n} w_i x_i - b$.

The classic SVR approach has two hyper-parameters, the regularization constant C and the tube width ε, that are typically selected by cross validation based on (generalization) estimates of the mean square error (MSE) or mean absolute deviation (MAD) measured on the out-of-sample data. In what follows, we focus on the latter and introduce additional parameters for feature selection and improved regularization. The bilevel model can be formulated to find hyper-parameters such that the mean average deviation of the validation sets (generalization estimate) is minimized:

$$\begin{array}{ll}
\underset{C,\varepsilon,\mathbf{w}^t,\underline{\mathbf{w}},\overline{\mathbf{w}}}{\text{minimize}} & \displaystyle\frac{1}{T}\sum_{t=1}^{T}\frac{1}{|\mathcal{N}_t|}\sum_{i\in\mathcal{N}_t}|\mathbf{x}^{i\prime}\mathbf{w}^t - b_t - y_i| \\
\text{subject to} & \varepsilon, C, \geq 0, \quad \underline{\mathbf{w}} \leq \overline{\mathbf{w}}, \\
& \text{and for } t = 1,\ldots,T, \\
& (\mathbf{w}^t, b_t) \in \underset{\substack{\underline{\mathbf{w}}\leq\mathbf{w}\leq\overline{\mathbf{w}} \\ b\in\mathbb{R}}}{\arg\min}\left\{\displaystyle\frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{j\in\overline{\mathcal{N}}_t}\max(|\mathbf{x}^{j\prime}\mathbf{w} - b_t - y_j| - \varepsilon, 0)\right\},
\end{array} \tag{15.4}$$

The parameters, $\underline{\mathbf{w}}$ and $\overline{\mathbf{w}}$, are related to feature selection and regularization. The bound constraints $\underline{\mathbf{w}} \leq \mathbf{w} \leq \overline{\mathbf{w}}$ enforce the fact that the weights on each descriptor must fall in a range for all of the cross-validated solutions. This effectively constrains the capacity of each of the functions, leading to an increased likelihood of improving the generalization performance. It also forces all the subsets to use the same descriptors, a form of variable selection. This effect can be enhanced by adopting the one-norm, which forces $\mathbf{w}$ to be sparse. The box constraints will ensure that consistent but not necessarily identical sets will be used across the folds. This represents a fundamentally new way to do feature selection, embedding it within cross validation for model selection.

Note that the loss functions used in the first level and second level—to measure errors—need not match. For the inner-level optimization, we adopt the ε-insensitive loss function because it produces robust solutions that are sparse in the dual space. But typically, ε-insensitive loss functions are not employed in the outer cross-validation objective; so here we use mean absolute deviation (as an example). Also, to facilitate comparison with grid search, we restrict C and ε to be within prescribed upper bounds in our implementation.

15.3.1. Bilevel Problems as MPECs

To solve (15.4), we rewrite the t-th inner-level optimization problem by introducing additional slack variables, $\boldsymbol{\xi}^t \geq 0$, within the t-th fold as follows: for given ε, C, $\underline{\mathbf{w}}$ ($= -\overline{\mathbf{w}}$), and $\overline{\mathbf{w}}$,

$$\begin{array}{ll}
\underset{\mathbf{w}^t,\boldsymbol{\xi}^t}{\text{minimize}} & \displaystyle\frac{1}{2}\|\mathbf{w}^t\|_2^2 + C\sum_{j\in\overline{\mathcal{N}}_t}\xi_j^t \\
\text{subject to} & -\overline{\mathbf{w}} \leq \mathbf{w}^t \leq \overline{\mathbf{w}}, \\
& \left.\begin{array}{l}
\xi_j^t \geq \mathbf{x}^{j\prime}\mathbf{w}^t - b_t - y_j - \varepsilon \\
\xi_j^t \geq y_j - \mathbf{x}^{j\prime}\mathbf{w}^t + b_t - \varepsilon \\
\xi_j^t \geq 0
\end{array}\right\} \quad j \in \overline{\mathcal{N}}_t,
\end{array} \tag{15.5}$$

which is easily seen to be a convex quadratic program in the variables $\mathbf{w}^t$, b_t and $\boldsymbol{\xi}^t$. By letting $\gamma^{t,\pm}$ be the multipliers of the bound constraints, $-\overline{\mathbf{w}} \leq \mathbf{w} \leq \overline{\mathbf{w}}$, respectively, and $\alpha_j^{t,\pm}$ be the multipliers of the constraints $\xi_j^t \geq \mathbf{x}^{j\prime}\mathbf{w}^t - b_t - y_j - \varepsilon$ and $\xi_j^t \geq y_j - \mathbf{x}^{j\prime}\mathbf{w}^t + b_t - \varepsilon$, respectively, we obtain the Karush-Kuhn-Tucker optimality conditions of (15.5) as the following *linear complementarity problem* in the variables $\mathbf{w}^t$, b_t, $\gamma^{t,\pm}$, $\alpha_j^{t,\pm}$, and $\boldsymbol{\xi}_j^t$:

$$\begin{aligned}
&0 \leq \gamma^{t,-} \perp \overline{\mathbf{w}} + \mathbf{w}^t \geq 0,\\
&0 \leq \gamma^{t,+} \perp \overline{\mathbf{w}} - \mathbf{w}^t \geq 0,\\
&\left.\begin{aligned}
&0 \leq \alpha_j^{t,-} \perp \mathbf{x}^{j\prime}\mathbf{w}^t - b_t - y_j + \varepsilon + \xi_j^t \geq 0\\
&0 \leq \alpha_j^{t,+} \perp y_j - \mathbf{x}^{j\prime}\mathbf{w}^t + b_t + \varepsilon + \xi_j^t \geq 0\\
&0 \leq \xi_j^t \quad \perp C - \alpha_j^{t,+} - \alpha_j^{t,-} \geq 0
\end{aligned}\right\} \forall j \in \overline{\mathcal{N}}_t,\\
&0 = \mathbf{w}^t + \sum_{j \in \overline{\mathcal{N}}_t} (\alpha_j^{t,+} - \alpha_j^{t,-})\mathbf{x}^j + \gamma^{t,+} - \gamma^{t,-},\\
&\sum_{j \in \overline{\mathcal{N}}_t} (\alpha_j^{t,+} - \alpha_j^{t,-}) = 0,
\end{aligned} \tag{15.6}$$

where $a \perp b$ means $a'b = 0$. The orthogonality conditions in (15.6) express the well-known complementary slackness properties in the optimality conditions of the inner-level (parametric) quadratic program. If we consider linear models with no bias term, b_t, then the last equality constraint, $\sum_{j \in \overline{\mathcal{N}}_t} (\alpha_j^{t,+} - \alpha_j^{t,-}) = 0$, drops out. Each of the inner-level problems in (15.4) can be replaced by their corresponding KKT conditions, (15.6).

The outer-level objective, Θ, is not differentiable due to the presence of the absolute value function. This can be fixed by the standard linear programming trick of introducing additional slack variables $\mathbf{z}^t$ similar to the variables $\boldsymbol{\xi}^t$ introduced in (15.5). The overall two-level regression problem is therefore

$$\begin{aligned}
\text{minimize} \quad & \frac{1}{T}\sum_{t=1}^{T} \frac{1}{|\mathcal{N}_t|} \sum_{i \in \mathcal{N}_t} z_i^t\\
\text{subject to} \quad & \varepsilon, C, \overline{\mathbf{w}} \geq 0,\\
\text{and} \quad & \text{for all } t = 1, \ldots, T\\
& -z_i^t \leq \mathbf{x}^{i\prime}\mathbf{w}^t - b_t - y_i \leq z_i^t, \qquad \forall i \in \mathcal{N}_t,\\
& \left.\begin{aligned}
&0 \leq \alpha_j^{t,-} \perp \mathbf{x}^{j\prime}\mathbf{w}^t - b_t - y_j + \varepsilon + \xi_j^t \geq 0\\
&0 \leq \alpha_j^{t,+} \perp y_j - \mathbf{x}^{j\prime}\mathbf{w}^t + b_t + \varepsilon + \xi_j^t \geq 0\\
&0 \leq \xi_j^t \quad \perp C - \alpha_j^{t,+} - \alpha_j^{t,-} \geq 0
\end{aligned}\right\} \forall j \in \overline{\mathcal{N}}_t,\\
& 0 \leq \gamma^{t,-} \perp \overline{\mathbf{w}} + \mathbf{w}^t \geq 0,\\
& 0 \leq \gamma^{t,+} \perp \overline{\mathbf{w}} - \mathbf{w}^t \geq 0,\\
& 0 = \mathbf{w}^t + \sum_{j \in \overline{\mathcal{N}}_t} (\alpha_j^{t,+} - \alpha_j^{t,-})\mathbf{x}^j + \gamma^{t,+} - \gamma^{t,-},\\
& \sum_{j \in \overline{\mathcal{N}}_t} (\alpha_j^{t,+} - \alpha_j^{t,-}) = 0.
\end{aligned} \tag{15.7}$$

The most noteworthy feature of the above optimization problem is the complementarity conditions in the constraints, making the problem an instance of a *Linear Program with (linear) Equilibrium Constraints* (LPEC).

15.3.2. Kernel Bilevel Cross Validation

The LPEC (15.7) is unable to handle non-linear data sets effectively as it is restricted to a linear class of functions. One of the most powerful features of SVMs is their ability to deal with high-dimensional, highly nonlinear data and this is achieved via the *kernel trick*. We show that this can be incorporated into the bilevel model as well.

The model (15.4) can perform simultaneous parameter and feature selection as the bilevel framework can handle multiple hyper-parameters. A glance at the first-order conditions, (15.6), shows that $\mathbf{w}^t$ depends, not only on the training data, but also on the multipliers of the box constraints, $\gamma^{t,\pm}$. In order to apply the kernel trick the hyperplane, $\mathbf{w}^t$, needs to be expressed solely as a linear combination of the training data. To enable this, we set aside feature selection for a moment and drop the box constraints. and the variables $\gamma^{t,\pm}$ from 15.7. The first order conditions within each fold now include the constraints:

$$\mathbf{w}^t = \sum_{k\in\overline{\mathcal{N}}_t} (\alpha_k^{t,-} - \alpha_k^{t,+})\mathbf{x}^k, \quad \forall t = 1,\ldots,T. \tag{15.8}$$

We can eliminate $\mathbf{w}^t$ within each fold of (15.7) using (15.8) and then apply the kernel trick, i.e., the resulting linear inner-product terms, $\mathbf{x}^{i\,\prime}\mathbf{x}^j$, are replaced with known, symmetric, positive semi-definite kernel functions, $\kappa(\mathbf{x}^i, \mathbf{x}^j)$. We have:

$$\begin{array}{rl}
\text{minimize} & \dfrac{1}{T}\displaystyle\sum_{t=1}^{T}\frac{1}{|\mathcal{N}_t|}\sum_{i\in\mathcal{N}_t} z_i^t \\[2ex]
\text{subject to} & \varepsilon, C, \geq 0, \\[1ex]
\text{and} & \text{for all } t = 1,\ldots,T \\[1ex]
& -z_i^t \leq \left[\displaystyle\sum_{k\in\overline{\mathcal{N}}_t}(\alpha_k^{t,-} - \alpha_k^{t,+})\kappa(\mathbf{x}^i, \mathbf{x}^k) - b_t\right] - y_i \leq z_i^t, \qquad \forall i \in \mathcal{N}_t, \\[3ex]
& \left.\begin{array}{l}
0 \leq \alpha_j^{t,-} \perp \left[\displaystyle\sum_{k\in\overline{\mathcal{N}}_t}(\alpha_k^{t,-} - \alpha_k^{t,+})\kappa(\mathbf{x}^j, \mathbf{x}^k) - b_t\right] - y_j + \varepsilon + \xi_j^t \geq 0 \\[3ex]
0 \leq \alpha_j^{t,+} \perp \left[b_t - \displaystyle\sum_{k\in\overline{\mathcal{N}}_t}(\alpha_k^{t,-} - \alpha_k^{t,+})\kappa(\mathbf{x}^j, \mathbf{x}^k)\right] + y_j + \varepsilon + \xi_j^t \geq 0 \\[3ex]
0 \leq \xi_j^t \quad \perp C - \alpha_j^{t,+} - \alpha_j^{t,-} \geq 0
\end{array}\right\} \forall j \in \overline{\mathcal{N}}_t, \\[3ex]
& \displaystyle\sum_{j\in\overline{\mathcal{N}}_t}(\alpha_j^{t,+} - \alpha_j^{t,-}) = 0.
\end{array} \tag{15.9}$$

While it may not appear so at first glance, when the kernel is known in advance and hence fixed, the optimization problem above is still an instance of an LPEC. Unfortunately, the kernel parameter is also usually one of the hyper-parameters that needs to be selected during model selection and is usually included in the cross-validation procedure. One further drawback is that in order to derive (15.9), we jettisoned feature selection which the model above is unable to perform.

These issues of parameter selection (for regularization and the kernel) and feature selection can be addressed together as in the linear model. This is achieved through the use of a parameterized kernel, $\kappa(\mathbf{x}^i, \mathbf{x}^k; \mathbf{p}, \mathbf{q})$. The nonnegative vector, $\mathbf{p} \in \mathbb{R}^n_+$, is the feature selection or scaling vector and $\mathbf{q} \geq 0$ is a vector of kernel parameters. The parameterized versions of some commonly used kernels are shown below, with $P = \text{diag}(\mathbf{p})$:

$$\begin{array}{rl} \text{Linear kernel} & \kappa(\mathbf{x}^i, \mathbf{x}^k; \mathbf{p}) = \mathbf{x}^{i\prime} P \mathbf{x}^k, \\ d-\text{th order Polynomial kernel} & \kappa(\mathbf{x}^i, \mathbf{x}^k; \mathbf{p}, c) = (\mathbf{x}^{i\prime} P \mathbf{x}^k + c)^d, \\ \text{Gaussian kernel} & \kappa(\mathbf{x}^i, \mathbf{x}^k; \mathbf{p}) = \exp\left(-(\mathbf{x}^i - \mathbf{x}^k)' P (\mathbf{x}^i - \mathbf{x}^k)\right). \end{array} \tag{15.10}$$

Other kernels can be similarly extended and used in the model. Consequently, the new kernel parameters, $\mathbf{p}$ and $\mathbf{q}$, enter the outer level of the kernel model as variables in the problem. The introduction of the parameterized kernel is a very powerful extension to the linear model (15.7) as it is capable of picking the regularization parameters, kernel parameters and features leaving only the choice of kernel family to the user. However, the optimization problem, (15.9), is an MPEC with nonlinear complementarity constraints and, in general, is a very difficult problem to solve. For the remainder of the discussion, we will restrict ourselves to the linear case (15.7).

15.4. Alternative Bilevel Optimization Methods

The bilevel cross-validation model described above searches the continuous domain of hyper-parameters as opposed to classical cross validation, which relies on the discretization of the domain. In this section, we describe two alternative methods for solving the model. We also describe the details of the classical grid search approach.

The difficulty in solving the LPEC reformulation (15.7) of the bilevel optimization problem (15.4) stems from the linear complementarity constraints formed from the optimality conditions of the inner problem (15.6); all of the other constraints and the objective are linear. It is well recognized that a straightforward solution using the LPEC formulation is not appropriate because of the complementarity constraints, which give rise to both theoretical and computational anomalies that require special attention. Among various proposals to deal with these constraints, two are particularly effective for finding a local solution: one is to relax the complementarity constraints and retain the relaxations in the constraints. The other proposal is via a penalty approach that allows the violation of these constraints but penalizes the violation by adding a penalty term in the objective function of (15.7). There are extensive studies of both treatments, including detailed convergence analyses and numerical experiments on realistic applications and random problems (Ralph and Wright, 2004; Mangasarian, 1994; Bennett and Mangasarian, 1993; Outrata et al., 1998; Luo et al., 1996). In this work, we experiment with both approaches.

15.4.1. A Relaxed NLP Reformulation

The first solution method we consider for solving (15.7) employs a relaxation of the complementarity constraints. In this relaxed complementarity formulation, we let $\mathsf{tol} > 0$ be a prescribed tolerance of the complementarity conditions. This method simply involves replacing all instances of "hard" complementarity constraints of the form

$$0 \leq a \perp b \geq 0 \qquad \equiv \qquad a \geq 0, b \geq 0, a'b = 0$$

with relaxed, "soft" complementarity constraints of the form

$$0 \leq a \perp_{\mathsf{tol}} b \geq 0 \qquad \equiv \qquad a \geq 0, b \geq 0, a'b \leq \mathsf{tol} \tag{15.11}$$

If we apply the relaxation (15.11) to the bilevel problem (15.7), we obtain the *relaxed bilevel support-vector regression problem* that we employ to determine the hyper-parameters C, ε and

$\overline{\mathbf{w}}$; the computed parameters are then used to define the desired support-vector model for data analysis.

The relaxed complementary slackness is a novel feature that aims at enlarging the search region of the desired regression model; the relaxation corresponds to *inexact cross validation* whose accuracy is dictated by the prescribed scalar, tol. This reaffirms an advantage of the bilevel approach mentioned earlier, namely, it adds flexibility to the model selection process by allowing early termination of cross validation, and yet not sacrificing the quality of the out-of-sample errors.

The above NLP remains a non-convex optimization problem; thus, finding a global optimal solution is hard, but the state-of-the-art general-purpose NLP solvers such as FILTER (Fletcher and Leyffer, 1999, 2002) and SNOPT (Gill et al., 2002) that are available on the NEOS server. To solve a given problem, the user first specifies the problem in an algebraic language, such as AMPL or GAMS, and submits the code as a job to NEOS. Upon receipt, NEOS assigns a number and password to the job, and places it in a queue. The remote solver unpacks, processes the problem, and sends the results back to the user.

The nonlinear programming solver, FILTER, was chosen to solve our problems. We also experimented with SNOPT but as reported in Bennett et al. (2006), we found FILTER to work better overall. FILTER is a sequential quadratic programming (SQP) based method, which is a Newton-type method for solving problems with nonlinear objectives and nonlinear constraints. The method solves a sequence of approximate convex quadratic programming subproblems. FILTER implements a SQP algorithm using a trust-region approach with a "filter" to enforce global convergence. It terminates either when a Karush-Kuhn-Tucker point is found within a specified tolerance or no further step can be processed (possibly due to the infeasibility of a subproblem).

15.4.2. Penalty Reformulation

Another approach to solving the problem (15.7) is the penalty reformulation. Penalty and augmented Lagrangian methods have been widely applied to solving LPECs and MPECs, for instance, by Huang et al. (2006). These methods typically require solving an unconstrained optimization problem. In contrast, exact penalty methods penalize only the complementarity constraints in the objective by means of a penalty function.

Consider the LPEC, (15.7), resulting from the reformulation of the bilevel regression problem. Define S_t, for $t = 1, \ldots, T$, to be the constraint set within the t-th fold, without the complementarity constraints:

$$S_t := \left\{ \begin{array}{c|l} & -z_i^t \le {\mathbf{x}^i}'\mathbf{w}^t - y_i \le z_i^t, \quad \forall i \in \mathcal{N}_t, \\ & \left.\begin{array}{l} {\mathbf{x}^j}'\mathbf{w}^t - y_j + \varepsilon + \xi_j^t \ge 0 \\ y_j - {\mathbf{x}^j}'\mathbf{w}^t + \varepsilon + \xi_j^t \ge 0 \\ C - \alpha_j^{t,+} - \alpha_j^{t,-} \ge 0 \end{array}\right\} \forall j \in \overline{\mathcal{N}}_t, \\ z^t, \boldsymbol{\alpha}^{t,\pm}, \boldsymbol{\xi}^t, & -\overline{\mathbf{w}} \le \mathbf{w}^t \le \overline{\mathbf{w}}, \\ \boldsymbol{\gamma}^{t,\pm}, \mathbf{r}^t, s_t, u_t & 0 = \mathbf{w}^t + \sum_{j \in \overline{\mathcal{N}}_t} (\alpha_j^{t,+} - \alpha_j^{t,-})\mathbf{x}^j + \boldsymbol{\gamma}^{t,+} - \boldsymbol{\gamma}^{t,-}, \\ & \sum_{j \in \overline{\mathcal{N}}_t} (\alpha_j^{t,+} - \alpha_j^{t,-}) = 0, \\ & \mathbf{w}^t = \mathbf{r}^t - \mathbb{1}\, u_t, \; b_t = s_t - u_t, \\ & z^t, \boldsymbol{\alpha}^{t,\pm}, \boldsymbol{\xi}^t, \boldsymbol{\gamma}^{t,\pm}, \mathbf{r}^t, s_t, u_t \ge 0. \end{array} \right\}, \tag{15.12}$$

where we rewrite $(\mathbf{w}^t, b_t)$ within each fold as $\mathbf{w}^t = \mathbf{r}^t - \mathbb{1}\, u_t$ and $b_t = s_t - u_t$ with $\mathbf{r}^t, s_t, u_t \ge 0$, and $\mathbb{1}$ denotes a vector of ones of appropriate dimension. Also, let S_0 be defined as

$$S_0 := \left\{ \; C, \varepsilon, \overline{\mathbf{w}} \;\middle|\; C, \varepsilon, \overline{\mathbf{w}} \ge 0 \; \right\}, \tag{15.13}$$

Then, the overall constraint set for the LPEC (15.7), without the complementarity constraints is defined as $S_{\mathsf{LP}} := \bigcup_{t=0}^{T} S_t$. Let all the variables in (15.12) and (15.13) be collected into the vector $\boldsymbol{\zeta} \ge 0$.

In the penalty reformulation, all the complementarity constraints of the form $a \perp b$ in (15.7) are moved into the objective via the penalty function, $\phi(a, b)$. This effectively converts the LPEC (15.7) into a penalty problem of minimizing some, possibly non-smooth, objective function on a *polyhedral set*. Typical penalty functions include the differentiable quadratic penalty term, $\phi(a, b) = a'b$, and the non-smooth piecewise-linear penalty term, $\phi(a, b) = \min(a, b)$. In this paper, we consider the quadratic penalty. The penalty term, which is a product of the complementarity terms is

$$\phi(\zeta) = \sum_{t=1}^{T} \left(\overbrace{\frac{1}{2}\| \mathbf{w}^t \|_2^2 + C \sum_{j \in \overline{\mathcal{N}}_t} \xi_j^t}^{\Theta_p^t} + \underbrace{\begin{array}{l} \frac{1}{2} \sum_{i \in \overline{\mathcal{N}}_t} \sum_{j \in \overline{\mathcal{N}}_t} (\alpha_i^{t,+} - \alpha_i^{t,-})(\alpha_j^{t,+} - \alpha_j^{t,-}){\mathbf{x}^i}'\mathbf{x}^j \\ + \varepsilon \sum_{j \in \overline{\mathcal{N}}_t} (\alpha_j^{t,+} + \alpha_j^{t,-}) + \sum_{j \in \overline{\mathcal{N}}_t} y_j (\alpha_j^{t,+} - \alpha_j^{t,-}) \\ - \overline{\mathbf{w}}'\boldsymbol{\gamma}^{t,+} + \overline{\mathbf{w}}'\boldsymbol{\gamma}^{t,-} \end{array}}_{-\Theta_d^t} \right). \tag{15.14}$$

The first two terms in the quadratic penalty constitute the primal objective, Θ_p^t, while the last five terms constitute the negative of the dual objective, Θ_d^t, for support vector regression in the t-th fold, with fixed parameter values. Consequently, the penalty function is a combination of T differences between the primal and dual objectives of the regression problem in each fold. Thus,

$$\phi(\zeta) = \sum_{t=1}^{T} \left(\Theta_p^t(\zeta_p^t) - \Theta_d^t(\zeta_d^t) \right),$$

where $\zeta_p^t \equiv (\mathbf{w}^t, b_t, \boldsymbol{\xi}^t)$, the vector of primal variables in the t-th primal problem and $\zeta_d^t \equiv (\boldsymbol{\alpha}^{t,\pm}, \boldsymbol{\gamma}^{t,\pm})$, the vector of dual variables in the t-th dual problem. However, the penalty function also contains the hyper-parameters, C, ε and $\overline{\mathbf{w}}$ as variables, rendering $\phi(\zeta)$ non-convex. Recalling that the linear cross-validation objective was denoted by Θ, we define the penalized objective: $P(\zeta;\mu) = \Theta(\zeta) + \mu\,\phi(\zeta)$, and the penalized problem, $PF(\mu)$, is

$$\begin{array}{ll} \min\limits_{\zeta} & P(\zeta;\mu) \\ \text{subject to} & \zeta \in S_{\mathsf{LP}}. \end{array} \tag{15.15}$$

This general penalty formulation can be applied to bilevel programs for cross-validation model selection for many types machine learning problems beyond regression. The penalty formulation results and algorithms in the remainder of this section are directly valid in general for learning linear functions with convex choices of the upper and lower loss functions, regularization with 2-norm, 1-norm, or infinity-norm and other parameters λ as long as the resulting bilevel program can be reduced to an LPEC. In Kunapuli (2008), the penalty approach and successive linearization algorithm are expanded to missing value model selection problems that do not reduce to LPECs.

It should be noted that, in general, KKT points do not exist for LPECs. An alternative local optimality condition, strong stationarity or S-stationarity, is defined for LPECs. A point ζ^* is strongly stationary to an LPEC if it solves an LP formed by fixing the LPEC complementarity conditions appropriately. See (Ralph and Wright, 2004, Definition 2.2) for precise details on strong stationarity. Keeping the definitions of S-stationarity in mind, the penalized problem (15.15) has very useful properties that have been extensively studied. The following results show that finite values of μ can be used and the penalty formulation $PF(\mu)$ is *exact* i.e., local solutions, (S-stationary points) of the LPEC, correspond to stationarity points of $PF(\mu)$.

Theorem 15.1 (Finite penalty parameter) *[(Ralph and Wright, 2004), Theorem 5.1a] Suppose that ζ^* is a strongly stationary point of (15.7), then for all μ sufficiently large, there exists a Lagrangian multiplier vector $\boldsymbol{\rho}^*$, such that $(\zeta^*, \boldsymbol{\rho}^*)$ is a KKT point of $PF(\mu)$, (15.15).*

Finiteness ensures that the penalty parameter can be set to reasonable values, contrasting with other approaches in which the penalty problem only solve the original problem in the limit. It is perhaps not surprising to note that the zero penalty corresponds to a point where the primal and dual objectives are equal in (15.4.2). These strongly stationary solutions correspond to solutions of (15.15) with $\phi(\zeta) = 0$, i.e., a zero penalty.

The quadratic program, $PF(\mu)$, is non-convex, since the penalty term is not positive definite. Continuous optimization algorithms will not necessarily find a global solution of $PF(\mu)$. However, we do know know that local solutions of $PF(\mu)$ that are feasible for the LPEC are also local optimal for the LPEC. This leads us to the following partial converse of Theorem 15.1.

Theorem 15.2 (Complementary $PF(\mu)$ solution solves LPEC) *[(Ralph and Wright, 2004), Theorem 5.2] Suppose ζ^* is a stationary point of $PF(\mu)$ (15.15) and $\phi(\zeta^*) = 0$. Then ζ^* is strongly stationary for (15.7).*

One approach to solving exact penalty formulations like (15.15) is the successive linearization algorithm, where a sequence of problems with a linearized objective, $(\nabla\Theta(\zeta^k) + \mu\,\nabla\phi(\zeta^k))'(\zeta - \zeta^k)$, is solved to generate the next iterate. This approach is called the *Successive Linearization Algorithm for Model Selection* (SLAMS), which we describe below.

15.4.3. Successive Linearization Algorithm for Model Selection

The successive linearization algorithm (SLA), which is based on the finite Frank-Wolfe algorithm, is used to solve the QP, (15.15). This simply involves solving a sequence of LPs until either a global minimum or some locally stationary solution of (15.7) is reached. The convergence properties of SLA are well-studied and it has been extensively applied to solving several mathematical programs that arise in the study of various machine learning methods. This includes bilinear programming based separation of two sets (Bennett and Mangasarian, 1993), SVM feature selection via smooth approximations (Bradley and Mangasarian, 1998), semi-supervised SVMs via concave minimization (Fung and Mangasarian, 2001) and LPECs from misclassification minimization problems (Mangasarian, 1994). In addition, one other motivation for adopting this approach is that there is a definite structure in the LPECs that are studied here that can be exploited and makes this approach a target for decomposition approaches. This issue is currently under investigation and the results will be reported elsewhere.

One concern while using SLA might be the introduction of yet another parameter into the methodology. However, in practice, a sufficiently large value of μ will lead to the penalty term vanishing from the penalized objective, $P(\zeta^*;\mu)$. In such cases, the locally optimal solution to (15.15) will also be feasible and locally optimal to the LPEC (15.7).

Algorithm 15.1: Successive linearization algorithm for model selection.

```
Fix $\mu > 0$.

  1. *Initialization*:
     Start with an initial point, $\zeta^0 \in S_{\mathrm{LP}}$.

  2. *Solve Linearized Problem*:
     Generate a vertex, $\bar{\zeta}^k$, from the previous iterate, $\zeta^k$, by solving the linearized penalty
     problem, $\bar{\zeta}^k \in \arg\,\text{vertex}\min_{\zeta \in S_{\mathrm{LP}}} \nabla_\zeta P(\zeta^k;\mu)'(\zeta - \zeta^k)$.

  3. *Termination Condition*:
     Stop if the minimum principle holds, i.e., if $\nabla_\zeta P(\zeta^k;\mu)'(\bar{\zeta}^k - \zeta^k) = 0$.

  4. *Compute Step Size*:
     Compute step length $\lambda \in \arg\min_{0 \le \lambda \le 1} P\left((1-\lambda)\zeta^k + \lambda\bar{\zeta}^k;\mu\right)$, and get the next iterate,
     $\zeta^{k+1} = (1-\lambda)\zeta^k + \lambda\bar{\zeta}^k$. Go to Step 2.
```

Algorithm 15.1 gives the details of SLAMS. In Step 2, the notation *arg vertex min* indicates that $\bar{\zeta}^k$ is a vertex solution of the LP in Step 2. The step size in Step 4 has a simple closed form solution since a quadratic objective subject to bounds constraints is minimized. The objective has the form $f(\lambda) = a\lambda^2 + b\lambda$, so the optimal solution is either 0, 1 or $\frac{-b}{2a}$, depending on which value yields the smallest objective. SLAMS is a special case of the Frank-Wolfe algorithm and a convergence proof of the Frank-Wolfe algorithm with no assumptions on the convexity of $P(\zeta;\mu)$ can be found in Bennett and Mangasarian (1993). Convergence results of this type are fairly standard and we offer them without proof.

Theorem 15.3 (Convergence of SLAMS) *[Bennett and Mangasarian (1993)] Algorithm 15.1 terminates at ζ^k that satisfies the minimum principle necessary optimality condition of $PF(\mu)$: $\nabla_\zeta P(\zeta^k;\mu)'(\zeta - \zeta^k) \ge 0$ for all $\zeta \in S_{LP}$, or each accumulation $\bar{\zeta}$ of the sequence $\{\zeta^k\}$ satisfies the minimum principle.*

Furthermore, for the case where SLAMS generates a complementary solution, SLAMS finds a strongly stationary solution of the LPEC.

Theorem 15.4 (SLAMS solves LPEC) *Let ζ^k be the sequence generated by* SLAMS *that accumulates to $\bar{\zeta}$. If $\phi(\bar{\zeta}) = 0$, then $\bar{\zeta}$ is strongly stationary for LPEC (15.7).*

Proof For notational convenience let the set $S_{\mathsf{LP}} = \{\zeta \,|\, \mathbf{A}\zeta \geq \mathbf{b}\}$, with an appropriate matrix, $\mathbf{A}$, and vector, $\mathbf{b}$. We first show that $\bar{\zeta}$ is a KKT point of the problem

$$\begin{array}{ll} \min\limits_{\zeta} & \nabla_\zeta P(\zeta;\mu) \\ \text{s.t.} & \mathbf{A}\zeta \geq \mathbf{b}. \end{array}$$

We know that $\bar{\zeta}$ satisfies $\mathbf{A}\bar{\zeta} \geq \mathbf{b}$ since ζ^k is feasible at the k-th iteration. By Theorem 15.3 above, $\bar{\zeta}$ satisfies the minimum principle; thus, we know the systems of equations

$$\nabla_\zeta P(\bar{\zeta};\mu)'(\zeta - \zeta^k) < 0, \quad \zeta \in S_{\mathsf{LP}},$$

has no solution for any $\zeta \in S_{\mathsf{LP}}$. Equivalently, if $I = \{i | A_i\zeta = \mathbf{b}_i\}$, then

$$\nabla_\zeta P(\bar{\zeta};\mu)'(\zeta - \bar{\zeta}) < 0, \;\; A_i\zeta \geq 0, \; i \in I,$$

has no solution. By Farkas' Lemma, there exists $\bar{\mathbf{u}}$ such that

$$\nabla_\zeta P(\bar{\zeta};\mu) - \sum_{i \in I} \bar{\mathbf{u}}_i A_i = 0, \;\; \bar{\mathbf{u}} \geq 0.$$

Thus $(\bar{\zeta}, \bar{\mathbf{u}})$ is a KKT point of $PF(\mu)$ and $\bar{\zeta}$ is a stationary point of $PF(\mu)$. By Theorem 15.2, $\bar{\zeta}$ is also a strongly stationary point of LPEC (15.7). ■

15.4.4. Early Stopping

Typically, in many machine learning applications, emphasis is placed on generalization and scalability. Consequently, inexact solutions are preferred to globally optimal solutions as they can be obtained cheaply and tend to perform reasonably well. Noting that, at each iteration, the algorithm is working to minimize the LPEC objective as well as the complementarity penalty, one alternative to speeding up termination at the expense of the objective is to stop as soon as *complementarity is reached.* Thus, as soon as an iterate produces a solution that is feasible to the LPEC, (15.7), the algorithm is terminated. We call this approach *Successive Linearization Algorithm for Model Selection with Early Stopping* (EZ-SLAMS). This is similar to the well-known machine learning concept of early stopping, except that the criterion used for termination is based on the status of the complementarity constraints i.e., feasibility to the LPEC. We adapt the finite termination result in Bennett and Mangasarian (1993) to prove that EZ-SLAMS terminates finitely for the case when complementary solutions exist, which is precisely the case of interest here. Note that the proof relies upon the fact that S_{LP} is polyhedral with no straight lines going to infinity in both directions.

Theorem 15.5 (Finite termination of EZ-SLAMS) *Let ζ^k be the sequence generated by* SLAMS *that accumulates to $\bar{\zeta}$. If $\phi(\bar{\zeta}) = 0$, then* EZ-SLAMS *terminates at an LPEC (15.7) feasible solution ζ^k in finitely many iterations.*

Proof Let $\mathscr{V}$ be the finite subset of vertices of S_{LP} that constitutes the vertices $\{\bar{\mathbf{v}}^k\}$ generated by SLAMS. Then,

$$\{\zeta^k\} \in \text{convex hull}\{\zeta^0 \cup \mathscr{V}\},$$
$$\bar{\zeta} \in \text{convex hull}\{\zeta^0 \cup \mathscr{V}\}.$$

If $\bar{\zeta} \in \mathscr{V}$, we are done. If not, then for some $\zeta \in S_{\mathsf{LP}}$, $\mathbf{v} \in \mathscr{V}$ and $\lambda \in (0,1)$,

$$\bar{\zeta} = (1-\lambda)\zeta + \lambda \mathbf{v}.$$

For notational convenience define an appropriate matrix M and vector b such that $0 = \phi(\bar{\zeta}) = \bar{\zeta}'(M\bar{\zeta} + q)$. We know $\bar{\zeta} \geq 0$ and $M\bar{\zeta} + q \geq 0$. Hence,

$$v_i = 0, \text{ or } M_i\mathbf{v} + q_i = 0.$$

Thus, $\mathbf{v}$ is feasible for LPEC (15.7). ■

The results comparing SLAMS to EZ-SLAMS are reported in Sections 15.6.1 and 15.6.2. It is interesting to note that there is always a significant decrease in running time with no significant degradation in validation or generalization performance when early stopping is employed.

15.4.5. Grid Search

In classical cross-validation, parameter selection is performed by discretizing the parameter space into a grid and searching for the combination of parameters that minimizes the validation error (which corresponds to the upper level objective in the bilevel problem). This is typically followed by a local search for fine-tuning the parameters. Typical discretizations are logarithmic grids of base 2 or 10 on the parameters. In the case of the classic SVR, cross validation is simply a search on a two-dimensional grid of C and ε.

This approach, however, is not directly applicable to the current problem formulation because, in addition to C and ε, we also have to determine $\overline{\mathbf{w}}$, and this poses a significant combinatorial problem. In the case of k-fold cross validation of n-dimensional data, if each parameter takes d discrete values, cross validation would involve solving roughly $O(kd^{n+2})$ problems, a number that grows to intractability very quickly. To counter the combinatorial difficulty, we implement the following heuristic procedures:

- Perform a two-dimensional grid search on the unconstrained (classic) SVR problem to determine C and ε. We call this the *unconstrained grid search* (Unc. Grid). A coarse grid with values of 0.1, 1 and 10 for C, and 0.01, 0.1 and 1 for ε was chosen.

- Perform an n-dimensional grid search to determine the features of $\overline{\mathbf{w}}$ using C and ε obtained from the previous step. Only two distinct choices for each feature of $\overline{\mathbf{w}}$ are considered: 0, to test if the feature is redundant, and some large value that would not impede the choice of an appropriate feature weight, otherwise. Cross validation under these settings would involve solving roughly $O(3.2^N)$ problems; this number is already impractical and necessitates the heuristic. We label this step the *constrained grid search* (Con. Grid).

- For the synthetic data sets (see Section 15.5.1) *recursive feature elimination* is used Guyon et al. (2002). For real data sets (which have 25 features, see Section 15.5.2), features are eliminated aggressively ie., recursive feature elimination is used to rank the features and only the 10 best features are chosen.

15.5. Experimental Design

Our experiments aim to address several issues. The experiments were designed to compare the successive linearization approaches (with and without early stopping) to the classical grid search method with regard to generalization and running time. The data sets used for these experiments consist of randomly generated synthetic data sets and real world chemoinformatics (QSAR) data.

15.5.1. Synthetic Data

A 16-d data set was generated of which, for the purposes of feature selection, only $n_r = 10$ features were relevant. We trained on sets of $\ell = 30$, 60, 90, 120 and 150 points using 5-fold cross validation and tested on a hold-out set of a further 1,000 points. For each combination of training set size, 10 trials were conducted and the test errors were averaged. In this subsection, we assume the following notation: $U(a,b)$ represents the uniform distribution on $[a,b]$, $N(\mu,\sigma)$ represents the normal distribution with probability density function $\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-(x-\mu)^2/2\sigma^2\right)$.

For each data set, the data, $(\mathbf{w}_{\text{real}}, b_{\text{real}})$ and labels were generated as follows. For each point, 25% of the features were drawn from $U(-1,1)$, 25% from $U(-2.5,2.5)$, another 25% from $U(-5,5)$, and the last 25% from $U(-10,10)$. Each feature of the regression hyperplane $(\mathbf{w}_{\text{real}}, b_{\text{real}})$ was drawn from $U(-1,1)$ and the smallest $n-n_r$ features were set to 0 and considered irrelevant. The noise-free regression labels were computed as $y_i = \mathbf{x}^{i'}\mathbf{w}_{real} - b_{\text{real}}$. Note that these labels now depend only on the relevant features. Gaussian noise drawn from $N(0,0.25)$ was added to the labels.

15.5.2. Real-world QSAR Data

We examined four real-world regression chemoinformatics data sets: Aquasol, Blood/Brain Barrier (BBB), Cancer, and Cholecystokinin (CCK), previously studied in Demiriz et al. (2001). The goal is to create Quantitative Structure Activity Relationship (QSAR) models to predict bioactivities typically using the supplied descriptors as part of a drug design process.

Table 15.1: The Chemoinformatics (QSAR) data sets

Data set	# Obs.	# Train	# Test	# Vars.	# Vars. (stdized)	# Vars. (postPCA)
AQUASOL	197	100	97	640	149	25
B/B BARRIER (BBB)	62	60	2	694	569	25
CANCER	46	40	6	769	362	25
CHOLECYSTOKININ (CCK)	66	60	6	626	350	25

The data is scaled and preprocessed to reduce the dimensionality. As was done in Demiriz et al. (2001), we standardize the data at each dimension and eliminate the uninformative variables that have values outside of ± 4 standard deviations range. Next, we perform principle components analysis (PCA), and use the top 25 principal components as descriptors. The training and hold out set sizes and the dimensionalities of the final data sets are shown in Table 15.1. For each of the training sets, 5-fold cross validation is optimized using bilevel programming. The results are averaged over 10 runs.

The LPs within each iterate in both SLA approaches were solved with CPLEX. The penalty parameter was uniformly set to $\mu = 10^3$ and never resulted in complementarity failure at termination. The hyper-parameters were bounded as $0.1 \le C \le 10$ and $0.01 \le \varepsilon \le 1$ so as to be consistent with the hyper-parameter ranges used in grid search.

15.5.3. Post-processing

The outputs from the bilevel approach and grid search yield the bound $\overline{\mathbf{w}}$ and the parameters C and ε. With these, we solve a constrained support vector problem on all the data points:

$$\begin{array}{ll} \text{minimize} & C \sum_{i=1}^{\ell} \max(\,|\,\mathbf{x}_i'\mathbf{w} - y_i\,| - \varepsilon, 0\,) + \frac{1}{2}\,\|\,\mathbf{w}\,\|_2^2 \\ \text{subject to} & -\overline{\mathbf{w}} \leq \mathbf{w} \leq \overline{\mathbf{w}} \end{array}$$

to obtain the vector of model weights $\widehat{\mathbf{w}}$, which is used in computing the generalization error on the hold-out data:

$$\text{MAD} \equiv \frac{1}{1000} \sum_{(\mathbf{x},y)\ \text{hold-out}} |\,\mathbf{x}'\widehat{\mathbf{w}} - y\,|.$$

15.6. Computational Results

In the following sections, constrained (abbreviated con.) methods refer to the bilevel models that have the box constraint $-\overline{\mathbf{w}} \leq \mathbf{w} \leq \overline{\mathbf{w}}$, while unconstrained (abbreviated unc.) methods refer to the bilevel models without the box constraint. In this section, we compare the performance of several different methods on synthetic data sets.

Four methods are compared are compared to unconstrained grid search (Unc. Grid): constrained grid search (Con. Grid), constrained SLAMS (Con. SLAMS), constrained SLAMS with early stopping (Con. EZ-SLAMS) and FILTER based sequential quadratic programming (Filter SQP). We compare based on CV objective (training error), generalization (test error), computation time and efficacy of feature selection (i.e., the number of features selected).

15.6.1. Synthetic Results

For each set of problems, 5 methods (as described above) were employed. For each method, 10 random instances of the same problem are solved and the averaged results are shown in Table 15.2 respectively. For MAD, the results in bold refer to those that are significantly different than those of the unconstrained grid as measured by a two-sided t-test with significance of 0.1. The results that are significantly better and worse are tagged with a check (✓) or a cross (✗) respectively.

From an optimization perspective, the bilevel programming methods consistently tend to outperform the grid search approaches. The objective values found by the bilevel methods, especially FILTER, are much smaller than those found by their grid-search counterparts especially as data set size increases. Of all the methods, FILTER finds a lower objective most often. The coarse grid size and feature elimination heuristics used in the grid search cause it to find relatively poor objective values.

The reported times provide a rough idea of the computational effort of each algorithm. As noted above, the computation times for the NEOS solver, FILTER, includes transmission, and waiting times as well as solve times. For grid search methods, smart restart techniques were used to gain a considerable increase in speed. It is interesting to note that for smaller data sets Con. Grid is highly inefficient as the underlying recursive feature elimination cannot eliminate too many variables to enable a more efficient grid search. However, the running times for Con. Grid decrease steadily until and it is most efficient at 120 pts which represents an optimal tradeoff between RFE efficiency and running time. However, as the number of points increases again, Con. Grid does worse.

Table 15.2: 16-d synthetic data with Gaussian noise under 5-fold cross validation.

Solution Method	CV Objective (Training Error)	Generalization (Test Error)	Time (sec.)	$\|\|\overline{\mathbf{w}}\|\|_0$ (# feat. selected)
30 pts				
UNC. GRID	0.294 ± 0.07	0.316 ± 0.05	1.1	16.0
CON. GRID	**0.236 ± 0.06** ✓	**0.300 ± 0.05** ✓	2065.0	11.3
FILTER	**0.166 ± 0.05** ✓	0.311 ± 0.04	61.5	9.0
CON. SLAMS	**0.164 ± 0.06** ✓	0.320 ± 0.09	7.3	9.2
CON. EZ-SLAMS	0.293 ± 0.10	0.301 ± 0.07	0.8	8.9
60 pts				
UNC. GRID	0.241 ± 0.04	0.238 ± 0.01	1.5	16.0
CON. GRID	**0.223 ± 0.04** ✓	**0.234 ± 0.01** ✓	1442.7	9.9
FILTER	**0.195 ± 0.03** ✓	**0.254 ± 0.02** ✗	96.5	9.0
CON. SLAMS	**0.197 ± 0.02** ✓	0.243 ± 0.04	20.4	9.5
CON. EZ-SLAMS	0.242 ± 0.04	0.257 ± 0.05	1.8	9.3
90 pts				
UNC. GRID	0.222 ± 0.03	0.229 ± 0.01	2.1	16.0
CON. GRID	**0.213 ± 0.03** ✓	0.226 ± 0.01	1013.5	10.1
FILTER	**0.191 ± 0.02** ✓	0.241 ± 0.03	173.7	9.1
CON. SLAMS	**0.199 ± 0.02** ✓	**0.223 ± 0.01** ✓	37.9	9.3
CON. EZ-SLAMS	0.224 ± 0.02	0.236 ± 0.04	4.1	9.4
120 pts				
UNC. GRID	0.214 ± 0.02	0.222 ± 0.01	3.1	16.0
CON. GRID	0.212 ± 0.01	0.219 ± 0.01	201.6	9.3
FILTER	**0.199 ± 0.01** ✓	0.230 ± 0.01	307.6	9.2
CON. SLAMS	**0.202 ± 0.01** ✓	0.230 ± 0.04	65.9	9.3
CON. EZ-SLAMS	0.215 ± 0.01	0.231 ± 0.04	8.7	9.2
150 pts				
UNC. GRID	0.209 ± 0.01	0.219 ± 0.01	4.6	16.0
CON. GRID	0.211 ± 0.01	0.217 ± 0.01	293.0	9.3
FILTER	**0.171 ± 0.01** ✓	**0.233 ± 0.02** ✗	366.9	8.9
CON. SLAMS	**0.203 ± 0.01** ✓	0.227 ± 0.04	83.7	9.4
CON. EZ-SLAMS	0.210 ± 0.01	0.227 ± 0.04	14.3	9.2

While the computation times of FILTER are less than that of Con. Grid, it is the SLA approaches that really dominate. The efficiency of the SLA approaches is vastly superior to both grid search and FILTER, especially for smaller problems. However, as problem size grows, Unc. SLAMS and Con. SLAMS become more expensive as bigger LPs have to be solved at each iteration while the algorithm converges. However, the early stopping methods, Unc. EZ-SLAMS and Con. EZ-SLAMS are very competitive, even as the problem size grows.

The bilevel approach is much more computationally efficient than grid search on the fully parameterized problems. The results, for FILTER, are relatively efficient and acceptable. It is reasonable to expect that a FILTER implementation on a local machine (instead of over the internet) would require significantly less computation times, which could bring it even closer to the times of Unc. Grid or the SLA methods. The FILTER approach does have a drawback, in that is that it tends to struggle as the problem size increases as is evidenced by its performance on the 150 point data set.

With regard to generalization error, compared to classic SVR optimized with Unc. Grid, FILTER and the SLA approaches yield solutions that are better or comparable to the test problems and never significantly worse especially as the data set size increases. Of particular interest is the fact that SLA approaches that employ early stopping tend to generalize identically to the SLA approaches that do not stop early. This is a very important discovery because it suggests that allowing the SLA approaches to iterate to termination is expensive and is usually without corresponding improvement in the cross-validation objective or the generalization performance.

15.6.2. Computational Results: QSAR Data

Table 15.3: QSAR data under 5-fold cross validation.

Solution Method	CV Objective (Val. Error)	Generalization (Test Error)	Time (sec.)	$\|\|\bar{\mathbf{w}}\|\|_0$ (# feat. selected)
Aquasol				
UNC. GRID	0.719 ± 0.10	0.644 ± 0.07	17.1	25.0
CON. GRID	**0.778 ± 0.09 ✗**	**0.849 ± 0.18 ✗**	1395.9	8.7
FILTER (SQP)	**0.551 ± 0.07 ✓**	**0.702 ± 0.04 ✗**	678.8	13.3
CON. SLAMS	**0.670 ± 0.09 ✓**	0.647 ± 0.06	137.8	12.7
CON. EZ-SLAMS	0.710 ± 0.09	0.643 ± 0.06	19.1	14.0
Blood/Brain Barrier				
UNC. GRID	0.364 ± 0.05	0.314 ± 0.29	13.4	25.0
CON. GRID	**0.463 ± 0.08 ✗**	**0.733 ± 0.44 ✗**	1285.7	6.8
FILTER (SQP)	**0.176 ± 0.01 ✓**	0.332 ± 0.15	121.3	11.9
CON. SLAMS	0.363 ± 0.04	0.312 ± 0.30	17.1	15.4
CON. EZ-SLAMS	0.370 ± 0.04	0.315 ± 0.30	8.0	15.6
Cancer				
UNC. GRID	0.489 ± 0.03	0.502 ± 0.21	10.3	25.0
CON. GRID	0.477 ± 0.06	**0.611 ± 0.20 ✗**	1035.3	6.6
FILTER (SQP)	**0.210 ± 0.02 ✓**	**0.395 ± 0.08 ✓**	100.2	11.8
CON. SLAMS	0.476 ± 0.09	0.481 ± 0.12	25.5	14.0
CON. EZ-SLAMS	**0.567 ± 0.10 ✗**	0.483 ± 0.13	5.2	13.9
Cholecystokinin				
UNC. GRID	0.798 ± 0.06	1.006 ± 0.34	12.0	25.0
CON. GRID	0.783 ± 0.07	**1.280 ± 0.43 ✗**	1157.6	7.6
FILTER (SQP)	**0.499 ± 0.03 ✓**	1.022 ± 0.25	189.8	12.9
CON. SLAMS	**0.881 ± 0.11 ✗**	**1.235 ± 0.28 ✗**	35.1	14.7
CON. EZ-SLAMS	**0.941 ± 0.09 ✗**	**1.217 ± 0.26 ✗**	9.1	14.7

Table 15.3 shows the average results for the QSAR data. After the data is preprocessed, we randomly partition the data into 20 different training and testing sets. For each of the training sets, 5-fold cross validation is optimized using bilevel programming. The results are averaged over the 20 runs. We report results for the same 5 methods used for synthetic data.

Again, as with the synthetic data, FILTER finds solutions with the smallest validation errors. However, computation times for FILTER are not competitive with the SLA methods and not even with Unc. Grid. Unsurprisingly, constrained grid search has the worst computation time. The difficulty of the underlying bilevel optimization problem is underscored by the fact that the greedy Con. Grid search in Section 15.4.5 sometimes fails to find a better solution than the unconstrained grid search. The constrained search drops important variables that cause it to have bad generalization.

In terms of test set error, FILTER and the constrained SLA approaches outperform the unconstrained approaches on the cancer data and do as well on the remaining three data sets. However, on the remaining data sets, the SLA approaches generalize very well and tend to be competitive with Unc. Grid with regard to execution time. The best running times, however, are produced by the early stopping based SLA approaches, which SLAM the door on all other approaches while maintaining good generalization performance.

With regard to feature selection, it is clear that the aggressive feature selection strategy employed by Con. Grid is still inefficient and has a debilitating effect on validation and generalization error. However, FILTER tends to perform feature selection very well and produces correspondingly good generalization suggesting that the box constraints do indeed act as additional regularization that may improve generalization.

15.7. Discussion

We showed how the widely used model selection technique of cross validation for general machine learning problems could be formulated as a bilevel programming problem; the formulation is more flexible and can deal with many more hyper-parameters than the typical grid search strategy. The proposed bilevel problem is converted to an instance of a linear program with equilibrium constraints (LPEC) which is difficult to solve due to the non-convexity created by the complementarity constraints introduced in the reformulation. A major outstanding question has always been the development of efficient algorithms for LPECs and bilevel programs. To this end, we proposed two approaches to solve the LPEC: a relaxed NLP-based approach which was solved using the off-the-shelf, SQP-based, NLP solver, FILTER and a exact penalty-based approach which was solved using a finite successive linearization algorithm.

Our preliminary computational results indicate that general purpose SQP solvers can tractably find high-quality solutions that generalize well. Generalization results on random data show that FILTER yields are comparable, if not better results than current methods. The successive linearization algorithms for model selection (SLAMS) and their early stopping variants (EZ-SLAMS) performed even better than FILTER. In fact, it was shown that the SLAMS algorithms comprehensively outperform classical grid search approaches as the size of the problem grows.

This is despite the fact that neither the NLP- or the SLA-based approaches take advantage of the structure inherent in bilevel problems arising from machine learning applications. Machine learning problems, especially support vector machines, are highly structured, and yield elegant and sparse solutions, a fact that several decomposition algorithms such as sequential minimal optimization target. Despite the non-convexity of the LPECs, bilevel programs for machine learning problems retain the structure inherent in the original machine learning problems. In addition, the variables in these LPECs tend to decouple, for example, in cross validation, the variables may be decoupled along the folds. This suggests that applying decomposition methods to bilevel approaches can make them even more efficient. An avenue for future research is developing a decomposition-based algorithm that can train on data sets containing tens of thousands of points.

While support vector regression was chosen as the machine learning problem to demonstrate the potency of the bilevel approach, the methodology can be extended to several machine learning problems including classification, semi-supervised learning, multi-task learning and novelty detection. Some of these formulations have been presented in Kunapuli et al. (2008a), while others remain open problems. Aside from discriminative methods, bilevel programming can also be applied to generative methods such as Bayesian techniques. Furthermore, the ability to optimize a large number of parameters allows one to consider new forms of models, loss functions and regularization.

The most pressing question, however, arises from a serious limitation of the algorithms presented herein: the results are limited to bilevel models selection problems that can be reduced to LPECs. Thus only linear models can be addressed. Classical machine learning addresses this problem by means of the kernel trick. It was shown in Kunapuli et al. (2008a) that the kernel trick can be incorporated into a generic bilevel model for cross validation. The flexibility of the bilevel approach means that one can even incorporate input-space feature selection into the kernelized bilevel model. This type of bilevel program can be reformulated as an instance of a mathematical program with equilibrium constraints (MPEC). The MPECs arising from kernelized bilevel machine learning problems tend to have several diverse sources of non-convexity because they have nonlinear complementarity and inequality constraints; this leads to very challenging mathematical programs and an equally challenging opening for future pursuits in this field.

Acknowledgments

This work was supported in part by the Office of Naval Research under grant no. N00014-06-1-0014. The authors are grateful to Professor Olvi Mangasarian for his suggestions on the penalty approach.

References

Kristin P. Bennett and Olvi L. Mangasarian. Bilinear separation of two sets in n-space. *Computational Optimization and Applications*, 2(3):207–227, 1993.

Kristin P. Bennett, Jing Hu, Xiaoyun Ji, Gautam Kunapuli, and Jong-Shi Pang. Model selection via bilevel optimization. *International Joint Conference on Neural Networks, (IJCNN) '06.*, pages 1922–1929, 2006.

Jinbo Bi, Kristin P. Bennett, Mark Embrechts, Curt Breneman, and Minghu Song. Dimensionality reduction via sparse support vector machines. *Journal of Machine Learning Research*, 3:1229–1243, 2003.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, New York, NY, USA, 1992. ACM Press.

Jerome Bracken and James T. McGill. Mathematical programs with optimization problems in the constraints. *Operations Research*, 21(1):37–44, 1973.

P. S. Bradley and Olvi L. Mangasarian. Feature selection via concave minimization and support vector machines. In *Machine Learning Proceedings of the Fifteenth International Conference(ICML '98), Jude W. Shavlik, Editor*, pages 82–90, San Francisco, California, 1998. Morgan Kaufmann.

Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

Olivier Chapelle, Vladimir Vapnik, Olivier Bousquet, and Sayan Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1–3):131–159, 2002.

Corrina Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

Ayhan Demiriz, Kristin P. Bennett, Curt M. Breneman, and Mark Embrechts. Support vector regression methods in cheminformatics. *Computer Science and Statistics*, 33, 2001.

Stephan Dempe. *Foundations of Bilevel Programming*. Kluwer Academic Publishers, Dordrecht, 2002.

Stephan Dempe. Annotated bibliography on bilevel programming and mathematical programs with equilibrium constraints. *Optimization*, 52:333–359, 2003.

Kaibo Duan, Sathiya S. Keerthi, and Aun N. Poo. Evaluation of simple performance measures for tuning SVM hyperparameters. *Neurocomputing*, 51:41–59, 2003.

Theodoros Evgeniou and Massimiliano Pontil. Regularized multi–task learning. In *KDD '04: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 109–117, New York, NY, USA, 2004. ACM Press.

Francisco Facchinei and Jong-Shi Pang. *Finite-Dimensional Variational Inequalities and Complementarity Problems*. Springer-Verlag, New York, 2003.

Roger Fletcher and Sven Leyffer. User manual for filtersqp. Technical Report NA/181, Department of Mathematics, University of Dundee, 1999.

Roger Fletcher and Sven Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, 91:239–269, 2002.

Glenn Fung and Olvi L. Mangasarian. Semi-supervised support vector machines for unlabeled data classification. *Optimization Methods and Software*, 15:29–44, 2001.

Philip E. Gill, Walter Murray, and Michael A. Saunders. User's guide for snopt version 6: A fortran package for large-scale nonlinear programming. Technical report, Systems Optimization Laboratory, Stanford University, 2002. URL http://www.cam.ucsd.edu/~peg/papers/sndoc6.pdf.

Gene H. Golub, Michael Heath, and Grace Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21:215–223, 1979.

Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002.

Trevor Hastie, Saharon Rosset, Robert Tibshirani, and Ji Zhu. The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, 5:1391–1415, 2004.

X. X. Huang, X. Q. Yang, and K. L. Teo. Partial augmented lagrangian method and mathematical programs with complementarity constraints. *Journal of Global Optimization*, 35: 235–254, 2006.

Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence*, pages 1137–1145, 1995.

Gautam Kunapuli. *A Bilevel Optimization Approach to Machine Learning*. PhD thesis, Rensselaer Polytechnic Institute, Troy, NY, 2008.

Gautam Kunapuli, Kristin P. Bennett, Jing Hu, and Jong-Shi Pang. Bilevel model selection for support vector machines. In Pierre Hansen and Panos Pardalos, editors, *Data Mining and Mathematical Programming [CRM Proceedings and Lecture Notes]*, volume 45. American Mathematical Society, 2008a.

Gautam Kunapuli, Kristin P. Bennett, Jing Hu, and Jong-Shi Pang. Classification model selection via bilevel programming. In Jiming Peng and Katja Scheinberg, editors, *Optimization Methods and Software: Special Issue on Mathematical Programming for Data Mining*. Taylor and Francis, 2008b. to appear.

Gert R. G. Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El-Ghaoui, and Michael I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Maching Learning Research*, 5:27–72, 2004.

Zhi-Quan Luo, Jong-Shi Pang, and Daniel Ralph. *Mathematical Programs With Equilibrium Constraints*. Cambridge University Press, Cambridge, 1996.

Olvi L. Mangasarian. Misclassification minimization. *Journal of Global Optimization*, 5:309–323, 1994.

Cheng Soon Ong, Alexander J. Smola, and Robert C. Williamson. Learning the kernel with hyperkernels. *Journal of Machine Learning Research*, 6:1043–1071, 2005.

Jiri V. Outrata, Michal Kocvara, and Jochem Zowe. *Nonsmooth Approach to Optimization Problems with Equilibrium Constraints: Theory, Applications and Numerical Results*. Kluwer Academic Publishers, Dordrecht, 1998.

Daniel Ralph and Stephen J. Wright. Some properties of regularization and penalization schemes for mpecs. *Optimization Methods and Software*, 19:527–556, 2004.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *NeuroCOLT2 Technical Report NC2-TR-1998-030*, 1998.

M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society, Series B*, 36:111–147, 1974.

Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 2000.

Gang Wang, Dit-Yan Yeung, and Frederick H. Lochovsky. Two-dimensional solution path for support vector regression. In *ICML '06: Proceedings of the 23rd International Conference on Machine Learning*, pages 993–1000, 2006.

Appendix A

Dataset Description

Appendix A1

Datasets for the Agnostic Learning vs. Prior Knowledge Competition

Isabelle Guyon ISABELLE@CLOPINET.COM

This documents provides details about the five datasets used in the "Agnostic Learning vs. Prior Knowledge" competition organized for IJCNN 2007. We used the same 5 datasets already used for the WCCI 2006 performance prediction challenge.[1] However, the data were reshuffled. We make available two data formats: (1) the raw data as provided from the original data source, (2) the preprocessed data representation of the WCCI 2006 challenge. All tasks are two-class classification problems. The goal of the challenge is to determine **how much can be gained in performance when prior/domain knowledge is available** compared to using a "black-box" method on the preprocessed data, or whether the "black box" agnostic methods match or outperform "Gnostic" methods.

This document provides guidance to the competitors interested in the "prior knowledge" track. The competitors entering in the "agnostic learning" track should not exploit this document to reverse engineer the agnostic track data. The rule of the game is that they should ignore the information made available in this document to make entries into the challenge.

The competitors have several months to build classifiers with provided (labeled) training data. A web server is provided to submit prediction results on additional unlabeled data. Two unlabeled datasets are used for evaluation: a small validation set used during the development period and a very large test set to do the final evaluation and the ranking of the participants. During a development period, the validation set performance is published immediately upon submission of prediction results. The test set performance remains undisclosed until the end of the competition. The labels of the validation set are published shortly before the end of the competition.

The data sets were chosen to span a variety of domains (drug discovery, ecology, handwritten digit recognition, text classification, and marketing.) We chose data sets that had sufficiently many examples to create a large enough test set to obtain statistically significant results. The input variables are continuous or binary, sparse or dense. All problems are two-class classification problems.

The data characteristics are summarized in Table A1.1.

A1.1. Method

Preparing the feature representation used for the agnostic track included preprocessing data to obtain features in the same numerical range (0 to 999 for continuous data and 0/1 for binary data) and randomizing the order of the patterns and the features to homogenize the data.

For all data representations, the data were split into the same three training, validation and test sets. However, the data of the agnostic track and the prior knowledge track were reshuffled in a difference way within each set. The features of the agnostic track were shuffled

1. A report on these datasets is available at http://clopinet.com/isabelle/Projects/modelselect/Dataset.pdf.

Table A1.1: Datasets of the challenge. The columns "sparsity", "type", "Featnum" and "Tr/FN" refer to the feature representation used for the WCCI 2006 challenge and made available to the "agnostic learning" track.

Dataset	Domain	Sparsity (%)	Type	FracPos (%)	Tr/ FN	Feat-Num	Train	Valid	Test
ADA	Marketing	79.4	mixed	24.8	86.4	48	4147	415	41471
GINA	Handwriting	69.2	continuous	49.2	3.25	970	3153	315	31532
HIVA	Drug discovery	90.9	binary	3.5	2.38	1617	3845	384	38449
NOVA	Text mining	99.7	binary	28.5	0.1	16969	1754	175	17537
SYLVA	Ecology	77.9	mixed	6.2	60.58	216	13086	1308	130858

differently than in the WCCI 2006 challenge. The proportions of the data split are the same as in the WCCI 2006 challenge: the validation set is 100 times smaller than the test set to make it 10 times less accurate to compute the performances on the basis of the validation set only. The training set is ten times larger than the validation set.

The classification performance is evaluated by the Balanced Error Rate (BER), that is the average error rate of the two classes. Both validation and test set truth-values (labels) are withheld during the benchmark. The validation set serves as development test set to give on-line performance feed-back to the participants. One month before the end of the challenge, the validation set labels are made available. At the end of the benchmark, the participants send their test set results. The scores on the test set results are disclosed simultaneously to all participants after the benchmark is over.

A1.2. Data formats

For both track, the following four files in text format are used for each dataset:

dataname.param: Parameters and statistics about the data

dataname_train.labels: Binary labels (truth values of the classes) for training examples.

dataname_valid.labels: Binary validation set labels (withheld during the benchmark).

dataname_test.labels: Binary test set labels (withheld during the benchmark).

For the prior knowledge track, multi-class labels are provided in addition to the binary labels:

dataname_train.mlabels: Training multiclass labels.

dataname_valid.mlabels: Validation multiclass labels (withheld during the benchmark).

dataname_test.mlabels: Test multiclass labels (withheld during the benchmark).

Note: the challenge is about binary classification. Multi-class labels are provided as additional hints/prior knowledge.

For the agnostic track, the data matrices are provided in text format:

dataname_train.data: Training set (a sparse or a regular matrix, patterns in lines, features in columns).

dataname_valid.data: Development test set or "validation" set.

dataname_test.data: Test set.

The matrix data formats used are:

- For regular matrices: a space delimited file with a new-line character at the end of each line.
- For sparse matrices with binary values: for each line of the matrix, a space delimited list of indices of the non-zero values. A new-line character at the end of each line. In this challenge there are no sparse matrices with non-binary values.

For the prior knowledge track, the datasets are provided in various formats specified in more details in the sections devoted to the specific datasets.

The results on each dataset should be formatted in 6 ASCII files:

dataname_train.resu: ± 1 classifier outputs for training set examples (mandatory for all submission.).

dataname_valid.resu: ± 1 classifier outputs for validation set examples (mandatory for all submission.).

dataname_test.resu: ± 1 classifier outputs for final test set examples (mandatory for final submissions.)

dataname_train.conf: confidence values for training examples (optional.)

dataname_valid.conf: confidence values for validation examples (optional.)

dataname_test.conf: confidence values for test examples (optional.)

The confidence values can be the absolute discriminant values. They do not need to be normalized to look like probabilities. They will be used to compute ROC curves and Area Under such Curve (AUC).

Only entries containing results on the five datasets will qualify towards the final ranking. You can make mixed entries (using domain knowledge for some entries and preprocessed data for others). Mixed entries will be entered in the "prior knowledge" track.

A1.3. Model formats

There is also the possibility of submitting information about the models used. This is described in a separate document.

A1.4. Result rating

The scoring method retained is the *test set balanced error rate* (test_ber): the average of the class error rates (the class error rates are the error rates obtained with test examples of individual classes, using the predictions provided by the participants.)

In addition to test_ber, other statistics will be computed, but not used for scoring, including the *AUC*, i.e. the area under the ROC curve.

A1.5. Dataset A: SYLVA

A1.5.1. Topic

The task of SYLVA is to classify forest cover types. The forest cover type for 30 × 30 meter cells is obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. We brought it back to a two-class classification problem (classifying Ponderosa pine vs. everything else). The "agnostic data" consists in 216 input variables. Each pattern is composed of 4 records: 2 true records matching the target and 2 records picked at random. Thus ½ of the features are distracters. The "prior knowledge data" is identical to the "agnostic data", except that the distracters are removed and the identity of the features is revealed.

A1.5.2. Sources

A1.5.2.1. ORIGINAL OWNERS

Remote Sensing and GIS Program
Department of Forest Sciences
College of Natural Resources
Colorado State University
Fort Collins, CO 80523

(contact Jock A. Blackard, jblackard/wo_ftcol@fs.fed.us or Dr. Denis J. Dean, denis@cnr.colostate.edu)
Jock A. Blackard
USDA Forest Service
3825 E. Mulberry
Fort Collins, CO 80524 USA
jblackard/wo_ftcol@fs.fed.us

Dr. Denis J. Dean
Associate Professor
Department of Forest Sciences
Colorado State University
Fort Collins, CO 80523 USA
denis@cnr.colostate.edu

Dr. Charles W. Anderson
Associate Professor
Department of Computer Science
Colorado State University
Fort Collins, CO 80523 USA
anderson@cs.colostate.edu

ACKNOWLEDGEMENTS, COPYRIGHT INFORMATION, AND AVAILABILITY

Reuse of this database is unlimited with retention of copyright notice for Jock A. Blackard and Colorado State University.

A1.5.2.2. DONOR OF DATABASE

This version of the database was prepared for the WCCI 2006 performance prediction challenge and the IJCNN 2007 agnostic learning vs. prior knowledge challenge by Isabelle Guyon, 955 Creston Road, Berkeley, CA 94708, USA (isabelle@clopinet.com).

A1.5.2.3. DATE RECEIVED

August 28, 1998, UCI Machine Learning Repository, under the name Forest Cover Type.

A1.5.2.4. DATE PREPARED FOR THE CHALLENGES

June 2005–September 2006.

A1.5.3. Past usage

Blackard, Jock A. 1998. “Comparison of Neural Networks and Discriminant Analysis in Predicting Forest Cover Types.” Ph.D. dissertation. Department of Forest Sciences. Colorado State University. Fort Collins, Colorado.

Classification performance with first 11,340 records used for training data, next 3,780 records used for validation data, and last 565,892 records used for testing data subset: – 70% backpropagation – 58% Linear Discriminant Analysis

A1.5.4. Experimental design

The original data comprises a total of 581012 instances (observations) grouped in 7 classes (forest cover types) and having 54 attributes corresponding to 12 measures (10 quantitative variables, 4 binary wilderness areas and 40 binary soil type variables). The actual forest cover type for a given observation (30×30 meter cell) was determined from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. Independent variables were derived from data originally obtained from US Geological Survey (USGS) and USFS data. Data is in raw form (not scaled) and contains binary (0 or 1) columns of data for qualitative independent variables (wilderness areas and soil types).

A1.5.4.1. VARIABLE INFORMATION

Table A1.2 gives the variable name, variable type, the measurement unit and a brief description. The forest cover type is the classification problem. The order of this listing corresponds to the order of numerals along the rows of the database.

A1.5.4.2. CODE DESIGNATIONS

Wilderness Areas:

1 — Rawah Wilderness Area

2 — Neota Wilderness Area

3 — Comanche Peak Wilderness Area

4 — Cache la Poudre Wilderness Area

Soil Types:

Table A1.2: Variable Information

Name	**Data Type**	**Measurement**	**Description**
Elevation	quantitative	meters	Elevation in meters
Aspect	quantitative	azimuth	Aspect in degrees azimuth
Slope	quantitative	degrees	Slope in degrees
Horizontal_Distance_To_Hydrology	quantitative	meters	Horz Dist to nearest surface water features
Vertical_Distance_To_Hydrology	quantitative	meters	Vert Dist to nearest surface water features
Horizontal_Distance_To_Roadways	quantitative	meters	Horz Dist to nearest roadway
Hillshade_9am	quantitative	0 to 255 index	Hillshade index at 9am, summer solstice
Hillshade_Noon	quantitative	0 to 255 index	Hillshade index at noon, summer soltice
Hillshade_3pm	quantitative	0 to 255 index	Hillshade index at 3pm, summer solstice
Horizontal_Distance_To_Fire_Points	quantitative	meters	Horz Dist to nearest wildfire ignition points
Wilderness_Area (4 binary columns)	qualitative	0 (absence) or 1 (presence)	Wilderness area designation
Soil_Type (40 binary columns)	qualitative	0 (absence) or 1 (presence)	Soil Type designation
Cover_Type (7 types)	integer	1 to 7	Forest Cover Type designation

1 to 40 : based on the USFS Ecological Landtype Units for this study area.

Forest Cover Types:

1 — Spruce/Fir

2 — Lodgepole Pine

3 — Ponderosa Pine

4 — Cottonwood/Willow

5 — Aspen

6 — Douglas-fir

7 — Krummholz

A1.5.4.3. CLASS DISTRIBUTION

Number of records of Spruce-Fir:	211840
Number of records of Lodgepole Pine:	283301
Number of records of Ponderosa Pine:	35754
Number of records of Cottonwood/Willow:	2747
Number of records of Aspen:	9493
Number of records of Douglas-fir:	17367
Number of records of Krummholz:	20510
Total records:	581012

Data preprocessing and data split We carved a binary classification task out these data. We decided to separate Ponderosa pine from all others. To disguise the data and render the task more challenging for the "agnostic track", we created patterns containing the concatenation of 4 patterns: two of the target class and two randomly chosen from either class. In this way there are pairs of redundant features and ½ of the features are non-informative. The "prior knowledge data" does not contain the distracters. The multi-class label information is provided with the "prior knowledge data" as a 2-digit number representing for each pattern the combination of 2 records used. All the examples of the positive class have code "33" (two Ponderosa pine records), others have different 2-digit numbers.

A1.5.5. Number of examples and class distribution

A1.5.6. Type of input variables and variable statistics

All variables are integer quantized on 1000 levels. There are no missing values. The data is not very sparse, but for data compression reasons, we thresholded the values. Approximately 78% of the variable values are zero. The data was saved as a dense matrix.

A1.5.7. Baseline results

The best entry in the "Performance prediction challenge" had a test_ber=0.53%.

Table A1.3: Prior knowledge data

	Positive ex.	**Negative ex.**	**Total**	**Check sum**
Training set	805	12281	13086	118996108
Validation set	81	1228	1309	11904801
Test set	8052	122805	130857	1191536355
All	8938	136314	145252	1322437264

Table A1.4: Agnostic data

	Positive ex.	**Negative ex.**	**Total**	**Check sum**
Training set	**805**	**12281**	**13086**	**238271607**
Validation set	81	1228	1309	23817234
Test set	8052	122805	130857	2382779242
All	8938	136314	145252	2644868083

Table A1.5: Prior knowledge data

Real variables	**Random probes**	**Total**
108	0	108

Table A1.6: Agnostic data

Real variables	**Random probes**	**Total**
108	108	216

A1.6. Dataset B: GINA

A1.6.1. Topic

The task of GINA is handwritten digit recognition. We chose the problem of separating the odd numbers from even numbers. We use 2-digit numbers. Only the unit digit is informative for that task, therefore at least ½ of the features are distracters. This is a two-class classification problem with sparse continuous input variables, in which each class is composed of several clusters. It is a problems with heterogeneous classes.

A1.6.2. Sources

A1.6.2.1. ORIGINAL OWNERS

The data set was constructed from the MNIST data that is made available by Yann LeCun of the NEC Research Institute at http://yann.lecun.com/exdb/mnist/. The digits have been size-normalized and centered in a fixed-size image of dimension 28×28. We show examples of digits in Figure A1.1.

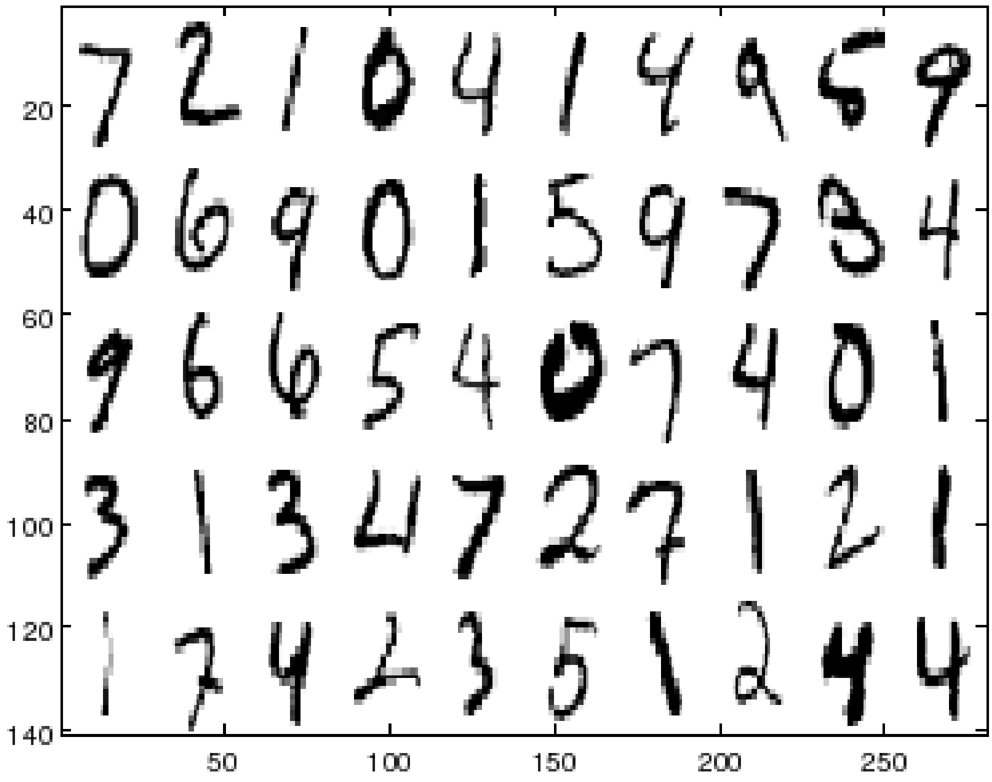


Figure A1.1: Examples of digits from the MNIST database.

Table A1.7: Number of examples in the original data

Digit	0	1	2	3	4	5	6	7	8	9	Total
Training	5923	6742	5958	6131	5842	5421	5918	6265	5851	5949	60000
Test	980	1135	1032	1010	982	892	958	1028	974	1009	10000
Total	6903	7877	6990	7141	6824	6313	6876	7293	6825	6958	70000

A1.6.2.2. DONOR OF DATABASE

This version of the database was prepared for the WCCI 2006 performance prediction challenge and the IJCNN 2007 agnostic learning vs. prior knowledge challenge by Isabelle Guyon, 955 Creston Road, Berkeley, CA 94708, USA (isabelle@clopinet.com).

A1.6.2.3. DATE PREPARED FOR THE CHALLENGES

June 2005–September 2006.

A1.6.3. Past usage

Many methods have been tried on the MNIST database, in its original data split (60,000 training examples, 10,000 test examples, 10 classes.) Table A1.8 is an abbreviated list from http://yann.lecun.com/exdb/mnist/.

Table A1.8: Abbreviated List

METHOD	TEST ERROR RATE (%)
linear classifier (1-layer NN)	12.0
linear classifier (1-layer NN) [deskewing]	8.4
pairwise linear classifier	7.6
K-nearest-neighbors, Euclidean	5.0
K-nearest-neighbors, Euclidean, deskewed	2.4
40 PCA + quadratic classifier	3.3
1000 RBF + linear classifier	3.6
K-NN, Tangent Distance, 16x16	1.1
SVM deg 4 polynomial	1.1
Reduced Set SVM deg 5 polynomial	1.0
Virtual SVM deg 9 poly [distortions]	0.8
2-layer NN, 300 hidden units	4.7
2-layer NN, 300 HU, [distortions]	3.6
2-layer NN, 300 HU, [deskewing]	1.6
2-layer NN, 1000 hidden units	4.5
2-layer NN, 1000 HU, [distortions]	3.8
3-layer NN, 300+100 hidden units	3.05
3-layer NN, 300+100 HU [distortions]	2.5
3-layer NN, 500+150 hidden units	2.95
3-layer NN, 500+150 HU [distortions]	2.45
LeNet-1 [with 16x16 input]	1.7
LeNet-4	1.1

Table A1.8: Abbreviated List (continued)

METHOD	TEST ERROR RATE (%)
LeNet-4 with K-NN instead of last layer	1.1
LeNet-4 with local learning instead of ll	1.1
LeNet-5, [no distortions]	0.95
LeNet-5, [huge distortions]	0.85
LeNet-5, [distortions]	0.8
Boosted LeNet-4, [distortions]	0.7
K-NN, shape context matching	0.67

A1.6.3.1. REFERENCE:

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278-2324, November 1998. http://yann.lecun.com/exdb/publis/index.html#lecun-98

The dataset restricted to a selection of digits "4" and "9" was used in the NIPS 2003 feature selection challenge http://clopinet.com/isabelle/Projects/NIPS2003/ and http://www.nipsfsc.ecs.soton.ac.uk/, under the name GISETTE.

A1.6.4. Experimental design

To construct the "agnostic" dataset, we performed the following steps:

- We removed the pixels that were 99% of the time white. This reduced the original feature set of 784 pixels to 485.
- The original resolution (256 gray levels) was kept.
- In spite of the fact that the data are rather sparse (about 30% of the values are non-zero), we saved the data as a dense matrix because we found that it can be compressed better in this way (to 19 MB.)
- The feature names are the (i, j) matrix coordinates of the pixels (in a 28×28 matrix.)
- We created 2 digit numbers by dividing the datasets into to parts and pairing the digits at random.
- The task is to separate odd from even numbers. The digit of the tens being not informative, the features of that digit act as distracters.

To construct the "prior" dataset, we went back to the original data and fetched the "informative" digit in its original representation. Therefore, this data representation consists in a vector of concatenating the lines of a 28×28 pixel map.

A1.6.5. Number of examples and class distribution

Table A1.9: Prior knowledge data

	Positive ex.	Negative ex.	Total	Check sum
Training set	1550	1603	3153	82735983
Validation set	155	160	315	8243382
Test set	15504	16028	31532	825458881
All	17209	17791	35000	916438246

Table A1.10: Agnostic data

	Positive ex.	Negative ex.	Total	Check sum
Training set	1550	1603	3153	164947945
Validation set	155	160	315	16688946
Test set	15504	16028	31532	1646492864
All	17209	17791	35000	1828129755

A1.6.6. Type of input variables and variable statistics

Table A1.11: Prior knowledge data

Real variables	Random probes	Total
784	0	784

Table A1.12: Agnostic data

Real variables	Random probes	Total
485	485	970

All variables are **integer** quantized on 256 levels. There are **no missing values**. The data is rather **sparse**. Approximately 69% of the entries are zero for the agnostic data. The data was saved as a **dense** matrix, because it compresses better in that format.

A1.6.7. Baseline results

The best entry of the “performance prediction challenge” obtained a test_ber=2.88%.

A1.7. Dataset C: NOVA

A1.7.1. Topic

The task of NOVA is text classification from the 20-Newsgroup data. We selected the separation of politics and religion topics from all the other topics. This is a two-class classification problem. The raw data comes as text files for the “prior knowledge” track. The preprocessed data for the “agnostic” track is a sparse binary representation using a bag-of-word representation with a vocabulary of approximately 17000 words.

A1.7.2. Sources

A1.7.2.1. ORIGINAL OWNERS

Tom Mitchell
School of Computer Science
Carnegie Mellon University
tom.mitchell@cmu.edu

Available from the UCI machine learning repository. The version we are using for the agnostic track was preprocessed by Ron Bekkerman http://www.cs.technion.ac.il/~ronb/thesis.html into the "bag-of-words" representation.

A1.7.2.2. DONOR OF DATABASE

This version of the database was prepared for the WCCI 2006 performance prediction challenge and the IJCNN 2007 agnostic learning vs. prior knowledge challenge by Isabelle Guyon, 955 Creston Road, Berkeley, CA 94708, USA (isabelle@clopinet.com).

A1.7.2.3. DATE PREPARED FOR THE CHALLENGES

June 2005 – September 2006.

A1.7.3. Past usage

T. Mitchell. Machine Learning, McGraw Hill, 1997.

T. Joachims (1996). A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization, Computer Science Technical Report CMU-CS-96-118. Carnegie Mellon University.

Ron Bekkerman, Ran El-Yaniv, Naftali Tishby, and Yoad Winter. Distributional Word Clusters vs. Words for Text Categorization. JMLR 3(Mar):1183-1208, 2003.

A1.7.4. Experimental design

We selected 8 newsgroups relating to politics or religion topics as our positive class (Table A1.13)

For the "prior knowledge" data, we kept the original text, but we removed the header. The format of the data files is as follows. Each entry corresponding to a newsgroup message is encoded as:

- 1st line: Subject: xxx
- 2nd line: Lines: yyy
- 3rd line: Blank
- The message with yyy lines.
- $$$$

Each entry corresponds to an example.

For the "agnostic" data, the vocabulary selection includes the following filters:

- remove words containing digits and convert to lowercase
- remove words appearing less than twice in the whole dataset.
- remove short words with less than 3 letters.
- exclude $\approx$ 2000 words found frequently in all documents.
- truncate the words at a max of 7 letters.

Table A1.13: Twenty newsgroup database.

Newsgroup	**Number of examples**
alt.atheism	1114
comp.graphics	1002
comp.os.ms-windows.misc	1000
comp.sys.ibm.pc.hardware	1028
comp.sys.mac.hardware	1002
comp.windows.x	1000
misc.forsale	1005
rec.autos	1004
rec.motorcycles	1000
rec.sport.baseball	1000
rec.sport.hockey	1000
sci.crypt	1000
sci.electronics	1000
sci.med	1001
sci.space	1000
soc.religion.christian	997
talk.politics.guns	1008
talk.politics.mideast	1000
talk.politics.misc	1163
talk.religion.misc	1023

A1.7.5. Number of examples and class distribution

Table A1.14: Prior knowledge data

	Positive ex.	Negative ex.	Total
Training set	499	1255	1754
Validation set	50	125	175
Test set	4990	12547	17537
All	5539	13927	19466

Table A1.15: Agnostic data

	Positive ex.	Negative ex.	Total	Check sum
Training set	499	1255	1754	103583
Validation set	50	125	175	9660
Test set	4990	12547	17537	984667
All	5539	13927	19466	1097910

A1.7.6. Type of input variables and variable statistics (agnostic data only)

Real variables	Random probes	Total
16969	0	16969

All variables are binary. There are no missing values. The data is very sparse. Over 99% of the entries are zero. The data was saved as a sparse-binary matrix.

A1.7.7. Baseline results

The best performance of the "Performance prediction challenge" was test_ber=4.44%.

A1.8. Dataset D: HIVA

A1.8.1. Topic

The task of HIVA is to predict which compounds are active against the AIDS HIV infection. The original data has 3 classes (active, moderately active, and inactive). We brought it back to a two-class classification problem (active vs. inactive). The problem is therefore to relate structure to activity (a QSAR=quantitative structure-activity relationship problem) to screen new compounds before actually testing them (a HTS=high-throughput screening problem.) The molecules in the original data are described by their chemical formula. We provide additionally the 3d structure for the "prior knowledge" track. For the "agnostic track" we represented the data as 2000 sparse binary input variables. The variables represent properties of the molecule inferred from its structure.

A1.8.2. Sources

A1.8.2.1. ORIGINAL OWNERS

The data is made available by the National Cancer Institute (NCI), via the DTP AIDS Antiviral Screen program at: `http://dtp.nci.nih.gov/docs/aids/aids_data.html`.

The DTP AIDS Antiviral Screen has checked tens of thousands of compounds for evidence of anti-HIV activity. Available are screening results and chemical structural data on compounds that are not covered by a confidentiality agreement.

A1.8.2.2. DONOR OF DATABASE

This version of the database was prepared for the WCCI 2006 performance prediction challenge and the IJCNN 2007 agnostic learning vs. prior knowledge challenge by Isabelle Guyon, 955 Creston Road, Berkeley, CA 94708, USA (`isabelle@clopinet.com`).

A1.8.2.3. DATE PREPARED FOR THE CHALLENGES

June 2005–September 2006.

A1.8.3. Past usage

An earlier release of the database was uses in an Equbits case study: `http://www.limsfinder.com/community/articles_comments.php?id=1553_0_2_0_C75`. The feature set was obtained by a different method.

A1.8.4. Experimental design

The screening results of the May 2004 release containing the screening results for 43,850 compounds were used. The results of the screening tests are evaluated and placed in one of three categories:

- CA - Confirmed active
- CM - Confirmed moderately active
- CI - Confirmed inactive

We converted this into a two-class classification problem: Inactive (CI) vs. Active (CA or CM.)

Chemical structural data for 42,390 compounds was obtained from the web page. It was converted to structural features by the program ChemTK version 4.1.1, Sage Informatics LLC. Four compounds failed parsing. The 1617 features selected include:

- unbranched_fragments: 750 features
- pharmacophores: 495 features
- branched_fragments: 219 features
- internal_fingerprints: 132 features
- ring_systems: 21 features

Only binary features having a total number of ones larger than 100 (> 400 for unbranched fragments) and at least 2% of ones in the positive class were retained. In all cases, the default program settings were used to generate keys (except for the pharmacophores for which "max number of pharmacophore points" was set to 4 instead of 3; the pharmacophore keys for Hacc, Hdon, ExtRing, ExtArom, ExtAliph were generated, as well as those for Hacc, Hdon, Neg, Pos.) The keys were then converted to attributes.

We briefly describe the attributes/features:

Branched fragments: each fragment is constructed through an "assembly" of shortest-path unbranched fragments, where each of the latter is required to be bounded by two atoms belonging to one or more pre-defined "terminal-atom".

Unbranched fragments: unique non-branching fragments contained in the set of input molecules.

Ring systems: A ring system is defined as any number of single or fused rings connected by an unbroken chain of atoms. The simplest example would be either a single ring (e.g., benzene) or a single fused system (e.g., naphthalene).

Pharmacophores: ChemTK uses a type of pharmacophore that measures distance via bond connectivity rather than a typical three-dimensional distance. For instance, to describe a hydrogen-bond acceptor and hydrogen-bond donor separated by five connecting bonds, the corresponding key string would be "HAcc.HDon.5". The pharmacophores were generated from the following features:

Neg. Explicit negative charge.

Pos. Explicit positive charge.

HAcc. Hydrogen-bond acceptor.

HDon. Hydrogen-bond donor.

ExtRing. Ring atom having a neighbor atom external to the ring.

ExtArom. Aromatic ring atom having a neighbor atom external to the ring.

ExtAliph. Aliphatic ring atom having a neighbor atom external to the ring.

Internal fingerprints: small, fixed catalog of pre-defined queries roughly similar to the MACCS key set developed by MDL.

We matched the compounds in the structural description files and those in the compound activity file, using the NSC id number. We ended up with 42678 examples.

A1.8.5. Data format, number of examples and class distribution

Table A1.16: Prior knowledge data

	Positive ex.	Negative ex.	Total
Training set	135	3710	3845
Validation set	14	370	384
Test set	1354	37095	38449
All	1503	41175	42678

The raw data is formatted in the MDL-SD format (hiva_train.sd, hiva_valid.sd, hiva_test.sd). It represents the 3-dimensional structure of the molecule. It was produced from the chemical formulas by the program Corina (`http://www.molecular-networks.de/software/corina/index.html`). Each record is separated by `$$$$`. One record contains:

Header block — line 1: molecule name; line 2: molecule header; line 3: comment line.

Connection Table — count line in Fortran format 2I3; atom block: One line per atom, including the atomic co-ordinates (X, Y, Z), symbol (SYM), mass difference for the isotope (MASS), formal charge (CHARGE), and stereo parity indicator (STEREO); bond block: One line per bond specifying the two atoms connected by the bond (ATOM1, ATOM2), the bond type (TYPE), stereochemistry (BONDST), and topology (TOPOL).

Data Block — data header: Indicated by the greater than symbol >; data: may extend over multiple lines, up to a maximum of 200 characters in total (up to 80 characters per line); black line.

Table A1.17: Agnostic data

	Positive ex.	**Negative ex.**	**Total**	**Check sum**
Training set	135	3710	3845	564954
Validation set	14	370	384	56056
Test set	1354	37095	38449	5674217
All	1503	41175	42678	6295227

A1.8.6. Type of input variables and variable statistics (agnostic data only)

Real variables	**Random probes**	**Total**
1617	0	1617

All variables are binary. The data was saved as a non-spase matrix, even though it is 91% sparse because dense matrices load faster in Matlab and the ASCII format compresses well.

A1.8.7. Baseline results

The best entry of the "Performance Prediction Challenge" had a test_ber=27.56%.

A1.9. Dataset E: ADA

A1.9.1. Topic

The task of ADA is to discover high revenue people from census data. This is a two-class classification problem. The raw data from the census bureau is known as the Adult database in the UCI machine-learning repository. It contains continuous, binary and categorical variables. The "prior knowledge track" has access to the original features and their identity. The agnostic track has access to a preprocessed numeric representation eliminating categorical variables.

A1.9.2. Sources

A1.9.2.1. ORIGINAL OWNERS

This data was extracted from the census bureau database found at http://www.census.gov/ftp/pub/DES/www/welcome.html

Donor: Ronny Kohavi and Barry Becker,
Data Mining and Visualization
Silicon Graphics.
e-mail: ronnyk@sgi.com for questions.

The information below is exerpted from the UCI machine learning repository:

> Extraction was done by Barry Becker from the 1994 Census database. The prediction task is to determine whether a person makes over 50K a year. The attributes are:
>
> **age:** continuous.
>
> **workclass:** Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked. fnlwgt: continuous.
>
> **education:** Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
>
> **education-num:** continuous.
>
> **marital-status:** Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
>
> **occupation:** Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
>
> **relationship** Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
>
> **race:** White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
>
> **sex:** Female, Male.
>
> **capital-gain:** continuous.
>
> **capital-loss:** continuous.
>
> **hours-per-week:** continuous.
>
> **native-country:** United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.
>
> **income:** > 50K, ≤ 50K.
>
> Split into train-test using MLC++ GenCVFiles (2/3, 1/3 random).
> 48842 instances, mix of continuous and discrete (train=32561, test=16281)
> 45222 if instances with unknown values are removed (train=30162, test=15060)

Duplicate or conflicting instances : 6
Class probabilities for adult.all file
Probability for the label "$>$ 50K" : 23.93% / 24.78% (without unknowns)
Probability for the label "$\leq$ 50K" : 76.07% / 75.22% (without unknowns)
Description of fnlwgt (final weight)
The weights on the CPS files are controlled to independent estimates of the civilian noninstitutional population of the US. These are prepared monthly for us by Population Division here at the Census Bureau. We use 3 sets of controls. People with similar demographic characteristics should have similar weights.

A1.9.2.2. DONOR OF DATABASE

This version of the database was prepared for the WCCI 2006 performance prediction challenge and the IJCNN 2007 agnostic learning vs. prior knowledge challenge by Isabelle Guyon, 955 Creston Road, Berkeley, CA 94708, USA (isabelle@clopinet.com).

A1.9.2.3. DATE PREPARED FOR THE CHALLENGES

June 2005–September 2006.

A1.9.3. Past usage

First cited in:

Ron Kohavi. Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996.

Error Accuracy reported as follows, (after removal of unknowns from train/test sets):

C4.5	: 84.46 ± 0.30
Naive-Bayes	: 83.88 ± 0.30
NBTree	: 85.90 ± 0.28

The following algorithms were later run with the following error rates, all after removal of unknowns and using the original train/test split. All these numbers are straight runs using MLC++ with default values.

	Algorithm	Error
1	C4.5	15.54
2	C4.5-auto	14.46
3	C4.5 rules	14.94
4	Voted ID3 (0.6)	15.64
5	Voted ID3 (0.8)	16.47
6	T2	16.84
7	1R	19.54
8	NBTree	14.10
9	CN2	16.00
10	HOODG	14.82
11	FSS Naive Bayes	14.05
12	IDTM (Decision table)	14.46
13	Naive-Bayes	16.12
14	Nearest-neighbor (1)	21.42
15	Nearest-neighbor (3)	20.35
16	OC1	15.04
17	Pebls	Crashed. Unknown why (bounds WERE increased)

Note: The performances reported are error rates, not BER. We tried to reproduce these performances and obtained 15.62% error with a linear ridge regression classifier. The performances slightly degraded when we tried to group features (15.67% when we replace the country code by a binary US/nonUS value and 16.40% with further reduction to 33 features.)

A1.9.4. Experimental design

To generate the "agnostic track" data, we performed the following steps:

- Convert the features to 14 numeric values a $1 \ldots n$.
- Convert the numeric values to binary codes (a vector of n zeros with value one at the a^{th} position. This results in 88 features. The missing values get an all zero vector.
- Downsize the number of features to 48 by replacing the country code by a binary US/nonUS feature.
- Randomize the feature and pattern order.
- Remove the entries with missing values for workclass.

Table A1.18: Features of the ADA datasets.

Feature name	min	max	numval	comments
age	0.19	1	continuous	No missing value.
workclass_Private	0	1	2	2799 missing values (corresponding entries removed.)
workclass_Self_emp_not_inc	0	1	2	
workclass_Self_emp_inc	0	1	2	
workclass_Federal_gov	0	1	2	
workclass_Local_gov	0	1	2	
workclass_State_gov	0	1	2	
workclass_Without_pay	0	1	2	
workclass_Never_worked	0	1	2	

Table A1.18: Features of the ADA datasets (continued).

Feature name	**min**	**max**	**numval**	**comments**
fnlwgt	0.008	1	continuous	No missing value.
educationNum	0.06	1	16	Number corresponding to 16 discrete levels of education
maritalStatus_Married_civ_spouse	0	1	2	No missing value.
maritalStatus_Divorced	0	1	2	
maritalStatus_Never_married	0	1	2	
maritalStatus_Separated	0	1	2	
maritalStatus_Widowed	0	1	2	
maritalStatus_Married_spouse_absent	0	1	2	
maritalStatus_Married_AF_spouse	0	1	2	
occupation_Tech_support	0	1	2	2809 missing values (corresponding entries removed.)
occupation_Craft_repair	0	1	2	
occupation_Other_service	0	1	2	
occupation_Sales	0	1	2	
occupation_Exec_managerial	0	1	2	
occupation_Prof_specialty	0	1	2	
occupation_Handlers_cleaners	0	1	2	
occupation_Machine_op_inspct	0	1	2	
occupation_Adm_clerical	0	1	2	
occupation_Farming_fishing	0	1	2	
occupation_Transport_moving	0	1	2	
occupation_Priv_house_serv	0	1	2	
occupation_Protective_serv	0	1	2	
occupation_Armed_Forces	0	1	2	
relationship_Wife	0	1	2	No missing value.
relationship_Own_child	0	1	2	
relationship_Husband	0	1	2	
relationship_Not_in_family	0	1	2	
relationship_Other_relative	0	1	2	
relationship_Unmarried	0	1	2	
race_White	0	1	2	No missing value.
race_Asian_Pac_Islander	0	1	2	
race_Amer_Indian_Eskimo	0	1	2	
race_Other	0	1	2	
race_Black	0	1	2	
sex	0	1	2	0=female, 1=male. No missing value.
capitalGain	0	1	continuous	No missing value.
capitalLoss	0	1	continuous	No missing value.
hoursPerWeek	0.01	1	continuous	No missing value.
nativeCountry	0	1	2	0=US, 1=non-US. 857 missing values replaced by 1.

A1.9.5. Data format, number of examples and class distribution

Table A1.19: Prior knowledge dataset

	Positive ex.	**Negative ex.**	**Total**
Training set	1029	3118	4147
Validation set	103	312	415
Test set	10290	31181	41471
All	11422	34611	46033

The data are stored in coma-separated files (ada_train.csv, ada_valid.csv, and ada_est.csv).

Table A1.20: Agnostic dataset

	Positive ex.	**Negative ex.**	**Total**	**Check sum**
Training set	1029	3118	4147	6798109
Validation set	103	312	415	681151
Test set	10290	31181	41471	67937286
All	11422	34611	46033	75416546

A1.9.6. Type of input variables and variable statistics

Table A1.21: Prior knowledge dataset

Real variables	**Random probes**	**Total**
14	0	14

Six variables are continuous, two are binary, the others are categorical. The missing values were eliminated.

Table A1.22: Agnostic dataset

Real variables	**Random probes**	**Total**
48	0	48

Six variables are continuous, the others are binary. There are no missing values. The data is 80% sparse. The data was saved as a dense matrix because once compressed it makes almost no difference and it loads much faster.

A1.9.7. Baseline results

The best entry in the Performance Prediction Challenge had a test_ber=16.96%.

Appendix B

Fact Sheets

Appendix B1

Performance Prediction Challenge

B1.1. LogitBoost with trees

Contact

Roman Werner Lutz, Seminar for Statistics, ETH Zurich, CH-8092 Zurich, Switzerland, lutz@stat.math.ethz.ch

Acronym of your best entry

LB tree mix cut adapted

Reference

LogitBoost with Trees Applied to the WCCI 2006 Performance Prediction Challenge, Roman Werner Lutz, In Proceedings IJCNN06, to appear.

Method

As preprocessing we used PCA for Nova with centered and scaled variables and took the first 400 principal components. No preprocessing was used for the other datasets. Then we applied LogitBoost with trees of prefixed depth. The number of iterations, the tree depth (in each iteration a tree of the same depth is fitted) and the BER guess were chosen/computed by 10-fold cross-validation. Shrinkage was added to make LogitBoost more stable: in each iteration only a fraction ν (0.3, 0.1 or 0.03) of the fitted learner was added. ν was chosen by visual inspection of the cross-validated BER curve (as a function of the boosting iteration). As a result, LogitBoost yielded probabilities of class membership for each sample. The cut point for the final classification was the proportion of class $+1$ in the data.

For the second entry we used the Wilcoxon test (for continuous variables) and the Fisher exact test (for binary variables) for variable pre-selection (variables with a p-value above 0.1 were dropped). For the third entry we averaged the predicted probabilities of LogitBoot with and without variable pre-selection. For the fourth entry we made an intercept adaption (on the logit scale) so that the average of the predicted probabilities on the test set equals the proportion of class $+1$ in the data.

Results

In the challenge, we rank 1^{st} as a group and our best entry (our fourth) is the 1^{st}, according to the average rank computed by the organizers. Our method is quite simple: no preprocessing is needed (except for Nova) and the tuning parameters are chosen by cross-validation. Additionally, LogitBoost with trees does variable selection, because in each iteration only a few variables are chosen.

Dataset	Our best entry					The challenge best entry				
	Test AUC	Test BER	BER guess	Guess error	Test score (rank)	Test AUC	Test BER	BER guess	Guess error	Test score (rank)
ADA	0.8304	0.1696	0.1550	0.0146	0.1843 (3)	0.9149	0.1723	0.1650	0.0073	0.1793 (1)
GINA	0.9639	0.0361	0.0388	0.0027	0.0386 (5)	0.9712	0.0288	0.0305	0.0017	0.0302 (1)
HIVA	0.7129	0.2871	0.2700	0.0171	0.3029 (8)	0.7671	0.2757	0.2692	0.0065	0.2797 (1)
NOVA	0.9542	0.0458	0.0503	0.0045	0.0499 (8)	0.9914	0.0445	0.0436	0.0009	0.0448 (1)
SYLVA	0.9937	0.0063	0.0058	0.0005	0.0067 (7)	0.9991	0.0061	0.0060	0.0001	0.0062 (1)
Overall	0.8910	0.1090	0.1040	0.0079	0.1165(6.2)	0.8910	0.1090	0.1040	0.0079	0.1165(6.2)

Code

Our implementation was done in R.

Keywords

PCA, Wilcoxon test, Fisher exact test, LogitBoost, trees of fixed depth, 10-fold cross-validation, shrinkage.

B1.2. Weighted LS-SVM + Leave-One-Out Cross-Validation + Repeated Hold-Out

Contact

Gavin Cawley, University of East Anglia, Norwich, United Kingdom, gcc@cmp.uea.ac.uk

Acronym of your best entry

Final #2

Reference

G. C. Cawley, "Leave-one-out cross-validation based model selection criteria for weighted LS-SVMs", Proceedings of the International Joint Conference on Neural Networks (IJCNN-2006), to appear.
http://theoval.cmp.uea.ac.uk/~gcc/publications/pdf/ijcnn2006a.pdf

Method

Preprocessing: All continuous features are standardised, also for ADA a log transform used on features 1 and 3 and thresholding of features 4 and 5. For the SYLVA dataset, features 11 and 12 are never positive for positive examples; this observation was used to reduce the number of training patterns so that the LS-SVM could be applied directly.

Feature selection: No feature selection was used, other than deleting constant features in the NOVA benchmark. Regularisation was the only mechanism used to avoid over-fitting.

Classification: Least-squares support vector machines (LS-SVMs), with linear, quadratic, cubic, Boolean and radial basis function (RBF) kernel functions. LS-SVMs with and without a bias term were evaluated. The LS-SVMs could optionally be weighted to equalise the importance of positive and negative patterns (as the balanced error rate is used as the primary performance indicator). Validation set targets were used in training and model selection.

Model selection: The optimisation of regularisation and kernel parameters was achieved by minimising leave-one-out cross-validation based estimates of generalisation performance via the Nelder-Mead simplex method. A variety of model selection criterion were investigated including sum-of-squares error (i.e. Allen's PRESS statistic), hinge loss, squared hinge loss, a smoothed approximation of the error rate and the smoothed Wilcoxon-Mann-Whitney statistic (i.e. the area under the ROC curve). The selection criterion could optionally be weighted to compensate for the disparity in class frequencies. The final choice of model, including the choice of kernel, use of a bias term, use of weighting in training and/or model selection and the choice of model selection criterion were all made by minimising the leave-one-out BER. A total of 180 experiments were conducted; this was somewhat computationally expensive!

Performance prediction: The test BER was estimated via 100 random 90%/10% training/test partitions of the available data, with model selection performed independently in each trial in order to avoid selection bias.

Results

This entry is a joint winner of the competition, having the lowest average test score and finishing second in terms of average rank. This entry also has the lowest overall guess error of any submission and the second highest overall test AUC. It is interesting that the models performed so well on the HIVA and NOVA benchmarks, given that no feature selection was used. It is reassuring that regularisation is effective in avoiding over-fitting, given a good value for the regularisation parameter. Also, the leave-one-out procedure is often (rightly) criticised as having a high variance, so it is interesting that it performed so well. This is probably because there were relatively few degrees of freedom to be optimised in model selection. If leave-one-out cross-validation were used in feature selection, there would probably be a much higher degree of over-fitting. Model details are as follows:

- ADA: Unweighted LS-SVM with bias, Radial Basis Function kernel, Wilcoxon-Mann-Whitney model selection criterion.
- GINA: Unweighted LS-SVM without bias, inhomogeneous cubic kernel, unweighted smoothed error rate model selection criterion.
- HIVA: Unweighted LS-SVM with bias, inhomogeneous quadratic kernel, Wilcoxon-Mann-Whitney model selection criterion.
- NOVA: Weighted LS-SVM with bias, linear kernel, weighted mean-squared error model selection.
- SYLVA: Weighted LS-SVM with bias, inhomogeneous cubic kernel, Wilcoxon-Mann-Whitney model selection.

Dataset	Our best entry					The challenge best entry				
	Test AUC	**Test BER**	**BER guess**	**Guess error**	**Test score (rank)**	**Test AUC**	**Test BER**	**BER guess**	**Guess error**	**Test score (rank)**
ADA	0.8965	0.1845	0.1742	0.0103	0.1947 (13)	0.9149	0.1723	0.1650	0.0073	0.1793 (1)
GINA	0.9900	0.0461	0.0470	0.0009	0.0466 (13)	0.9712	0.0288	0.0305	0.0017	0.0302 (1)
HIVA	0.7464	0.2804	0.2776	0.0028	0.2814 (2)	0.7671	0.2757	0.2692	0.0065	0.2797 (1)
NOVA	0.9914	0.0445	0.0470	0.0025	0.0464 (3)	0.9914	0.0445	0.0436	0.0009	0.0448 (1)
SYLVA	0.9990	0.0067	0.0065	0.0002	0.0067 (8)	0.9991	0.0061	0.0060	0.0001	0.0062 (1)
overall	0.9246	0.1124	0.1105	0.0034	0.1152 (7.8)	0.8910	0.1090	0.1040	0.0079	0.1165 (6.2)

Code

The LS-SVM, leave-one-out model selection, Nelder-Mead simplex optimisation methods were all implemented by the author in MATLAB. Extensive use was made of automatically generated scripts to run the individual experiments. A demonstration of the approach used will shortly be made available from `http://theoval.cmp.uea.ac.uk/~gcc/matlab/default.html#loo`.

Keywords

standardisation, no feature selection, kernel method, least-squares support vector machine, L2 norm regularisation, leave-one-out model selection, pattern weighting, Nelder-Mead simplex, repeated hold-out validation.

B1.3. Bayesian Neural Networks for the Performance Prediction Challenge

Contact

Radford M. Neal, University of Toronto, `radford@stat.utoronto.ca`

Acronym of your best entry

Bayesian Neural Networks

References

The general methods I used are described in my book:

Neal, R. M. (1996) *Bayesian Learning for Neural Networks,* Lecture Notes in Statistics No. 118, New York: Springer-Verlag.

The detailed models used are similar to those I used for two other prediction competitions, described in :

Neal, R. M. and Zhang, J. (2006) “High Dimensional Classification with Bayesian Neural Networks and Dirichlet Diffusion Trees”, in I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh (editors) *Feature Extraction, Foundations and Applications,* Physica-Verlag, Springer

Neal, R. M. (2006) "Classification with Bayesian Neural Networks", in J. Quinonero-Candela, I. Dagan, B. Magnini, and F. D'Alche-Buc (editors) *Evaluating Predictive Uncertainty, Visual Object Classification and Textual Entailment,* Springer.

Method

I used Bayesian neural network models, implemented using Markov chain Monte Carlo.

Preprocessing: I transformed some features to improve correlation with the response. Specifically, for ADA, I squared feature 15 and took the square roots of features 25 and 32, and for GINA, I took the cube roots of all features. Non-binary features were rescaled to have standard deviation of approximately one.

Feature selection or dimensionality reduction: For all datasets except NOVA, I considered looking at the first 10 or 20 principal components instead of or in addition to the original features. I ended up using the first 20 principal components plus the original features for GINA and HIVA. For NOVA, I did not directly use features that were non-zero in less than four training or validation cases, though these features were incorporated into some constructed features, as described below. No other feature selection was done. However, in all models, hyperparameters were present that could adjust as the model discovered how relevant each of the features was to predicting the class (a method known as Automatic Relevance Determination (ARD)).

Classification: I used multilayer perceptron neural networks with at least two hidden layers of non-linear units (tanh activation function). Sometimes an additional hidden layer of units with identity activation function was added before these two, in order to effectively reduce dimensionality.

Model exploration and selection: Some exploration of various models was done by looking at properties of the data by hand, by seeing how different models trained on the training set performed on the validation set, and by looking at the models' own assessments of their expected performance on the test set. I also checked whether models trained only on the training set appeared to be well calibrated in their predictions for the validation set (eg, whether among cases that were predicted to be in class $+1$ with probability approximately 0.7, the fraction that were actually in class $+1$ was about 0.7). No calibration problems were found with any of the models tried.

Note that hyperparameters within each model can have the effect of smoothly adjusting various aspects of the model, such as the degree of non-linearity in the predictions. These hyperparameters were automatically updated as part of the Markov chain Monte Carlo procedure.

Since the focus of this competition was model selection, I submitted only one final entry (though the rules allowed up to five). The models for each dataset were chosen by hand, largely on the basis of the models' own assessments of performance. For HIVA, I averaged the predictions of three models. This process of manual model selection might work much better in real problems, when one would not have to guess at the meaning of peculiar aspects of the data such as described below for HIVA and NOVA.

Models for individual datasets: The results of choosing amongst various models were as follows:

ADA: I used a network with two hidden layers, containing 25 units and 10 tanh units.

GINA: I used a network containing a layer of 20 hidden units with identity activation function that looked at the original features. The outputs of these 20 hidden units along with the first 20 principal component values were fed into two subsequent hidden layers of 20 and 10 tanh units.

HIVA: I averaged the predictive probabilities produced by three models. One model used a layer of 10 hidden units with identity activation function to reduce dimensionality, with the values of these units being fed into two subsequent hidden layers with 20 and 10 tanh units. The other two models looked at the first 20 principal components plus a special composite input (but not the features themselves). They both used two hidden layers of tanh units (one had 10 and 5 units, the other 20 and 10). The special composite input was obtained by first selecting only those features that did not have a statistically significant negative correlation with the class (most have a positive correlation). For each case, the weighted average of these features was computed, with the weights being inversely proportional to the fraction of all cases in which the feature was 1, raised to the power 1.75. This was done because exploratory analysis of the data indicated that most features were positively correlated with the class, more so for those that were mostly 0.

NOVA: This dataset has a large number of features that are almost always zero. I eliminated those that were non-zero in less than four cases (training, validation, or test), but also used three derived features that were intended to capture any useful information contained in the omitted features (eg, perhaps cases with many such rare features are more likely to be in class $+1$). The network had a layer of 10 hidden units with identity activation function that looked at the common features. The outputs of these 10 hidden units along with the three derived features were fed into two subsequent hidden layers of 20 and 10 tanh units.

SYLVA: I used a network with two hidden layers, containing 25 units and 10 tanh units.

Performance prediction guess: My prediction for the class of a test case was obtained by thresholding the predictive probability of class $+1$ as produced by the model (averaging over many networks from the posterior distribution). The threshold was set to the fraction of cases that were in class $+1$. From the probabilities of class $+1$ for each test case, along with the predictions made, I also computed the expected number of errors in each class, and from this obtained an expected balanced error rate.

Results

I submitted only one entry (not counting entries before validation labels were released). This entry was ninth overall. Only two other entrants submitted better entries, so I ranked third in terms of entrants. My entry was the best in terms of average AUC on the test set, indicating that the predictive probabilities produced by my Bayesian methods are a good guide to the accuracy of predictions on individual test cases.

Dataset	**My entry**					**The challenge best entry**				
	Test AUC	**Test BER**	**BER guess**	**Guess error**	**Test score (rank)**	**Test AUC**	**Test BER**	**BER guess**	**Guess error**	**Test score (rank)**
ADA	0.9107	0.1753	0.1656	0.0097	0.1850 (4)	0.9149	0.1723	0.1650	0.0073	0.1793 (1)
GINA	0.9915	0.0418	0.0635	0.0216	0.0635 (31)	0.9712	0.0288	0.0305	0.0017	0.0302(1)
HIVA	0.7627	0.2824	0.2937	0.0113	0.2916 (4)	0.7671	0.2757	0.2692	0.0065	0.2797 (1)
NOVA	0.9878	0.0528	0.0706	0.0178	0.0706 (29)	0.9914	0.0445	0.0436	0.0009	0.0448 (1)
SYLVA	0.9991	0.0066	0.0070	0.0005	0.0069 (11)	0.9991	0.0061	0.0060	0.0001	0.0062 (1)
Overall	0.9304	0.1118	0.1201	0.0122	0.1235 (15.8)	0.8910	0.1090	0.1040	0.0079	0.1165 (6.2)

Code

I used my Software for Flexible Bayesian Modeling, which is available from my web page, at http://www.cs.utoronto.ca/~radford/. However, the scripts for this competition are not available, as they use features of my current development version, which has not yet been released. Scripts for the two other competitions referenced above are available from my web page.

Keywords

Bayesian learning, neural networks, PCA, Automatic Relevance Determination.

B1.4. Random Forests

Contact

Corinne Dahinden, Seminar for Statistics, ETH Zurich, CH-8092 Zurich, Switzerland, dahinden@stat.math.ethz.ch

Acronym of your best entry

RF

Reference

Classification with Tree-Based Ensembles Applied to the WCCI 2006 Performance Challenge Datasets, Corinne Dahinden, In Proceedings IJCNN06, to appear.

Method

For the NOVA dataset, PCA with centered and scaled variables were computed and the first 400 principal components were taken. For the other datasets, no preprocessing was applied. No variable-selection was performed for any dataset. Then Random Forests was used with 4000 trees fitted with CART. Instead of the theoretical cutoff, which is the proportion of labels with +1 in the dataset, this cutoff is estimated by Cross-Validation, which considerably improves the performance of Random Forests for unbalanced datasets.

The performance prediction was guessed by 10-fold Cross-Validation.

Results

In the challenge, we rank 4th as a group and our best entry is the 11th, according to the average rank computed by the organizers. The Random Forests algorithm can be applied to a wide range of datasets and is not subject to the "small n — large p" problem. Plain standard Random Forests is very simple, yet highly efficient. The cutoff-adaptation has even shown to improve this performance, still the computational cost is kept low. The procedure requires minimal human interaction, can be used for variable selection and internally computes unbiased estimate of the generalization error. Random Forests achieves results which can keep up with the most sophisticated algorithms.

Dataset	**Our best entry**					**The challenge best entry**				
	Test AUC	**Test BER**	**BER guess**	**Guess error**	**Test score (rank)**	**Test AUC**	**Test BER**	**BER guess**	**Guess error**	**Test score (rank)**
ADA	0.8200	0.1800	0.1650	0.0150	0.1950 (16)	0.9149	0.1723	0.1650	0.0073	0.1793 (1)
GINA	0.9587	0.0413	0.0490	0.0077	0.0490 (17)	0.9712	0.0288	0.0305	0.0017	0.0302(1)
HIVA	0.7009	0.2994	0.2700	0.0294	0.3284 (32)	0.7671	0.2757	0.2692	0.0065	0.2797 (1)
NOVA	0.9470	0.0530	0.0530	0.0000	0.0530 (15)	0.9914	0.0445	0.0436	0.0009	0.0448 (1)
SYLVA	0.9946	0.0054	0.0065	0.0011	0.0065 (3)	0.9991	0.0061	0.0060	0.0001	0.0062 (1)
Overall	0.8842	0.1158	0.1087	0.0106	16.6	0.8910	0.1090	0.1040	0.0079	0.1165 (6.2)

Code

The computations were done in R. Random Forests is implemented in the randomForest package available from CRAN (http://cran.r-project.org). In addition, we have also implemented the cutoff-adaptation in R.

Keywords

PCA, embedded feature selection, RF, 10-fold cross-validation, ensemble method, CART

B1.5. Kernel Classifier

Contact

Wei Chu, Center of Computational Learning Systems, Columbia University,
chuwei@cs.columbia.edu

Acronym of your best entry

SVM/GPC

Reference

I have not written up a document on the procedure I applied. For reference or code, see http://www.gatsby.ucl.ac.uk/~chuwei/.

Method

Classifiers with different kernels were trained on the five datasets respectively. More specifically, I tried two kernel classifiers, support vector classifier and Gaussian process classifier. Profile of my methods as follows:

Preprocessing: Training and test data, except NOVA, were jointly normalized to zero-mean and unit variance.

Feature selection: On Gina, ranksum test was applied that reduced 970 features to 453 features; On Hiva, hypergeometry test was applied that reduced 1617 features to 425. On other datasets, we used all normalized features.

Classification

- A linear support vector classifier was used on Nova; Gaussian process classifier with a Gaussian kernel was used on Sylva; while non-linear support vector classifiers with Gaussian kernels were used on other datasets.
- Did you use ensemble methods? No.
- Did you use "transduction" or learning from the unlabeled test set? No.

Model selection/hyperparameter selection: Model evidence was used to decide optimal values of hyperparameters, whereas 10-fold cross validation was applied for model selection in support vector classifiers.

Performance prediction guess. (How did you compute the value in the .guess file). Validation outputs were used for support vector classifiers to estimate predictive performance, while leave-one-out validation outputs were used in Gaussian process classifiers.

Results

Dataset	**Our best entry**					**The challenge best entry**				
	Test AUC	**Test BER**	**BER guess**	**Guess error**	**Test score (rank)**	**Test AUC**	**Test BER**	**BER guess**	**Guess error**	**Test score (rank)**
ADA	0.8101	0.1899	0.174	0.016	0.2059					
GINA	0.9619	0.0381	0.0379	0.0002	0.0381					
HIVA	0.7095	0.2905	0.27	0.0205	0.31					
NOVA	0.952	0.048	0.045	0.003	0.0503					
SYLVA	0.99	0.01	0.0076	0.0024	0.0124					
Overall	0.8847	0.1153	0.1069	0.0084	0.1233					

challenge performances (group rank is fifth). qualitative advantages (Gaussian process classifiers provide predictive probability).

Code

An implementation of Gaussian process classifiers can be found `http://www.gatsby.ucl.ac.uk/~chuwei/`, which is designed for more general cases of ordinal regression. Binary classification is treated as a special case of ordinal regression.

Keywords

- Preprocessing or feature construction: standardization.
- Feature selection approach: filter.
- Feature selection engine: miscellaneous classifiers.
- Feature selection search: feature ranking.
- Feature selection criterion: K-fold cross-validation.
- Classifier: SVM, kernel-method, Gaussian processes.
- Hyper-parameter selection: grid-search, evidence, K-fold cross-validation.
- Other: No.

B1.6. Random Linear Matching Pursuit

Contact

Nicolai Meinshausen, nicolai@stat.math.ethz.ch

Acronym of your best entry

ROMA

Reference

none yet, unfortunately

Method

- Preprocessing: none, except for NOVA (PCA)
- Feature selection feature selection is achieved automatically; no preprocessing with feature selection
- Classification
  - Generalized linear model (Binomial family); linear in the variables and all interaction terms between variables; forward selection of variables and interactions (somewhat similar to MARS), yet not the best candidate is chosen from all variables but the best in a randomly selected subset (in this regard being similar to Random Forests). An ensemble of these predictors was formed; The goals was to have a good classifier which is linear in the variables and interactions
- Model selection/hyperparameter selection

  Hyperparameter selection is not very important for this method; some tuning was done on on out-of-bag samples
- Performance prediction guess. (How did you compute the value in the guess file). Cross-validation

Results

Dataset	Our best entry					The challenge best entry				
	Test AUC	Test BER	BER guess	Guess error	Test score (rank)	Test AUC	Test BER	BER guess	Guess error	Test score (rank)
ADA	0.8190	0.1810	0.1590	0.0220	0.2029 (15)	0.9149	0.1723	0.1650	0.0073	0.0000
GINA	0.9442	0.0558	0.0534	0.0024	0.0578 (24)	0.9712	0.0288	0.0305	0.0017	0.0000
HIVA	0.7057	0.2943	0.2698	0.0245	0.3182 (18)	0.7671	0.2757	0.2692	0.0065	0.0000
NOVA	0.9542	0.0458	0.0506	0.0048	0.0502 (9)	0.9914	0.0445	0.0436	0.0009	0.0000
SYLVA	0.9935	0.0065	0.0053	0.0012	0.0076 (19)	0.9991	0.0061	0.0060	0.0001	0.0000
Overall	0.8833	0.1167	0.1076	0.0110	0.1274 (21)	0.8910	0.1090	0.1040	0.0079	0.0000

easy interpretation of results as result is linear in variables and interactions; computationally attractive

Code

Implementation in R; code is to be made available later

B1.7. Regularized and Averaged Selective Naïve Bayes Classifier

Contact

Marc Boullé, France Telecom R&D, 2, avenue Pierre Marzin, 22307 Lannion cedex - France
marc.boulle@francetelecom.com

Acronym of your best entry

SNB(CMA) + 10k F(3D) tv

Reference

References

M. Boullé, “Regularization and Averaging of the Selective Naïve Bayes classifier”, International Joint Conference on Neural Networks, 2006.

M. Boullé, “MODL: a Bayes Optimal Discretization Method for Continuous Attributes”, Machine Learning, to be published.

Method

Our method is based on the Naïve Bayes assumption.

All the input features are preprocessed using the Bayes optimal MODL discretization method. We use a Bayesian approach to compromise between the number of selected features and the performance of the Selective Naïve Bayes classifier: this provides a regularized feature selection criterion. The feature selection search is performed using alternate forward selection and backward elimination searches on randomly ordered feature sets: this provides a fast search heuristic, with super-linear time complexity with respect to the number of instances and features. Finally, our method introduces a variant of feature selection: feature “soft” selection.

Whereas feature "hard" selection gives a "Boolean" weight to the features according to whether they selected or not, our method gives a continuous weight between 0 and 1 to each feature. This weighing schema of the features comes from a new classifier averaging method, derived from Bayesian Model Averaging.

The method computes the posterior probabilities of the classes, which is convenient when the classical accuracy criterion or the area under the ROC curve is evaluated. For the challenge, the Balanced Error Rate (BER) criterion is the main criterion. In order to improve the BER criterion, we adjusted the decision threshold in a post-optimization step. We still predict the class having the highest posterior probability, but we artificially adjust the class prior probabilities in order to optimize the BER criterion on the train dataset. For the challenge, several trials of feature construction have been performed in order to evaluate the computational and statistical scalability of the method, and to naively attempt to escape the naïve Bayes assumption:

- 10k F(2D): 10 000 features constructed for each dataset, each one is the sum of two randomly selected initial features,
- 100k F(2D): 100 000 features constructed (sums of two features),
- 10k F(3D): 10 000 features constructed (sums of three features).

The performance prediction guess is computed using a stratified tenfold cross-validation.

Results

In the challenge, we rank 7th as a group and our best entry is 26th, according to the average rank computed by the organizers. On 2 of the 5 five datasets (ADA and SYLVA), our best entry ranks 1st.

Our method is highly scalable and resistant to noisy or redundant features: it is able to quickly process about 100 000 constructed features without decreasing the predictive performance. Its main limitation comes from the Naïve Bayes assumption. However, when the constructed features allow to "partially" break the naïve Bayes assumption, the method succeeds in significantly improve its performances. This is the case for example for the GINA dataset, which does not fit well the naïve Bayes assumption: adding randomly constructed features allows to improve the BER from 12.83% down to 7.30%.

The AUC criterion, which evaluates the ranking of the class posterior probabilities, indicates high performances for our method.

Dataset	**Our best entry**					**The challenge best entry**				
	Test AUC	**Test BER**	**BER guess**	**Guess error**	**Test score (rank)**	**Test AUC**	**Test BER**	**BER guess**	**Guess error**	**Test score (rank)**
ADA	0.9149	0.1723	0.1650	0.0073	0.1793 (1)	0.9149	0.1723	0.1650	0.0073	0.1793
GINA	0.9772	0.0733	0.0770	0.0037	0.0767	0.9712	0.0288	0.0305	0.0017	0.0302
HIVA	0.7542	0.3080	0.3170	0.0090	0.3146	0.7671	0.2757	0.2692	0.0065	0.2797
NOVA	0.9736	0.0776	0.0860	0.0084	0.0858	0.9914	0.0445	0.0436	0.0009	0.0448
SYLVA	0.9991	0.0061	0.0060	0.0001	0.0062 (1)	0.9991	0.0061	0.0060	0.0001	0.0062
Overall	0.9242	0.1307	0.1306	0.0096	0.1399 (26.4)	0.8910	0.1090	0.1040	0.0079	0.1165 (6.2)

Code

Our implementation was done in C++.

Keywords

Discretization, Bayesianism, Naïve Bayes, Wrapper, Regularization, Model Averaging

B1.8. Artificial Contrasts with Ensembles and Regularized Least Squares Classifiers

Contact

Kari Torkkola and Eugene Tuv

Acronym of your best entry

ACE+RLSC

References

1. **ACE:** Feature Selection Using Ensemble Based Ranking Against Artificial Contrasts, Eugene Tuv, Alexander Borisov and Kari Torkkola, IJCNN06, to appear.
2. **RLSC:** Ensembles of Regularized Least Squares Classifiers for High-Dimensional Problems, Kari Torkkola and Eugene Tuv, in Feature Extraction, Foundations and Applications, Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti Zadeh (eds.), 2006

Method

No preprocessing was used. For feature selection, we used feature ranking against artificial contrasts (ACE). ACE is an embedded feature selection method using Gradient Boosting Trees as the internal engine. For classification, we use regularized least squares classifiers with Gaussian kernels. Hyper-parameters (kernel width, regularization coefficient) are adjusted after feature selection, using grid search and 10-fold CV with the same training data. Our test BER prediction is based also on the same 10-fold CV.

Results

In the challenge, we rank 8th as a group and our best entry is the 33rd, according to the average rank computed by the organizers. As advantages, compact feature sets could be mentioned.

Dataset	Our best entry					The challenge best entry				
	Test AUC	Test BER	BER guess	Guess error	Test score (rank)	Test AUC	Test BER	BER guess	Guess error	Test score (rank)
ADA	0.8178	0.1822	0.1673	0.0149	0.197 (24)	0.9149	0.1723	0.1650	0.0073	0.1793 (1)
GINA	0.9712	0.0288	0.0305	0.0017	0.0302 (1)	0.9712	0.0288	0.0305	0.0017	0.0302(1)
HIVA	0.6986	0.3014	0.2461	0.0553	0.3567 (47)	0.7671	0.2757	0.2692	0.0065	0.2797 (1)
NOVA	0.9256	0.0744	0.0469	0.0275	0.1018 (55)	0.9914	0.0445	0.0436	0.0009	0.0448 (1)
SYLVA	0.9911	0.0089	0.0044	0.0045	0.0134 (33)	0.9991	0.0061	0.0060	0.0001	0.0062 (1)
Overall	0.8809	0.1191	0.099	0.0208	0.1398 (32)	0.8910	0.1090	0.1040	0.0079	0.1165 (6.2)

Code

ACE experiments were run using Intel's IDEAL (not available). RLSC was written using MATLAB (one-liner), and all cross-validation experimentation was done using MATLAB (code not available).

Keywords

embedded feature selection, gradient boosting trees, artificial contrast variables, Regularized Least Squares Classifier (RLSC) with Gaussian kernels, hyperparameter grid-search,10-fold cross validation.

B1.9. SVM-LOO

Contact

Olivier Chapelle

Acronym of your best entry

SVM-LOO

Reference

Relevant material can be found at http://www.kyb.mpg.de/publication.html?publ=1436

Method

- preprocessing: all components were normalized to have variance = 1.
- feature selection: none
- classifier: SVM with L2 penalization of the slacks and RBF kernel. No transduction. Threshold adjusted afterwards to optimize an (approximate) leave-one-out BER.
- Hyperparameters (C and σ) optimized by gradient descent on eiter: leave-one-out, radius/margin bound, validation error (in that case, half of the training set is used as validation), evidence (Bayesian treatment).

- Performance prediction based on an approximate leave-one-out procedure (no use of the test set).

Code

code available at: `http://www.kyb.tuebingen.mpg.de/bs/people/chapelle/ams/ams.m`

B1.10. Model Selection in an Ensemble Framework

Contact

Joerg Wichard
Institute of Molecular Pharmacology
Molecular Modelling Group
Robert Rössle Straße 10,
D-13125 Berlin-Buch, Germany.
`JoergWichard@web.de`

Acronym of your best entry

submission 13

Reference

Model Selection in an Ensemble Framework, J. Wichard, In Proceedings IJCNN06, to appear.

Method

Our method uses a simple normalizing and balancing of the data sets. In one experiment we were using a SVM-based feature selection method. We build heterogeneous ensembles of classifiers, consisting of linear models (LDA, ridge), nearest neighbor models, trees, neural networks and SVMs. We use a cross-validation approach for model selection and hyperparameter selection. The performance prediction guess was calculated by averaging the BER (balanced error rate) on the validation sets in the cross-validation folds.

Results

In the challenge, we rank 38.2 in the group rank and our best entry is the 37th according to the test score computed by the organizers.

Dataset	Our best entry					The challenge best entry				
	Test AUC	Test BER	BER guess	Guess error	Test score (rank)	Test AUC	Test BER	BER guess	Guess error	Test score (rank)
ADA	0.8614	0.1801	0.1636	0.0165	0.1965	0.8304	0.1696	0.155	0.1046	0.1843
GINA	0.9866	0.0523	0.05	0.0023	0.0543	0.9639	0.0361	0.0388	0.0027	0.0386
HIVA	0.7172	0.3057	0.338	0.0323	0.3377	0.7129	0.2871	0.27	0.0171	0.03029
NOVA	0.9459	0.0611	0.08	0.0189	0.08	0.9542	0.0458	0.0503	0.0045	0.0499
SYLVA	0.9956	0.0267	0.007	0.0197	0.0464	0.9937	0.0063	0.0058	0.0005	0.0067
Overall	0.9013	0.1252	0.1277	0.0179	0.143 (38.2)	0.891	0.109	0.104	0.0079	0.1165 (6.2)

Code

Our method is implemented in MATLAB. We developed a toolbox for regression tasks and a toolbox for classification problems, both are based on an ensemble approach. The code is available at: `http://chopin.zet.agh.edu.pl/~wichtel/`.

Keywords

Cross-validation, Heterogeneous Ensembles, Mixed Models

B1.11. Advanced Analytical Methods, INTEL

Contact

Borisov Alexander (`alexander.borisov@intel.com`) and Eugene Tuv (`eugene.tuv@intel.com`)

Acronym of your best entry

IDEAL

Reference

Borisov A., Eruhimov V. and Tuv, E. *Tree-Based Ensembles with Dynamic Soft Feature Selection*, In Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti Zadeh, editors, Feature Extraction, Foundations and Applications. Springer, 2006. (in press)

Method

Gradient Tree Boosting with Dynamic Feature Selection.

No preprocessing was done. We used gradient tree boosting with dynamic feature selection. The method builds a serial binomial logistic regression tree ensemble. Each new expert — a shallow tree is built on the residuals from the previous iteration using a random subset of cases. For very unbalanced datasets a stratified sampling was used to up weight the rare class. At each node, a random small subset of variables [sqrt(total number of variables)] is selected. The vars sampling probabilities are proportional to sum of priors (initially equal, then decreasing influence as trees are added to the ensemble) and current variable importances computed using split scores evaluated over the ensemble.

All hyper-parameters (tree depth, sampling scheme, regularization, importance adjustment rate for the dynamic FS, class priors) were selected using test sample estimation (over multiple train/test partitions). Performance prediction guess error was done using the same test sample estimation. Learning from the unlabeled test set was not used.

Results

In the challenge, we ranked 7th as a group and our best entry is the 17.6 th, according to the average rank computed by the organizers. Our method is accurate and fast on wide variety of datasets with complex dependencies (for example, we ranked in top ten on NIPS2003 also with the same method). Our method is generally more accurate and incomparably faster (on massive in both dimensions datasets) than original Freidman's MART and Breiman's RF. It does not require any data preprocessing.

Dataset	**Our best entry**					**The challenge best entry**				
	Test AUC	**Test BER**	**BER guess**	**Guess error**	**Test score (rank)**	**Test AUC**	**Test BER**	**BER guess**	**Guess error**	**Test score (rank)**
ADA	0.9110	0.1779	0.162	0.0159	0.1939(12)	0.8304	0.1696	0.155	0.0146	0.1843(3)
GINA	0.9893	0.035	0.044	0.009	0.044(7)	0.9639	0.0361	0.0388	0.0027	0.0386(5)
HIVA	0.7193	0.3085	0.285	0.0235	0.3312(3)	0.7129	0.2871	0.27	0.0171	0.3029(8)
NOVA	0.9837	0.0622	0.062	0.0002	0.0622(21)	0.9542	0.0458	0.0503	0.0045	0.0499(8)
SYLVA	0.9986	0.0132	0.0085	0.0047	0.0179(45)	0.9937	0.0063	0.0058	0.0005	0.0067(7)
Overall	0.92038	0.11936	0.1123	0.01066	0.12984(17.6)	0.891	0.109	0.104	0.0079	0.1165(6.2)

Code

Method is implemented in C++ as a part of Intel's IDEAL ML software product. It is not publicly available.

Keywords

Ensembles, boosting, feature selection, tree

B1.12. Learning with Mean-Variance Filtering, SVM and Gradient-based Optimization

Contact

Vladimir Nikulin, the Australian National University, Canberra, Australia,
vladimir.nikulin@anu.edu.au

Acronym of your best entry

GbO+MVf+SVM2

Reference

Learning with Mean-Variance Filtering, SVM and Gradient-based Optimization, Vladimir Nikulin. In Proceedings IJCNN06, to appear.

Method

We consider several models, which employ gradient-based method as a core optimization tool. Experimental results were obtained in a real time environment during WCCI-2006 Performance Prediction Challenge. None of the models were proved to be absolutely best against all five datasets. Furthermore, we can exploit the actual difference between different models and create an ensemble system as a complex of the base models where the balances may be regulated using special parameters or confidence levels.

Overfitting is a usual problem in the situation when dimension is comparable with sample size or even higher. Using mean-variance filtering we can reduce the difference between training and test results significantly considering some features as a noise.

Results

In the challenge we rank 12th as a group and our best entry is 47th, according to the average rank computed by the organizers.

Dataset	Our best entry					The challenge best entry				
	Test AUC	Test BER	BER guess	Guess error	Test score (rank)	Test AUC	Test BER	BER guess	Guess error	Test score (rank)
ADA	0.8225	0.1851	0.165	0.0201		0.8304	0.1696	0.155	0.0146	
GINA	0.9403	0.0566	0.05	0.0066		0.9639	0.0361	0.0388	0.0027	
HIVA	0.6588	0.3536	0.245	0.1086		0.7129	0.2871	0.27	0.0171	
NOVA	0.9474	0.0507	0.05	0.0007		0.9542	0.0458	0.0503	0.0045	
SYLVA	0.9644	0.0212	0.012	0.0092		0.9937	0.0063	0.0058	0.0005	
Overall	0.8667	0.1334	0.1044	0.029		0.891	0.109	0.104	0.0079	

Keywords

Gradient-based optimization, support vector machines, feature selection, logit model, ensemble method

B1.13. Large margin linear classifiers with bias adjustment for skewed two-class distributions.

Contact

Edward F. Harrington, Defence Science and Technology Organisation (DSTO) Australia, edward.harrington@dsto.defence.gov.au

Acronym of your best entry

ba4

Reference

no specific paper as yet.

Method

The data was preprocessed with centering and scaling (standardization). In the case of the SVC sub-sampling was also done.

No feature selection was done prior to classification.

Three linear classifiers were used: SVC, Online Bayes Point Machine (OBPM) and Approximate Large Margin Algorithm (ALMA).

Model and parameter selection was done using a simple cross-validation. To select the model of OBPM and ALMA we considered the results for each random permutation of the training set. To tune the bias parameter of OBPM and ALMA we used a line search to minimize the affect of any skew between the two classes in the training set. The criteria to select the "best" model for each classifier was simply the minimum BER from the validation set. The classifier with the lowest validation BER amongst the three was selected as the "best" classifier.

Results

Dataset	**Our best entry**					**The challenge best entry**				
	Test AUC	**Test BER**	**BER guess**	**Guess error**	**Test score (rank)**	**Test AUC**	**Test BER**	**BER guess**	**Guess error**	**Test score (rank)**
ADA	0.8900	0.1964	0.1761	0.0203	0.2168	0.8304	0.1550	0.1550	0.0146	0.1843
GINA	0.9758	0.0750	0.0600	0.0086	0.0836	0.9639	0.0388	0.0388	0.0027	0.0386
HIVA	0.7043	0.3509	0.3500	0.0009	0.35	0.7129	0.2871	0.2700	0.0171	0.3029
NOVA	0.9369	0.0631	0.0500	0.0131	0.0762	0.9542	0.0458	0.0503	0.0045	0.0499
SYLVA	0.9886	0.0114	0.0042	0.0072	0.0187	0.9937	0.0063	0.0058	0.0005	0.0067
Overall	0.8991	0.1394	0.1294	0.01	0.1492(43.8)	0.8910	0.1090	0.1040	0.0079	0.1165(6.2)

Keywords

standardization, sub-sampling, cross-validation, permutation, bias adjustment, linear classifier, OBPM, ALMA, SVC.

B1.14. A Study of Supervised Learning with Multivariate Analysis on Unbalanced Datasets

Contact

Yu-Yen Ou, Department of Computer Science and Engineering, Yuan-Ze University, Chung-Li, Taiwan, yien@csie.org

Acronym of your best entry

svm+ica

Reference

Yu-Yen Ou, Hao-Geng Hung and Yen-Jen Oyang, A Study of Supervised Learning with Multivariate Analysis on Unbalanced Datasets, IJCNN06.

Method

Our study aimed at providing effective solutions to these two challenges. For handling unbalanced datasets, we proposed that a different value of the cost parameter in Support Vector Machine (SVM) is employed for each class of samples. For handling high-dimensional datasets, we resorted to Independent Components Analysis (ICA), which is a multivariate analysis algorithm, along with the conventional univariate analysis.

Preprocessing
Independent Components Analysis (ICA)
Noise Reduction

Feature selection
Univariate Analysis

Classification
Support Vector Machine (SVM)

- RBF kernel and linear kernel
- give different cost parameter to the each class of data

Model selection/hyperparameter selection
Cross Validation

Performance prediction guess.
Cross Validation

Results

In the challenge, we rank 16th as a group and our best entry is the 46th, according to the average rank computed by the organizers. Also, our method yields the second best results for GINA dataset.

Dataset	Our best entry					The challenge best entry				
	Test AUC	Test BER	BER guess	Guess error	Test score (rank)	Test AUC	Test BER	BER guess	Guess error	Test score (rank)
ADA	0.8041	0.1959	0.151	0.0449	0.2408	0.8965	0.1845	0.1742	0.0103	0.1947
GINA	0.9672	0.0328	0.04	0.0072	0.04	0.99	0.0461	0.047	0.0009	0.0466
HIVA	0.676	0.324	0.24	0.084	0.4081	0.7464	0.2804	0.2776	0.0028	0.2814
NOVA	0.9347	0.0653	0.05	0.0153	0.0805	0.9914	0.0445	0.047	0.0025	0.0464
SYLVA	0.9812	0.0188	0.002	0.0168	0.0356	0.999	0.0067	0.0065	0.0002	0.0067
Overall	0.8727	0.1273	0.0966	0.0336	0.161 (46)	0.9246	0.1124	0.1105	0.0034	0.1152 (1)

Keywords

centering, scaling, ICA, univeriate feature selection, Chi-square, F-score, training error, leave-one-out, K-fold cross-validation, SVM, kernel-method, grid-search, cross-validation

B1.15. Cross-indexing

Contact

Juha Reunanen, Juha.Reunanen@iki.fi

Acronym of your best entry

CLOP-models-only-5

Reference

J. Reunanen (2006), *Less Biased Measurement of Feature Selection Benefits.* In C. Saunders et al. (Eds.): SLSFS 2005, LNCS 3940, pp. 198–208. *To appear.*

Method

Cross-indexing is a recent approach for assessing the outcome of a model selection process. Compared to traditional cross-validatory model selection and assessment, using cross-indexing may in some special cases either provide less biased results in a similar amount of time, or results of similar accuracy in significantly less time (depending on whether an outer loop of cross-validation is used). The method has been described in the context of feature selection in the reference mentioned above. In this challenge, it was used to select the model architecture and the corresponding parameters, and to estimate their performance when applied together. The models compared were introduced already in the sample code: *Prepro+naiveBayes*, *PCA+kernelRidge*, *GS+kernelRidge*, *Prepro+linearSVC*, *Prepro+nonlinearSVC*, and *Relief+neuralNet*. For each model type, a couple of parameters were subjected to optimization, but in other respects the models were treated as black boxes. The final ensemble consisted of nine members for each dataset.

In more detail, the selection took place as follows: First, the data available were split into nine folds. Then, during each of the nine iterations, eight of these folds were pooled and used during the search, while the remaining *k*th fold was utilized as a validation set, using which the optimal model and the corresponding parameters for the *k*th ensemble member were chosen. The union of the eight folds was further divided into only three folds (to save some time) in order to facilitate standard cross-validation to guide a simple stochastic search for the optimal parameters. The search was interleaved to give equal possibilities for all the model architectures being considered: the execution scheduler basically tried to round-robin the time spent (instead of the number of evaluations), with the exception that more time was allocated to the optimization of those models that were able to demonstrate good performance estimates.

The performance estimate obtained for the optimal parameter set using the remaining fold was potentially overfitted when a large number of comparisons had been performed. Thus, this score was not used as such to assess the performance of the corresponding ensemble member — instead, the cross-indexing approach was adopted to recall the estimated performance on the other folds after a similar number of iterations. This score was not used to select the model, and thus it had not been overfitted due to a multiple-selection process. The final performance guess was obtained as the median of these nine guesses. This might have introduced a pessimistic bias, as the ensemble can be expected to perform better than its individual members, but based on the results, it looks like this did not really happen. However, the variance of these nine guesses could have been used to estimate *the accuracy of the BER guess*, had that been the goal of the challenge.

Results

While no competitive BER was obtained for any of the datasets, the guess error remains at an acceptable level, and the AUC is good. Moreover, with respect to the test score, the method beats the reference entries that were using the same CLOP models, although such a comparison is hardly a fair one, as the reference models were probably trained without the validation labels. Still, it can be said that the selection of the final model, and the estimation of its performance using cross-indexing, were performed successfully.

Dataset	**Our best entry**					**The challenge best entry**				
	Test AUC	**Test BER**	**BER guess**	**Guess error**	**Test score (rank)**	**Test AUC**	**Test BER**	**BER guess**	**Guess error**	**Test score (rank)**
ADA	0.8825	0.2037	0.1884	0.0153	0.2190	0.8304	0.1696	0.1550	0.0146	0.1843
GINA	0.9631	0.0980	0.0840	0.0141	0.1121	0.9639	0.0361	0.0388	0.0027	0.0386
HIVA	0.7392	0.3088	0.3172	0.0084	0.3148	0.7129	0.2871	0.2700	0.0171	0.3029
NOVA	0.9874	0.0892	0.0917	0.0025	0.0907	0.9542	0.0458	0.0503	0.0045	0.0499
SYLVA	0.9971	0.0341	0.0320	0.0021	0.0357	0.9937	0.0063	0.0058	0.0005	0.0067
overall	0.9138	0.1468	0.1427	0.0085	0.1545 (45.8)	0.8910	0.1090	0.1040	0.0079	0.1165 (6.2)

Appendix B2

AL vs PK Challenge

B2.1. LogitBoost with trees

Contact

Roman Werner Lutz, Seminar for Statistics, ETH Zurich, CH-8092 Zurich, Switzerland, lutz@stat.math.ethz.ch

Acronym of your best entry

LB tree mix cut adapted

Reference

LogitBoost with Trees Applied to the WCCI 2006 Performance Prediction Challenge, Roman Werner Lutz, In Proceedings IJCNN06, to appear.

Method

As preprocessing we used PCA for Nova with centered and scaled variables and took the first 400 principal components. No preprocessing was used for the other datasets. Then we applied LogitBoost with trees of prefixed depth. The number of iterations, the tree depth (in each iteration a tree of the same depth is fitted) and the BER guess were chosen/computed by 10-fold cross-validation. Shrinkage was added to make LogitBoost more stable: in each iteration only a fraction ? (0.3, 0.1 or 0.03) of the fitted learner was added. ? was chosen by visual inspection of the cross-validated BER curve (as a function of the boosting iteration). As a result, LogitBoost yielded probabilities of class membership for each sample. The cut point for the final classification was the proportion of class $+1$ in the data.

For the second entry we used the Wilcoxon test (for continuous variables) and the Fisher exact test (for binary variables) for variable pre-selection (variables with a p-value above 0.1 were dropped). For the third entry we averaged the predicted probabilities of LogitBoot with and without variable pre-selection. For the fourth entry we made an intercept adaption (on the logit scale) so that the average of the predicted probabilities on the test set equals the proportion of class $+1$ in the data.

Results

In the challenge, we rank 1^{st} as a group and our best entry (our fourth) is the 1^{st}, according to the average rank computed by the organizers. Our method is quite simple: no preprocessing is needed (except for Nova) and the tuning parameters are chosen by cross-validation. Additionally, LogitBoost with trees does variable selection, because in each iteration only a few variables are chosen.

Dataset	Our best entry					The challenge best entry				
	Test AUC	Test BER	BER guess	Guess error	Test score (rank)	Test AUC	Test BER	BER guess	Guess error	Test score (rank)
ADA	0.8304	0.1696	0.1550	0.0146	0.1843 (3)	0.9149	0.1723	0.1650	0.0073	0.1793 (1)
GINA	0.9639	0.0361	0.0388	0.0027	0.0386 (5)	0.9712	0.0288	0.0305	0.0017	0.0302 (1)
HIVA	0.7129	0.2871	0.2700	0.0171	0.3029 (8)	0.7671	0.2757	0.2692	0.0065	0.2797 (1)
NOVA	0.9542	0.0458	0.0503	0.0045	0.0499 (8)	0.9914	0.0445	0.0436	0.0009	0.0448 (1)
SYLVA	0.9937	0.0063	0.0058	0.0005	0.0067 (7)	0.9991	0.0061	0.0060	0.0001	0.0062 (1)
Overall	0.8910	0.1090	0.1040	0.0079	0.1165(6.2)	0.8910	0.1090	0.1040	0.0079	0.1165(6.2)

Code

Our implementation was done in R.

Keywords

PCA, Wilcoxon test, Fisher exact test, LogitBoost, trees of fixed depth, 10-fold cross-validation, shrinkage.

B2.2. Feature selection with redundancy elimination + gradient boosted trees.

Contact

ASML team, INTEL Corporation, alexander.borisov@intel.com

Acronym of your best entry

out1-fs-nored-val (Intel final 1)
out3-fs-red-valid (Intel final 2)
out5-valid-no-fs (Intel final 3)

Reference

Paper to be resubmitted to the JMLR

Method

No preprocessing was done.

The method consists of the following steps

1. Feature selection using ensemble classifiers (ACE FS). Random probes that are permutation of original features are added. Importance of each variable in RF ensemble is compared versus importance of probes using t-test over several ensembles. Variables that are more important in statistical sense then most of probes are selected as important. Variables are ordered according to sum of gini index reduction in tree splits.

2. Variable masking it estimated on important variables with GBT ensemble using surrogate splits (if a more important variable has surrogate on less important one, the second variable is masked by the first). Again, statistically significant masking pairs are selected, then subset of mutually non-masked variables with high importance is chosen

3. Effect of found variables is removed using RF ensemble.

Steps 1-3 are repeated until no more important variables remain.

Next GBT with embedded feature selection (to prevent over fitting) is built on selected variable set. The following parameters of GBT were optimized : number of trees, tree depth, shrinkage, number of selected features per tree node and importance adjust rate (for embedded FS), stratified sampling 0/1 class proportions, priors. For FS, #of trees in series, importance and masking quantile were chosen.

Optimization strategy (manual) was to set reasonable parameter values, then try to adjust each parameter (sequentially), so that test error decreases (model was trained on 60% of training data during parameter optimization). Several passes over all GBT parameters was done, one for FS parameters.

Priors were selected using cross validation (FS+GBT run was done on K partitions of the data, optimal priors were selected on remaining part).

Results

Table 23: Our Best Results

Dataset	**Entry name**	**Entry ID**	**Test BER**	**Test AUC**	**Score**	**Track**
ADA	out1-fs-nored-val (Intel final 1)	1051	0.1737	0.8259	0.0143	Agnos
GINA	out1-fs-nored-val (Intel final 1)	1051	0.0373	0.9631	0.2436	Agnos
HIVA	out3-fs-red-valid (Intel final 2)	1052	0.2899	0.7123	0.1124	Agnos
NOVA	out1-fs-nored-val (Intel final 1)	1051	0.0547	0.9468	0.2756	Agnos
SYLVA	out1-fs-nored-val (Intel final 1)	1051	0.0135	0.9865	0.5126	Agnos
Overall	out1-fs-nored-val (Intel final 1)	1051	0.1142	0.8859	0.2373	Agnos

- quantitative advantages

  Method is very fast ($\sim$a minute for one FS iteration on NOVA dataset with 16K+ vars) (20 ensembles with 70 trees) (faster than all known to us minimal subset selection methods). Complexity is proportional to

  $$(\text{Fsel} + \text{Fimpvar}) \times N \times \log N \times \text{Ntrees} \times \text{Nensembles} \times \text{Niter} + \text{Niter} \times \text{Fimpvar}^2,$$

  – Niter — #of iteration of ACE FS algorithm always < 10, usually 3–4
  – Nensembles = 20 (number of ensembles for t-test)
  – Ntrees = 20-100 (number of trees in RF or ensemble)
  – N — number of samples,
  – Fsel = number of selected important vars per tree split (sqrt(total number features) or less)

  - Fimvar — total number of selected important variable, for NOVA — 400–800 depending on parameters.

  Works with any variable types, mixed values, requires no preprocessing.

- qualitative advantages

  This method allows to find a small subset of features with the same predictive capacity as the original set.

Original # of features, CV-err using all features / best subset size, CV-err using best subset

Ada:	47, 0.190902 / 16, 0.185584
Gina:	970, 0.052740 / 75, 0.050629
Hiva:	1617, 0.284723 / 221, 0.255898
Nova:	12993, 0.059070 / 400, 0.051794
Sylva:	212, 0.013268 /69, 0.012852

Keywords

- Preprocessing or feature construction: —
- Feature selection approach: embedded feature selection.
- Feature selection engine:miscellaneous classifiers (RF, GBT).
- Feature selection search: variable masking estimation, redundancy elimination, statistical test.
- Feature selection criterion: 5-fold cross-validation.
- Classifier: RF, Gradient boosting trees.
- Hyper-parameter selection: manual optimization.
- Other: —.

B2.3. Cross-indexing

Contact

Juha Reunanen, Juha.Reunanen@iki.fi

Acronym of your best entry

cross-indexing-7a (AL), cross-indexing-prior-1 (PK)

Reference

J. Reunanen (2006): Less Biased Measurement of Feature Selection Benefits. In C. Saunders et al. (Eds.), *Subspace, Latent Structure and Feature Selection: Statistical and Optimization Perspectives Workshop* (SLSFS 2005; LNCS 3940), Revised Selected Papers, pp. 198–208.

Method

Cross-indexing is a recent approach for assessing the outcome of a model selection process. Compared to traditional cross-validatory model selection and assessment, using cross-indexing may in some special cases either provide less biased results in a similar amount of time, or results of similar accuracy in significantly less time (depending on whether an outer loop of cross-validation is used). The method has been described in the context of feature selection in the reference mentioned above. In this challenge, it was used to select the model architecture and the corresponding parameters, and to estimate their performance when applied together. The models compared were introduced already in the sample code: *Prepro+naiveBayes*, *PCA+kernelRidge*, *GS+kernelRidge*, *Prepro+linearSVC*, *Prepro+nonlinearSVC*, *Relief+neuralNet*, *RF*, and *Boosting* (with *neuralNet*, *SVC* and *kernelRidge*). For each model type, a couple of parameters were subjected to optimization, but in other respects the models were treated as black boxes. The final ensemble consisted of three, five or nine members, depending on the dataset.

In more detail, the selection took place as follows: First, the data available were split into K (five or nine[2]) folds, depending on the dataset (no magic here: just varied it depending on the time, memory etc. available). Then, during each of the K iterations, $K-1$ of these folds were pooled and used during the search, while the remaining kth fold was utilized as a validation set, using which the optimal model and the corresponding parameters for the kth ensemble member were chosen. The union of the $K-1$ folds was further divided into only three folds (to save some time) in order to facilitate standard cross-validation to guide a simple stochastic search for the optimal parameters. The search was interleaved to give equal possibilities for all the model architectures being considered: the execution scheduler basically tried to round-robin the time spent (instead of the number of evaluations), with the exception that more time was allocated to the optimization of those models that were able to demonstrate good performance estimates for the present dataset.

The performance estimate obtained for the optimal parameter set using the remaining fold was potentially overfitted when a large number of comparisons had been performed. Therefore, this score was not used as such to assess the performance of the corresponding ensemble member — instead, the cross-indexing approach was adopted to recall the estimated performance *on the other folds* after a similar number of iterations. These scores had not been used to select this model, thus they had not been overfitted due to a multiple-selection process. The final performance guess (which was not required in this challenge, but is always useful for development purposes) was obtained as the median of the K guesses. This may have introduced a pessimistic bias, as the ensemble can be expected to perform better than its individual members.

2. In some of my submissions, including cross-indexing-7a, the HIVA model only contains three ensemble members. This is because two of the five (that were searched for) were manually removed, due to their apparently bad performance.

Results

Table 24: Our Best Results

Dataset	Entry name	Entry ID	Test BER	Test AUC	Score	Track
ADA	cross-indexing-prior-2	905	0.1807	0.907	0.0961	Agnos
GINA	cross-indexing-prior-3	996	0.0236	0.997	0.1068	Prior
HIVA	cross-indexing-7a	882	0.2863	0.7662	0.1004	Agnos
NOVA	cross-indexing-prior-1	743	0.0472	0.9903	0.0769	Agnos
SYLVA	cross-indexing-prior-1	743	0.0066	0.9989	0.0603	Prior
Overall	cross-indexing-prior-1a	883	0.11	0.9312	0.1294	Prior

Code

CLOP models used:

Dataset	Track	Ensemble members
ADA	AL	2*{sns,std,norm,gentleboost(neural),bias}; 2*{std,norm,gentleboost(kridge),bias}; 1*{rf,bias}
GINA	AL	6*{std,gs,svc(degree=1)}; 3*{std,svc(degree=2)}
	PK	4*{std,svc(degree=2)}; 1*{rf}
HIVA	AL	3*{norm,svc(degree=1),bias}
NOVA	AL	5*{norm,gentleboost(kridge),bias}
SYLVA	AL	4*{std,norm,gentleboost(neural),bias}; 4*{std,neural}; 1*{rf,bias}
	PK	3*{sns,std,norm,gentleboost(neural),bias}; 2*{rf,b}

(sns = shift'n'scale, std = standardize, norm = normalize)

The ensemble members were chosen during the model selection loop according to their estimated performance (using the cross-indexing criterion).

Keywords

- Preprocessing: centering, scaling, standardization.
- Feature selection: Gram-Schmidt (only GINA on the AL track).
- Classifier: boosting, neural networks, ridge regression, kernel method, RF.
- Hyper-parameter selection: stochastic search, cross-validation, cross-indexing.
- Other: ensemble method.

B2.4. Classification with Random Sets, Boosting and Distance-based Clustering

Contact

Vladimir Nikulin
Airservices Australia, 25 Constitution Ave., Canberra, ACT 2601

vnikulin@digisurf.com.au,
vladimir.nikulin@airservicesaustralia.com

Acronym of your best entry

vn3

Reference

The paper of 24 pages will be resubmitted to JMLR

Method

Overfitting represents usual problem associated with classification of high-dimensional data. According to the proposed approach we can use large number of classifiers where any single classifier is based on the subset of relatively small number of randomly selected features or random sets (RS) of features.

Consequently, any single RS-classifier 1) will not overfit, and 2) may be evaluated very quickly. The property of limited overfitting is a very important. As a result, feature selection in the final model will be made according to several best performing subsets of features.

The proposed method is an essentially different comparing with Breiman's Random Forests (voting system)[3] where the final classifier represents a sample average of the single classifiers. Note, also, that any single RS-classifier may be evaluated using different methods and it is not necessarily a decision tree.

Secondly, we propose a new boosting approach, which is based on experience-innovation principles. Assuming that overfitting is limited, it is logical to increase weights of randomly selected mis-classified patterns (innovation) in order to improve training results. As a starting point for any iteration we can use weights, which correspond to the best past result (experience). Again, the proposed system is not a voting one in difference to AdaBoost or LogitBoost[4].

Thirdly, using some criterion we can split data under expectation that the corresponding clusters will be more uniform in the sense of relations between features and target variable. The final model may be constructed as an ensemble of several models, which were evaluated independently using particular data from the corresponding clusters.

3. L. Breiman (2001) "Random Forests", Machine Learning, 45, 1, pp.5–32.
4. J. Friedman and T. Hastie and R. Tibshirani (2000) "Additive logistic regression: a statistical view of boosting", Annals of Statistics, 28, pp.337–374.

Results

Table 25: Our Best Results

Dataset	Entry name	Entry ID	Test BER	Test AUC	Score	Track
ADA	vn5	1026	0.1751	0.8331		Agnos
ADA	vn3	1024	0.1788	0.8225		Prior
GINA	vn2	1023	0.0226	0.9777		Prior
GINA	vn4	1025	0.0503	0.9507		Agnos
HIVA	vn3	1024	0.2904	0.7343		Agnos
NOVA	vn5	1026	0.0471	0.9456		Agnos
SYLVA	vn3	1024	0.0071	0.9959		Prior
SYLVA	vn4	1025	0.0096	0.9933		Agnos
Overall	vn3	1024	0.1095	0.8949		Prior
Overall	vn5	1026	0.1177	0.8891		Agnos

Keywords

random forests, gradient-based optimization, boosting, cross-validation, distance-based clustering.

We used an opportunity of the Challenge to test CLOP Version 1.1 – October, 2006. The most basic (and sufficient) instructions may be found on the last page of Ref.[5] The package is a quite efficient and can produce competitive results in application to any dataset of the Challenge. It is very easy to arrange suitable cross validations with required number of folds in order to evaluate any particular model, and there is a wide range of choices. For example, we can recommend my_model='boosting'. All necessary details in relation to this model may be found in the file "model_examples.m" in the directory `../CLOP/sample_code`. Definitely, Isabelle Guyon and her team have done an excellent work.

Also, it is worth to mention that ADA-prior task is a very similar to the recent task of PAKDD-2007 Data Mining Competition[6]. Accordingly, we applied the same preprocessing technique. Firstly, using standard methods we reduced categorical features to the numerical (dummy) values. Also, we normalized continuous values to the range [0..1]. As a result of the above transformation we created totally numerical dataset with 127 features. Then, using soft Mean-Variance Filtering[7] the number of features was reduced to 108.

Some concluding remarks

Certainly, practical experience is the best way to learn, and I am pleased with results of the Table 25, which demonstrate significant improvement over all previous results dated July 2006. The proper feature selection is a very essential in order reduce overfitting. The following models

5. I. Guyon and A. Alamdari and G. Dror and J. Buhmann (2006) "Performance Prediction Challenge", IJCNN, Vancouver, BC, Canada, July 16–21, pp.2958–2965.
6. `http://lamda.nju.edu.cn/conf/pakdd07/dmc07/`
7. V. Nikulin (2006) "Learning with mean-variance filtering, SVM and gradient-based optimization", IJCNN, Vancouver, BC, Canada, July 16–21, pp.4195–4202.

appears to be the most suitable: LogitBoost for ADA and SYLVA; RBF-SVM for GINA and LinearSVM for NOVA; regularized linear model for HIVA.

Currently, the areas of my primary interests are decision trees, random forest and a variety of boosting modifications. According to my experience, such existing packages as “randomForest” or “ADA” (R-environment) are efficient, but there may be problems with memory allocation. The performance of “TreeNet”, Salford Systems, is a very good in the case of regression in difference to classification. Also, it is not easy to arrange a satisfactory cross-validation using TreeNet. Respectively, a new package (written in C with dynamic memory allocation) is under construction at the moment.

I was in Orlando, FL, twice in 2005 and 2006 and wish all participants of IJCNN-2007 very pleasant and productive work during the Conference.

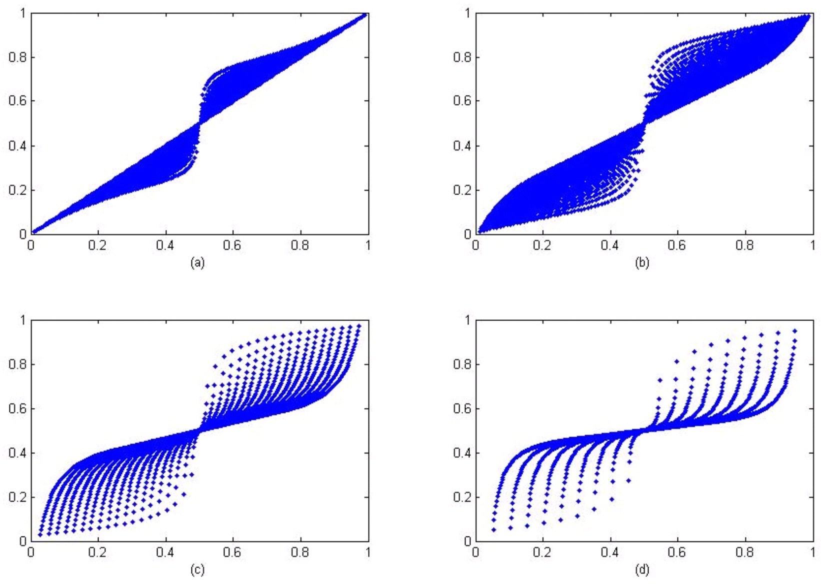


Figure 2: [Fact Sheet B2.4] BER vs BER where true-labels and expected-labels were replaced with each other; (a) balanced case, ... , (d) imbalanced case. These nice figures illustrate non-symmetrical properties of the BER loss function.

B2.5. PSMS for Neural Networks

Contact

Hugo Jair Escalante. Luis Enrique Erro # 1, Tonantzintla, 72840, Puebla, México
hugojair@ccc.inaoep.mx

Acronym of your best entry

Corrida_Final (according the March 1st milestone results)

Reference

H. Jair Escalante and M. Montes and L. E. Sucar, *PSMS for Neural Networks on the IJCNN 2007 Agnostic vs Prior Knowledge Challenge*, In INNS-IEEE Proceedings of the 20th International Joint Conference on Neural Networks 2007 (IJCNN-2007), Orlando, FL, USA.

Method

Preprocessing The following preprocessing methods from the CLOP package were considered for the model selection process: *normalization (normalize), standardization (standardize) and scaling (shif_n_scale).*

Feature selection The feature selection methods considered for the model selection process were: Signal-to-noise ratio (s2n), relief (relief), and Gram-Schmidt orthogonalization (gs).

Classification Experiments were performed with a linear classifier, naïve Bayes and neural networks, though the best ranked entries were obtained with neural networks, a non-linear classifier.

No ensemble methods were considered for the model selection process.

No methods for learning from unlabeled data were used.

Model selection/hyperparameter selection For model and hyperparameter selection a bio- inspired search algorithm called: particle swarm optimization (*PSO*) was used. *PSO* is a population-based algorithm that aims to simulate the social behavior of birds within a flock. Candidate solutions of an optimization problem are considered particles that fly through the search space. Each particle has a velocity that is influenced by the global best solution and the best solution the particle has found so far. A fitness function is used to evaluate each candidate solution.

For model and hyperparameter selection CLOP models (preprocessing method + feature selection method + hyperparameters for the neural net) were codified as real valued vectors. The BER value obtained by 5-fold cross validation was used as fitness function. A standard *PSO* algorithm was implemented using default parameter values, see the reference.

The algorithm was run for 100 iterations for the HIVA, GINA and SYLVA datasets, and for 500 iterations for the ADA dataset; though it was not applied to NOVA because of its high dimensionality and the complexity of the learning machine. Instead, for NOVA the model was selected empirically by trial and error.

The PSO algorithm has the same computational drawbacks that any search algorithm applied to the task of model selection, namely they depend on the complexity of the learning machine used, which in turn depends on the size and/or dimensionality of the data. However PSO has fast convergence to minima, although it can be a local one. Therefore strategies for avoiding local minima should be included. The simplest (and the one we considered) is the insertion of an inertia weight into the updating velocity equation (see reference paper) that allows the algorithm to perform global and local search, avoiding in a sense local minima.

Results

Table 26: Our Best Results

Dataset	Entry name	Entry ID	Test BER	Test AUC	Score	Track
ADA	Corrida_final_10CV	922	0.1804	0.9015	0.09	Agnos
GINA	AdaBoost	170	0.053	0.9878	0.3846	Agnos
HIVA	Corrida_final	919	0.2854	0.7551	0.0884	Agnos
NOVA	AdaBoost	170	0.0504	0.9895	0.1987	Agnos
SYLVA	PSMS_100_4all_NCV	987	0.0084	0.9989	0.2362	Agnos
Overall	PSMS_100_4all_NCV	987	0.1178	0.925	0.2464	Agnos

quantitative advantages (e.g. compact feature subset, simplicity, computational advantages)

The PSO algorithm for model selection provides competitive models even when the available algorithms to select from are very simple. Such models are obtained by running the algorithm only a few iterations. It can be considered a black-box method in the sense that no experience on machine learning is required to use it.

qualitative advantages (e.g. compute posterior probabilities, theoretically motivated, has some elements of novelty).

The PSO algorithm has been already used for training neural network (adjusting weights), though it has not been used for model selection with hyperparameter selection, a slight modification that allows for the selection of preprocessing and feature selection algorithms as well. The algorithm can be used with any other learning algorithm and preprocessing/feature selection methods.

Code

Dataset	Spider command used to build the model
ADA	chain({standardize({'center=0'}), normalize({'center=1'}), shift_n_scale({'take_log=0'}), neural({'units=5', 'shrinkage=1.4323', 'balance=0', 'maxiter=257'}), bias})
GINA	chain({gs({'f_max=48'}), shift_n_scale({'take_log=1'}), neural({'units=16', 'shrinkage=0.29191', 'balance=1', 'maxiter=456'}), bias})
HIVA	chain({standardize({'center=1'}), normalize({'center=0'}), neural({'units=5', 'shrinkage=3.028','balance=0','maxiter=448'}), bias})
NOVA	chain({normalize({'center=0'}), gentleboost(neural({'units=1', 'shrinkage=0.2' , 'balance=1', 'maxiter=50'}) , {'units=10', 'rejNum=3'}), bias})
SYLVA	chain({standardize({'center=1'}), normalize({'center=1'}), neural({'units=6', 'shrinkage=0.02882', 'balance=1', 'maxiter=359'}), bias})

Keywords

- Preprocessing or feature construction: centering, scaling, standardization.

- Feature selection approach: embedded feature selection.
- Feature selection engine: correlation coefficient, Relief.
- Feature selection search: feature ranking.
- Feature selection criterion: K-fold cross-validation.
- Classifier: neural networks.
- Hyper-parameter selection: pattern search, bio-inspired search cross-validation, K-fold.
- Other: swarm optimization.

B2.6. Hybrid approach for learning

Contact

Mehreen Saeed, FAST National University of Computer & Emerging Science, Lahore, Pakistan, mehreen.saeed@nu.edu.pk

Acronym of your best entry

SubmitA

Reference

N/A

Method

Preprocessing : Standardize, shift-n-scale, normalize

Feature selection : Removed the sparse features whose percentage of non-zero entries was below as certain number.

Classification

- Used mixture models for finding clusters within data. I wrote down my own routine for generating the mixture parameters
- For binary data, Bernoulli mixture models were used. For continuous data Gaussian mixture models were used
- Neural network and gentleboost from CLOP / spider was used for learning the conditional class probabilities as a function of class label
- Did you use ensemble methods? No
- Did you use "transduction" or learning from the unlabeled test set? No

Model selection/hyperparameter selection : 5-fold cross-validation was used

Results

Table 27: Our Best Results

Dataset	**Entry name**	**Entry ID**	**Test BER**	**Test AUC**	**Score**	**Track**
ADA	Submit D final	1037	0.181	0.8185	0.1022	Agnos
GINA	Submit A final	1034	0.0495	0.9507	0.3162	Agnos
HIVA	Submit D final	1037	0.305	0.6976	0.4418	Agnos
NOVA	Submit E final	1038	0.0456	0.9552	0.0385	Agnos
SYLVA	Submit C final	1036	0.0094	0.9906	0.2864	Agnos
Overall	Submit D final	1037	0.1194	0.8812	0.2786	Agnos

quantitative advantages : Dimensionality reduction

qualitative advantages : Novel approach that combines both the advantages of a generative and a discriminative classifier.

Code

I used spider/clop commands and added my own objects to the spider library. I implemented my own version of the expectation maximization algorithm in C++ and called this routine from matlab.

Keywords

- Preprocessing or feature construction: centering, scaling, standardization
- Feature selection approach: frequency count
- Feature selection engine: Very simple matlab routine
- Feature selection search: Brute force
- Feature selection criterion: data statistics
- Classifier: Neural networks, SVM, boosting, mixture model
- Hyper-parameter selection: cross-validation
- Other: post-processing, “bias” option

B2.7. Linear Programming SVM (Liknon)

Contact

Dr. Erinija Pranckeviciene. Institute for Biodiagnostics, NRC Canada, Ellice ave 435, Winnipeg, MB. erinija.pranckevie@nrc-cnrc.gc.ca

Reference

E.Pranckeviciene and R.Somorjai, Liknon feature selection for Microarrays, in F.Masulli, S.Mitra, and G.Pasi(Eds.):WILF 2007, LNAI 4578, 580–587, 2007.

Method

Linear Programming SVM (Liknon) feature selection combined with state of the art classification rules.

Code/engine:

1. For the Matlab implementation of the LP SVM we refer to C. Bhattacharrya, LR Grate, A Rizki and et. al, "Simultaneous relevant feature identification and classification in high-dimensional spaces:application to molecular profiling data", Signal Processing, vol 83(4), 729–743, 2003.
2. For the Matlab implementation of the state of the art classifiers we refer to R. Duin, P. Juscak, P. Paclick, E.Penkalska, D. deRidder, D.Tax, PRTools4 A Matlab toolbox for pattern recognition, February, 2004.

Implementation strategy: The dataset first is divided into 10 parts (10 fold crossvalidation). The training set of the single fold is further subdivided into balanced training and unbalanced (the rest samples) monitoring sets. A number of the subdivisions is arbitrary, we did 31. In every subdivision a number of LP SVM models/discriminants is trained on the balanced training set and the BER -balanced error rate- of each is estimated on the monitoring set. The number of the models depends on the set of the chosen values of regularization parameter C. The data, number of features and available computational time determine the range of C values. The model with smallest monitoring BER is selected in every subdivision. As a result, in the single fold, we have an ensemble of the linear discriminants and a feature profile possibly to be investigated with the other classification rules.

Preprocessing: none.

Feature selection: Feature selection relies on the property of LP SVM to produce sparse solutions. The identities of the features, corresponding to non zero weights of the discriminants, are included in the profile. Every fold reveals slightly different feature profiles. However relevant feature identities consistently appear in many subdivisions. First it was noticed in the experiments with synthetic data and then in the experiments with microarrays.

Classification:

1. ensemble of linear discriminants;
2. the rules tested on the derived feature profile included linear and nonlinear, all available in the PRTools: fisher classifier, linear logistic classifier, subspace classifier, nearest neighbors, decision tree, neares mean classifier, quadratic classifier.
3. NO TRANSDUCTION.

Results

Table 28: Our Best Results

Dataset	**Entry name**	**Entry ID**	**Test BER**	**Test AUC**	**Score**	**Track**
ADA	liknon feature selection + state of art (1)	1012	0.1818	0.8702	0.135	Agnos
GINA	liknon feature selection + state of the art (3)	1014	0.0533	0.974	0.3889	Agnos
HIVA	liknon feature selection+ state of art classifiers	814	0.2939	0.7589	0.1647	Agnos
NOVA	liknon feature selection + state of art 2	713	0.0725	0.9814	0.5064	Agnos
SYLVA	liknon feature selection + state of the art (2)	1013	0.019	0.9949	0.7085	Agnos
Overall	liknon feature selection + state of art (1)	1012	0.127	0.9133	0.4358	Agnos

Quantitative advantages: Simplicity and interpretability of the results in terms of feature identities, important for discrimination. The stability of the discovered feature identities in different folds suggests that the feature selection via LP SVM is robust to the sample bias. In case of high dimensional data, the discovered features provide a reduced representation of the data for testing other classifiers. A success of the suggested approach was apparent for GINA dataset. DISADVANTAGE- high computational burden.

Qualitative advantages: The sequence of values of the regularization parameter determines the increasing number of features given by the increasing number of non-zero weights of the linear discriminant. This can be considered as a feature selection structure. The sequence of the LP SVM solutions- linear discriminants of increasing complexity- forms a nested structure, where the principles of the structural risk minimization may apply.

Keywords

- Preprocessing or feature construction: none.
- Feature selection approach: embedded feature selection.
- Feature selection engine: SVM.
- Feature selection search: feature ranking, ordered FS (ordered feature selection) Feature selection criterion: monitoring error
- Classifier: nearest neighbors, tree classifier, L1 norm regularization, ensemble method, bagging.
- Hyper-parameter selection: grid-search.
- Other: sample bias.

B2.8. Agnostic Learning with Ensembles of Classifiers

Contact

Joerg Wichard. FMP - Molecular Modelling Group Robert Roessle Strasse 10 13125 Berlin, Germany joergwichard@web.de

Reference

Agnostic Learning with Ensembles of Classifiers

Method

Ensembles of Classifiers, mainly Classification and Regression Trees (CART). Ensemble building based on a modified bagging scheme. We used a simple balancing strategy for data preprocessing. No feature selection and no transduction method was used. Model selection and parameter selection was combined in a modified cross-validation approach.

Results

Table 29: Our Best Results

Dataset	Entry name	Entry ID	Test BER	Test AUC	Score	Track
ADA	Final	1039	0.181	0.906	0.1002	Agnos
GINA	boosted trees 2	1008	0.0731	0.979	0.6197	Agnos
HIVA	cv-trees	1010	0.295	0.7702	0.2008	Agnos
NOVA	boosted trees 2	1008	0.1416	0.9344	0.8462	Agnos
SYLVA	Final	1039	0.0115	0.9981	0.402	Agnos
Overall	Final	1039	0.1481	0.9137	0.4811	Agnos

quantitative advantages: Ensembles of CART trees are robust, conceptually simple and fast. The critical issue of feature selection is done by the training algorithm, if "tree pruning" is used.

qualitative advantages: Huge ensembles (almost like random forests) are possible with this method.

Code

Our own Matlab Ensemble Toolbox was used: http://www.j-wichard.de/entool/

B2.9. Modified multi-class SVM formulation; Efficient LOO computation

Contact

Vojtech Franc
Fraunhofer Institut FIRST IDA
Kekulestr. 7, 12489 Berlin
fravoj@first.fraunhofer.de

Acronym of your best entry

SVM-RBF

Reference

Unpublished.

Method

PRIOR KNOWLEDGE — GINA

Preprocessing: Each input image was normalized such that variance of pixels was one. No feature selection was applied.

We modified multi-class SVM by Crammer et al. 2001. A decision {odd,even} is taken based on the output of the multi-class classifier, i.e. we take decision "odd" whenever the multi-class classifier returns 1,3,. . . 9 and the decision "even" otherwise. The standard multi-class SVM minimizes the number misclassification. In our modified formulation, we minimize the number of wrong decisions made on account of the output of the multi-class classifier. For example, if the multi-class classifier returns 1 and the true label is 2 we penalizes the decision by one. However, if the classifier returns 4 (or any even number) and the true label is 2 we pay no penalty since the final decision is correct. This modification leads to a QP task similar to the original multi-class SVM but the number of linear constraints is reduced. We used a cutting plane algorithm similar to Tsochantaridis et al. 2005 to solve the underlying QP task.

We implemented a kernel version of this modified SVM and used the RBF (Gaussian) kernel in all experiments. The optimal hyper-parameters (regularization constant and the kernel with) were selected from a chosen finite set by minimizing the BER estimate on the validation set.

AGNOSTIC LEARNING — HIVA, NOVA, SILVA

Features were normalized such that the covariance matrix was the identity matrix. No feature selection was applied.

We used standard binary SVM with RBF kernel in all the experiments.

The hyper-parameters were tuned using the Leave-One-Out (LOO) estimate of the BER error. To make computation of the LOO feasible we designed a new method which allows for early stopping of the QP optimizer. The new stopping condition guarantees that the decision on the test example equals to the decision of the classifier with optimal (minimizing the QP exactly) parameters, i.e. this method gives the exact LOO estimate and it is computationally feasible. In contrast to the standard stopping conditions used in QP-SVM optimization (e.g those based on KKT conditions or duality gap) our stopping condition is directly related to the desired output of the optimization, i.e. we stop the optimization when the decision (label) on the testing example can be reliably determined.

Results

Table 30: Our Best Results

Dataset	**Entry name**	**Entry ID**	**Test BER**	**Test AUC**	**Score**	**Track**
ADA	RBF SVM	716	0.1833	0.8987	0.1984	Agnos
GINA	RBF SVM	933	0.023	0.9962	0.0598	Prior
HIVA	RBF SVM	734	0.2827	0.7707	0.0763	Agnos
NOVA	RBF SVM	734	0.0877	0.9834	0.6795	Agnos
SYLVA	RBF SVM	734	0.0205	0.997	0.7437	Agnos
Overall	RBF SVM	933	0.1211	0.9289	0.408	Prior

Add GINA: In contrast to treating the problem as standard binary classification, the proposed method exploits the information about the clustering of the data within the decision classes by modeling each digit with a single normal vector. The proposed formulation is a natural extension of the multi-class SVM when an addition information about clustering within the classes is available.

Add Agnostic Learning: Our method allows for exact and efficient computation of the LOO estimate for problems with thousands of examples.

Code

We implemented all the methods in Matlab. The critical time consuming parts (like QP solvers) are written in C. In principal, we can make the code available but currently it is not in the state ready for publication.

Keywords

- Preprocessing or feature construction: feature normalization.
- Feature selection approach: none.
- Classifier: SVM, kernel-method, L2 norm regularization.
- Hyper-parameter selection: leave-one-out.
- Other: QP optimization.

B2.10. Report on Preliminary Experiments with Data Grid Models in the Agnostic Learning vs. Prior Knowledge Challenge

Contact

Marc Boullé
France Telecom R&D
2, avenue Pierre Marzin
22307 Lannion Cedex, France
marc.boulle@orange-ftgroup.com

Acronym of your best entry

Data Grid (Coclustering)

Reference

Paper published in IJCNN 2007

Method

Data grids extend the MODL discretization and value grouping methods to the multivariate case.

They are based on a partitioning of each input variable, in intervals in the numerical case and in groups of values in the categorical case. The cross-product of the univariate partitions forms a multivariate partition of the input representation space into a set of cells. This multivariate partition, called data grid, allows to evaluate the correlation between the input variables and the output variable. The best data grid is searched owing to a Bayesian model selection approach and to combinatorial algorithms.

Three classification techniques exploiting data grids differently are presented and evaluated in the Agnostic Learning vs. Prior Knowledge Challenge:

- Data Grid (MAP): use the MAP data grid as a classifier
- Data Grid (CMA): use an ensemble of data grids
- Data Grid (Coclustering): apply a bivariate unsupervised data grid to learn a coclustering on the instance*variable space, using all the unlabelled train+valid+test data. The clusters of instances are used for prediction using the available labels (train+valid).

Summary of the method:

- Preprocessing : multivariate partition (discretization/value grouping)
- Feature selection: variables whose univariate partition contains at least two parts are selected
- Classification
  - Data Grid (MAP): the best multivariate partition forms a classifier
  - Data Grid (CMA): use an ensemble method
  - Data Grid (Coclustering): use learning from the unlabeled test set
- Model selection/hyperparameter selection: model are selected using a Bayesian approach (no hyper-parameter)

Results

Table 31: Our Best Results

Dataset	**Entry name**	**Entry ID**	**Test BER**	**Test AUC**	**Score**	**Track**
ADA	Data Grid (CMA)	920	0.1756	0.8464	0.0245	Prior
GINA	Data Grid (Coclustering)	921	0.0516	0.9768	0.3718	Prior
HIVA	Data Grid (Coclustering)	921	0.3127	0.7077	0.5904	Agnos
NOVA	Data Grid (Coclustering)	921	0.0488	0.9813	0.141	Agnos
SYLVA	Data Grid (CMA)	918	0.0158	0.9873	0.6482	Agnos
Overall	Data Grid (Coclustering)	921	0.1223	0.8984	0.3813	Prior

quantitative advantages compact feature subset, works with any variable type, ease of interpretation, no parameter tuning, use all the available data, computational efficiency

qualitative advantages compute posterior probabilities, model selection based on a Bayesian approach, data grids are a new machine learning technique.

B2.11. Dimensionality Reduction Techniques

Contact

Stijn Vanderlooy & Laurens van der Maaten
MICC-IKAT, Universiteit Maastricht, P.O. Box 616,
6200 MD Maastricht, the Netherlands

Acronym of your best entry

micc-ikat

Method

Our method to the challenge was to apply dimensionality reduction techniques in order to find a small set of discriminative features. As a preprocessing step we made the data zero mean and unit variance. We then applied principal components analysis and linear discriminant analysis to find a linear subspace of the original data space. In addition, the following six nonlinear dimensionality reduction techniques were applied: isomap, kernel principal components analysis with Gaussian kernel, (Hessian) locally linear embedding, Laplacian eigenmaps, and local tangent space alignment. Algorithms are run with default settings.

For classification we tried the following five classifiers: naïve Bayes, linear discriminant classifier, quadratic discriminant classifier, one nearest neighbour, and least squares support vector machine with Gaussian kernel. Complexity parameter C and bandwidth h are optimized with values $C = [1\ 5\ 10\ 20\ 50\ 100\ 500]$ and $h = 0.1$ to 1.5 in steps of size 0.1 for each dataset separately using a ten-fold cross validation procedure.

The best overall results, measured by means of error rate, were obtained using the support vector machine on the data representation found by linear discriminant analysis.

Results

Table 32: Our Best Results

Dataset	Entry name	Entry ID	Test BER	Test AUC	Score	Track
ADA	micc-ikat	1059	0.2805	0.7195	0.953	Agnos
GINA	LDA and LSSVM	945	0.1648	0.835	0.9145	Agnos
HIVA	LDA and LSSVM	945	0.3837	0.6157	0.9237	Agnos
NOVA	LDA and LSSVM	945	0.4248	0.5741	0.9679	Agnos
SYLVA	LDA and LSSVM	945	0.0495	0.9505	0.9397	Agnos
Overall	micc-ikat	1059	0.2606	0.739	0.9398	Agnos

quantitative advantages (e.g. compact feature subset, simplicity, computational advantages)

qualitative advantages (e.g. compute posterior probabilities, theoretically motivated, has some elements of novelty).

Keywords

- Preprocessing or feature construction: centering, scaling, standardization, PCA.
- Feature selection approach: filter, wrapper, embedded feature selection.
- Feature selection engine: correlation coefficient, Relief, single variable classifier, mutual information, miscellaneous classifiers, including neural network, SVM, RF.
- Feature selection search: feature ranking, ordered FS (ordered feature selection), forward selection, backward elimination, stochastic search, multiplicative updates
- Feature selection criterion: training error, leave-one-out, K-fold cross-validation.
- Classifier: neural networks, nearest neighbors, tree classifier, RF, SVM, kernel-method, least-square, ridge regression, L1 norm regularization, L2 norm regularization, logistic regression, ensemble method, bagging, boosting, Bayesian, transduction.
- Hyper-parameter selection: grid-search, pattern search, evidence, bound optimization, cross-validation, K-fold.
- Other: ensemble method, transduction.

B2.12. DoubleBoost

Contact

Lutz, Roman, `lutz@stat.math.ethz.ch`

Acronym of your best entry

DoubleBoost

Method

DoubleBoost is a special version of LogitBoost (Lutz, WCCI 2006) that can only be used for the prior knowledge track for Sylva. It uses the fact, that each pattern is composed of two true records: "each pattern contains each variable twice".

A new dataset is constructed by putting the second half of the data (variables 55 to 108) below the first half (variables 1 to 54). The new dataset is of dimension $2n$ times 54 (instead of n times 108). This new dataset is used for fitting the base learner (tree). The output of the base learner is averaged over the two records belonging to the same pattern.

Results

Table 33: Our Best Results

Dataset	**Entry name**	**Entry ID**	**Test BER**	**Test AUC**	**Score**	**Track**
ADA	LogitBoost with trees	13	0.166	0.9168	0.002	Agnos
GINA	LogitBoost with trees	892	0.0339	0.9668	0.2308	Agnos
HIVA	LogitBoost with trees	13	0.3018	0.7512	0.3414	Agnos
NOVA	LogitBoost with trees	892	0.0463	0.9538	0.0449	Agnos
SYLVA	Doubleboost	893	0.0043	0.9957	0.005	Prior
Overall	Doubleboost	893	0.1114	0.8896	0.1381	Prior

B2.13. Boosting with SVM

Contact

Jorge Sueiras, jorge.sueiras@neo-metrics.com

Acronym of your best entry

Boost tree

Method

For NOVA

1. Parsing each record, generating a word count table. Usage of a thesaurus and a word exclusion list.

2. Application of SVM techniques for dimensional reduction.

3. Selection of input variables: Top 50 variables from the SVM process and top 100 variables from word counting.

4. Model building through application of bootstrapping techniques to decision tree models with no more than 8 leaves per tree. Tree splitting search criterion was based on the Chi-square test.

Results

Table 34: Our Best Results

Dataset	Entry name	Entry ID	Test BER	Test AUC	Score	Track
ADA	boost tree	906	0.1825	0.8075	0.1575	Prior
GINA	Boost mix	915	0.0896	0.9655	0.6581	Prior
HIVA	boost tree	906	0.3257	0.7127	0.739	Agnos
NOVA	boost tree	906	0.0507	0.9869	0.2115	Agnos
SYLVA	boost tree	906	0.0081	0.9996	0.2211	Prior
Overall	boost tree	906	0.1314	0.8944	0.3983	Prior

B2.14. High-Throughput Screening with Two-Dimensional Kernels

Contact

Chloe-Agathe Azencott and Pierre Baldi
Institute for Genomics and Bioinformatics
236 Info & Computer Science Bldg. 2
University of California, Irvine
Irvine, California 92697-3445
cazencot@ics.uci.edu

Acronym of your best entry

SVM

Reference

None

Method

- Preprocessing

  The molecules of the HIVA dataset are represented as two-dimensional graphs where nodes are atoms and edges are bounds. The nodes are labelled by different schemes, including atomic number or element-connectivity label (atomic number together with number of connected atoms); the edges are labelled by bond type. Molecular graphs are then translated into fingerprints, vectors for which each component accounts for a given two-dimensional feature. More specifically, circular fingerprints [1] are used.

- Feature selection

  No feature selection is applied.

- Classification

  - Specific kernels, namely Tanimoto and MinMax, are used on top of the fingerprints to derive support vector machines. Generally speaking, these kernels allow for comparing fingerprints by comparing the number of features they share.
  - SVMTorch implementation [2] is used
  - Over-sampling of the active class is used to compensate for the unbalanceness of the dataset

- Model selection/hyperparameter selection
  - Optimal labeling schemes, depth of fingerprints and kernels are chosen by 10-fold cross-validation on the training set
  - Optimal SVM parameters C and epsilon are chosen among grid values by 10-fold cross-validation on the training set

Results

Table 35: Our Best Results

Dataset	**Entry name**	**Entry ID**	**Test BER**	**Test AUC**	**Score**	**Track**
ADA	final svm # 1	936	0.2751	0.8084	0.9448	Agnos
GINA	final svm # 1	936	0.1984	0.8915	0.9872	Agnos
HIVA	SVM	992	0.2693	0.7643	0.008	Prior
NOVA	final svm # 1	936	0.2005	0.9574	0.9423	Agnos
SYLVA	final svm # 1	936	0.0434	0.9912	0.9246	Agnos
Overall	SVM	992	0.1973	0.8826	0.7614	Prior

- quantitative advantages
  1. computation of features is relatively fast
  2. no feature selection is needed
- qualitative advantages
  - cross-validation results are in favor of low over-fitting
  - generic enough to be applied to other problems of the chemistry domain
  - the method can easily be adapted to other problems where the data can be represented as graphs

Code

- Computation of the molecular graph (Python Module)
  With the help of the OpenBabel library [3], we create a graph where the nodes are the atoms and the edges the bonds between them
- Computation of the fingerprints (Python Module)

  - Assign an initial label to each heavy (i.e. non hydrogen) atom. The label can be, for instance, the atomic number of the atom (atomic number scheme), or its atomic number paired with its number of connections to other heavy atoms (element-connectivity scheme)
  - For each iteration (typically: 2 to 4), update each atom label in the following way: the new label is a hash of the old label with the labels of all the connected atoms
  - Eventually, each atom label (across the whole dataset) is a possible feature and feature vectors (that are very sparse) are created

- SVM implementation
  - The SVMTorch module [2] is used (the MinMax kernel, that can be used on binary vectors as the Tanimoto kernel, is added to the kernels)
  - A Python wrapper is used to perform grid-search hyper-parameters optimization

Keywords

- Preprocessing or feature construction: molecular graph, circular fingerprints
- Feature selection approach: None
- Feature selection engine: None
- Feature selection search: None
- Feature selection criterion: None
- Classifier: SVM, active class over-sampling
- Hyper-parameter selection: grid-search,10-fold cross-validation.
- Other: 2D kernels for small molecules

[1] Dubois, J. E. **Chemical Applications of Graph Theory**; Academic Press, London: **1976** and Hassan, M., Brown, R. D., Varma-O'Brien, S., Rogers, D. **Cheminformatics analysis and learning in a data pipelining environment**. *Molecular Diversity* **2006**, *10*, pp 283–299.

[2] `http://www.idiap.ch/machine-learning.php`

[3] `http://openbabel.sourceforge.net`

Appendix C

CLOP: The challenge learning object package

Appendix C1

Quick Start Guide for CLOP

Amir Reza Saffari Azar Alamdari AMIR@YMER.ORG
Institute for Computer Graphics and Vision
Graz University of Technology, Graz, Austria
Isabelle Guyon ISABELLE@CLOPINET.COM
Clopinet Enterprise, 955 Creston road
Berkeley, CA 94708, USA

C1.1. Introduction

The main goal of this guide is to provide the *Model Selection Game* participants a quick starting point for using *Challenge Learning Object Package* (CLOP). This document will not cover information about the game, goals, rules, and etc, for those information please refer to the game's official website at http://clopinet.com/isabelle/Projects/modelselect/challenge/.

C1.1.1. What is CLOP?

CLOP is a software package, which contains several ready-to-use machine learning algorithms to be used during the game by participants. It is based on Spider package [1] from Department of Empirical Inference for Machine Learning and Perception, Max Planck Institute for Biological Cybernetics, Tuebingen, Germany. CLOP has more algorithms provided by challenge organizers compared to Spider and it runs on MATLAB http://www.mathworks.com, but it doesn't depend on any particular toolbox of MATLAB. CLOP together with this manual can be downloaded from http://clopinet.com/isabelle/Projects/modelselect/Clop.zip.

Briefly, to provide a sufficient toolbox for the challenge, we did the following modifications to the original Spider package:

- The entire spider is provided, but for the purpose of participating to the game, you are allowed to build models only from the following list of CLOP objects:
  - bias
  - chain
  - ensemble
  - gentleboost
  - gs
  - kridge
  - naive
  - neural

1. http://www.kyb.tuebingen.mpg.de/bs/people/spider/index.html

  - normalize
  - pc_extract
  - relief
  - rf
  - rffs
  - shift_n_scale
  - standardize
  - subsample
  - svc
  - svcrfe
  - s2n

  This list can also be obtained by typing `whoisclop` at the MATLAB prompt. Please refer to Section C1.4 for details.

- Some of the CLOP objects were not part of the original Spider; some have been modified (methods overloaded) for several reasons, including providing a simpler hyperparameterization (hiding some hyperparameters as *private*), returning always a discriminant value as output, providing more efficient implementations, and providing good default values to all hyperparameters.

For a valid challenge submission, please refrain from using other Spider learning objects, except using cross-validation objects, which allow you searching for the best model, but do not affect the final model. In addition, you should not modify the code of the CLOP objects, except if you find an error (please let us know) or make a change that does not affect the end result, but affects computational efficiency (we should be able to reproduce your results, given the model and the same set of parameters that you used). You can verify that your object is a valid CLOP object with the function `isclop`.

C1.1.2. How to install CLOP?

First of all you should obtain CLOP from the website mentioned above, and save it into any directory that you prefer. Then you have to open and extract the zip file using any archiving software of your convenient, into any directory that you want to install the CLOP. For the rest of this manual we will call this destination directory, where the extracted files are stored, as MyProjects/CLOP/ and we will refer to MyProjects/ as root directory. For your case, you have to replace it with the path to your CLOP directory. Now you have CLOP installed on your computer. If you are running CLOP under Linux OS, you might need to compile some of our models, please refer to Section C1.1.5.

The followings are the directory structures of CLOP and a brief description of what is inside:

1. MyProjects/CLOP/challenge_objects/: Challenge objects provided by the organizers.
2. MyProjects/CLOP/sample_code/: Sample code and lots of different functions written for the game.
3. MyProjects/CLOP/spider/: Original Spider is located here.

C1.1.3. What I need to run CLOP?

Since CLOP runs under MATLAB, you have to have MATLAB running on your computer. Additionally, in order to use CLOP for this game you will need to download the datasets from `DATASET'S WEBSITE LINK`. For compatibility with the current settings inside the sample code, we suggest you to have your datasets extracted into the following directory: MyProjects/Data/. Of course you can choose other places, but then you have to modify manually some parts of the main.m program which is located inside the sample_code directory, see bellow for more information.

C1.1.4. How to run CLOP?

We will describe first how to run a sample program using CLOP. Start MATLAB. Now change the current directory to where the sample code is located (in our case MyProjects/CLOP/sample_code/) using:

```
» cd MyProjects/CLOP/sample_code/
```

and run main.m program:

```
» main
```

If you have the datasets in MyProjects/Data/, this probably will run the main program successfully. You should notice from the MATLAB command window that the program displays some license agreement terms first and then loads a specific dataset and trains some algorithms on it, and finally tests the trained models and saves the results. If you experienced some problems check that the directory structure and path are correct.

C1.1.5. Compilation of SVC

The `svc` model is originally based on a C code, so depending on your machines configuration, there might be a need for compilation. We have provided pre-compiled versions for Windows and they usually run without problems. But for Linux, you need to compile them again. The source code for `svc` is located in CLOP/challenge_objects/packages/libsvm-mat-2.8-1. For Linux users, there are two different Makefiles: Makefile_orig is the one which was provided by the authors of the SVM package, and Makefile_amir is what I used on my machine to compile it. The only difference is these two files is the name of the C compiler. So you might need to go to one of these files and change environmental variables in the beginning of the file according your system's settings and which version of C compiler you have installed. For Windows users, you can run make.m to compile this object again, if needed. There is a README.TXT file which describes installation procedure in more details.

C1.1.6. More Details on Objects and Classes

Spider and CLOP both use object oriented programming style provided in MATLAB, which are called *class*es. If you do not know anything about MATLAB objects and/or object oriented programming, don't be scared away. You can learn how to use CLOP from examples, and in principle you will not need to deal with objects and classes. But you may definitely benefit from reading the (short) MATLAB help on objects. Briefly:

- An object is a structure (i.e. has data members), which has a number of programs (or methods) associated to it. The methods modify eventually the data members.
- The methods of an object *myObject* are stored in a directory named @myObject, which must be in your MATLAB path if you want to use the object (e.g. call addpath).
- One particular method, called *constructor* is named myObject. It is called (with eventually some parameters) to create a new object. For example:
  `» myObjectInstance = myObject(someParameters);`
- Once an instance is created, you can call a particular method. For example:
  `» myResults = myObjectMethod(myObjectInstance, someOtherParameters);`
- Note that myObjectInstance should be the first argument of myObjectMethod. Matlab *knows* that because myObjectInstance is an instance of myObject, it must call the method myObjectMethod found in the directory @myObject. This allows methods overloading (i.e. calling methods the same name for different objects.)
- Inheritance is supported in MATLAB, so an object may be derived from another object. A child object inherits from the methods of its parents. For example:
  `» myResults = myParentMethod(myObjectInstance, someOtherParameters);`
  In that case, the method myParentMethod is found in the parent directory @myObject-Parent, unless of course it has been overloaded by a method of the same name found in @myObject.

Refer to MATLAB's help for more information about classes and objects at `http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_prog/index.html`. Some useful functions for dealing with classes and objects in MATLAB are:

- **isa**: checks the class type
- **class**: returns the class
- **methods**: returns all the methods
- **struct**: lets you examine the data members
- **fieldnames**: returns a cell array with the field names

C1.2. Sample Program

C1.2.1. What is inside the main.m program?

The main program is written to provide participants with an easy-to-use template of how to utilize CLOP and build their own model with it [2]. Here we describe what are the different parts of this program and how to modify them for your own preferences.

1. **Initialization** This part of the code specifies some initial values for different variables which will be used through out the code:
    - It starts with cleaning the variables from workspace, cleaning the command window, and closing all figures. IF YOU DON'T WANT TO LOOSE YOUR AVAILABLE DATA AND FIGURES, REMOVE THIS PART OF THE CODE.

2. With *model*, here we mean any combination of algorithms that one might define to apply to a dataset. For example the sequence of normalizing the data, and then classifying it using a neural network can be described as a model.

- The next section defines the directory structure of your system. It is assumed that you want to have the following directories for your data and results, which is well designed to keep track of different activities and results required for a valid submission:
  - `my_root`: root directory where everything is there,
    Example: MyProjects/.
  - `data_dir`: data directory where datasets are,
    Example: MyProjects/Data/.
  - `code_dir`: code directory where CLOP is located,
    Example: MyProjects/CLOP/.
  - `resu_dir`: directory where results will be stored,
    Example: MyProjects/Results/.
  - `zip_dir`: directory where zip files will be stored,
    Example: MyProjects/Zipped/.
  - `model_dir`: directory where models will be stored,
    Example: MyProjects/Models/.
  - `score_dir`: directory where model scores will be stored,
    Example: MyProjects/Scores/.
- `ForceOverWrite` defines if you want the system to overwrite your previous results (if they are available in results, models, and zip directories). The default value is 1, which will not disturb you for overwriting questions, but be careful that in this case there is always risk of loosing valuable information and results.
- `DoNotLoadTestData` defines if you want the system to load the test set or not. This is used for cases where the size of test set is very large and loading it to memory will not be efficient for training phase. The default value is 1, which will not load the test sets.
- `MergeDataSets` defines if you want the system to merge training and validation sets. This is useful when you have labels for validation sets and you want to use them as extra training samples. The default value is 0, which will not merge the training and validation sets.
- `FoldNum` defines if you want the system to do a k-fold cross-validation or not, where k=FoldNum. The default value is 0, which will not perform cross-validation.
- The next step is to define which datasets you want to train, you can define this in the `dataset` cell array. For starting it is a better idea to start with just one dataset and check how your models are performing on it. In addition note that not always a single model is a good choice for all datasets.
- In order to keep your models separate from your current code, by default in training and testing section the program will call another function named model_examples.m with a user chosen model name and the dataset. You can easily define there which algorithms you would like to try over datasets under the chosen name, without a need to change the code in main program. This procedure will facilitate the model selection process too. For more information about model_examples.m, please refer to Section C1.4. You can define your model names in the `modelset` cell array.
- Now that you have defined your preferences, the system tries to generate and add the current directories (and their subsequent directories) to MATLAB's search path. This will finalize the initialization section.

2. **Train/Test** Here the code will start a loop for training and testing your models over different datasets you have specified above. Each loop consists of the following steps:

   - Loading the dataset and creating data structures: Since the original datasets provided from challenge website are stored in text files, this section will load all of them into proper data structures suitable for MATLAB. Because you will need these variables in your later experiments again and the process of loading them from text files usually takes longer time compared to loading variables from a MATLAB save file (.mat format), it also saves the data structures in the same data locations. The next time that you run the code over the same dataset, this section will automatically check whether the .mat format is available or not, and in the case of existing file it will load data from that .mat file, resulting in a very low loading time. Please refer to Section C1.3 for more information about the data variables.
   - Looping over different models: With the `modelset` array you can define several models to be tested over different datasets. This part of the code starts a loop over those models, which includes the following parts:
     - First of all, it calls `model_examples.m` function with the model and dataset names, in order to get back a valid CLOP or Spider model. Please refer to Section C1.4 for more information on how to define valid models.
     - If you have defined the algorithm to perform a cross-validation step, the system starts this step and trains the model FoldNum times.
     - Now that the program has the data and model, it is time to train the algorithms specified within the model, and obtain the training results. This is done simply by calling `train` function with model and training data.
     - After training, it will calculate the balanced error rate of the trained model, followed by computing a very tentative guess for test BER.
     - Now that the model is trained over the training set, it is time for testing it over validation and test sets, this task is accomplished easily by calling `test` function with model and proper data set.
     - The next step is to save the results into specified directories in appropriate official format which can be sent directly to the challenge website. It follows by saving the models too, which are needed for a valid final entry of the game.

3. **Make Archive** The finishing part of the code is to make a zip archive of all necessary files needed for submission to the challenge website for verification.

As it can be seen from these steps, the main program contains almost everything that a participant needs to enter the competition and produce proper results. We suggest you to make a backup copy of this program, and then modify different parts of it according to your interests. Since this is a model selection competition, you may want to replace the simple k-fold cross-validation provided as example by a more elaborate model selection strategy to select an optimum model or set of hyperparameters.

C1.3. Data Structure

The data object that has been used in CLOP, consists of the following fields (let's name the data variable `D`):

1. `D.train`: training data is stored here.

2. `D.valid`: validation data is stored here.
3. `D.test`: test data is stored here.

Each of these fields has two additional subfields, called `.X` and `.Y`. In `.X` field the raw data are stored in a matrix format with example arranged in rows and features in columns. The `.Y` field contains the labels for the corresponding set of examples in a one dimensional vector format. Note that if the labels are not provided in the datasets (always for test set and just for validation set before release of validation labels), then this field will be an empty vector. Additionally one can get statistics about the data using function `data_stats(D)`.

C1.4. Defining Models

For the purpose of the game, a valid *model* is defined as a combination of learning objects from a predefined list (type `whoisclop` at the MATLAB prompt to get a list of allowed CLOP learning objects; to check that a particular object is a valid CLOP object, type `isclop(object)`).

A typical model usually (but not necessarily) consists of the following parts[3]:

1. Preprocessing
2. Feature Selection
3. Classification
4. Postprocessing

The simplest model for our challenge can be just a classifier. Defining a model is a very simple task within CLOP framework. For example the following code makes a neural network classifier available with its default hyperparameters:

```
» myClassifier = neural
```

and this code defines a linear support vector machine classifier with a shrinkage (regularization) value of 0.1:

```
» myClassifier = svc({'shrinkage=0.1'})
```

Note the way that a hyperparameter value (shrinkage in this case) is passed to the object constructor (svc in this case). This is the general method to assign different values for hyperparameters rather than their default values. If you want to know what are the hyperparameters of each model, you can simply type » `help OBJECT_NAME` in MATLAB command window, where `OBJECT_NAME` is the name of a model. This will bring up information available for each model together with the list of hyperparameters used inside the model.

In general, there are two types of hyperparameters for each model: one is named as *public* and the other is *private*. For the competition, only the public hyperparameters can be tuned. The method to do that is via the constructor, as showed in examples. To find out which hyperparameters are public, use `default(OBJECTNAME)` in MATLAB command window. It is also possible to set the hyperparameters directly (outside of the constructor), but do this *at your own risks* since this may generate inconsistencies.

3. Note that the two first steps should not necessarily be in this order. In particular, since feature selection changes the number of features, normalization, which is a preprocessing method, may need to be done/redone after feature selection

C1.4.1. How to combine different models?

There exist two different ways to combine several models with each other. The first one is serial combination where outputs of each model is fed to the inputs of the next model. For example suppose that we want to normalize the data and then classify it with a neural network. This is a serial combination of algorithms where output of normalization step is supposed to be an input for neural network classifier. This type of combinations can be done easily using `chain` object. The following is the sample code you would need for the example described above:

```
» myModel = chain({normalize , neural})
```

Now when you train or test `myModel` with an input data, first the `normalize` algorithm will operate on the data and then it will pass the resulted data to `neural` object to classify it:

```
» [Outputs, myModelTrained] = train(myModel, D.train)
```

In this example it is supposed that `D.train` contains the training data. The `myModelTrained` is the resulted model after training, while `Outputs` contains information about predicted labels and other output variables. For more information about training and testing models, please refer to Section C1.5.

The other way of combining different algorithms is to combine different models with each other in a parallel style. This is usually known as *ensemble* methods in machine learning, and the goal is often to combine outputs of different learning algorithms to improve the classification performances. For example we want to have two classifiers, one neural network and one naive Bayes, trained on the same data, and then add their outputs to create the final results. The following code will generate the desired model and train it on `D.train` set:

```
» myNeural = chain({normalize, neural({'units=3', 'balance=1'})})
» myNaive = naive
» myClassifier = ensemble({myNeural, myNaive})
» [Outputs, myModelTrained] = train(myClassifier, D.train)
```

Note that for ensemble methods there are more sophisticated algorithms to train and combine outputs of different classifiers, like *bagging* and *boosting*. We consider those methods as individual classifiers and they will be described in next sections in details.

Note that the nth model of a chain or ensemble can be easily accessed with the curly bracket notation. For example in the following model, we want to access to the `neural` object, which is the 2nd element in chain and 3rd in ensemble object:

```
» myChain = chain({normalize, ensemble({svc, kridge, neural}), bias})
       chain
      {
      1:normalize center=0
      2:ensemble
      3:bias option=1
      }
» C{2}{3}
      neural units=10 shrinkage=1e-014 balance=0 maxiter=100
```

C1.4.2. Preprocessing Methods

The following section shows different preprocessing models available within CLOP for participants.

C1.4.2.1. STANDARDIZE

- **Description:** Standardization of the features (the columns of the data matrix are divided by their standard deviation; optionally, the mean is first subtracted if center=1). Note that a lot of methods benefit from this preprocessing, particularly neural networks.
- **Hyperparameters:** $center \in \{0,1\}$
- **Default Values:** $center = 1$
- **Example:**
  `» myModel = chain({normalize({'center=1'}) , naive})`

C1.4.2.2. NORMALIZE

- **Description:** Normalization of the lines of the data matrix (optionally the mean of the lines is subtracted first if center=1). Some methods benefit from this preprocessing, particularly the polynomial kernel methods. It is sometimes best to normalize after feature selection or both before and after.
- **Hyperparameters:** $center \in \{0,1\}$
- **Default Values:** $center = 0$
- **Example:**
  `» myModel = chain({standardize({'center=1'}) , naive})`

C1.4.2.3. SHIFT_N_SCALE

- **Description:** Performs this transformation globally on the data matrix $X = (X - \mathit{offset})/scale$, while offset and factor are set as hyperparameters, or subject to training. Optionally performs in addition $\log(1+X)$.
- **Hyperparameters:** $\mathit{offset} \in [-\infty, \infty], \mathit{factor} \in [0^+, \infty]$, take_log $\in \{0,1\}$
- **Default Values:** $\mathit{offset} = \min(X), \mathit{factor} = \max(X - \mathit{offset})$, take_log $= 0$
- **Example:**
  `» myModel = chain({shift_n_scale({'take_log=1'}) , naive})`

C1.4.2.4. PC_EXTRACT

- **Description:** Extract f_max number of features with principal component analysis.
- **Hyperparameters:** f_max $\in [0, \infty]$
- **Default Values:** f_max $= \infty$
- **Example:**
  `» myModel = chain({pc_extract({'f_max=50'}) , naive})`

C1.4.2.5. SUBSAMPLE

- **Description:** Make a subset of p_max number of the training patterns. It is possible to specify which patterns should be included in the resulting subset, by giving additional input to the `subsample` function which contains the indexes of those patterns. With balance hyperparameter you can specify whether the resulting subset should be a balanced set according to the number of class members or not. Note that subsampling can be combined with the ensemble object to implement bagging, for example[4]:

```
 for k=1:100
  baseModel{k}=chain({subsample({'p_max=1000', 'balance=1'}),
kridge});
 end
 myModel = chain({standardize, ensemble(baseModel, 'signed_output=1'),
bias});
```

- **Hyperparameters:** p_max $\in [0,\infty]$, *balance* $\in \{0,1\}$
- **Default Values:** p_max $= \infty$, *balance* $= 0$
- **Example:**
  `» myModel = chain({subsample({'p_max=100'}) , naive})`

C1.4.3. Feature Selection Methods

The following section shows different feature selection models available within CLOP for participants. The following notation for hyperparameters is commonly used for all feature selection algorithms:

- `f_max` defines the maximum number of features to be selected using the target model.
- `w_min` defines a threshold on the ranking criterion W of the target model. If $W(i) <=$ w_min, the feature i is eliminated. W is vector with non-negative values, so a negative value of w_min means all the features are kept.

C1.4.3.1. S2N

- **Description:** Signal-to-noise ratio coefficient for feature ranking. This method ranks features with the ratio of the absolute difference of the class means over the average class standard deviation. This criterion is similar to the Fisher criterion, the Ttest criterion, and the Pearson correlation coefficient. It can be thought of as a *linear univariate* feature ranking method. The top ranking features are selected and the new data matrix returned. The hyperparameters can be changed after construction of the object to allow users to vary the number of features without retraining.
- **Hyperparameters:** f_max $\in [0,\infty]$, w_min $\in [-\infty,\infty]$
- **Default Values:** f_max $= \infty$, w_min $= -\infty$
- **Example:**
  `» myModel = chain({s2n({'f_max=100'}) , naive})`

4. Note that this does not take advantage of the *out-of-bag examples* to compute an estimate of the test error, but can be considered as an approximation to bagging algorithm

C1.4.3.2. RELIEF

- **Description:** This method ranks features with the Relief coefficient. Relief is a method based on the nearest neighbors scoring features according to their relevance, in the context of others. It is a non-linear multivariate feature ranking method. In our implementation, it is slow for large numbers of patterns because we compute the entire distance matrix. We chunk it if it is very large, to avoid memory problems. The top ranking features are selected and the new data matrix returned. The hyperparameters can be changed after construction of the object to allow users to vary the number of features without retraining. `k_num` defines the number of neighbors in the Relief algorithm.
- **Hyperparameters:** f_max $\in [0,\infty]$, w_min $\in [-\infty,\infty]$, k_num $\in [0,\infty]$
- **Default Values:** f_max $=\infty$, w_min $=-\infty$, k_num $=4$
- **Example:**
  ```
  » myModel = chain({relief({'f_max=100', 'k_num=5'}) , naive})
  ```

C1.4.3.3. GS

- **Description:** Forward feature selection with Gram-Schmidt orthogonalization. This is a forward selection method creating nested subsets of complementary features. The top ranking features are selected and the new data matrix returned. Note that if you want to change the value of `f_max` after training, it cannot be set to a larger number than the number `f_max` used for training (or it will be chopped at f_max).
- **Hyperparameters:** f_max $\in [0,\infty]$
- **Default Values:** f_max $=\infty$
- **Example:**
  ```
  » myModel = chain({gs({'f_max=100'}) , naive})
  ```

C1.4.3.4. RFFS

- **Description:** Random Forest used as feature selection filter. The *child* argument, which may be passed in the argument array, is an `rf` object, with defined hyperparameters. If no child is provided, an `rf` with default values is used.
- **Hyperparameters:** f_max $\in [0,\infty]$, w_min $\in [-\infty,\infty]$, child
- **Default Values:** f_max $=\infty$, w_min $=-\infty$, child=rf
- **Example:**
  ```
  » myModel = chain({rffs({'w_min=0.2'}) , naive})
  ```

C1.4.3.5. SVCRFE

- **Description:** Recursive Feature Elimination filter using SVC. This is a backward elimination method creating nested subsets of complementary features. The *child* argument, which passed in the argument array, is an `svc` object, with defined hyperparameters. If no child is provided, a linear `svc` with default values is used.
- **Hyperparameters:** f_max $\in [0,\infty]$, child

- **Default Values:** f_max $= \infty$, child=svc
- **Example:**
  ```
  » myModel = chain({svcrfe({'f_max=100'}) , naive})
  ```

C1.4.4. Classification Methods

The following section shows different classification models available within CLOP for participants. The following notation for hyperparameters is commonly used for all classification algorithms:

- For kernel methods, the general kernel equation is

  $$k(x_1,x_2) = (coef0 + x_1.x_2)^{degree} \exp(-gamma\,\|x_1 - x_2\|^2)$$

  Note that $(.)$ is vector dot product operator. Some examples of mostly used kernels are:

  1. *Linear Kernel:* $degree = 1, coef0 = 0, gamma = 0$, $k(x_1,x_2) = (x_1.x_2)$
  2. *Polynomial degree N Kernel:* $degree = N, coef0 = a, gamma = 0$, $k(x_1,x_2) = (a + x_1.x_2)^N$
  3. *RBF Kernel:* $degree = 0, coef0 = 0, gamma = \gamma$, $k(x_1,x_2) = \exp(-\gamma\|x_1 - x_2\|^2)$

- `units` usually defines the number of substructures needed for each algorithm. For example, in the case of a `neural` object, it defines the number of hidden neurons, while in boosting methods, it is the number of weak learners to be used in the algorithm.
- `balance` is a flag used to specify if the algorithm should balance the number of class members before training using subsampling method.
- `shrinkage` defines usually the regularization parameter used in each algorithm.

C1.4.4.1. KRIDGE

- **Description:** Kernel ridge regression. This object trains a regression learning machine using the least square loss and a weight-decay or *ridge* specified by the *shrinkage* parameter.
- **Hyperparameters:** coef0 $\in [0,\infty]$, degree $\in [0,\infty]$, gamma $\in [0,\infty]$, shrinkage $\in [0,\infty]$, balance $\in \{0,1\}$
- **Default Values:** coef0 $= 1$, degree $= 1$, gamma $= 0$, shrinkage $= 1e-14$, balance $= 0$
- **Example:**
  ```
  » myModel = chain({normalize , kridge({'coef0=0.1', 'degree=2'})})
  ```

C1.4.4.2. SVC

- **Description:** Support vector classifier. This object trains a *2-norm* SVC, i.e. the shrinkage parameter is similar to the ridge in kridge. There is no box constraint (bound on the alphas).
- **Hyperparameters:** coef0 $\in [0,\infty]$, degree $\in [0,\infty]$, gamma $\in [0,\infty]$, shrinkage $\in [0,\infty]$
- **Default Values:** coef0 $= 0$, degree $= 1$, gamma $= 0$, shrinkage $= 1e-14$

- **Example:**
  » `myModel = chain({normalize , svc({'degree=0', 'gamma=0.1', 'shrinkage=0.1'})})`

C1.4.4.3. NAIVE

- **Description:** Naive Bayes classifier. Two separate implementations are made for binary and continuous variables (the object switches automatically). For binary variables, the model is based on frequency counts; for continuous variable, the model is a Gaussian classifier.
- **Hyperparameters:** None
- **Default Values:** None
- **Example:**
  » `myModel = chain({normalize , naive})`

C1.4.4.4. NEURAL

- **Description:** Neural networks classifier. Two layer neural network (a single layer of hidden units). The shrinkage corresponds to a weight decay or the effect of a Bayesian Gaussian prior. `units` is the number of hidden neurons, and `maxiter` defines the number of training epochs.
- **Hyperparameters:** units $\in [0,\infty]$, maxiter $\in [0,\infty]$, shrinkage $\in [0,\infty]$, balance $\in \{0,1\}$
- **Default Values:** units $= 10$, maxiter $= 100$, shrinkage $= 1e-14$, balance $= 0$
- **Example:**
  » `myModel = chain({normalize , neural({'units=5'})})`

C1.4.4.5. RF

- **Description:** Random Forest classifier. This object builds an ensemble of tree classifiers with 2 elements of randomness: (1) each tree is trained on a randomly drawn *bootstrap* subsample of the data (approximately 2/3 of the examples); (2) for each node, the feature to split the node is selected among a random subset of all features. `units` is the number of trees, and `mtry` defines the number of candidate feature per split.
- **Hyperparameters:** units $\in [0,\infty]$, mtry $\in [0,\infty]$
- **Default Values:** units $= 100$, mtry $= \sqrt{\text{Feat\_Num}}$
- **Example:**
  » `myModel = chain({normalize , rf({'units=200'})})`

C1.4.4.6. GENTLEBOOST

- **Description:** This object builds an ensemble of classifiers (called weak learners) by sequentially adding weak learners trained on a subsample of the data. The subsamples are biased toward the examples misclassified by the previous weak learner. Gentleboost is a variant of the adaboost algorithm, which is less sensitive to data outliers because it puts less weight on misclassified examples and weighs the weak learners evenly. The base classifier is defined separately and can be any of the classification methods defined above. `units` is the number of weak learners, `subratio` defines the ratio of subsampling compared to the original dataset, and `rejNum` is the number of different trials to get a weak learner before stopping the whole iteration, if the weighted error of a weak learner is over 0.5.

- **Hyperparameters:** units $\in \{1,2,...,\infty\}$, subratio $\in [0,1]$, rejNum $\in \{1,2,...,\infty\}$, balance $\in \{0,1\}$
- **Default Values:** units $= 5$, subratio $= 0.9$, rejNum $= 3$, balance $= 1$
- **Example:**

```
» myBase = naive
» myModel = chain({normalize , gentleboost(myBase, {'units=10'})})
```

C1.4.5. Postprocessing Methods

The following section shows different postprocessing models available within CLOP for participants.

C1.4.5.1. BIAS

- **Description:** Bias optimization. It calculates a threshold value that will be applied to the outputs of classifier, optimizing several factors listed bellow. The `option` can be one of following items:
  1. minimum of the BER.
  2. break-even-point between sensitivity and specificity.
  3. average of the two previous results if they do not differ a lot, otherwise zero.
  4. values that gives the same fraction of positive responses on the test data than on the training data (transduction).
- **Hyperparameters:** $option \in \{1,2,3,4\}$
- **Default Values:** $option = 1$
- **Example:**

```
» myModel = chain({normalize({'center=1'}) , naive, bias({'option=2'})}
```

C1.4.6. Model Selection Methods

In the main.m example script we provide, we illustrate how the Spider can be used to perform a selection of the CLOP models. In the script, we use the Spider object cv, which implements the traditional k-fold cross-validation algorithm. Since this is a model selection contest, we encourage you to develop your own algorithms for this purpose. The main goal of developing CLOP was to provide the participants with a diverse set of learning algorithms that should be sufficient to achieve competitive results. By restricting them to the use of CLOP models, they can focus on model selection rather than spending time tuning their own classification algorithms. So use this opportunity and develop your ideas for better model selection systems. This may include developing better:

- hyperparameter search strategies

- model architectures (by combining CLOP modules with combinations of chains and ensembles)
- model assessment (e.g. cross-validation, complexity penalization, etc.)

The Spider provides a few objects implementing some of these tasks:

- `param`: this object allows you to specify ranges of parameter values
- `cv`: this object implements k-fold cross-validation
- `gridsel`: this object integrates the functions of param and cv to perform model selection with grid search

It is noteworthy that the model selection game is as much an *ensemble method* game as it is a *model selection* game, since hyperparameter selection can to a large extent be circumvented by using an ensemble of methods.

Note that objects like `r2w2_sel` and `bayessel` are specific to the Spider implementation of SVMs, which is different from the one of CLOP. `r2w2_sel` concerns feature selection. Please do not use it since we restricted the feature selection methods to the ones provided in this manual. `bayessel` adjust the C hyperparameter in 1-norm SVM. This is not useful to CLOP users, since in CLOP, we provide only a 2-norm SVM.

C1.4.7. How to use model_examples.m?

In order to keep the model creation separate from the main code that might be used by participants, we use another program which is called model_examples.m. Using this scheme, one can easily define lots of different models without a need to modify the main.m program. This style is not necessary but we recommend you to follow the proposed structural system.

In model_examples.m, the first section is used to checks empty calls of this function. Next, there exist several proposed models and methods which some of them are common for all datasets, and some has specific instructions to deal with different characteristics of different datasets. Check this section carefully to get a general idea of how to create and write your own models. Note that the proposed models are not optimal for this challenge, and we have created them just to show different capabilities of CLOP. In order to be competitive in the challenge, you would need to create your models and tune their parameters according to any model selection strategy that you have in your mind.

C1.5. Training and Testing

After you have defined your models, you can easily train them using `train` command as bellow:

```
» myModel = chain({normalize({'center=1'}) , naive, bias({'option=2'})})
» [TrainOutputs, myModelTrained] = train(myModel, D.train)
```

In this example, we assume that there is a data object named `D` already available in the workspace which contains training data and labels. The `train` function returns the original model with tuned parameters in `myModelTrained`, while `TrainOutputs` will have predicted labels as `.X` subfield, (`TrainOutputs.X`) and original target training labels as `.Y` subfield, (`TrainOutputs.Y`). The `TrainOutput` object can later be used to evaluate the performance of training phase, like computing the BER, AUC, and etc.

After training a model, we usually are interested in testing the algorithm over unseen data examples. This can be easily done with `test` function as follows:

```
» TestOutputs = test(myModelTrained, D.test)
```

Now the `TestOutput` has the predicted labels again in `.X` subfield, but since the labels for test sets will not be provided, the `.Y` field will be empty. The `TestOutput.X` can be passed further to other functions such as `save_outputs` to generate output files which are suitable to send directly to online website.

C1.6. Results

The previous sections provided a variety of models available in CLOP, in order to successfully create an optimal learning machine to enter the competition. In order to show you that CLOP models are competitive algorithms compared to the ones used by the Performance Prediction Challenge (PPC) participants, we show in Table C1.1 the results we obtained with some CLOP objects designed without extensively searching for an optimum model (CLOP baseline models). All these objects are within the tenth best percentile of the methods submitted by the challenge finalists. We will not reveal the structure of these models until the end of contest to encourage people to come up with their own architecture and develop efficient hyperparameter optimization strategies.

Table C1.1: **Results comparison.** We report the test BER over different datasets. The PPC results correspond to the data split used for the Performance Prediction Challenge, with validation labels. There is a new data split of the same data that should be used for the game (last column), for which the validation set labels are not available. For the PPC datasets, the performance reported correspond to training with both training and validation data. For the game, only training labels are available during the development period.

Entries	Best PPC entries	CLOP baseline	CLOP baseline
Data split	Original PPC split with valid. labels	Original PPC split with valid. labels	New game split NO valid. labels
ADA	0.1696 ± 0.0021	0.1832	0.1808
GINA	0.0288 ± 0.0009	0.0293	0.0262
HIVA	0.2757 ± 0.0068	0.3038	0.2988
NOVA	0.0433 ± 0.0017	0.0489	0.0440
SYLVA	0.0053 ± 0.0002	0.0100	0.0108

C1.7. Credits

The organization of this competition was a team effort to which many have participated. We are particularly grateful to Olivier Guyon (MisterP.net) who implemented the back-end of the web

site. The front-end follows the design of Steve Gunn (University of Southampton), formerly used for the NIPS 2003 feature selection challenge. We are thankful to Bernd Fischer (ETH Zurich) for administering the computer resources. Other advisors and beta-testers are gratefully acknowledged: Gideon Dror, Joachim Buhmann, Yoshua Bengio (University of Montreal), Asa Ben-Hur (Colorado State university), Lambert Schomaker (University of Groningen), and Vladimir Vapnik (NEC, Princeton). The Challenge Learning Object Package (CLOP) is based on code to which many people have contributed: The creators of the spider: Jason Weston, Andre Elisseeff, Gikhan Bakir, Fabian Sinz. The developers of the packages attached: Chih-Chung Chang and Chih-JenLin Jun-Cheng (LIBSVM), Chen, Kuan-Jen Peng, Chih-Yuan Yan, Chih-Huai Cheng, and Rong-En Fan (LIBSVM Matlab interface), Junshui Ma and Yi Zhao (second LIBSVM Matlab interface), Leo Breiman and Adele Cutler (Random Forests), Ting Wang (RF Matlab interface), Ian Nabney and Christopher Bishop (NETLAB), Thorsten Joachims (SVM-Light), Ronan Collobert (SVM Torch II), Jez Hill, Jan Eichhorn, Rodrigo Fernandez, Holger Froehlich, Gorden Jemwa, Kiyoung Yang, Chirag Patel, Sergio Rojas. This project is supported by the National Science Foundation under Grant N0. ECS-0424142. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Predicant biosciences, Microsoft, and Unipen provided additional support permitting to grant prizes to the winners.