UNIVERSIDAD AUTÓNOMA DE MADRID

Prediction Based on Averages over Automatically Induced Learners: Ensemble Methods and Bayesian Techniques

by

Daniel Hernández-Lobato

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the
Escuela Politécnica Superior
Computer Science Department

under the supervision of Alberto Suárez González

November 2009

"Roads? Where we're going we don't need roads."

Dr. Emmett Brown

Abstract

Ensemble methods and Bayesian techniques are two learning paradigms that can be useful to alleviate the difficulties associated with automatic induction from a limited amount of data in the presence of noise. Instead of considering a single hypothesis for prediction, these methods take into account the outputs of a collection of hypotheses compatible with the observed data. Averaging the predictions of different learners provides a mechanism to produce more accurate and robust decisions. However, the practical use of ensembles and Bayesian techniques in machine learning presents some complications. Specifically, ensemble methods have large storage requirements. The predictors of the ensemble need to be kept in memory so that they can be readily accessed. Furthermore, computing the final ensemble decision requires querying every predictor in the ensemble. Thus, the prediction cost increases linearly with the ensemble size. In general, it is also difficult to estimate an appropriate value for the size of the ensemble. On the other hand, Bayesian approaches require the evaluation of multi-dimensional integrals or summations with an exponentially large number of terms that are often intractable. In practice, these calculations are made using approximate algorithms that can be computationally expensive. This thesis addresses some of these shortcomings and proposes novel applications of ensemble methods and Bayesian techniques in supervised learning tasks of practical interest.

In the first part of this thesis we analyze different pruning methods that reduce the memory requirements and prediction times of ensembles. These methods replace the original ensemble by a subensemble with good generalization properties. We show that identifying the subensemble that is optimal in terms of the training error is possible only in regression ensembles of intermediate size. For larger ensembles two approximate methods are analyzed: ordered aggregation and SDP-pruning. Both SDP-pruning and ordered aggregation select subensembles that outperform the original ensemble. In classification ensembles it is possible to make inference about the final ensemble prediction by querying only a fraction of the total classifiers in the ensemble. This is the basis of a novel ensemble pruning method: instance-based (IB) pruning. IB-pruning produces a large speed-up of the classification process without significantly deteriorating the generalization performance of the ensemble. This part of the thesis also describes a statistical procedure for determining an adequate size for the ensemble. The probabilistic framework introduced in IB-pruning can be used to infer the size of a classification ensemble so that the resulting ensemble predicts the same class label as an ensemble of infinite size with a specified confidence level.

The second part of this thesis proposes novel applications of Bayesian techniques with a focus on computational efficiency. Specifically, the expectation propagation (EP) algorithm is used as an alternative to more computationally expensive methods such as Markov chain Monte Carlo or type-II maximum likelihood estimation. In this part of the thesis we introduce the Bayes machine for binary classification. In this Bayesian classifier the posterior distribution of a parameter that quantifies the level of noise in the class labels is inferred from the data. This posterior distribution can be efficiently approximated using the EP algorithm. When EP is used to compute the approximation, the Bayes machine does not require any re-training to estimate this parameter. The cost of training the Bayes machine can be further reduced using a sparse representation. This representation is found by a greedy algorithm whose performance is improved by considering additional refining iterations. Finally, we show that EP can be used to approximate the posterior distribution of a Bayesian model for the classification of microarray data. The EP algorithm significantly reduces the training cost of this model and is useful to identify relevant genes for subsequent analysis.

Resumen

Los métodos basados en conjuntos y las técnicas Bayesianas son dos paradigmas de aprendizaje que pueden ser útiles para aliviar los problemas asociados a la inducción automática cuando la cantidad de ejemplos de los que se dispone es limitada y cuando estos datos están contaminados por ruido. En lugar de considerar una sola hipótesis para predecir, estos métodos tienen en cuenta las predicciones de una colección de hipótesis compatibles con los datos observados. Promediar estas predicciones proporciona un mecanismo para obtener decisiones finales más precisas y robustas. Sin embargo, el uso práctico de los métodos basados en conjuntos y las técnicas Bayesianas presenta ciertas complicaciones. En particular, los requisitos de memoria de los métodos basados en conjuntos son elevados. Los elementos del conjunto necesitan ser almacenados en memoria para ser accesibles de manera eficiente. Además, el cálculo de la decisión final del conjunto requiere obtener la predicción de todos y cada uno de los elementos del conjunto. De este modo, el coste de predicción del conjunto se incrementa linealmente con su tamaño. En general, también es difícil estimar un valor apropiado para este tamaño. Por otro lado, los enfoques Bayesianos requieren la evaluación de integrales multidimensionales o sumas con un número exponencial de términos, que frecuentemente son intratables. En la práctica estos cálculos se llevan a cabo utilizando algoritmos aproximados que pueden ser costosos computacionalmente. En esta tesis se introducen y analizan una serie de mejoras para los métodos de aprendizaje automático basados en conjuntos y las técnicas Bayesianas. Así mismo, se proponen nuevas aplicaciones de estos métodos en problemas de interés práctico.

En la primera parte de esta tesis analizamos distintos métodos de poda que reducen los requisitos de memoria y el tiempo de predicción de los métodos basados en conjuntos. Estos métodos sustituyen el conjunto original por un subconjunto con buenas propiedades de generalización. En la práctica, el subconjunto que es óptimo en términos de error de entrenamiento sólo se puede encontrar en conjuntos de regresión de tamaño intermedio. Para conjuntos mayores, se analizan dos métodos aproximados: agregación ordenada y poda SDP. Tanto agregación ordenada como poda SDP seleccionan subconjuntos que mejoran la capacidad de generalización del conjunto original. En conjuntos de clasificación es posible hacer inferencia sobre la decisión final del conjunto tras obtener la predicción de sólo unos pocos de los clasificadores del conjunto. Esta es la base para un nuevo método de poda llamado poda basada en instancia (poda BI). La poda BI acelera el proceso de clasificación sin aumentar significativamente el error del conjunto. Esta parte de la tesis también describe un procedimiento estadístico para fijar un tamaño adecuado para el conjunto. En particular, el marco probabilístico introducido en la poda BI se puede utilizar para fijar el tamaño de un conjunto de clasificadores tal que el conjunto resultante prediga la misma clase que un conjunto de tamaño infinito con un nivel de confianza especificado.

La segunda parte de esta tesis propone nuevas aplicaciones de las técnicas Bayesianas con énfasis en la eficiencia computacional. Específicamente, el algoritmo de propagación de esperanzas (PE) se utiliza como alternativa a otros métodos computacionalmente más costosos como pueden ser los métodos Monte Carlo o la estimación basada en máxima verosimilitud de tipo II. En esta parte de la tesis introducimos la máquina de Bayes para clasificación binaria. En este clasificador Bayesiano la distribución posterior de un parámetro que cuantifica el nivel de ruido en las etiquetas de clase es inferida a partir de los datos. Esta distribución posterior se puede aproximar eficientemente utilizando el algoritmo PE. Cuando el algoritmo PE se utiliza para calcular la aproximación, la máquina de Bayes no requiere re-entrenamiento para estimar este parámetro. El coste de entrenar la máquina de Bayes se puede reducir aún más utilizando una representación dispersa. Esta representación se puede encontrar utilizando un algoritmo codicioso cuya eficiencia se puede mejorar considerando iteraciones adicionales de refinación. Finalmente, mostramos que el algoritmo PE se puede utilizar para aproximar la distribución posterior de un modelo Bayesiano para la clasificación de datos de microarray. El algoritmo PE reduce significativamente el coste de entrenamiento de este modelo y es útil para identificar genes relevantes para su posterior análisis.

Acknowledgements

First of all, I would like to thank my supervisor Alberto Suárez González for introducing me into the machine learning research community and for his guiding during all these years. I certainly appreciate the freedom he has given me for satisfying my curiosity about different research topics. Furthermore, his comments and discussions have been not only instructive, but also of invaluable help.

I would also like to thank Gonzalo Martínez-Muñoz for helping me at the beginning of my post-graduate studies and for many interesting talks.

I express my gratitude to Bert Kappen for giving me the opportunity to join his research group, SNN Nijmegen, for a three-month period in 2006. It is during this stay that I became interested in using (Bayesian) probabilities to solve machine learning problems. I would also like to thank all the members of his research group at that time, specially to Bastian Wemmenhove, for many interesting discussions, and Annet Wanders, for her help at my arrival.

I am grateful to my brother, José Miguel Hernández-Lobato, for joining me in the adventure of post-graduate studies, and for many interesting discussions and comments about different machine learning problems. I am also grateful to my parents, who have always supported and helped me throughout this dissertation.

I appreciate the friendship of Alexandra Dumitrescu and Cristina Bogdanschi during all these years. You are the best girls I have ever met!

I also express my gratitude to Hyun-Chul Kim for providing his implementation of the EM-EP Gaussian process classifier.

Finally, I would like to thank all the members of the B-408 Lab, the secretaries of the computer science department for their effort, my friends in Collado Villalba for their support and all the fellows from the climbing gym of Moralzarzal.

Contents

Abbreviations

ADF	**A**ssumed **D**ensity **F**iltering
BM	**B**ayes **M**achine
BPM	**B**ayes **P**oint **M**achine
CART	**C**lassification **A**nd **R**egression **T**rees
DLDA	**D**iagonal **L**inear **D**iscriminant **A**nalysis
EM	**E**xpectation **M**aximization
EP	**E**xpectation **P**ropagation
FR	**F**ixed **R**ate
GPC	**G**aussian **P**rocess **C**lassifier
GP	**G**aussian **P**rocess
IB	**I**nstance **B**ased
IVM	**I**nformative **V**ector **M**achine
KL	**K**ullback **L**iebler
KNN	**K**-**N**earest **N**eighbours
MCMC	**M**arkov **C**hain **M**onte **C**arlo
MC	**M**ax **C**ut
MSE	**M**ean **S**quared **E**rror
NSC	**N**earest **S**runken **C**entroids
OA	**O**rdered **A**ggregation
OOB	**O**ut-**O**f-**B**ag
PAC	**P**robably-**A**pproximately-**C**orrect
PCA	**P**rincipal **C**omponents **A**nalysis
RFE	**R**ecursive **F**eature **E**limination
RF	**R**andom **F**orest
RVM	**R**elevance **V**ector **M**achine
SDP	**S**emi **D**efinite **P**rogramming
SV	**S**upport **V**ector
SVM	**S**upport **V**ector **M**achine

To my family

Chapter 1

Introduction

Machine learning is a branch of computer science concerned with the design and implementation of methods to automatically induce patterns from data (Bishop, 2006; Hastie et al., 2001). In this context, the term pattern denotes a recurrent regularity in the data that is predictable by some rule. A successful machine learning method is expected to induce such a rule from the data with little or no human supervision. While machine learning is a broad research field with many different areas, this thesis focuses on a particular subfield called *supervised machine learning*. In a supervised learning setting we are given a training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ with observations of the attribute vector $\mathbf{x}$ and the corresponding target variable y. The task of interest is to learn a predictor for y when only the vector $\mathbf{x}$ is observed. If y takes values from a continuous set, e.g. $\mathbb{R}$, we talk about *regression* problems. If y takes values from a finite set, e.g. $\{-1, 1\}$, the problem is called *classification*. In classification problems, y is referred to as the class label.

A predictor for y is a rule or hypothesis f that produces a value of this variable given the attribute vector $\mathbf{x}$, i.e. $y = f(\mathbf{x})$. Let f_0 be the unknown optimal rule, in terms of prediction accuracy, that generates y from $\mathbf{x}$. An estimate of f_0, denoted $\hat{f}$, can be automatically induced by fitting a model to $\mathcal{D}$ (Bishop, 1996; Hastie et al., 2001). This model assumes that f_0 can be well approximated by some rule contained in a restricted space of candidate rules $\mathcal{F}$. The fitting process consists in estimating some parameters $\boldsymbol{\theta}$ that identify $\hat{f}$ within $\mathcal{F}$. The estimation of $\boldsymbol{\theta}$ is carried out so that the predictions of $\hat{f}$ are accurate for the instances in $\mathcal{D}$. However, the fitting process has to be implemented carefully to ensure that the predictions of $\hat{f}$ are also correct for instances that are not in $\mathcal{D}$. When this is satisfied we say that $\hat{f}$ has good generalization properties.

A common difficulty in supervised machine learning is the limited number of training instances. Furthermore, the training data may be contaminated with random noise (Bishop, 2006). Under these circumstances, there is uncertainty in the estimation of f_0. The estimates of the model parameters can have a large variance because of the reduced amount of data available for induction. Furthermore, we do not know which of the regularity patterns present in the training data are intrinsic or simply appear by chance as a result of random fluctuations. This latter issue affects the selection of the set of candidate rules $\mathcal{F}$. If the rules in $\mathcal{F}$ are too simple, the resulting hypothesis $\hat{f}$ may fail to capture the patterns in the data. By contrast, if the rules in $\mathcal{F}$ are too complex, $\hat{f}$ is likely to capture random fluctuations that are not characteristic of the learning task. Both lead to a reduction in the prediction accuracy of $\hat{f}$ when evaluated on new data.

1.1 An Illustrative Example: Fitting a Polynomial Curve

To illustrate the role of uncertainty in the problem of automatic hypothesis induction we analyze a supervised learning task similar to the one described in (Bishop, 2006). Assume that we have a training set $\mathcal{D} = \{(x_i, y_i) : i = 1, \ldots, n\}$, where each x_i is drawn from a uniform distribution in the interval $[0, 1]$ and the corresponding value of y_i is generated by a polynomial function f_0 with some additive noise

$$y_i = f_0(x) + \epsilon_i = -1.7x^4 + 27x^3 - 37.5x^2 + 12.7x - 0.25 + \epsilon_i \,. \tag{1.1}$$

The noise term ϵ_i follows a Gaussian distribution with zero mean and standard deviation equal to $1/4$. The properties of these data are characteristic of many real-world datasets. That is, there is an underlying regularity (the polynomial), which we wish to learn, but the individual observations are corrupted by random noise (Bishop, 2006).

We now use the information in $\mathcal{D}$ to construct an estimate $\hat{f}$ of f_0. The first step is to specify a model for the set of candidate rules $\mathcal{F}$. Because in this case we know the true parametric form of f_0, we choose $\mathcal{F}$ to be the set of polynomials of degree four with coefficients $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3, \theta_4, \theta_5)^T$, where the superscript T means transpose. The functions in $\mathcal{F}$ are therefore parameterized in terms of $\boldsymbol{\theta}$ as

$$f(x; \boldsymbol{\theta}) = \sum_{i=1}^{5} \theta_i x^{i-1} \,. \tag{1.2}$$

Using this representation f_0 is given by $f(x; \boldsymbol{\theta}_0)$, where

$$\boldsymbol{\theta}_0 = (-0.25, 12.7, -37.5, 27, -1.7)^T \,. \tag{1.3}$$

In actual applications the functional form of f_0 is often unknown. Therefore, we are left with the problem of choosing a set $\mathcal{F}$ that contains candidate rules similar to f_0. Even when the selection of $\mathcal{F}$ is appropriate, we need to estimate the true model parameters $\boldsymbol{\theta}_0$.

Let $\hat{\boldsymbol{\theta}}$ be an estimate of $\boldsymbol{\theta}_0$. The corresponding estimate $\hat{f}$ of f_0 is $f(x; \hat{\boldsymbol{\theta}})$. The estimation of $\boldsymbol{\theta}_0$ is carried out by fitting the model parameters to the training data. This fitting process can be implemented by minimizing with respect to $\boldsymbol{\theta}$ a loss function $L(\boldsymbol{\theta})$ that measures the average disagreement between $f(x_i; \boldsymbol{\theta})$ and y_i for the pairs (x_i, y_i) in $\mathcal{D}$ (Bishop, 2006). A typical choice for the loss function L in regression is the sum of squared differences between the predictions of the model and the corresponding true target variables

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left(f(x_i; \boldsymbol{\theta}) - y_i\right)^2 \,. \tag{1.4}$$

The minimizer of (1.4) is $\hat{\boldsymbol{\theta}}$, a consistent estimate of $\boldsymbol{\theta}_0$, i.e. $\hat{\boldsymbol{\theta}} \to \boldsymbol{\theta}_0$ as $n \to \infty$. However, for datasets of finite size $\hat{\boldsymbol{\theta}}$ need not be close to $\boldsymbol{\theta}_0$. The reason for these deviations is the presence of random fluctuations in the training data and the noise in the target variables. Figure 1.1 (left) illustrates this result for a training set $\mathcal{D}$ with $n = 10$ instances by displaying the pairs (x_i, y_i), the true underlying function f_0, and the estimate $\hat{f}$ that is obtained from the minimization of (1.4). While $\hat{f}$ is close to f_0, there is still a small discrepancy between these two functions. The origin of this discrepancy is that $\hat{\boldsymbol{\theta}}$ is different from $\boldsymbol{\theta}_0$. In particular, $\hat{\boldsymbol{\theta}}$ behaves like a random variable whose value fluctuates for different realizations of the training set (Hastie et al., 2001). Figure 1.1

(right) illustrates this result by displaying 100 estimates of f_0 obtained for 100 different realizations of $\mathcal{D}$. This large variability in the estimation of the model parameters may result in an estimate $\hat{\boldsymbol{\theta}}$ that differs significantly from $\boldsymbol{\theta}_0$. When this happens, $\hat{f}$ can have a poor predictive performance on new data.

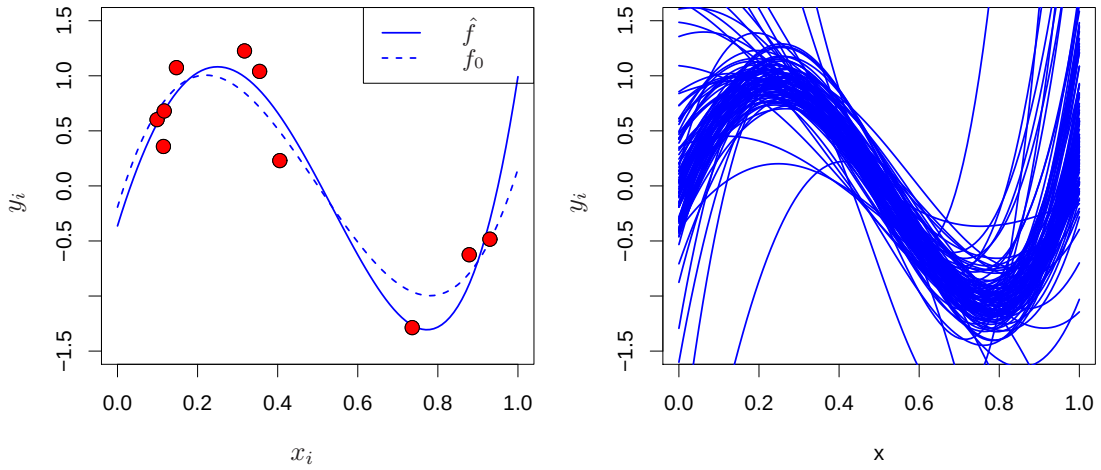

FIGURE 1.1: (left) Graphical representation of a training set $\mathcal{D}$ of size $n = 10$ (red dots), the true underlying function f_0 that associates each y_i to each x_i and the estimate $\hat{f}$ that is obtained from the minimization of (1.4). (right) Graphical representation of 100 estimates of f_0 obtained under 100 different realizations of $\mathcal{D}$.

Note that the functional form of f_0 is generally unknown. In this case, machine learning methods consider a set $\mathcal{M} = \{m_1, m_2, \ldots, m_M\}$ of alternative candidate models or functional representations for f_0. This set can include, for example, polynomial functions of different degrees. From $\mathcal{M}$, the learning method must then select the functional representation that gives the best prediction accuracy on new data instances. The problem is that the prediction accuracy in the training set can be a poor indicator of how well a model performs on new data. Figure 1.2 illustrates this point by displaying the fit of a linear model (left) and a polynomial of degree nine (right) to $\mathcal{D}$. This is the interpolating polynomial that goes through all the observed pairs (x_i, y_i) in $\mathcal{D}$. Therefore, its prediction error is zero on the training data. Considering only the training set it seems that the interpolating polynomial achieves the best fit. However, when comparing both fits with f_0, it is apparent that the linear model has better generalization accuracy. In particular, the interpolating polynomial exhibits large oscillations in regions without observations. This behavior is known as over-fitting and is generally the consequence of using an unnecessarily complex model to describe the training data (Bishop, 2006). Over-fitting is an undesirable result that should be avoided. When the number of training instances increases, over-fitting typically becomes less severe because the sampling of the space of attributes is more complete.

In summary, supervised machine learning methods face severe problems when the amount of data available for induction is small and when the training instances are contaminated with noise. First, the selection of an appropriate model is difficult. If the model is too simple, it can fail to describe some of the regularities of the data. On the other hand, if the model is too complex, it can fit spurious patterns in the training set. This typically leads to over-fitting. Second, even assuming that the selection of the model is correct, the estimation of the model parameters can also be problematic. In

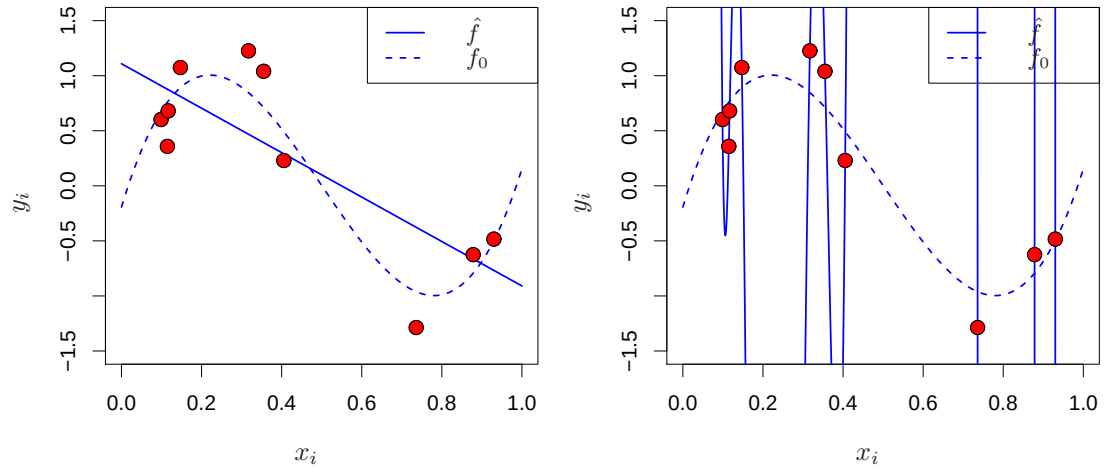

FIGURE 1.2: Graphical representation of the training set $\mathcal{D}$ of size $n = 10$ (red dots), the true underlying function f that associates each y_i to each x_i and the estimates that are obtained from fitting a linear model (left) and a polynomial of degree nine (right) to this dataset.

particular, the empirical estimate $\hat{\boldsymbol{\theta}}$ can have a large variance for different realizations of the training set. The consequence of this variability is that $\hat{\boldsymbol{\theta}}$ may differ significantly from the actual value $\boldsymbol{\theta}_0$. As a result, $\hat{f}$ can have a poor generalization performance. When the amount of data available to perform the estimations is large the severity of these problems is reduced. However, for small datasets it is important to take them into account to obtain reliable predictions. In the next section we describe how ensemble methods and Bayesian machine learning can be used to address these shortcomings.

1.2 Ensemble Methods and Bayesian Machine Learning

The uncertainties about the appropriate type of model and about the correct model parameters that arise in the process of learning from a limited amount of training data contaminated with noise can be alleviated using two supervised machine learning paradigms: ensemble methods and Bayesian machine learning. Instead of considering a single hypothesis $\hat{f}$ for prediction, these learning paradigms combine the decisions of a collection of different hypotheses using averages. This combination process has several advantages over standard machine learning methods. In particular, it provides a mechanism to obtain more accurate and robust predictions.

1.2.1 Ensemble Methods

Ensemble methods generate a collection of complementary hypotheses whose predictions are compatible with the observed data. These hypotheses are induced by fitting different models to the training data or by fitting the same model under different training conditions. For instance, by using randomization techniques in the learning algorithm or by using different heuristics for the estimation of the model parameters. The prediction of the ensemble is then computed by combining the decisions of the different elements in the ensemble using averages or voting rules (Sharkey, 1996; Xu et al., 1992). Voting rules are simply averages in which the target variable y is discrete (Kittler, 1998).

Combining the decisions of a collection of complementary predictors has several advantages over relying on an individual hypothesis for prediction. First, the risk of choosing an incorrect rule for making predictions is reduced because the ensemble considers simultaneously alternative hypotheses (Dietterich, 2000a). Second, some ensemble methods have proved to be very robust when the training data are contaminated with noise (Breiman, 2001; Dietterich, 2000b; Martínez-Muñoz et al., 2009; Opitz and Maclin, 1999). Ensemble methods can also be useful to reduce over-fitting. In particular, the combination of several complex models that are prone to over-fitting is beneficial in numerous problems of practical interest (Husmeier and Althoefer, 1998; Sollich and Krogh, 1996). Finally, the prediction of the ensemble is often significantly better than the individual predictions of the different members of the ensemble (Breiman, 1996a, 1998, 2001; Bühlmann, 2003; Dietterich, 2000b; Martínez-Muñoz and Suárez, 2005; Opitz and Maclin, 1999; Rodríguez et al., 2006).

In spite of the advantages described, the practical implementation of ensemble methods can be difficult in practice. Specifically, the number of predictors that are required to guarantee the convergence of the prediction error of the ensemble can be very large (Banfield et al., 2007; Latinne et al., 2001; Margineantu and Dietterich, 1997). In consequence, ensembles are costly to generate. Furthermore, considerable memory resources are necessary to store all the predictors. A more serious drawback is that the time needed to compute the prediction of the ensemble increases linearly with the ensemble size. Therefore, it can be significantly larger than the prediction time of a single model. These aspects can be critical in online applications (Margineantu and Dietterich, 1997; Prodromidis and Stolfo, 2001).

Another problem of ensemble methods is that, in general, it is difficult to determine an appropriate size for the ensemble. In practice, the ensemble size is set to a large number for which the predictive error of the ensemble is assumed to have converged (Breiman, 2001; Freund and Schapire, 1996; Geurts et al., 2006; Martínez-Muñoz and Suárez, 2005). However, according to the findings of some recent investigations, the number of predictors required to guarantee convergence strongly depends on the particular problem under analysis: While some learning problems require just a few tens of predictors to be included in the ensemble, others require many more (Banfield et al., 2007; Latinne et al., 2001). Over-estimating the ensemble size can result in a waste of resources. By contrast, under-estimating this number can result in a loss of prediction accuracy.

The first part of this thesis analyzes different ensemble pruning methods. These techniques can be used to alleviate the large storage requirements and the large prediction times of ensembles. Additionally, the problem of determining how many individual predictors should be included in an ensemble is investigated. This question is of practical interest because it involves a trade-off between computational resources and prediction accuracy. The following paragraphs describe the contributions of this thesis to the field of ensemble methods:

- The problem of extracting a subensemble with good generalization properties from an original pool of regressors generated by bagging (Breiman, 1996a) is studied in detail. For this purpose, we attempt to identify the subensemble whose training error is as low as possible. This problem can be shown to be NP-hard. In consequence, exact solutions become infeasible for ensembles of realistic sizes. Two

approximate methods are described to find near-optimal subensembles in these ensembles: SDP-pruning and ordered aggregation. Both methods select subensembles that are smaller and have better generalization performance than the original ensemble in the problems investigated (Hernández-Lobato et al., 2006b, 2009b).

- We analyze a probabilistic framework that can be used to infer the class label predicted by a classification ensemble after computing the predictions of only a fraction of the total classifiers in the ensemble. This framework is the basis of a novel ensemble pruning method called instance-based (IB) pruning (Hernández-Lobato et al., 2009). IB-pruning significantly reduces the prediction time of classification ensembles with a negligible deterioration in the generalization performance of the ensemble.

- The previous probabilistic framework is used to determine the size of a classification ensemble so that, with a specified confidence level, this ensemble generates the same predictions as an ensemble of infinite size (Hernández-Lobato et al., 2009a). The classification accuracy of this optimal ensemble of finite size is only slightly inferior to the accuracy estimated for an ensemble of infinite size. The application of this method to different classification problems shows that the optimal ensemble size strongly depends on the particular problem under study: While some classification problems require a few tens of predictors in the ensemble, others require several thousands.

1.2.2 Bayesian Machine Learning

In Bayesian machine learning probabilities are used to express an initial degree of belief in the different candidate hypotheses (Bishop, 2006; MacKay, 2003). Assuming a particular form for the model that generates the target variables, Bayes' theorem can then be used to compute a final set of posterior probabilities for these hypotheses on the basis of the empirical evidence given by the training data. Prediction is then implemented by computing a weighted average of the predictions of the individual hypotheses. The weights in this average are the corresponding posterior probabilities.

Bayesian methods have several advantages over other machine learning techniques. First, the Bayesian framework is useful for discriminating among different candidate models. In particular, those models that are unnecessarily complex for the learning task are automatically penalized (Bishop, 2006; MacKay, 2003). Second, if we assume that the correct model has been selected, Bayesian methods account for the variability in the estimation of the model parameters. Instead of computing a single point estimate for these parameters, the Bayesian approach considers all possible parameter values and weighs each individual value by the corresponding posterior probability. Finally, expert knowledge about the particular problem domain can be readily incorporated in the Bayesian framework in a very intuitive manner. For this, the prior probabilities are chosen to reflect the particular characteristics of the learning problem (Bishop, 2006; MacKay, 2003). This prior knowledge can yield significant improvements in the performance when the number of training instances is small.

Despite these advantages, the practical implementation of Bayesian machine learning also poses challenging problems. Specifically, computing the posterior probabilities often requires solving integrals in high dimensions or evaluating summations that involve an exponential number of terms (Bishop, 2006; MacKay, 2003). In consequence, these probabilities have to be approximated in most practical applications. A standard method

for computing the approximation is Markov chain Monte Carlo (MCMC) (Neal, 1993). This method is based on the construction of a Markov chain whose stationary distribution coincides with the posterior probabilities of the model. The approximation of the posterior probabilities is then performed by sampling from the Markov chain designed. A disadvantage of MCMC techniques is their high computational cost (Bishop, 2006; MacKay, 2003). More recently, efficient deterministic methods have been proposed for approximate Bayesian inference (Jaakkola, 2001; Minka, 2001b). These methods approximate the posterior probabilities using simpler distributions for which computations are tractable. However, the applicability of these methods is limited, and sometimes it can be difficult to compute approximate probabilities for all the model parameters. Type-II maximum likelihood is an alternative technique that can be used to estimate those parameters whose posterior probabilities are difficult to approximate (Bishop, 2006). An inconvenient of this technique is that generally demands the running of the approximate inference algorithm repeatedly (Bishop, 2006). This significantly increases the total cost of training the Bayesian model.

The second part of this thesis proposes novel applications of the deterministic algorithm expectation propagation (EP) for approximate Bayesian inference (Minka, 2001b). EP provides a computationally efficient alternative to methods such as Markov chain Monte Carlo (MCMC) sampling or type-II maximum likelihood estimation. The following paragraphs describe the contributions of this thesis to the field of Bayesian machine learning:

- The Bayes machine is introduced as an extension of the Bayes point machine (Herbrich et al., 2001). In the Bayes machine the posterior distribution of a parameter that quantifies the level of noise in the class labels is inferred from the data (Hernández-Lobato and Hernández-Lobato, 2008). In general, the posterior distribution of this parameter is difficult to approximate. Existing Bayesian models use type-II maximum likelihood for its estimation. In non-Bayesian models, this parameter is typically estimated by cross-validation. Both type-II maximum likelihood and cross validation require re-training the model multiple times. By contrast, in the Bayes machine the posterior distribution of this parameter is efficiently approximated using EP. Experiments on benchmark classification problems show that the Bayes machine is competitive with support vector machines (Vapnik, 1995) and Gaussian process classifiers (Kim and Ghahramani, 2006).

  A disadvantage of the Bayes machine is that its computational cost is still very high; namely $\mathcal{O}(n^3)$ where n is the number of training instances. To reduce this cost, a sparse representation for the Bayes machine is proposed (Hernández-Lobato, 2008). In particular, the Bayes machine is trained using only a reduced set of active instances that are selected by a greedy algorithm similar to the one considered in the informative vector machine (Lawrence et al., 2003; Seeger, 2003). Nevertheless, additional refining iterations are introduced to correct some of the mistakes of the original approach. These extra iterations also improve the quality of the posterior approximation. The cost of training a sparse Bayes machine is $\mathcal{O}(nd^2)$, where $d \leq n$ is the number of active instances. Experimental evaluation of this sparse representation shows that it is competitive with support vector machines and non-sparse Bayes machines.

- A novel application of the EP algorithm to the Bayesian model for microarray data classification proposed by Lee et al. (2003) is described in this thesis (Hernández-Lobato et al., 2009). Bayesian inference in this model has been previously implemented by MCMC sampling methods. However, in this case EP is a much faster alternative to MCMC. The cost of EP is linear in both the number of data instances and in the number of attributes. Experiments on a wide range of microarray datasets show that EP is competitive in terms of prediction accuracy with MCMC and with other representative microarray classification techniques. Additional experiments confirm that the posterior approximation provided by EP is also useful for ranking relevant genes for subsequent analysis. Furthermore, this ranking is stable against small perturbations of the training set.

1.3 Publications

This section lists (in chronological order) the work published during the postgraduate period in which this thesis was written. The manuscripts are organized in different categories. The category *Related Work* includes manuscripts that are related to this thesis but that are not described in detail. The category *Submitted Work* refers to manuscripts that have been submitted for publication.

Ensemble Methods

- Hernández-Lobato, D., Martínez-Muñoz, G., and Suárez, A. (2009). Statistical instance-based pruning in ensembles of independent classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):364–369.

- Hernández-Lobato, D., Martínez-Muñoz, G., and Suárez, A. (2006). Pruning in ordered regression bagging ensembles. In *International Joint Conference on Neural Networks*, pages 1266–1273. IEEE.

Bayesian Techniques

- Hernández-Lobato, D. and Hernández-Lobato, J. M. (2008). Bayes machines for binary classification. *Pattern Recognition Letters*, 29(10):1466–1473.

- Hernández-Lobato, D. (2008). Sparse Bayes machines for binary classification. In Kurková, V., Neruda, R., and Koutník, J., editors, *Proceedings of the 18th International Conference on Artificial Neural Networks*, volume 5163 of *Lecture Notes in Computer Science*, pages 205–214. Springer.

Related Work

- Martínez-Muñoz, G., Hernández-Lobato, D., and Suárez, A.(2009). Statistical instance-based ensemble pruning for multi-class problems. In Alippi C., Polycarpou, M. M., Panayiotou C., and Ellinas, G. editors. *Proceedings of the 19th International Conference on Artificial Neural Networks*, volume 5768 of *Lecture Notes in Computer Science*, pages 90–99. Springer.

- Martínez-Muñoz, G., Hernández-Lobato, D., and Suárez, A. (2009). An analysis of ensemble pruning techniques based on ordered aggregation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31:245–259.

- Martínez-Muñoz, G., Sánchez-Martínez, A., Hernández-Lobato, D., and Suárez, A. (2008). Class-switching neural network ensembles. *Neurocomputing*, 71(13-15):2521–2528.

- Hernández-Lobato, D., Martínez-Muñoz, G., and Suárez, A. (2007). Out of bootstrap estimation of generalization error curves in bagging ensembles. In Yin, H., Tiño, P., Corchado, E., Byrne, W., and Yao, X., editors, *Proceedings of the 8th International Conference on Intelligent Data Engineering and Automated Learning*, volume 4881 of *Lecture Notes in Computer Science*, pages 47–56. Springer.

- Hernández-Lobato, J. M., Hernández-Lobato, D., and Suárez, A. (2007). GARCH processes with non-parametric innovations for market risk estimation. In Marques de Sá, J., Alexandre, L. A., Duch, W., and Mandic, D. P., editors, *Proceedings of the 17th International Conference on Artificial Neural Networks*, volume 4668 of *Lecture Notes in Computer Science*, pages 718–727. Springer.

- Martínez-Muñoz, G., Hernández-Lobato, D., and Suárez, A. (2007). Selection of decision stumps in bagging ensembles. In Marques de Sá, J., Alexandre, L. A., Duch, W., and Mandic, D. P., editors, *Proceedings of the 17th International Conference on Artificial Neural Networks*, volume 4668 of *Lecture Notes in Computer Science*, pages 319–328. Springer.

- Hernández-Lobato, D., Hernández-Lobato, J. M., Ruiz-Torrubiano, R., and Valle, Á. (2006). Pruning adaptive boosting ensembles by means of a genetic algorithm. In Corchado, E., Yin, H., Botti, V. J., and Fyfe, C., editors, *Proceedings of the 7th International Conference on Intelligent Data Engineering and Automated Learning*, volume 4224 of *Lecture Notes in Computer Science*, pages 322–329. Springer.

- Martínez-Muñoz G., Sánchez-Martínez A., Hernández-Lobato D., and Suárez A. (2006). Building ensembles of neural networks with class-switching. In Kollias, S. D., Stafylopatis,A., Duch, W., and Oja, E. editors, *Proceedings of the 16th International Conference on Artificial Neural Networks*, volume 4131 of *Lecture Notes in Computer Science*, pages 178–187. Springer.

Submitted Work

- Hernández-Lobato, D., Hernández-Lobato, J. M., and Suárez, A. Expectation propagation for microarray data classification.

- Hernández-Lobato, D., Martínez-Muñoz, G., and Suárez, A. How large should ensembles of classifiers be?

- Hernández-Lobato, D., Martínez-Muñoz, G., and Suárez, A. Pruning regression bagging ensembles.

- Hernández-Lobato, D., Martínez-Muñoz, G., and Suárez, A. Inference on the asymptotic prediction of classification ensembles.

1.4 Summary by Chapters

The organization of the remaining chapters of this thesis is as follows:

Chapter 2 gives an introduction to ensemble methods and describes why ensembles are useful in supervised machine learning tasks. Most ensemble methods work in a similar way. First, the individual ensemble members are generated and second, the predictions of these elements are combined. This chapter reviews different techniques that can be used to implement these two phases. Additionally, it describes in detail three representative ensemble methods that have been used in our investigations. These are bagging, *random forest* and boosting.

Chapter 3 addresses the problem of reducing the large storage requirements and prediction costs of ensemble methods. This chapter shows that the problem of finding the optimal subensemble from an original regression bagging ensemble is NP-hard. Ordered aggregation and SDP-pruning are then introduced as two approximate pruning methods that can be used to identify near optimal subensembles. Experimental results show that ordered aggregation and SDP-pruning not only reduce the size of the ensemble but also improve its generalization performance. These improvements are a consequence of a reduction in the bias (individual error) and the covariance of the predictors selected. Pruning classification ensembles is then analyzed from a different perspective. Specifically, instance-based (IB) pruning is described as an efficient procedure to speed-up classification. Additional experiments illustrate the performance of this pruning method in several classification problems.

Chapter 4 focuses on the problem of estimating the optimal size of parallel ensembles. The statistical framework employed in IB-pruning to describe the majority voting process is now used to estimate the size of a classification ensemble so that this ensemble generates the same predictions as an ensemble of infinite size with a specified confidence level. Several experiments with two representative ensemble methods (bagging and random forest) illustrate the application of this procedure for determining the ensemble size. These experiments confirm that the optimal ensemble size strongly depends on the classification task considered.

Chapter 5 provides a review of Bayesian machine learning. First, it discusses how Bayesian probabilities can be used to describe the uncertainty about the exact value of the model parameters. Then, Bayesian model selection is introduced as a principled approach to discriminate among different models using these probabilities. Bayesian probabilities are updated using Bayes' theorem in a procedure called Bayesian inference which is typically intractable. Thus, this chapter also describes different methods that can be used to perform approximate Bayesian inference. In particular, the expectation propagation (EP) algorithm is reviewed in detail. This algorithm will be extensively used in the remainder of this thesis.

Chapter 6 introduces the Bayes machine, a Bayesian model for binary classification. In the Bayes machine the EP algorithm is used to approximate the posterior distribution of a parameter that quantifies the level of noise in the labels of the training data. Experiments on a simple classification problem show that this model can learn the intrinsic noise in the class labels of the data. Additional experiments compare the performance of Bayes machines with the performance of support vector machines and Gaussian process classifiers. Finally, this chapter proposes a sparse representation for the Bayes machine that reduces the cost of training this model.

Chapter 7 shows how the EP algorithm can be used to perform approximate inference in a Bayesian model for microarray data classification. Experiments on real-world data compare the performance of this model trained with EP and with Markov chain Monte Carlo sampling, with the performance of other representative microarray classification methods. Additional experiments demonstrate that EP can be used to identify relevant genes for classification. Furthermore, the ranking of the different genes obtained by this algorithm is stable to small perturbations of the training set.

Chapter 8 summarizes the conclusions of this thesis and proposes some lines for future research.

Part I

Ensemble Methods

Chapter 2

Ensemble Methods in Machine Learning

This chapter presents an overview of methods based on inducing a collection of predictors from some labeled data and then combining their outputs to produce a consensus response. There are several reasons that explain why combining the decisions of different predictors instead of using a single one could be beneficial. In particular, the aggregation process can provide a mechanism to reduce the bias and/or the variance components of the generalization error with respect to the individual ensemble members. These methods can also reduce the risk of selecting suboptimal values for the parameters of the model used to describe the data. Furthermore, they can extend the representation capacity of a single predictor. In practice, the algorithms used to build ensembles follow similar strategies. In a first phase, the ensemble predictors are generated from the training data. Typically, this phase is implemented to encourage the generation of accurate predictors that are also diverse. In a second phase, the outputs of these predictors for a new instance are combined to compute the final ensemble response. In this chapter we review some algorithms that can be used to implement these phases. Finally, we also describe in detail some representative ensemble methods.

2.1 Introduction

The induction of predictors from a finite amount of data, which can furthermore be contaminated by noise, poses some difficult problems. In particular, there is some uncertainty about the form of the model that should be used to represent the observed data. Furthermore, it is also difficult to estimate accurate values for the model parameters from these data. Ensemble methods can be used to address these difficulties. This learning paradigm builds a collection of different predictors whose individual responses are then combined to label test instances (Dietterich, 1998, 2000a; Hansen and Salamon, 1990; Kuncheva, 2004; Opitz and Maclin, 1999; Sharkey, 1996; Valentini and Masulli, 2002). The different predictors can be obtained by fitting the same model to the observed data under different training conditions (homogeneous ensembles) (Breiman, 1996a, 2001; Bühlmann, 2003; Freund and Schapire, 1996) or by fitting different models to the same data (heterogeneous ensembles) (Bahler and Navarro, 2000; Caruana and Niculescu-Mizil, 2004; Tsoumakas et al., 2004). Combining the outputs of several predictors results in improved accuracy in many regression and classification problems. Specifically, the generalization performance of the ensemble is often much better than

a single individual ensemble member (Bauer and Kohavi, 1999; Breiman, 1996a; Dietterich, 2000b; Freund and Schapire, 1996; Friedman, 2001; Opitz and Maclin, 1999; Webb, 2000). These improvements in performance arise from the combination of accurate predictors whose errors are complementary (Dietterich, 2000a; Fumera and Roli, 2005; Hansen and Salamon, 1990; Krogh and Vedelsby, 1995; Sharkey and Sharkey, 1997; Ueda and Nakano, 1996). Combining accurate predictors that are similar does not lead to any substantial gain in the prediction accuracy (Tumer and Ghosh, 1996). On the other hand, combining predictors whose accuracy is worse than random guessing can lead to a significant deterioration of the performance of the ensemble (Hansen and Salamon, 1990). In consequence, ensemble methods attempt to generate predictors that are both accurate and that err in different data instances. This can be obtained by training each predictor of the ensemble using a perturbed version of the training set (Breiman, 1996a; Freund and Schapire, 1996; Ho, 1998; Martínez-Muñoz and Suárez, 2005) or by introducing some type of randomness in the training process of the ensemble members (Breiman, 2001; Geurts et al., 2006).

To reach a final ensemble decision, it is necessary to combine the individual predictions of the ensemble members. In practice, different algorithms can be employed for this purpose (Bahler and Navarro, 2000; Hashem, 1993; Kittler, 1998; Sharkey, 1996; Xu et al., 1992). Nevertheless, in many ensemble methods the combination scheme is very simple, e.g. (Breiman, 1996a, 2001; Geurts et al., 2006; Liu and Yao, 1999; Martínez-Muñoz and Suárez, 2005; Rodríguez et al., 2006). In regression problems, the average prediction of the ensemble elements is commonly used as the ensemble output. In classification problems, majority voting is used instead. In majority voting the final ensemble prediction is the class label that is predicted more frequently by the individual ensemble members. Majority voting and simple averaging are two combination methods that are very robust (Fumera and Roli, 2005; Kittler et al., 1998; Kuncheva et al., 2003; Lam and Suen, 1997; Tumer and Ghosh, 1996; Ueda and Nakano, 1996). Besides these two simple algorithms there are other methods that employ non-linear decision functions for output combination or place the ensemble elements in complex tree-like structures (Gama and Brazdil, 2000; Jordan and Jacobs, 1994; Wolpert, 1992). However, in general there is no strong evidence supporting that more complex combination schemes exhibit better performance.

The organization of this chapter is as follows: In Section 2.2 we motivate the use of ensemble methods to solve supervised machine learning problems. In Section 2.3 some representative methods for generating the individual predictors of the ensemble are reviewed. Section 2.4 discusses alternative procedures for combining the decisions of these elements. In Section 2.5 we describe in detail three commonly used ensemble learning algorithms. Finally, the conclusions of this chapter are summarized in Section 2.6.

2.2 Reasons for Using Ensembles

In machine learning applications a given learning system is often found to be superior to other models only for a specific learning task. It can be shown that no single inductive learning method can achieve a generalization performance better than some other method in all possible classification tasks (Schaffer, 1994; Wolpert, 1996). The apparent superiority of a learning algorithm is only due to the nature of the problems investigated and/or to the distribution of the observed data. Despite these considerations, ensemble methods have shown an excellent performance in numerous learning tasks of practical

interest (Hansen and Salamon, 1990; Ho et al., 1994; Perrone and Cooper, 1993; Xu et al., 1992). In particular, the errors of the individual predictors of the ensemble can be compensated by the decisions of the other ensemble members. This intuitive idea is illustrated in the case of classification problems by the simple example described by Dietterich (2000a). Consider a binary classification task. Assume that the classifiers in an ensemble of size M make independent errors with common probability $p < 1/2$ and that the outputs of these classifiers are combined by a simple majority voting. That is, the different classifiers assign a class label to the new instance and the class label that receives the largest number of votes is the final ensemble prediction. The overall error of a classification ensemble containing these classifiers is given by the probability that more than half of the classifiers simultaneously predict the erroneous class label

$$\mathrm{Err}_{\mathrm{ens}}(p, M) = \sum_{m=\lceil \frac{M}{2} \rceil}^{M} \binom{M}{m} p^m (1-p)^{M-m} = I_p \left(\lfloor \frac{M}{2} \rfloor + 1, M - \lfloor \frac{M}{2} \rfloor \right) , \qquad (2.1)$$

where $I_x(a, b)$ is the regularized incomplete beta function (Abramowitz and Stegun, 1964). Figure 2.1 displays the value of (2.1) as a function of p, for different values of the ensemble size M. The figure shows that $\mathrm{Err}_{\mathrm{ens}}(p, M) = p$ for $M = 1$, as expected. However, if $p < 1/2$ $\mathrm{Err}_{\mathrm{ens}}(p, M) \ll p$ for large M. Thus, the majority voting combination of the different classifiers has the effect of reducing the prediction error.

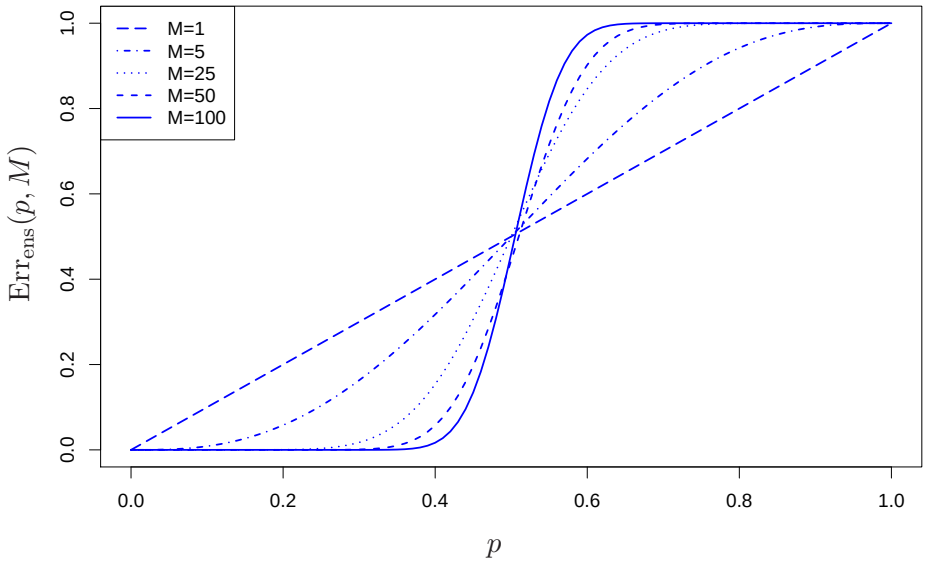

FIGURE 2.1: Prediction error of the ensemble as a function of M, the ensemble size, and p, the misclassification probability of an ensemble member.

The key assumptions in the previous analysis are that the errors of the classifiers are independent and that the individual error probabilities are p with $p < 1/2$. In the case that p is larger than $1/2$, a different behavior for the ensemble error is observed in Figure 2.1. In particular, the aggregation process by majority voting has the opposite effect. That is, it increases the classification error of a single element, $\mathrm{Err}_{\mathrm{ens}}(p, M) \gg p$ for large M. This simple example brings up an important issue in ensemble methods. The effectiveness of the ensemble relies on both the individual accuracy of the different ensemble members and the independence of their errors (ensemble diversity) (Dietterich,

2000a; Hansen and Salamon, 1990; Krogh and Vedelsby, 1995; Ueda and Nakano, 1996). Ideally, the ensemble predictors should perform better than random guessing and their errors should be uncorrelated. Predictors with these properties are complementary. That is, their individual errors are compensated in the ensemble aggregation process.

The previous example is illustrative of why building ensembles of predictors can be beneficial. However, the assumption of uncorrelated errors does not hold in practice. The classifiers usually err on the same instances (typically those located near the decision boundary). In spite of this limitation, there are several reasons that explain why aggregating many different predictors can be an effective way of improving the generalization performance, as described by Dietterich (2000a).

The first reason is statistical. In particular, almost all learning algorithms perform some search in the space $\mathcal{F}$ of possible hypotheses $\hat{f}$. The task of the learning algorithm is to find the hypothesis in that space that best fits the observed data. If the amount of training data is limited there may be different hypotheses in $\mathcal{F}$ with a similar performance on the training data. Aggregating these hypotheses in an ensemble reduces the risk of selecting the incorrect one. Figure 2.2 (top left) illustrates this situation. The outer curve in blue represents the space $\mathcal{F}$ of candidate hypotheses $\hat{f}$ where the learning algorithm performs its search. The inner curve in yellow depicts the set of all the hypotheses that perform well when evaluated on the training data. The point labeled f_0 is the target rule to approximate. The picture shows that by averaging all these hypotheses it is possible to obtain a better estimate of f_0.

The second reason is computational. As described earlier, the learning algorithm performs a search in the space $\mathcal{F}$ of candidate hypotheses $\hat{f}$ to find a good estimate of f_0. In this search, the algorithm can become trapped in a local minimum, which is a suboptimal solution (Hastie et al., 2001). Aggregating many hypotheses $\hat{f}$, obtained by repeating the search process from different starting points, may result in a better approximation to the target predictive rule f_0 than any of the single hypotheses $\hat{f}$ found. This situation is illustrated in Figure 2.2 (top right).

The third reason is related to the extended representation capacity of ensembles. In many machine learning applications the target rule f_0 can not be accurately approximated by any single hypotheses in $\mathcal{F}$. As an example consider a classification problem with a non-linear decision boundary. This learning task can not be solved by a single linear model. However, by combining the outputs of different linear models it is possible to obtain a non-linear decision boundary. Ensemble methods can hence be used to overcome the problem described. In particular, by aggregating the hypotheses $\hat{f}$ contained in $\mathcal{F}$ it is possible to expand the space of candidate hypotheses $\mathcal{F}$, obtaining a more expressive model. Figure 2.2 (bottom) displays this situation.

In addition to these reasons, extensive empirical evidence shows that combining the predictions of the ensemble members reduces the variance component of the generalization error (Breiman, 1996a, 1998; Geurts et al., 2006; Ueda and Nakano, 1996) and in some cases also the bias (Breiman, 1998, 2001; Schapire et al., 1998; Wolpert, 1992). Furthermore, some parallel classification ensembles such as bagging or *random forest* are very robust to the presence of data instances in the training set contaminated with different levels of noise (Breiman, 2001; Dietterich, 2000b; Martínez-Muñoz et al., 2009; Opitz and Maclin, 1999). Ensemble methods can also be useful to alleviate the problem of over-fitting. In particular, combining the outputs of very complex models that are prone to severe over-fitting seems to be beneficial in some ensemble learning algorithms (Husmeier and Althoefer, 1998; Martínez-Muñoz et al., 2009; Sollich and Krogh, 1996). Finally, it can also be desirable to combine different predictors in some situations. For

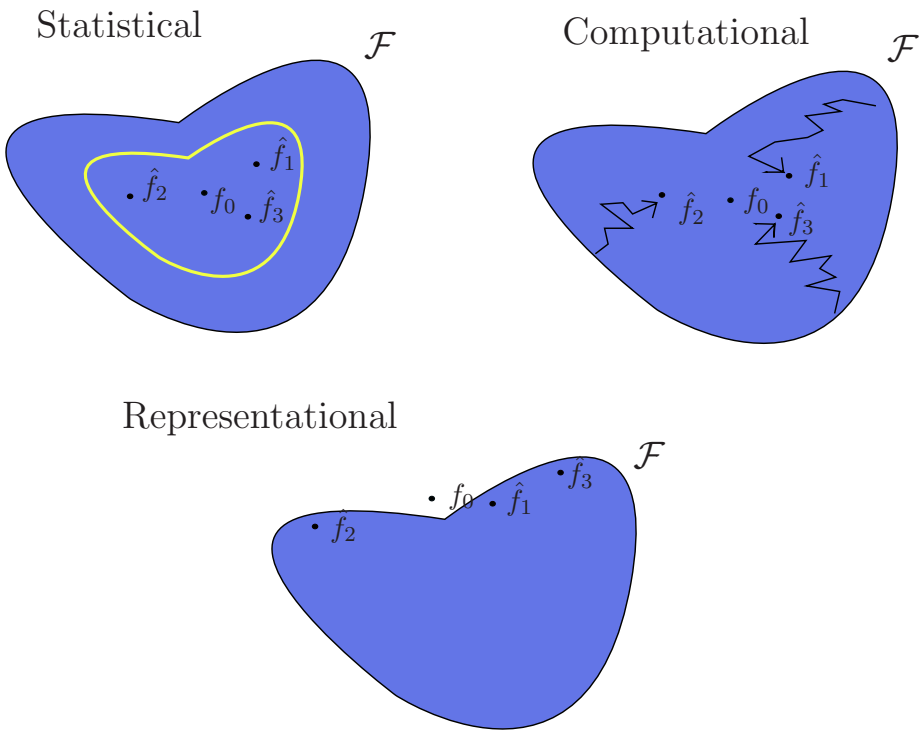

FIGURE 2.2: Graphical illustration of some fundamental reasons by which ensemble methods might perform better than a single predictor. This picture has been adapted from (Dietterich, 2000a).

instance, when there is access to different models, each one designed to focus on different parts of the learning task, or when there are several training sets, collected at different times or in different environments (Jain et al., 2000). A representative example of the beneficial effects of considering different models for prediction can be found in the Netflix prize challenge. The winning solution combines nearest neighbors, restricted Boltzmann machines, methods based on matrix factorization and other models (Koren, 2009; Piotte and Chabbert, 2009; Töscher et al., 2009).

2.3 Methods for Building the Individual Predictors

Different techniques have been proposed to build the individual predictors of the ensemble. These can be grouped in the following categories, as described in (Brown et al., 2005a; Dietterich, 2000a; Kotsiantis et al., 2006; Kuncheva, 2004; Sharkey, 1996; Valentini and Masulli, 2002):

- **Manipulating Training Examples:**

  This is the most frequently used strategy to build the ensemble members. It consists in perturbing the initial training data by removing or adding training instances or by modifying their relative weights in the training set. Diversity among the predictors in the ensemble is obtained by training each ensemble member on a different perturbed version of the initial training set. For this method to be effective, the individual predictors should exhibit some instabilities. That is, small changes in the training set should lead to changes in the predictions of the fitted

model. Neural networks and decision trees are learning models known to have this property (Breiman, 1998). Therefore, they are suited for building ensembles using this mechanism. By contrast, linear models, support vector machines (Vapnik, 1995) and K-nearest neighbors classifiers (Hastie et al., 2001) are fairly stable (Breiman, 1998). Using this strategy with stable models is not expected to be useful. Nevertheless, there are some studies showing that if the modifications of the training set are sufficiently large, these models also become unstable. In particular, ensembles of K-nearest neighbors (Hall and Samworth, 2005), support vector machines (Kim et al., 2003) and linear classifiers (Skurichina and Duin, 2002) with a good generalization capacity can also be obtained.

A method that can be used to obtain different modified versions of the training set is bootstrap sampling (Efron and Tibshirani, 1994). Bootstrap samples are obtained by drawing with replacement from the training set. A bootstrap sample of the same size as the training set contains on average only 63.2% different instances. The remaining data are repeated instances (Efron and Tibshirani, 1994). Thus, bootstrap samples are very different one from another because, on average, they share only 39.9% of their instances. Bagging is an example of ensemble learning algorithm that uses bootstrap samples to build the different predictors of the ensemble (Breiman, 1996a). Random forests is another example (Breiman, 2001).

Instead of removing data instances from the training set, other methods assign different weights to the different instances. By modifying these weights, the influence in the training set of the corresponding data instances can be controlled. This procedure is used in boosting ensembles to induce a collection of complementary predictors (Drucker, 1997; Freund and Schapire, 1997; Friedman, 2001; Meir and Rätsch, 2003). In boosting, the individual ensemble members are generated sequentially. In this sequence, the first predictor is obtained using equal training weights. Subsequent predictors are generated by modifying the weights of the training instances. Larger weights are assigned to instances that are incorrectly labeled by the predictor generated most recently in the sequence. By contrast, the weights of correctly labeled instances are lowered. This mechanism has shown to generate very diverse ensemble members (Dietterich, 2000b).

Other ensemble methods that generate diversity by manipulating the training instances are subagging and bragging (Bühlmann, 2003), wagging (Bauer and Kohavi, 1999), multiboosting (Webb, 2000) and bag-boosting (Dettling, 2004).

- **Manipulating Input Features:**

  This technique selectively removes some of the input features contained in the attribute vector $\mathbf{x}$ of the training instances. This process needs to be implemented carefully because removing some of the attributes can lead to a dramatic decrease in the accuracy of the resulting predictors (Tumer and Ghosh, 1996). The random subspace method is an example of ensemble learning algorithm that uses this technique to generate the individual predictors (Ho, 1998). In this method each predictor is generated using only a subset of randomly chosen features from the initial vector of attributes $\mathbf{x}$. The random subspace method generates predictors that can be as diverse as those produced in bagging or boosting ensembles (Ho, 1998). Another example of this technique is input decimation (Tumer and Oza, 2003; Turner and Oza, 1999). In input decimation features are selected according to their correlation with a fixed value of the target variable. Other ensemble

methods use genetic algorithms for feature selection (Bacauskiene et al., 2009; Kuncheva, 1993). The idea of using only a subset of the attributes for training has also been employed in a method called *attribute bagging* (Bryll et al., 2003).

Instead of selecting a subset of input features, new input features can be generated. For instance, new features can be obtained by projecting the original feature space into a transformed space using the principal component analysis (PCA). In particular, Skurichina and Duin (2005) find that ensembles build with this method perform better than ensembles generated with random feature selection. The PCA projection technique is also employed in the *rotation forest* ensemble learning algorithm (Rodríguez et al., 2006), a method that is competitive with both boosting and bagging. There are other projection operators besides PCA that can be used to generate new input features (Cherkauer, 1996; Duin and Tax, 2000; Fern and Brodley, 2003).

- **Manipulating Target Variables:**

  In this technique the target variable y associated to each attribute vector $\mathbf{x}$ in the training set is perturbed before training each individual classifier. Error-correcting output coding (ECOC) is an example of classification ensemble method that uses this technique to generate the ensemble predictors (Dietterich and Bakiri, 1995). In particular, before training the j-th ensemble member, this method randomly partitions the set of the different class labels considered in the classification task in two disjoint sets $\mathcal{A}_j$ and $\mathcal{B}_j$. The training instances are then relabeled according to their membership in one of these two sets. The j-th classifier is subsequently built on the relabeled data. Once the ensemble has been generated, the prediction of the class label of a test instances is computed using majority voting, where the j-th classifier from the ensemble votes for all the class labels included in either $\mathcal{A}_j$ or $\mathcal{B}_j$. Experimental evaluation of this method shows that it improves the performance of a single classification tree and a single neural network on several multi-class problems (Dietterich and Bakiri, 1995). Additionally, Schapire (1997) has shown that combining ECOC with boosting is also effective in multi-class problems. A limitation of ECOC is that the number of different class labels has to be large to obtain significant improvements. The idea of binarizing the classification task has also been employed in (Fürnkranz, 2002), where a different predictor is generated for each different pair of class labels.

  Instead of relabeling the training instances other techniques inject random noise in the class labels. These ideas are employed in (Breiman, 2000) to build ensembles based on *output randomization*. In regression problems, Gaussian noise can be injected in the target variable of each instance before training the predictors of the ensemble (*output smearing*). In classification problems, the class labels of the training data are modified at random (*output flipping*). The flipping of the class labels is made in such a way that the proportions of the classes in the training set are preserved. Output flipping has a parameter, the flip rate, that has to be tuned to its optimal value. Experimental results show that both output smearing and output flipping perform better than bagging (Breiman, 2000).

  Another ensemble learning algorithm that injects noise in the class labels of the observed data is *class-switching* (Martínez-Muñoz et al., 2008; Martínez-Muñoz and Suárez, 2005). Unlike output flipping, class-switching does not maintain the original class distribution in the perturbed training data. Thus, larger values for

the flipping rate parameter are possible in this method when the class labels of the classification problem are unbalanced. Class switching usually outperforms output flipping in these problems (Martínez-Muñoz and Suárez, 2005). In particular, in un-balanced classification problems larger values of the flip rate are often required to reach better prediction accuracy (Martínez-Muñoz and Suárez, 2005).

- **Randomizing the Learning Algorithm:**

  A last technique for building ensembles consists in introducing some randomization in the learning algorithm of the ensemble members. In consequence, different realizations of the learning algorithm will produce different classifiers, increasing the diversity of the ensemble. This technique has to be employed carefully since introducing randomness in the learning algorithm can sometimes lead to a decrease in the accuracy of the individual predictors obtained.

  Injecting randomness in the learning algorithm of the ensemble members can be implemented in different ways. For instance, different neural networks can be built using different random values for the initial weights of the synapses in the back-propagation algorithm. Different starting points in this algorithm can lead to different local minima, producing in consequence a diverse collection of neural networks (Kolen and Pollack, 1991). The C4.5 learning algorithm for building decision trees can also be randomized. Specifically, instead of choosing at each node the best possible split according to some criterion, Dietterich and Kong (1995) propose to choose randomly among the best s splits, with $s = 20$. Experimental results show that aggregating randomized decision trees provides equivalent or better results than bagging (Dietterich, 2000b). A similar randomizing procedure is implemented in the random forests (RF) ensemble learning algorithm (Breiman, 2001). In this case the best splitting attribute is selected at each node among a random subset of m input features, where m is typically set to $\lfloor \log_2(k)+1 \rfloor$ or $\lceil \sqrt{k} \rceil$, with k the total input features (Bernard et al., 2009; Breiman, 2001). RF is a very powerful ensemble algorithm that outperforms bagging and boosting in numerous learning tasks (Breiman, 2001). Further randomizations are implemented in the *extra-trees* ensemble method (Geurts et al., 2006). In particular, the splits in the internal nodes are made using random attributes and random cut-points.

  In these techniques the amount of randomization injected in the learning algorithm is specified by some parameter (e.g. m in the case of RF). Because the performance of the ensemble algorithm actually depends on this parameter (Bernard et al., 2009; Geurts et al., 2006), it should be tuned to get the best possible prediction accuracy. Nonetheless, simple rules for choosing this parameter often perform well in a wide variety of learning problems (Breiman, 2001; Geurts et al., 2006).

2.4 Methods for Combining the Ensemble Members

In this section we review some methods that have been proposed to combine the decisions of the individual ensemble members into a final ensemble prediction. Following (Jain et al., 2000) these methods can be grouped in the following categories:

- **Parallel Combination:**

  In these methods the predictors of the ensemble are queried independently and their responses are then combined. Examples of parallel combination methods include simple majority voting (also referred to as *plurality* voting when more than

two class labels are possible (Auda et al., 1995)). In simple majority voting, the different classifiers in the ensemble predict a class label. The final ensemble decision is the class label that receives the most votes. This method is employed in bagging, random forest, subagging or class-switching ensembles (Breiman, 1996a, 2001; Bühlmann, 2003; Martínez-Muñoz and Suárez, 2005). In regression problems the final prediction of the ensemble is often computed by averaging over the predictions of the different ensemble members. Majority voting and averaging are very robust combination methods (Fumera and Roli, 2005; Kittler et al., 1998; Kuncheva et al., 2003; Lam and Suen, 1997; Tumer and Ghosh, 1996; Ueda and Nakano, 1996). Instead of considering the same weight for each predictor, these two combination methods can also assign different weights to the different predictors in the ensemble. In particular, weighted majority voting is employed in boosting algorithms like Adaboost (Freund and Schapire, 1996). Weighted averaging is often used in regression ensembles as well (Hashem, 1993; Perrone and Cooper, 1993).

Another example of parallel combination method is found in the mixtures of experts learning paradigm (Jacobs et al., 1991). After querying the different predictors (experts) of the ensemble (mixture), this method employs a gating network to compute a linear combination of their outputs. Because the coefficients of the linear combination actually depend on the instance that is being processed, this method allows for the different elements in the ensemble to specialize in different regions of the input space.

Instead of using a linear combination, there are several methods that employ non-linear functions to generate the final ensemble prediction. In particular, the *stacking* combination method is based on fitting a non-linear classifier using as input data the outputs of the different ensemble members for the training set (Wolpert, 1992). This method can deduce the biases of the individual predictors of the ensemble with respect to the different aspects of the learning task. Stacking has been applied to regression tasks in (Breiman, 1996d) and has been further extended in other works (Ortega et al., 2001; Todorovski and Džeroski, 2003). A review of stacking is provided in (LeBlanc and Tibshirani, 1993).

Finally, there are other methods that can be employed to combine the outputs of the different ensemble members, namely naive Bayes combination, multinomial combination, Dempster-Shafer combination, the fuzzy integral, etc. See (Kuncheva, 2004) for a description of these methods and (Kuncheva et al., 2001) for an empirical comparison.

- **Cascade Generalization:**

  Cascading is a multistage method (Pudil et al., 1992) that can be used to combine the predictions of an ensemble of classifiers (Alpaydin and Kaynak, 1998; Gama and Brazdil, 2000). It is based on sequentially querying the different predictors of the ensemble. This querying process is implemented by enlarging the input space of the next predictor in the sequence with the concatenation of the outputs of the predictors previously queried and the attribute vector $\mathbf{x}$ (Gama and Brazdil, 2000). The predictors that are simpler are queried first in the sequence so that instances that are easier to classify can be quickly identified. The predictors that are more complex and costly to evaluate are queried later to recognize those instances that are more difficult to classify (Alpaydin and Kaynak, 1998).

Cascading is different from the combination methods described previously. In parallel techniques the ensemble members only employ the attribute vector $\mathbf{x}$ to output a prediction and hence, they can be queried independently. Cascading is a multistage method where the information computed by each predictor is used by the next predictors in the sequence to label the instance. The ensemble decision for a test instance is the output of the last predictor from the sequence. This predictor uses the information computed by the previous ensemble members and also the attribute vector $\mathbf{x}$. Cascading has been shown to outperform other combination methods like stacking and is competitive with boosting (Gama and Brazdil, 2000).

- **Dynamic Integration of Classifiers:**

  This combination method aims to solve the problem of estimating and then selecting the most appropriate ensemble member to classify a given test instance (Puuronen et al., 1999). This selection process is implemented dynamically for each different test instance to be classified. In consequence, the attributes of the instance are considered for the selection of the classifier. This method makes the assumption that each ensemble member has its own competence domain in the learning task. Thus, this domain has to be estimated first before it can be employed in the dynamic integration of the different ensemble members. Puuronen et al. (1999) propose to estimate the competence domain of the different ensemble members before the training phase using classification error estimates. In particular, a meta-level classifier is proposed to learn the individual errors of each ensemble member on each new test instance. This meta-level classifier can then be used to identify regions of the input space where the ensemble members correctly classify the data. When a new test instance is available for prediction, the ensemble member with the lowest estimated error on the instance is selected to make the final classification. When the method of dynamic integration of classifiers is applied to boosting and bagging ensembles it provides better prediction accuracy than the standard voting methods (Tsymbal and Puuronen, 2000). Dynamic integration has been improved in (Tsymbal et al., 2003).

- **Hierarchical Combination:**

  This combination method consists in placing the different experts (ensemble members) in the leaves of a hierarchical structure, similar to that of a decision tree (Jordan and Jacobs, 1994). Additionally, gating networks are placed at the different splitting nodes of the tree. These gating networks are employed to compute different input-dependent probabilities that are associated with the decisions that are made at each different splitting node. Given an unlabeled instance, the output of the hierarchical mixture is computed by starting at the root of the tree and randomly choosing a path down to a single expert whose output is then returned as the final output of the mixture. In this process, the probability of choosing a fixed path in the tree is given by the input-dependent probabilities of the different gating networks located in each split in the path. Such a partitioning allows each expert to specialize in different regions of the input space. This combination method takes advantage of the discriminative power of different features. Furthermore, the hierarchical architecture can be used to represent very complex data structures. Because both the experts and the tree structure are trained simultaneously, the resulting model is very complex and hence, it is prone to severe over-fitting. Over-fitting can be alleviated in this method by considering a Bayesian treatment

of the tree-structure (Bishop and Svensen, 2003). A limitation of hierarchical combination is that the architecture of the mixture needs to be specified beforehand.

2.5 Representative Ensemble Learning Algorithms

In this section we describe in more detail some representative ensemble learning algorithms that will be used in this thesis. These are bagging (Breiman, 1996a), random forests (Breiman, 2001) and boosting (Freund and Schapire, 1996).

2.5.1 Bagging

Bagging is a simple ensemble method in which the same type of individual predictors are built on different bootstrap samples from the training data (Breiman, 1996a). Thus, we begin by describing bootstrap sampling (Efron and Tibshirani, 1994). After this, we illustrate how the bagging ensemble algorithm can obtain improved predictions by averaging over the outputs of the ensemble members. The origin of these improvements in the prediction accuracy in bagging is a reduction in the variance term of the generalization error.

2.5.1.1 Bootstrap Sampling

Bootstrap sampling is a simple method that provides a direct computational way of assessing uncertainty in the estimation of the parameters of a model (Efron and Tibshirani, 1994; Hastie et al., 2001). This technique assumes that uncertainty arises from considering that the observed data $\mathcal{D}$ are a realization of a random variable. Assume that $\boldsymbol{\theta}$ are the parameters of a model and that the observed data $\mathcal{D}$ have been generated by this model for some value of $\boldsymbol{\theta}$, denoted $\boldsymbol{\theta}_0$, in whose estimation we are interested. As described in Chapter 1, an estimate of $\boldsymbol{\theta}_0$, denoted $\hat{\boldsymbol{\theta}}$, can be obtained by minimizing some loss function L that depends on $\mathcal{D}$ and $\boldsymbol{\theta}$

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \quad L(\boldsymbol{\theta}|\mathcal{D})\,. \tag{2.2}$$

Common choices for L are the sum of squared errors or the cross-entropy (Hastie et al., 2001). Because the observed data $\mathcal{D}$ used to perform the estimation are viewed as independent realizations of a random variable, $\hat{\boldsymbol{\theta}}_0$ is also a random variable that depends on $\mathcal{D}$. The bootstrap method provides a way to approximate the underlying distribution of the estimate $\hat{\boldsymbol{\theta}}$ without knowledge of the true distribution of $\mathcal{D}$. For this purpose, this technique builds a set of B datasets $\{\mathcal{D}_b, b = 1, \ldots, B\}$. Each dataset $\mathcal{D}_b$ is a *bootstrap sample* built by drawing with replacement from $\mathcal{D}$ as many instances as there are in $\mathcal{D}$. These bootstrap samples can be seen as modified versions of the original dataset $\mathcal{D}$. The probability distribution of $\hat{\boldsymbol{\theta}}_0$ can be approximated by computing a set of bootstrap replicates $\{\hat{\boldsymbol{\theta}}_0^{(b)}, i = 1, \ldots, B\}$, where each replicate $\hat{\boldsymbol{\theta}}_0^{(b)}$ is obtained as in (2.2) by using $\mathcal{D}_b$ instead of $\mathcal{D}$.

We now illustrate the bootstrap method in a simple problem that consists in inferring the mean μ of a Gaussian distribution with unit standard deviation. Assume that for this purpose, a set of data instances $\mathcal{D} = \{x_1, \ldots, x_n\}$ is available and that the true

mean, μ_0, is equal to 0. To estimate μ_0, we use the sample mean

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i \,. \tag{2.3}$$

A $(1-2\alpha)$-confidence interval for $\hat{\mu}_0$ derived from the statistical theory is

$$\left[-z_{1-\alpha}\sqrt{1/n}, z_{1-\alpha}\sqrt{1/n}\right] \,, \tag{2.4}$$

where $z_{1-\alpha}$ is the $1-\alpha$ percentile of a standard Gaussian distribution.

Similarly, the bootstrap approach can also be used to compute a rough confidence interval for $\hat{\mu}$. First, a set of B bootstrap samples $\mathcal{D}_b$ is built by drawing with replacement from $\mathcal{D}$. Next, we compute a set of B bootstrap replicates of $\hat{\mu}$, $\{\hat{\mu}^{(b)}, i = 1, \ldots, B\}$, using the maximum likelihood estimate (2.3) and the different bootstrap samples $\mathcal{D}_b$, i.e.

$$\hat{\mu}^{(b)} = \frac{1}{n}\sum_{i=1}^{n} x_i^{(b)} \,, \tag{2.5}$$

where $x_i^{(b)}$ are the different data instances contained in $\mathcal{D}_b$. The $(1-2\alpha)$ confidence interval for $\hat{\mu}$ can be computed using these estimates. Namely, we simply take those estimates placed in the positions $\lfloor \alpha B \rfloor$ and $\lceil B(1-\alpha) \rceil$ of the ordered sequence of the estimates.

Assume now that $n = 10$, $B = 1,000$, $\alpha = 2.5\%$, and that the dataset $\mathcal{D}$ is given by the following set of Gaussian observations

$$\{-0.636, -0.684, -0.877, 1.448, 0.424, 0.332, -1.19, 2.012, -0.242, -0.234\} \,. \tag{2.6}$$

The standard confidence interval derived from the statistical theory is

$$[-0.620, 0.620] \,. \tag{2.7}$$

The confidence interval derived from a computer simulation of the bootstrap method using $1,000$ replicates of $\hat{\mu}$ is

$$[-0.526, 0.655] \,. \tag{2.8}$$

These confidence intervals are very close to each other. Figure 2.3 displays a histogram for the $1,000$ bootstrap replicates of $\hat{\mu}$ computed using the bootstrap method and the probability distribution of $\hat{\mu}$, denoted $\mathcal{P}(\hat{\mu})$, derived from the statistical theory. The picture shows that the differences between both probability distributions for $\hat{\mu}$ are very small. These results indicate that the different bootstrap samples are a good approximation of independent realizations of the training set.

Under some conditions, the bootstrap method is asymptotically consistent (Efron and Tibshirani, 1994). In particular, as the size of the observed data grows, the empirical distribution of the observed data in $\mathcal{D}$ asymptotically approaches the true distribution of the data. In consequence, the bootstrap samples become more and more accurate. However, in general the method does not provide finite-sample guarantees. Furthermore, it can fail when there is a poor match between the true distribution of the observed data and the empirical distribution (Efron and Tibshirani, 1994). The main advantage of the bootstrap approach over analytical methods is that it is very simple to implement. It can be applied to very complex estimators for which it may be difficult to derive exact formulas for averages, standard errors or confidence intervals (Efron and Tibshirani,

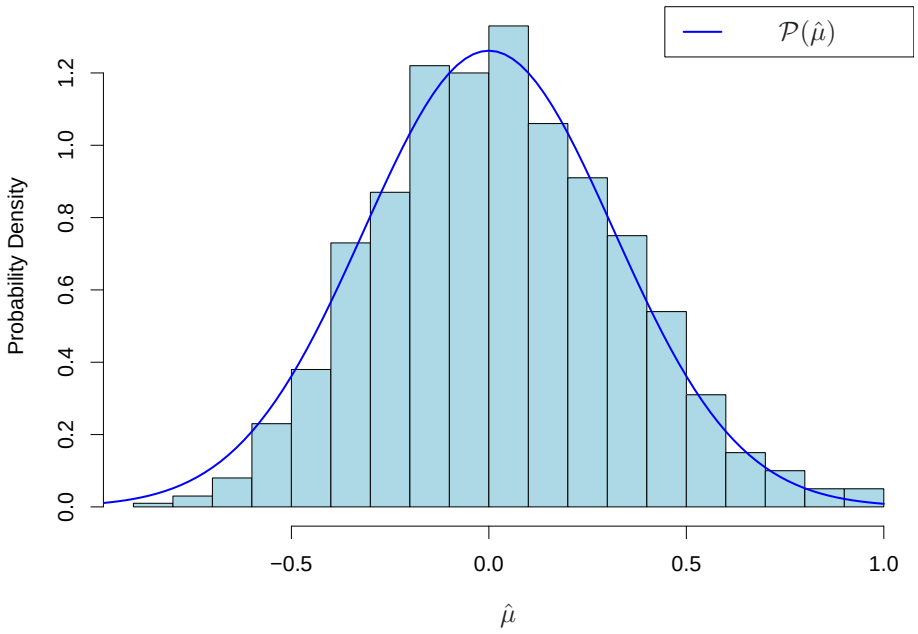

FIGURE 2.3: Histogram of 1,000 bootstrap replicates of $\hat{\mu}$ and probability density function of $\hat{\mu}$, $\mathcal{P}(\hat{\mu})$, derived from the statistical theory.

1994). The number B of bootstrap replicates to compute is often determined by the computational resources available. Typically, a few hundred replications are sufficient. However, if the underlying distribution of $\hat{\boldsymbol{\theta}}$ is heavy-tailed, a very large number of replicates might be required (Efron and Tibshirani, 1994). There is a procedure similar to the bootstrap called *subsampling*, that is based on sampling without replacement instead of with replacement (Politis et al., 1999).

2.5.1.2 Bootstrap Aggregation

Bagging is an acronym for *bootstrap aggregation* (Breiman, 1996a). As described in the previous section, the bootstrap method can be used to determine the uncertainty in the estimation of the model parameters from samples of finite size. In this section we show that the bootstrap can be also used to improve the prediction itself, leading to the bagging ensemble learning algorithm (Breiman, 1996a). In particular, the improvements obtained in bagging can be explained by a reduction of the variance component of the prediction error (Breiman, 1998).

To illustrate this point, consider a regression problem where the task of interest is to compute a predictor for the target value y given the observed attributes $\mathbf{x}$. Assume that M independent training sets $\mathcal{D}_i$, with $i = 1, \ldots, M$ are available for this purpose. For each one of these sets, we fit the same model M times, giving predictions $\{\hat{f}_i(\mathbf{x}), i = 1, \ldots, M\}$. To simplify the notation, the dependence of $\hat{f}_i(\mathbf{x})$ on $\mathcal{D}_i$ is assumed to be implicit. An improved estimator of the target value y can be obtained by averaging the outputs of these individual predictors

$$\hat{f}_{\text{avg}}(\mathbf{x}) = \frac{1}{M} \sum_{i=1}^{M} \hat{f}_i(\mathbf{x}) \,. \tag{2.9}$$

To show that this is in fact an improved estimator, consider the expected squared prediction error of a single model for the pair $(\mathbf{x}, y)$ given by

$$\begin{aligned} \mathrm{Err}_i &= \mathbb{E}_{\mathcal{D}_i}[(\hat{f}_i(\mathbf{x}) - y)^2] \\ &= \mathrm{Var}_i + \mathrm{Bias}_i^2 \,, \end{aligned} \tag{2.10}$$

where

$$\mathrm{Var}_i = \mathbb{E}_{\mathcal{D}_i}\left[(\hat{f}_i(\mathbf{x}) - \mathbb{E}_{\mathcal{D}_i}[\hat{f}_i(\mathbf{x})])^2\right] , \qquad \mathrm{Bias}_i^2 = \left(\mathbb{E}_{\mathcal{D}_i}[\hat{f}_i(\mathbf{x})] - y\right)^2 . \tag{2.11}$$

The first term, Var_i, is known as the variance term. It measures the magnitude of the fluctuations of $\hat{f}_i(\mathbf{x})$ around its expected value. The second term, bias_i^2, is known as the squared bias term, and it measures the difference between the expected value of $\hat{f}_i(\mathbf{x})$ and the target value y (Hastie et al., 2001).

Following the work of Ueda and Nakano (1996), we can similarly compute the expected squared prediction error of the averaged estimator defined in (2.9)

$$\begin{aligned} \mathrm{Err}_{\mathrm{avg}} &= \mathbb{E}_{\{\mathcal{D}_i\}_{i=1}^M}[(\hat{f}_{\mathrm{avg}}(\mathbf{x}) - y)^2] \\ &= \mathrm{Var}_{\mathrm{avg}} + \mathrm{Bias}_{\mathrm{avg}}^2 \,, \end{aligned} \tag{2.12}$$

where

$$\begin{aligned} \mathrm{Var}_{\mathrm{avg}} &= \mathbb{E}_{\{\mathcal{D}_i\}_{i=1}^M}\left[\left(\frac{1}{M}\sum_{i=1}^M \hat{f}_i(\mathbf{x}) - \mathbb{E}_{\{\mathcal{D}_i\}_{i=1}^M}\left[\frac{1}{M}\sum_{i=1}^M \hat{f}_i(\mathbf{x})\right]\right)^2\right] \\ &= \frac{1}{M^2}\sum_{i=1}^M \mathbb{E}_{\mathcal{D}_i}\left[\hat{f}_i(\mathbf{x}) - \mathbb{E}_{\mathcal{D}_i}[\hat{f}_i(\mathbf{x})]\right]^2 \\ &\quad + \frac{1}{M^2}\sum_{i\neq j} \mathbb{E}_{\mathcal{D}_i,\mathcal{D}_j}\left[\left(\hat{f}_i(\mathbf{x}) - \mathbb{E}_{\mathcal{D}_i}[\hat{f}_i(\mathbf{x})]\right)\left(\hat{f}_j(\mathbf{x}) - \mathbb{E}_{\mathcal{D}_j}[\hat{f}_j(\mathbf{x})]\right)\right] \\ &= \frac{1}{M^2}\sum_{i=1}^M \mathrm{Var}_i + \frac{1}{M^2}\sum_{i\neq j}\mathrm{Cov}_{ij} \\ &= \frac{1}{M^2}\sum_{i=1}^M \mathrm{Var}_i + \left(1 - \frac{1}{M}\right)\frac{1}{M(M-1)}\sum_{i\neq j}\mathrm{Cov}_{ij} \,, \end{aligned} \tag{2.13}$$

and

$$\mathrm{Bias}_{\mathrm{avg}}^2 = \left(\mathbb{E}_{\{\mathcal{D}_i\}_{i=1}^M}\left[\frac{1}{M}\sum_{i=1}^M \hat{f}_i(\mathbf{x})\right] - y\right)^2 = \left(\frac{1}{M}\sum_{i=1}^M \mathrm{Bias}_i\right)^2 . \tag{2.14}$$

In (2.13) Var_i is the variance of the estimate $\hat{f}_i(\mathbf{x})$ and Cov_{ij} the covariance between the estimates $\hat{f}_i(\mathbf{x})$ and $\hat{f}_j(\mathbf{x})$. In (2.14) Bias_i is the bias of the estimate $\hat{f}_i(\mathbf{x})$. Therefore,

$$\mathrm{Err}_{\mathrm{avg}} = \frac{1}{M}\overline{\mathrm{Var}} + \left(1 - \frac{1}{M}\right)\overline{\mathrm{Cov}} + \overline{\mathrm{Bias}}^2 \,, \tag{2.15}$$

where $\overline{\mathrm{Var}}$, $\overline{\mathrm{Cov}}$ and $\overline{\mathrm{Bias}}$ are respectively the average variance, covariance and bias of the individual predictors. Since the datasets D_i, with $i = 1, \ldots, M$, are assumed to be independent of each other, the covariances terms Cov_{ij} are zero. Additionally, if we

assume that all the models are of the same type, e.g. neural networks with the same architecture, all the variances and biases are also equal. In this case the decomposition is

$$\text{Err}_{\text{avg}} = \frac{1}{M}\text{Var}_i + \text{Bias}_i^2 \,. \tag{2.16}$$

Thus, the expected square error of the averaged estimator on the pair $(\mathbf{x}, y)$ becomes lower as M increases. The variance term is reduced proportionally to the number of predictors employed in the average. If the variance component of the squared error is large for a single predictor, this can be an important improvement. However, this improvement is obtained at the extra cost of combining several predictors.

In practice a single dataset $\mathcal{D}$ is available for training. Bagging overcomes this limitation by using datasets generated from different bootstrap samples. That is, the parameters of each predictor are estimated using different bootstrap samples of the observed data $\mathcal{D}$, as described in the previous section. A drawback of this procedure is that it introduces correlations among the predictors which means that the independence assumption is no longer valid. Furthermore, as consequence of using the bootstrap samples for training instead of $\mathcal{D}$, it is expected that the bias and the variance of the ensemble members increase slightly with respect to the bias and the variance of a predictor trained with all the available data. Nevertheless, the resulting aggregated estimator often has a better generalization performance than a single predictor (Bauer and Kohavi, 1999; Breiman, 1996a; Opitz and Maclin, 1999).

Bagging can also be used in classification problems. In this case, the ensemble members are classifiers trained using bootstrap samples and the output of each classifier in the ensemble is a vector whose components are all zero except one component which takes value one. This component is used to indicate the class label predicted by the ensemble member (Hastie et al., 2001). Therefore, the output of the bagging ensemble is a probability vector whose components indicate the fraction of classifiers from the ensemble that predict each different class label. The final ensemble prediction is the class label with the largest estimated probability. This procedure is equivalent to majority voting. Figure 2.4 displays the pseudo-code for bagging.

Input: Training set $\mathcal{D}$ and ensemble size M.

Output: Aggregated estimate $\hat{f}_{\text{bag}}(\mathbf{x})$.

1. For $i = 1, \ldots, M$
   (a) Draw $\mathcal{D}_i$ from $\mathcal{D}$ using bootstrap sampling.
   (b) Train a new model $\hat{f}_i(\mathbf{x})$ using $\mathcal{D}_i$.
2. Return an aggregated estimate

$$\hat{f}_{\text{bag}}(\mathbf{x}) = \frac{1}{M}\sum_i \hat{f}_i(\mathbf{x}) \,.$$

FIGURE 2.4: Algorithm that implements bagging.

We now illustrate the performance of bagging in a simple regression problem. Consider the following learning task: Assume that each input vector $\mathbf{x}_i$ is a 20-dimensional vector generated from a multivariate Gaussian distribution with mean vector $(r_i, r_i, \ldots, r_i)^T$ and unit standard deviation, where r_i is chosen independently for each different vector $\mathbf{x}_i$ with uniform probability from the interval $[0, 3]$. Given $\mathbf{x}_i$, the target variable y_i

associated to this input vector is

$$y_i = 25\frac{\sin(r_i)}{r_i} + \epsilon_i\,, \tag{2.17}$$

where ϵ_i is standard Gaussian noise. Using these rules we randomly generate 100 different training sets $\mathcal{D}_i$, with $i = 1, \ldots, 100$, each containing 25 instances. We also generate a test set, $\mathcal{D}_{\text{test}}$, of $1,000$ instances. For each different training set $\mathcal{D}_i$ we build a bagging ensemble of 100 un-pruned CART trees. A single tree is also built for comparison. Next, the mean squared error (MSE) of the ensemble and of the single tree is evaluated in the test set. The generalization error of these two different predictors is estimated by averaging the different test error estimates. Finally, we carry out a bias-variance decomposition of the generalization error, as described in (Bishop, 1996; Hastie et al., 2001). Table 2.1 summarizes the results of this experiment. This table shows the average MSE of each method and the average estimates of the variance and the squared bias. In this regression problem the generalization error of the bagging ensemble is smaller than the generalization error of the single tree. This improvement in the prediction accuracy arises due to a reduction in the variance component of the error. In particular, the variance of bagging is smaller than the variance of the single tree. By contrast, the biases of these two methods are very similar, although it is slightly higher the bias of the bagging ensemble. Because the ensemble members generated by bagging are not independent, the variance of the ensemble is not 100 times smaller than the variance of the single tree. As indicated by (2.13), this variance term includes the covariances among the different ensemble members that are small but different from zero in general.

TABLE 2.1: Experimental results comparing the performance of bagging and a single tree in the toy dataset.

Method	**MSE**	**Bias^2**	**Variance**
Bagging	15.75	11.16	4.60
Single Tree	36.80	9.49	27.30

Besides bagging there are other parallel ensemble methods that work in a similar way. These methods introduce some form of randomization in the process of building the ensemble members or instead of re-sampling, they inject noise in the target variables to generate modified versions of the original training set. Some examples are subagging (Bühlmann, 2003), random forests (Breiman, 2001), rotation forest (Rodríguez et al., 2006), output randomization (Breiman, 2000), class-switching (Martínez-Muñoz and Suárez, 2005) and extra-trees (Geurts et al., 2006).

2.5.2 Random Forests

Initially inspired by the work of Amit and Geman (1997), random forests (RF) were introduced by Breiman (2001) as an improvement over bagging when the members of the ensemble are decision trees. RF can be thought as a bagging algorithm where decision trees are replaced by trees built with a randomized version of the CART algorithm (Breiman et al., 1984). The splits of the internal nodes of these trees are made in terms of randomly selected attributes. The purpose of introducing randomness in the selection of the attributes for the splits is to increase the diversity among the trees generated in

the ensemble. As in bagging, the different trees are generated in RF using different bootstrap samples of the training data. Their outputs are then combined by simple majority voting, in the case of classification, or by averaging, in the case of regression.

Given a dataset $\mathcal{D}$, the standard tree building algorithm performs some greedy search in the space $\mathcal{F}$ of different partitions of the input space to find a partition $\hat{\Omega}$ that explains the observed data (Breiman et al., 1984). Besides the partition found, there can be several other partitions that can explain the observed data equally well. This is illustrated in Figure 2.5. In particular, Figure 2.5 (left) shows a sample training set $\mathcal{D}$ from a binary classification task containing 25 data instances. This figure displays with a solid black line the decision boundary obtained from inducing an un-pruned CART tree from these data. The optimal decision boundary of the classification problem (a diagonal line across the unit square) is also displayed as a reference. We note that the partition found correctly labels the observed data $\mathcal{D}$. However, there are many partitions that obtain zero training error. Figure 2.5 (right) shows alternative partitions that also predict the correct class label for the training instances. RF aims to take into account these alternative partitions by considering at each splitting node a subset of m randomly chosen features. Building decision trees using these randomly chosen features has the effect of enlarging the space of candidate trees that can be used to describe the observed data. In consequence, the diversity of the resulting classifiers with respect to the ones generated in bagging is increased (Rodríguez et al., 2006). This increment of the ensemble diversity is typically related to an increment in the variance of the different predictors contained in the ensemble (Banfield et al., 2005; Domingos, 2000). Because the differences among the ensemble members are greater, it is also expected that this randomization mechanism reduces the average covariance between pairs of the resulting trees. As stated in (Tumer and Ghosh, 1996; Ueda and Nakano, 1996), the combined performance of the ensemble strongly depends on the average correlation between pairs of predictors from the ensemble. The variance component of the generalization error can be reduced simply by increasing the ensemble size. Thus, the performance of RF is often better than the performance of bagging (Geurts et al., 2006; Rodríguez et al., 2006). Nevertheless, RF typically demands the aggregation of more trees in the ensemble than bagging (Banfield et al., 2007).

We now compare the performance of RF and bagging in the simple regression problem investigated in the previous section using the same experimental set-up. For each different training set $\mathcal{D}_i$ we build a bagging ensemble of un-pruned CART trees and a RF ensemble using randomly generated trees with the parameter m set to 1. Both ensembles are composed of 100 trees. Then, the mean squared error (MSE) of the ensembles is evaluated in the test set. An estimate of the generalization error is obtained by averaging the different test error estimates. We also carry out a bias-variance-covariance decomposition of the generalization error of each ensemble member, as described by Ueda and Nakano (1996). Note that this decomposition is given for the individual ensemble members, while in the previous section the decomposition was calculated for the bagging ensemble as a whole. Table 2.2 summarizes the results of this experiment. This table shows the average MSE of bagging and RF in the test dataset and the average estimates of the squared bias, variance and covariance of the different trees generated by each ensemble method. We note that the error of RF is lower than the error of bagging in this regression problem. In particular, the variance of the trees generated by RF is larger than the variance of the trees generated by bagging. The opposite effect is observed for the covariance. That is, the covariance between pairs of ensemble members

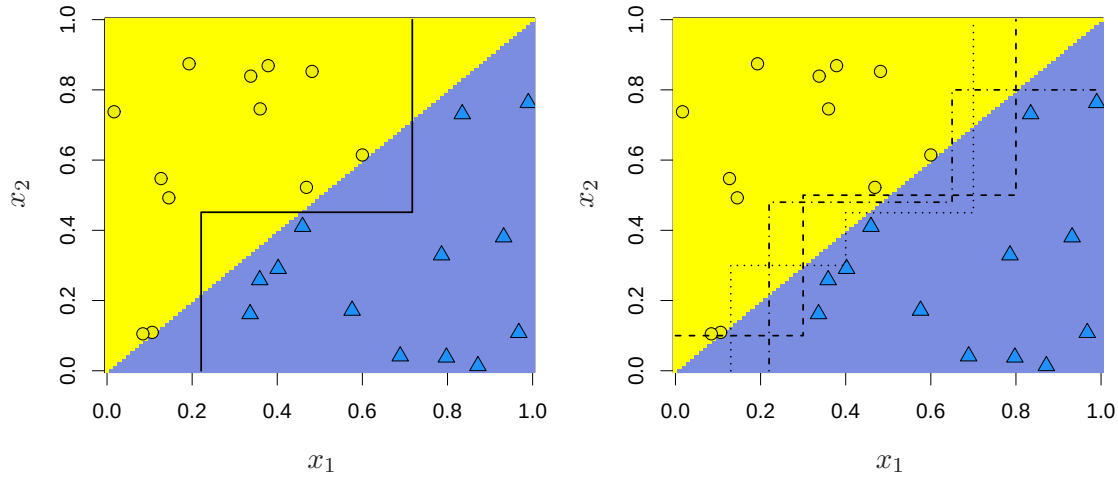

FIGURE 2.5: (left) Partitioning of the feature space provided by fitting a single un-pruned CART tree to the data displayed. The optimal decision boundary of the classification problem is displayed as a diagonal across the unit square. (right) Alternative partitions of the feature space that also explain the observed data.

in RF is smaller than the covariance among the ensemble members in bagging. Finally, the squared bias of these elements is similar for both ensemble methods.

TABLE 2.2: Experimental results comparing the performance of bagging and RF in the toy dataset.

Ensemble Method	**MSE**	$\overline{\textbf{Bias}}^2$	$\overline{\textbf{Var}}$	$\overline{\textbf{Cov}}$
Bagging	15.75	11.15	31.82	4.32
Random Forest	12.07	10.70	34.27	1.04

RF is faster than bagging because growing random trees takes less time than growing un-pruned CART trees (Breiman, 2001). In particular, the algorithm used to build the individual trees in RF has only to look for splits in m randomly chosen features. In both RF and bagging the out-of-bag (OOB) instances can be used to compute an estimate of the generalization error efficiently (Breiman, 2001). This estimate is very accurate if the ensemble size is sufficiently large. Otherwise, the estimate can have an upwards bias (Breiman, 2001; Bylander, 2002). Another advantage of RF is that it provides a mechanism to assess the importance of each predictive variable in the input vector $\mathbf{x}$ (Breiman, 2001). This can be useful for identifying variables that are relevant for the classification task. The mechanism consists in randomly permuting the values of the i-th input variable in the OOB samples used for computing the OOB error estimate. This process is repeated for each different input variable and the increment in the misclassification rate, as compared to the OOB error estimate with all the input variables intact, is recorded. The larger the increment in the misclassification rate, the higher the predictive importance of that input variable. This mechanism has been proved useful for carrying out feature selection in the medical domain (Díaz-Uriarte and Alvarez de Andrés, 2006). RF also allows to compute a proximity measure between pairs of data instances (Breiman, 2002). Given two training instances i and j, the proximity

measure between these two instances is computed as the fraction of random trees where the two instances lie in the same terminal node. This measure can be used for example to identify hidden structures in the data (Breiman, 2002). RF typically outperforms bagging in different learning domains and obtains error rates that are similar to the ones of Adaboost (Breiman, 2001). However, in noisy domains RF is more robust than Adaboost. In particular, if there is noise in the class labels of the training data the performance of Adaboost quickly deteriorates. By contrast, the performance of RF is fairly robust (Breiman, 2001). Typical choices for the value of m are $\lfloor \log_2(k) + 1 \rfloor$ or $\lceil \sqrt{k} \rceil$, where k is the total number input features (Bernard et al., 2009; Breiman, 2001).

2.5.3 Boosting

Boosting is a family of ensemble learning algorithms introduced to obtain more expressive learning models by combining simple predictors whose performance is required to be slightly better than random guessing (Avnimelech and Intrator, 1999; Drucker, 1997; Freund, 1995; Freund and Schapire, 1997; Friedman, 2001; Meir and Rätsch, 2003). Originally, boosting algorithms were designed to solve classification problems. However, they can also be extended to address regression tasks. In this section we focus on a representative boosting algorithm: Adaboost, introduced by Freund and Schapire (1996).

The working principle of Adaboost is to induce a sequence of predictors by applying the same learning algorithm to repeatedly modified versions of the training set. In this sequence, new predictors focus on those training instances that are difficult to classify by the previous predictors in the sequence. The decisions of all the predictors in the sequence are then combined using weighted majority voting. Consider a binary classification problem in which the target variable y takes values from the set $\mathcal{Y} = \{-1, 1\}$. Adaboost minimizes the average value of an exponential loss function (Friedman et al., 2000; Hastie et al., 2001). For a training instance $(\mathbf{x}_i, y_i)$ this loss function is defined as

$$\mathcal{L}(y_i, \hat{f}(\mathbf{x}_i)) = \exp\left(-y_i \hat{f}(\mathbf{x}_i\right) . \tag{2.18}$$

In this expression $\hat{f}$ is the Adaboost ensemble output function which is defined as weighted sum of the outputs of the individual ensemble members

$$\hat{f}(\mathbf{x}) = \sum_{m=1}^{M} \beta_m \hat{f}_m(\mathbf{x}) , \tag{2.19}$$

where the weights are restricted to be positive, i.e. $\beta_m > 0$, for $m = 1, \ldots, M$. Figure 2.6 displays the value of (2.18) as a function of the output function $\hat{f}$. The misclassification loss is also displayed for reference. The figure shows that the exponential loss function of Adaboost takes large values for data instances for which $y_i \hat{f}(\mathbf{x}_i)$ is negative. On the other hand, the values of (2.18) are small for positive values of $y_i \hat{f}(\mathbf{x}_i)$. In consequence, this loss function encourages large positive values for $y_i \hat{f}(\mathbf{x}_i)$ and hence, large values, either positive or negative, for $\hat{f}(\mathbf{x}_i)$, for correctly classified training instances. This suggests that given a test instance $\mathbf{x}$, the class label y that the Adaboost ensemble should output is

$$y = \text{sign}\left(\hat{f}(\mathbf{x})\right) = \text{sign}\left(\sum_{m=1}^{M} \beta_m \hat{f}_m(\mathbf{x})\right) , \tag{2.20}$$

where $\text{sign}(z)$ is a function returning -1 if $z \leq 0$ and 1 otherwise.

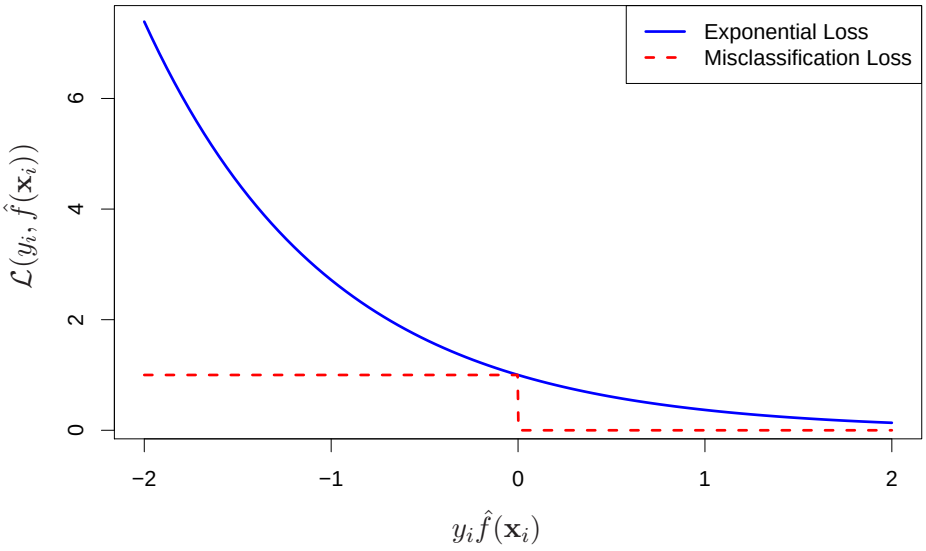

FIGURE 2.6: Exponential loss of the Adaboost algorithm for $y_i \hat{f}(\mathbf{x})$ alongside with the associated misclassification error.

We show now how the ensemble elements are computed sequentially in Adaboost. For this purpose, assume that we have built up to the $k-1$-th ensemble member and that the individual predictors $\hat{f}_m$, with $m = 1, \ldots, M$, output class labels from the set $\{-1, 1\}$. Using the exponential loss function defined before we have to solve the following optimization problem to find the next predictor and its associated weight β_k

$$\left(\hat{f}_k, \beta_k\right) = \arg\min_{(\tilde{f}, \tilde{\beta})} \sum_{i=1}^{n} \exp\left[-y_i \left(\sum_{m=1}^{k-1} \hat{f}_m(\mathbf{x}_i) + \tilde{\beta}\tilde{f}(\mathbf{x}_i)\right)\right] . \tag{2.21}$$

This problem can be rewritten as

$$\left(\hat{f}_k, \beta_k\right) = \arg\min_{(\tilde{f}, \tilde{\beta})} \sum_{i=1}^{n} w_i^k \exp\left[-y_i \tilde{\beta}\tilde{f}(\mathbf{x}_i)\right] , \tag{2.22}$$

where $w_i^k = \exp(-y_i \sum_{m=1}^{k-1} \hat{f}_k(\mathbf{x}_i))$ can be seen as the weight that is assigned to each different observation. We note that the values of these weights actually depend on the errors of the predictors available and that they change when a newly built predictor is incorporated into the ensemble.

As described by Hastie et al. (2001) the solution to (2.22) is obtained in two separated steps. The solution for $\hat{f}_k$ can be computed independently of the value of β_k by finding the predictor that minimizes a weighted training error rate

$$\hat{f}_k = \arg\min_{\tilde{f}} \sum_{i=1}^{n} w_i^k \mathbb{I}\left(y_i \neq \tilde{f}(\mathbf{x}_i)\right) , \tag{2.23}$$

where $\mathbb{I}(z)$ is an indicator function that takes value one when z is satisfied and zero otherwise. To see this, consider the following transformation of the function that is

being optimized in (2.22)

$$e^{-\tilde{\beta}} \sum_{y_i=\tilde{f}(\mathbf{x}_i)} w_i^k + e^{\tilde{\beta}} \sum_{y_i\neq\tilde{f}(\mathbf{x}_i)} w_i^k \,. \tag{2.24}$$

This is equivalent to

$$\left(e^{\tilde{\beta}} - e^{-\tilde{\beta}}\right) \sum_{i=1}^{n} w_i^k \mathbb{I}\left(y_i = \tilde{f}(\mathbf{x}_i)\right) + e^{-\tilde{\beta}} \sum_{i=1}^{n} w_i^k \,. \tag{2.25}$$

Because $e^{\tilde{\beta}} - e^{-\tilde{\beta}}$ is a strictly positive function of $\tilde{\beta}$ (recall that the weights are restricted to be positive), the optimal solution of (2.22) with respect to $\tilde{f}$ is given by (2.23). β_k can now be found in terms of $\hat{f}_k$

$$\beta_k = \frac{1}{2} \log \frac{1-\epsilon_k}{\epsilon_k} \,, \tag{2.26}$$

where ϵ_k is the weighted training error rate of the predictor $\hat{f}_k$. That is,

$$\epsilon_k = \frac{1}{\sum_{i=1}^{n} w_i^k} \sum_{i=1}^{n} w_i^k \mathbb{I}\left(\hat{f}_k(\mathbf{x}_i) = y_i\right) \,. \tag{2.27}$$

Finally, we note that when the predictor $\hat{f}_{k+1}$ has to be estimated at the next iteration of Adaboost, the new weights will be given by

$$\begin{aligned} w_i^{k+1} &= w_i^k \exp\left[-\beta_k y_i \hat{f}_k(\mathbf{x}_i)\right] \\ &= w_i^k \exp\left[\alpha_k \mathbb{I}\left(\hat{f}_k(\mathbf{x}_i \neq y_i\right)\right] e^{-\beta_k} \,, \end{aligned} \tag{2.28}$$

where $\alpha_k = 2\beta_k$ and we have used that $y_i \hat{f}_k(\mathbf{x}_i) = 2\mathbb{I}\left(\hat{f}_k(\mathbf{x}_i) \neq y_i\right) - 1$. Because the factor $e^{-\beta_k}$ in (2.28) affects all the weights independently, it can be safely ignored in the computations.

The first predictor in Adaboost is built by fitting a single model to the training data. Each new predictor that is incorporated into the ensemble reduces the exponential loss over the training data. In consequence, Adaboost will eventually reach zero training error. The pseudo-code of the Adaboost algorithm is displayed in Figure 2.7.

We now illustrate how Adaboost works in a simple classification problem. Consider a learning task in which each input vector $\mathbf{x}_i$ is uniformly extracted from the square $[-1, 1] \times [-1, 1]$. In terms of the attribute vector, the target variable is

$$y_i = \begin{cases} 1 & \text{if } x_1^2 + x_2^2 \geq \frac{2}{\pi} \,, \\ -1 & \text{otherwise} \,. \end{cases} \tag{2.29}$$

Thus, the optimal decision boundary of this classification problem is a circumference of $\sqrt{2/\pi}$ radius centered at the origin. To show that Adaboost can turn a simple classifier into an accurate one, we employ Adaboost with a linear model as the base learner to solve this problem. The model employed is logistic regression. This model assumes that

Input: training set $\mathcal{D}$ of n instances and ensemble size M.

Output: Aggregated estimate $\hat{f}(\mathbf{x})$.

1. Initialize the observation weights to be uniform $w_i = 1/n$, $\forall i$.
2. For $k = 1, \ldots, M$
   (a) Train a new model $\hat{f}_k(\mathbf{x})$ using $\mathcal{D}$ and the weights w_i.
   (b) Compute
   $$\epsilon_k = \frac{1}{\sum_{i=1}^{n} w_i} \sum_{i=1}^{n} w_i \mathbb{I}\left(\hat{f}_k(\mathbf{x}_i) = y_i\right) .$$
   (c) Compute $\beta_k = 1/2 \log\left((1-\epsilon_k)/\epsilon_k\right)$ and $\alpha_k = 2\beta_k$.
   (d) Set $w_i \leftarrow w_i \exp\left[\alpha_k \mathbb{I}\left(\hat{f}_k(\mathbf{x}_i) \neq y_i\right)\right]$, $\forall i$.
3. Return an aggregated estimate
   $$\hat{f}(\mathbf{x}) = \text{sign}\left(\sum_{k=1}^{M} \beta_k \hat{f}_k(\mathbf{x})\right) .$$

FIGURE 2.7: Algorithm that implements Adaboost. Extracted from (Hastie et al., 2001).

the probability of y_i given $\mathbf{x}_i$ is

$$\mathcal{P}(y_i|\mathbf{x}_i, \boldsymbol{\theta}) = \sigma\left(y_i\left(x_1\theta_1 + x_2\theta_2 + \theta_3\right)\right) , \tag{2.30}$$

where σ is the sigmoid function, defined as $\sigma(z) = 1/(1+\exp(-z))$, and $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)^T$ are the model parameters. Under these circumstances, the class label predicted by this model for a given input vector $\mathbf{x}$ is

$$\hat{f}_k(\mathbf{x}; \boldsymbol{\theta}) = \text{sign}\left(x_1\theta_1 + x_2\theta_2 + \theta_3\right) . \tag{2.31}$$

Note that this simple model can only correctly represent linear decision boundaries. However, Adaboost can learn the non-linear decision boundary of the previous classification task very accurately by combining several of these models. To show this, we randomly generate a sample training set $\mathcal{D}$ containing $n = 500$ instances of this classification task. Then, an Adaboost ensemble of $1,000$ elements is built using $\mathcal{D}$. The base classifiers are trained using maximum likelihood. To set weights to the different training instances, before training the k-th predictor we compute a modified version of the training set $\mathcal{D}_k$. This new training set is obtained by drawing with replacement a set of n instances from $\mathcal{D}$, where the probability of drawing each different instance is proportional to the weights w_i, with $i = 1, \ldots, n$, of Adaboost. Thus, given $\mathcal{D}_k$ we maximize

$$\mathcal{L}(\mathcal{D}_k; \boldsymbol{\theta}) = \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_k} \log\left(\mathcal{P}(y_i|\mathbf{x}_i, \boldsymbol{\theta})\right) , \tag{2.32}$$

with respect to the model parameters $\boldsymbol{\theta}$. The predictor resulting from this optimization process can be inaccurate. However, Adaboost only requires that this optimization process generates a model whose weighted error on the training data is better than random guessing.

Figure 2.8 (left) displays the training set $\mathcal{D}$ used in Adaboost and the decision boundary that results from building an ensemble of 1,000 linear models using this algorithm. Figure 2.8 (right) displays $\mathcal{D}$ and the decision function provided by fitting a single linear model to this dataset. This figure shows that Adaboost generates a non-linear boundary by combining linear models. Furthermore, the decision boundary provided matches the actual decision boundary quite closely. By contrast, the decision boundary computed by a single linear model is very inaccurate. This is the expected result because of the limited expressive capacity of the base learners.

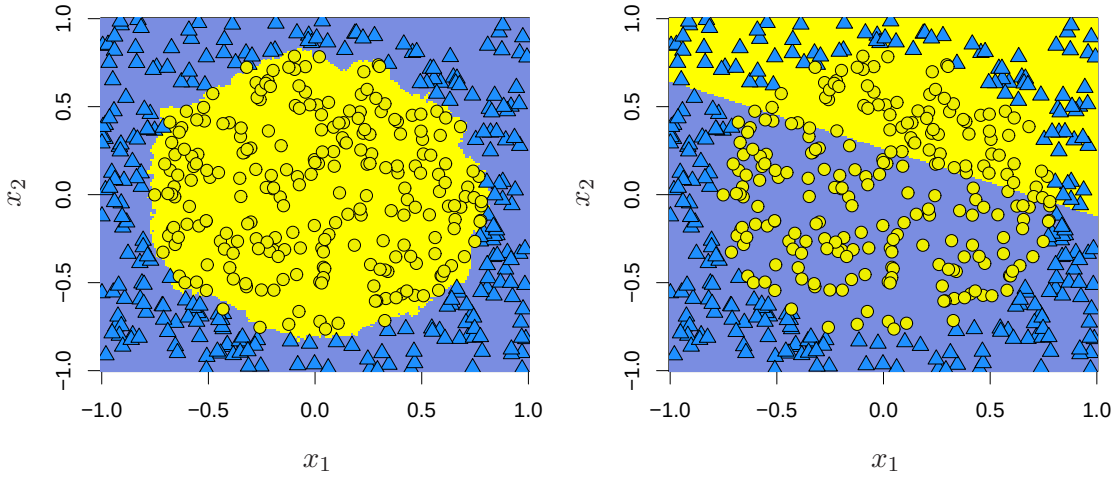

FIGURE 2.8: (left) Training set $\mathcal{D}$ and decision boundary provided by an Adaboost ensemble built using 1,000 linear models. (right) Training set $\mathcal{D}$ and decision boundary provided by a single linear model.

Adaboost has been shown to perform well in several benchmark classification problems (Bauer and Kohavi, 1999; Breiman, 1998; Quinlan, 1996). However, it is very sensitive to noise in the class labels. Its performance reduces in noisy domains (Breiman, 2001; Dietterich, 2000b; Martínez-Muñoz et al., 2009; Quinlan, 1996). This behavior arises because the weights w_i of incorrectly predicted training instances become larger in Adaboost. This encourages the next predictor in the sequence to correctly predict their associated class label. Typically, this is what you want Adaboost to do. However, if these instances are actually mislabeled data, the focus on them can lead to severe over-fitting. This problem can be alleviated in boosting algorithms by reducing the ensemble size M, by employing a shrinkage factor (Friedman, 2001) or by using alternative functions to update the training weights (Freund, 2009; Gómez-Verdejo et al., 2008, 2006).

Because each new predictor that is incorporated to the ensemble attempts to correct the erroneous predictions of the previous ensemble members, the performance improvements of Adaboost are often related to a decrease in the bias component of the generalization error (Bauer and Kohavi, 1999; Breiman, 1996b; Domingos, 2000; Schapire et al., 1998; Webb, 2000). However, most of these studies show that Adaboost is also effective reducing the variance. In fact, the first iterations of Adaboost primarily reduce the bias, while the later ones have been found to mainly reduce variance (Domingos, 2000). An alternative explanation of the good performance of Adaboost is the optimization of *classification margins*. The margin of an instance is defined as the difference between the weight assigned to the correct label and the maximal weight assigned to any single

incorrect label. In particular, Schapire et al. (1998) show that Adaboost is very aggressive maximizing the margins. These authors also provide non-asymptotic upper bounds on the generalization error of Adaboost in terms of the value of the margin. Nonetheless, direct optimization of the margin has not proved a successful strategy (Breiman, 1999a).

The success of Adaboost in many classification tasks has inspired a collection of alternative boosting algorithms. These algorithms differ in the method used to weigh the training instances or the different predictors contained in the ensemble. In particular, Breiman (1998) proposes a boosting algorithm similar to Adaboost that combines the different predictors using un-weighted voting. In this algorithm the re-weighting of the different training instances is done in terms of a polynomial function instead of an exponential function. In (Breiman, 1999a) the process of building a boosting ensemble is reformulated as a two-player game in which one player selects instances from the training set and the other a linear combination of predictors. Inspired by these results, a boosting algorithm that converges to the optimal game strategy is proposed. Some boosting algorithms to directly maximize the minimum classification margin of the training instances are described in (Rätsch et al., 2005). Rätsch et al. (2001) also introduce a regularization term in Adaboost that alleviates over-fitting in noisy problems. Friedman et al. (2000) show that Adaboost maximizes of a binomial log-likelihood function. A boosting algorithm is described to optimize this function directly. Finally, Schapire and Singer (1999) propose a version of Adaboost in which the individual predictors output real-valued probabilities instead of class labels. The function used by this algorithm to update the weights of the training instances can be decomposed in two factors. Gómez-Verdejo et al. (2006) balance between these two factors with one additional parameter to alleviate over-fitting. Additionally, Gómez-Verdejo et al. (2008) describe a procedure that can be used to adjust this parameter dynamically.

There have been several proposals to extend boosting to regression problems. The first of these techniques is described in (Freund and Schapire, 1997). This method assumes that the target variable y takes values from the unit interval $[0, 1]$. The regression problem is reduced to a binary classification task. Then, the standard boosting algorithm for classification is used. Each instance $(\mathbf{x}_i, y_i)$ in the training set $\mathcal{D}$ is mapped to an infinite set of binary questions, one for each $y \in [0, 1]$, of the form *is* $y_i \leq y$*?* In a similar manner, the output of each ensemble member $\hat{f}_k$ approximates the function $\mathbb{I}(y \geq y_i)$. Thus, $\hat{f}_k$ tends to give a natural answer to these binary questions. Although it is obviously infeasible to explicitly maintain an infinitely large training set, a method for the efficient implementation of this method is described in (Freund and Schapire, 1997). Even though a theoretical framework is also given, no experiments are carried out in (Freund and Schapire, 1997) to assess its performance. Finally, several other contributions have been inspired on the work described above, e.g. (Avnimelech and Intrator, 1999; Ridgeway et al., 1999).

Another boosting algorithm for regression is the one proposed by Friedman (2001). This author proposes a general gradient boosting paradigm developed for function approximation that takes into account different cost functions. In particular, specific algorithms are presented for squared and absolute errors. In (Friedman, 2001) function estimation is viewed from the perspective of numerical optimization in function space rather than in parameter space. Assuming that the goal is to minimize the sum of squared errors, a boosting algorithm is implemented by fitting each new predictor to the residual errors of the partially aggregated ensemble. The final prediction of the boosting ensemble is computed as a sum of the individual predictions of the different ensemble elements.

In (Drucker, 1997) an *ad hoc* regression modification of Adaboost is proposed. As in Adaboost, a weight vector $\mathbf{w}$ is introduced to assign different weights to the observations in the training set. At each iteration of the algorithm, a model $\hat{f}_k$ is fitted using a perturbed training set $\mathcal{D}_k$. This training set is obtained by drawing n samples from the original training set according to the distribution given by $\mathbf{w}$. Then, $\mathbf{w}$ is updated by considering the weighted loss $\mathcal{L}$ of the predictor $\hat{f}_k$ on the training set. For this purpose, three candidate loss functions are considered, all of them lying within the interval $[0, 1]$. These are *linear*, *square law* and *exponential*. Using $\mathcal{L}$, a confidence measure β_k in the model $\hat{f}_k$ is calculated by setting $\beta_k = \mathcal{L}/(1 - \mathcal{L})$. Finally, the ensemble output for an unlabeled instance is computed as the weighted median of the outputs of the individual ensemble predictors, where each predictor is weighted by the associated confidence measure β_k. Even though the authors do not present a theoretical justification for the modification, experiments on several regression problems illustrate the good performance of this method.

2.6 Conclusions

In this chapter we have given an overview of ensemble methods. These methods induce a collection of predictors from some data to then combine their outputs to produce a final response. Ensembles have several advantages over a single predictor. In particular, combining the decisions of a complementary set of predictors has been shown to be an effective procedure to alleviate the problems of learning from labeled data in the presence of a limited amount of data an noisy instances. The risk of choosing an incorrect rule for prediction is reduced in ensemble methods because they consider multiple rules at the same time. Furthermore, some ensemble methods have proved to be very robust when the training data are contaminated with noise. Ensemble methods can also be useful to reduce over-fitting. The combination of several complex models that are prone to over-fitting seems to be beneficial instead of detrimental in some ensemble algorithms. Finally, the prediction of the ensemble is often significantly better than the individual predictions of the individual ensemble members.

In spite of the advantages described, the practical implementation of ensemble methods presents some difficulties. Specifically, the number of predictors that are required to guarantee the convergence of the predictive error of the ensemble can be very large. Thus, ensembles can demand considerable memory resources to store all the different predictors. Additionally, the time needed to compute the prediction of the ensemble increases linearly with the ensemble size. Thus, it can be significantly longer than the prediction time of a single model. Another problem is that, in general, it is difficult to determine an appropriate size for the ensemble. The ensemble size is set in practice to a large number for which the predictive error of the ensemble is assumed to have converged. Over-estimating the ensemble size can result in a waste of resources. By contrast, under-estimating this number can result in a loss of prediction accuracy.

The following chapter of this thesis discusses different procedures that can be used to alleviate these problems.

Chapter 3

Ensemble Pruning Methods

The prediction time and storage requirements of ensemble methods can be improved by selecting a subset of complementary predictors whose combined generalization performance is similar or better than the original ensemble. Three ensemble pruning methods are investigated in this chapter. The first two are designed to extract a near-optimal subensemble from a regression ensemble generated by bagging. Identifying the optimal subset of predictors in this type of ensembles is a difficult task that has exponential cost in the size of the ensemble. Thus, the first strategy constructs a relaxed version of the problem that can be solved using semidefinite programming. The second one is based on modifying the order of aggregation of the predictors in the ensemble. Both methods identify subensembles that are close to the optimal ones. Experiments in several regression problems show that in the problems investigated pruned ensembles with only 20% of the initial predictors are more accurate than the original bagging ensembles. A bias-variance-covariance analysis relates these improvements to a reduction in the bias and the covariance of the selected ensemble elements. The last ensemble pruning method can be applied to parallel classification ensembles. Instead of finding a near-optimal subensemble, this pruning method determines for each instance to be labeled the number of classifiers that need to be queried to estimate the prediction of the complete ensemble with a specified confidence level. Thus, for a particular test instance, the voting process of the ensemble is halted when the probability that the predicted class will not change when taking into account the remaining votes is above the specified confidence level. Experiments on several classification problems using bagging and *random forests* confirm the validity of this dynamical instance-based ensemble pruning method.

3.1 Introduction

An important shortcoming of ensemble methods is that they often require a large number of predictors to guarantee the convergence of the generalization error to its asymptotic limit (Banfield et al., 2007; Latinne et al., 2001; Margineantu and Dietterich, 1997). Therefore, these methods demand large memory resources to store the ensemble members and take a long time to compute the prediction for an unlabeled instance. In consequence, using ensembles in online applications or in large-scale datasets remains a challenging task. A possible solution to these problems consists in replacing the original

ensemble by a representative subset of predictors from the initial ensemble. This approach is known as *ensemble pruning* and has been shown to be very effective. Besides needing less storage space and predicting faster, pruned subensembles can outperform the original classification ensembles from which they are extracted (Banfield et al., 2005; Caruana and Niculescu-Mizil, 2004; Margineantu and Dietterich, 1997; Martínez-Muñoz et al., 2007; Martínez-Muñoz et al., 2009; Martínez-Muñoz and Suárez, 2004; Martínez-Muñoz and Suárez, 2006, 2007; Ruta and Gabrys, 2005; Zhang et al., 2006). In practice, the task of selecting a representative subset from a pool of initial predictors (those induced in the original ensemble) using only information from the training set is a difficult problem. On the one hand, it is computationally expensive. The search is conducted in the space of $2^M - 1$ non-empty subensembles that can be extracted from an initial ensemble of size M. On the other hand, even if the search is feasible, the selection needs to be made in terms of an objective function estimated on the training data. Since we are interested in generalization performance, finding the optimum in the training set does not guarantee that the selected subensembles generalize well.

In this chapter we analyze three different ensemble pruning methods. The first two are devised to prune regression bagging ensembles. From these, the first method considered is SDP-pruning. SDP-pruning solves a relaxed version of the ensemble pruning problem using semidefinite programming (SDP). This method is an extension to regression ensembles of the method introduced by Zhang et al. (2006) to prune classification ensembles. The second method is ordered aggregation. Ordered aggregation is based on modifying the order in which the predictors are incorporated into the ensemble (Hernández-Lobato et al., 2006b). Starting with an empty subensemble, ordered aggregation incorporates the predictor that reduces the training error of the enlarged subensemble the most, until a stopping criterion is met. Subensembles obtained by these two strategies need not be optimal. However, comparisons with the exact solutions obtained by exhaustive search in ensembles of intermediate size show that they actually share a large fraction of the predictors with the optimal subensembles. Experiments in several real-world regression problems show that both strategies are effective in selecting subensembles that have a better generalization performance than complete ensembles built either with bagging or with the Adaboost.R2 algorithm (Drucker, 1997; Sharkey, 1999). A detailed analysis of the different components of the generalization error of the subensembles shows that the improvements in prediction accuracy arise because both SDP-pruning and ordered aggregation select from the original ensembles subsets of regressors that simultaneously have a low bias (i.e. their individual errors tend to be low) and small correlations (i.e. their predictions are complementary).

The last ensemble pruning method investigated in this chapter can be applied to parallel classification ensembles whose elements are generated independently when conditioned to the training data. Examples of these types of ensemble learning algorithms include bagging (Breiman, 1996a), but also other parallel ensemble methods like *random forests* (RF) (Breiman, 2001) or *class-switching* (Breiman, 2000; Martínez-Muñoz et al., 2008; Martínez-Muñoz and Suárez, 2005). Instead of identifying a near-optimal subensemble from the original classification ensemble, this pruning method follows a different approach. Specifically, the prediction of a classification ensemble whose elements are generated independently when conditioned to the initial training data is analyzed within a Bayesian framework. Assuming that majority voting is used for combining the decisions of the individual classifiers, it is possible to estimate with a specified confidence level the prediction of the complete ensemble by querying only a subset of the total classifiers of the ensemble. In consequence, for a particular instance that needs to

be classified, the voting process of the classifiers of the ensemble can be halted when the probability that the predicted class will not change when taking into account the remaining votes is above a specified confidence level. Experiments on a collection of benchmark classification problems using representative parallel ensembles, such as bagging and random forests, confirm the validity of the analysis and demonstrate the effectiveness of this instance-based (IB) pruning method. IB-pruning does not reduce the storage requirements, or improves the generalization performance of the original ensemble. The objective is to speed-up the prediction process for an unlabeled instance by determining the minimum number of classifiers that need to be queried to estimate the prediction of the complete ensemble with a high confidence level.

The chapter is organized as follows: Section 3.2 introduces the problem of selecting an optimal subensemble from a given bagging regression ensemble. This problem is shown to be NP-hard. Then, we review some ensemble pruning methods that are representative of the research in this area. Approximate solutions to the subensemble selection problem are then found by SDP-pruning or by ordered aggregation. The effectiveness of ordered aggregation and SDP-pruning is established by comparing the composition of the near-optimal subensembles identified by these methods and the composition of the optimal subensembles, obtained by exhaustive search. Then, we present the results of a series of experiments carried out to investigate the performance of these two ensemble pruning methods on a set of benchmark regression problems. This section includes a bias-variance-covariance analysis of the ensemble error that elucidates the origins of the improvements obtained. Section 3.3 analyzes the pruning of classification ensembles from a different perspective and introduces IB-pruning. The results of extensive experiments illustrate the performance of this ensemble pruning method. Finally, Section 3.4 summarizes the results and conclusions of this chapter.

3.2 Pruning Regression Ensembles

Consider a regression problem. The goal is to learn a function that predicts the dependent variable $y \in \mathbb{R}$ in terms of the attributes $\mathbf{x} \in \mathcal{X}$ using a set of training data $\mathcal{D} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$, generated from a probability distribution $\mathcal{P}(\mathcal{D})$. Bagging works by combining the outputs of a collection of diverse predictors. Each of these predictors is obtained by applying the same learning algorithm on a different bootstrap sample extracted from the original training data $\mathcal{D}$. Assume that $\hat{f}_i(\mathbf{x})$ is the prediction given by the i-th ensemble member, built using $\mathcal{D}_i$, the i-th bootstrap sample from the training data. The prediction of the ensemble is the average of the individual predictions of the M elements in the ensemble

$$M^{-1} \sum_{i=1}^{M} \hat{f}_i(\mathbf{x}), \;\; i = 1, 2, \ldots, M. \tag{3.1}$$

The generalization error of the ensemble is

$$\mathcal{L} = \int \left(M^{-1} \sum_{i=1}^{M} \hat{f}_i(\mathbf{x}) - y \right)^2 \mathcal{P}(\mathbf{x}, y) d\mathbf{x} dy, \tag{3.2}$$

where $y = f(\mathbf{x})$, f is the target function to approximate, and $\mathcal{P}(\mathbf{x}, y)$ is the joint probability distribution of the space of attributes and target variables. After some algebra

(3.2) can be expressed as

$$\mathcal{L} = M^{-2} \sum_{i=1}^{M} \sum_{j=1}^{M} \mathcal{C}_{ij}\,, \tag{3.3}$$

where

$$\mathcal{C}_{ij} = \int \left(\hat{f}_i(\mathbf{x}) - y\right) \left(\hat{f}_j(\mathbf{x}) - y\right) \mathcal{P}(\mathbf{x}, y) d\mathbf{x} dy. \tag{3.4}$$

The value $\mathcal{C}_{ii}$ is the average squared error of the i-th ensemble member. The off-diagonal terms $\{\mathcal{C}_{ij},\ i \neq j\}$ correspond to correlation-like values between the predictions of the i-th and j-th ensemble members (Zhou et al., 2002).

Assume that a bagging ensemble composed of M different predictors has been constructed. Our goal is to select the subensemble composed of u predictors $\{s_1, s_2, \ldots, s_u\}$ that minimizes the error

$$\mathcal{L}^{(u)} = u^{-2} \sum_{i=1}^{u} \sum_{j=1}^{u} \mathcal{C}_{s_i s_j}. \tag{3.5}$$

Because the true error is not available in the learning problem, we implement the selection of the optimal subensemble on the basis of the training error. The expression for the training error is identical to (3.5), except that the average over $\mathcal{P}(\mathbf{x}, y)$ in the calculation of $\mathcal{C}_{ij}$ is replaced by an average over the training data

$$\mathcal{C}_{ij}^{(tr)} = \frac{1}{N} \sum_{n=1}^{N} \left(\hat{f}_i(\mathbf{x}_n) - y\right) \left(\hat{f}_j(\mathbf{x}_n) - y\right). \tag{3.6}$$

Thus, the information needed for the optimization problem is contained in the matrix $\boldsymbol{\mathcal{C}}^{(tr)}$, which is estimated on the training set. This estimate is expected to be close to the true $\boldsymbol{\mathcal{C}}$ matrix, which is calculated as an average over the actual distribution of the data, so that minimizing the training error leads to the minimization of the generalization error. This is not necessarily the case in actual regression problems: minimizing the training error sometimes leads to over-fitting, and consequently, to the selection of subensembles whose generalization performance is suboptimal. The experiments carried out in this investigation show that, typically, the subensembles that minimize the error on the training data tend to be smaller than the subensembles that are optimal when the error is estimated on a test set that is independent of the training set.

Even if it were possible to accurately predict the generalization error from the training data only, finding the optimal subensemble is a computationally expensive problem that involves comparing the performance of all the possible $2^M - 1$ non-empty subensembles that can be extracted from the original ensemble. In fact, the problem of selecting the optimal subensemble from a given regression ensemble can be shown to be NP-hard. The proof proceeds by finding a problem that is known to be NP-complete and then showing that every instance of such problem can be reduced in polynomial time to the optimal subensemble selection problem (Cormen et al., 1990; Garey and Johnson, 1990). For this purpose, consider the *Subset Sum* problem (Cormen et al., 1990). This problem consists in extracting from a given set of integers $\mathcal{S} = \{n_1, \ldots, n_M : n_i \in \mathbb{Z}\}$ a subset of elements whose sum is equal to zero. Assume that there is an algorithm $\mathcal{A}$ that can find in polynomial time the subensemble of an ensemble of regressors whose combined prediction has the lowest mean squared error (MSE) on the dataset $\mathcal{D}$. If this were the case, the *Subset Sum* problem would also be solvable in polynomial time. To show how this would be accomplished, consider an ensemble of M regressors. Assume that the i-th

regressor in the ensemble outputs the integer value $n_i \in \mathcal{S}$ independently of the input. Define a regression problem that consists in predicting the value 0 independently of the input. The goal is then to select a non-empty subensemble $\{s_1, s_2, \ldots, s_u\}$, $1 \leq u \leq M$ whose combined prediction ($u^{-1}\sum_{i=1}^{u} n_{s_i}$, independent of the input) is as close to zero as possible. This can be achieved by minimizing the squared prediction error

$$MSE \equiv \left(u^{-1} \sum_{i=1}^{u} n_{s_i} - 0 \right)^2 = \left(u^{-1} \sum_{i=1}^{u} n_{s_i} \right)^2 . \tag{3.7}$$

Since the MSE is a non-negative value, if a subensemble whose mean squared error is zero exists, then algorithm $\mathcal{A}$ should find it in polynomial time. Finding a subensemble whose MSE is zero is equivalent to finding a subset $\{n_{s_1}, n_{s_2}, \ldots, n_{s_u}\} \subset \mathcal{S}$ whose elements sum to zero

$$MSE \equiv \left(u^{-1} \sum_{i=1}^{u} n_{s_i} \right)^2 = 0 \Longleftrightarrow \sum_{i=1}^{u} n_{s_i} = 0 . \tag{3.8}$$

Therefore, $\{n_{s_1}, n_{s_2}, \ldots, n_{s_u}\}$ would be a solution to the *Subset Sum* problem. Conversely, if no subset of $\mathcal{S}$ exists whose elements add up to zero, algorithm $\mathcal{A}$ would return a subset whose mean squared error is greater than zero. Since the *Subset Sum* is NP-complete, unless NP and P coincide, no algorithm with the properties of $\mathcal{A}$ exists. Hence, the problem of finding the optimal subensemble of a regression bagging ensemble is at least as hard as any NP-complete problem; i.e. it is NP-hard.

The fact that the optimal subensemble selection problem is NP-hard implies that finding the exact solution is infeasible for large ensembles. In this investigation we propose two methods that can be used to identify near-optimal subensembles at a reduced computational cost. The first method solves a relaxed version of the problem using semidefinite programming (SDP). The second one is based on modifying the order of aggregation of the predictors in the ensemble, which in standard bagging is random. Before describing these methods we review some of the available ensemble pruning techniques.

3.2.1 Related Work

The key to the improvements in the performance of the ensemble is the complementarity of the predictions given by the individual ensemble members (Brown et al., 2005b; Dietterich, 2000b; Krogh and Vedelsby, 1995; Tumer and Ghosh, 1996; Ueda and Nakano, 1996). Measures of accuracy or measures of diversity cannot be used in isolation to improve the performance of the ensemble. In consequence, most of the ensemble pruning methods attempt to extract a small subset of complementary predictors from the original ensemble. In this section we describe some of these methods. Since the pruning of regression ensembles is very similar to the pruning of classification ensembles, we also review in this section pruning methods for classification ensembles.

Margineantu and Dietterich (1997) investigate whether all the classifiers generated in a boosting ensemble are essential to obtain a good generalization performance. Using pruning heuristics based on measures of diversity and joint classification accuracy they show that, in the classification tasks investigated, the number of classifiers of a boosting ensemble can be substantially reduced (up to 60-80% in some classification problems) without a significant deterioration of the generalization accuracy of the ensemble. Similar pruning strategies are investigated in (Banfield et al., 2005; Tamon and Xiang, 2000).

In (Banfield et al., 2005), the initial ensemble is *thinned* by a sequential backward selection process that attempts to maximize the accuracy of the ensemble by eliminating classifiers whose contribution to the generalization performance (estimated in terms of measures of accuracy and/or diversity on the training set) is either detrimental or small.

Genetic algorithms (GAs) have also been proposed for the selection of a near-optimal subensemble from a complete bagging ensemble (Zhou and Tang, 2003; Zhou et al., 2002). In (Zhou et al., 2002) the output of the ensemble is a weighted average of the outputs of each ensemble member. The optimal set of weights of the ensemble members is found by minimizing a function that estimates the generalization error of the ensemble. The minimization problem is solved by a standard GA with a floating-point encoding scheme for the real-valued weights. Once the evolutive process has finished, the neural networks whose optimized weights are below a specified threshold are removed from the ensemble. The final ensemble output is the average over the predictions of the networks retained in the ensemble. The experiments carried out in (Zhou et al., 2002) establish GAs as a viable strategy to prune bagging ensembles. This approach has been applied to boosting in (Hernández-Lobato et al., 2006a).

Zhang et al. (2006) design an ensemble pruning method based on Semidefinite programming (SDP). In classification tasks the subensemble selection problem is formulated in terms of an $M \times M$ matrix $\mathbf{G}$, whose diagonal terms G_{ii} measure the individual errors of the ensemble members and whose off-diagonal terms G_{ij}, $i \neq j$ measure the number of common errors between classifiers i and j. The goal is then to find the sub-matrix of $\mathbf{G}$, of dimensions $u \times u$, corresponding to a subensemble of size u that minimizes the sum of the elements in the sub-matrix of $\mathbf{G}$. This is a standard 0-1 optimization problem that is also NP-hard. The problem can be reformulated so that it has the same optimal solution as an instance of the max-cut problem of size u (MC-u). This problem consists in partitioning the vertices of an edge-weighted graph into two sets, one of which has size u, so that the total weight of edges crossing the partition is maximized. The MC-u problem is known to have a good approximate algorithm based on SDP (Goemans and Williamson, 1995; Han et al., 2002). Therefore, a good approximate solution to the subensemble selection problem of size u can be found by solving the corresponding instance of the MC-u problem.

Tsoumakas et al. (2005, 2004) propose a pruning method called *selective fusion* that combines the outputs of a subset of classifiers selected from an heterogeneous ensemble using weighted voting. The selection of the optimal subensemble is approached as a multiple comparisons problem, which is solved by applying statistical tests to detect significant differences in cross-validation estimates of the prediction errors. In a separate work, these authors propose to use reinforcement learning to identify optimal subensembles (Partalas et al., 2006). More recently, Meynet and Thiran (2007) have proposed an information theoretic measure of the ensemble performance that can be used for the selection of a subset from an initial pool of classifiers.

Modifying the order in which the predictors are aggregated into the ensemble is introduced as a possible improvement in ensemble learning by Perrone and Cooper (1993). However, in that work predictors are ordered according to individual properties of the ensemble members (e.g. the mean squared error of each predictor). These heuristics do not take into account the complementarity of the predictions of the ensemble members. The joint performance of the predictors is only considered to decide whether a candidate neural network does not lead to a significant reduction of the error and should therefore be excluded from the ordered sequence. The ordered aggregation investigated in the next section represents an improvement over (Perrone and Cooper, 1993) because it

takes advantage of the complementarity of the predictions of the ensemble members by incorporating at each step the predictor that reduces the error of the current subensemble the most (Hernández-Lobato et al., 2006b). In (Partalas et al., 2008) this algorithm has been compared to other pruning heuristics in the domain of water quality prediction using ensembles of heterogeneous models.

A family of ensemble pruning methods based on modifying the order in which classifiers are aggregated in bagging ensembles is presented and evaluated in (Martínez-Muñoz et al., 2007; Martínez-Muñoz et al., 2009). In particular, the classifiers that are expected to perform better when combined are aggregated first. From the subensemble $\mathcal{S}_{u-1}$ of size $u-1$, the subensemble of size $\mathcal{S}_u$ is constructed by incorporating a single classifier from the remaining pool of classifiers. This classifier is selected by maximizing a metric that is expected to be correlated with the generalization performance of the ensemble. The metrics considered include prediction accuracy, ensemble diversity, complementarity among classifiers, margin distance, etc. The aggregation process is stopped when a fixed number of classifiers (typically 20% of the initial number of predictors) has been included in the ordered sequence. The results of extensive experiments show that these pruning methods can identify near-optimal subensembles and hence, they actually improve the generalization performance of standard bagging classification ensembles. Another investigation that evaluates different search algorithms and subensemble selection criteria for classification ensembles is (Ruta and Gabrys, 2005).

An alternative to selecting a subset of classifiers consists in replacing the ensemble by a single surrogate classifier that reproduces the decisions of the original ensemble. In particular, the technique of *Combined Multiple Models* proposed by Domingos (1997) builds a single classifier using a training set that, besides the original training examples, includes synthetic examples labeled by the ensemble. A related technique is suggested by Prodromidis and Stolfo (2001) to eliminate some of the classifiers from the ensemble. In this method a new classification task is defined. It consists in predicting the class label assigned by the ensemble to each one of the instances in the training set, using the outputs of the individual classifiers as predictor variables. A CART tree is fully grown using this auxiliary data and then pruned as described in (Breiman et al., 1984). Finally, those classifiers whose outputs are not used in the decisions of the CART tree are eliminated from the ensemble.

Instead of selecting a subset of complementary predictors from the ensemble, in negative correlation learning complementarities among the different ensemble members are encouraged by simultaneously training a collection of neural networks. These networks are trained using a cost function that includes a correlation penalty term in addition to the prediction error (Liu and Yao, 1999). The correlation penalty term encourages the specialization of the individual networks and cooperation among them. However, the strength of the penalty needs to be carefully tuned for each problem. If it is too small, the cooperation among the ensemble members will not be sufficient to produce significant improvements in performance. If the penalty is too large, the learning process is ineffective, i.e. the cost function is dominated by the penalty term and becomes insensitive to prediction errors. A further difficulty of this method is that the correlation penalty introduces a coupling among the parameters of the different ensemble members. This coupling increases the dimensionality of the parameter space in which the search is conducted and makes the learning process more difficult. In practice, only ensembles of small sizes can be built by means of this technique.

In (Demir and Alpaydin, 2005), the cost in time of classifying new instances is considered in the selection process. Ensembles are pruned by maximizing a utility function

that explicitly takes into account both accuracy and speed of classification. A different method for ensemble pruning is to replace the ensemble by a set of classifiers using clustering. The objective is to group classifiers by similarity and to retain one representative classifier per cluster (Bakker and Heskes, 2003; Giacinto and Roli, 2001a; Giacinto et al., 2000; Lazarevic and Obradovic, 2001). In (Chen et al., 2006), classification and regression ensembles that employ a weighted average for output combination are pruned using a probabilistic framework. Specifically, the weights of the combination are optimized using a regularization term that encourages sparsity. This term guarantees that some weights are driven to zero during the optimization process. The predictors corresponding to these weights can then be removed from the ensemble. Finally, Chan et al. (2007) propose to remove from the ensemble those classifiers that have less impact on the final ensemble output.

Several of the studies presented in this section focus on rather small ensembles. A notable exception is the work of Caruana and Niculescu-Mizil (2004), in which an extensive library of approximately $2,000$ models is compiled by inducing classifiers of different types (support vector machines, artificial neural nets, nearest-neighbors, decision trees, bagged decision trees, boosted decision trees, and boosted stumps), trained with different parameters for the learning algorithm. A subset of models is selected from this library according to different performance metrics, e.g. accuracy, cross entropy, mean precision, ROC area, etc.

3.2.2 Approximate Pruning Methods for Regression Ensembles

In this section we describe two approximate methods that can be used to prune regression bagging ensembles: SDP-pruning and ordered aggregation. Additionally, we compare the subensembles selected by these two methods with the optimal subensembles found by exhaustive search in ensembles of intermediate size.

3.2.2.1 Ensemble Pruning via Semidefinite Programming

The method based on semidefinite programming (SDP), which was proposed by Zhang et al. (2006) to prune classification ensembles, can be extended to prune regression ensembles. For this, we observe that the generalization error of the initial bagging ensemble can be expressed in terms of the matrix $\boldsymbol{\mathcal{C}}$ in (3.3). Thus, the subensemble selection problem consists in finding a subensemble of size u for which the sum of the elements in the corresponding sub-matrix of $\boldsymbol{\mathcal{C}}$ is as low as possible. An approximate solution to this problem can be obtained using semidefinite programming. The resulting algorithm is very similar to the one described in (Zhang et al., 2006) to prune classification ensembles. The difference lies in the matrix that is used to guide the optimization process. In (Zhang et al., 2006) it is the matrix $\mathbf{G}$, whose diagonal elements measure individual classification errors and whose off-diagonal elements measure common errors between the different classifiers in the ensemble. In the regression case this matrix is replaced by the matrix $\boldsymbol{\mathcal{C}}$, whose diagonal elements measure individual squared errors and whose off-diagonal elements correspond to correlation-like values.

The details of the approximate algorithm based on SDP are described in (Zhang et al., 2006). In this section we reproduce this algorithm for illustrative purposes. The algorithm is as follows: First, the subensemble selection problem is formulated as a 0-1

optimization problem

$$\begin{array}{ll} \min\limits_{\mathbf{u}} & \mathbf{u}^T \boldsymbol{\mathcal{C}}^{(tr)} \mathbf{u} \\ \text{s.t.} & \sum_i u_i = u, \\ & u_i \in \{0,1\}, \end{array} \tag{3.9}$$

where $\boldsymbol{\mathcal{C}}^{(tr)}$ is the estimate of $\boldsymbol{\mathcal{C}}$ in the training set and the superscript T means transpose. The binary variable u_i takes the value 1 if the i-th predictor is selected and 0 if it is not selected. Without the cardinality constraint, $\sum_i u_i = u$, there is a trivial solution to the problem in which none of the predictors is selected. Consider the transformation $u_i = (v_i + 1)/2$ with $v_i \in \{-1, 1\}$. With this change of variables, the objective function is proportional to $(\mathbf{v} + \mathbf{e})^T \boldsymbol{\mathcal{C}}^{(tr)} (\mathbf{v} + \mathbf{e})$, where $\mathbf{e}$ is a vector composed of ones. The constraint $\sum_i u_i = u$ becomes $(\mathbf{v} + \mathbf{e})^T \mathbf{I} (\mathbf{v} + \mathbf{e}) = 4u$, where $\mathbf{I}$ is the identity matrix. Defining the matrices

$$\begin{aligned} \mathbf{H} &= \begin{pmatrix} \mathbf{e}^T \boldsymbol{\mathcal{C}}^{(tr)} \mathbf{e} & \mathbf{e}^T \boldsymbol{\mathcal{C}}^{(tr)} \\ \boldsymbol{\mathcal{C}}^{(tr)} \mathbf{e} & \boldsymbol{\mathcal{C}}^{(tr)} \end{pmatrix}, \\ \mathbf{D} &= \begin{pmatrix} M & \mathbf{e}^T \\ \mathbf{e} & \mathbf{I} \end{pmatrix}, \end{aligned} \tag{3.10}$$

and extending the vector $\mathbf{v}$ with one additional component v_0 that takes value one, (3.9) becomes

$$\begin{array}{ll} \min\limits_{\mathbf{v}} & \mathbf{v}^T \mathbf{H} \mathbf{v} \\ \text{s.t.} & \mathbf{v}^T \mathbf{D} \mathbf{v} = 4u, \\ & v_0 = 1, \\ & v_i \in \{-1, 1\} \, \forall i \neq 0. \end{array} \tag{3.11}$$

This problem is equivalent to

$$\begin{array}{ll} \min\limits_{\mathbf{v}} & \mathbf{H} \bullet \mathbf{v}\mathbf{v}^T \\ \text{s.t.} & \mathbf{D} \bullet \mathbf{v}\mathbf{v}^T = 4u, \\ & v_0 = 1, \\ & v_i \in \{-1, 1\} \, \forall i \neq 0 \,, \end{array} \tag{3.12}$$

where $\mathbf{A} \bullet \mathbf{B} = \sum_{ij} A_{ij} B_{ij}$. This formulation of the problem is very similar to the max cut problem of size u (MC-u) (Zhang et al., 2006). The goal is to construct a relaxed version of the problem that can be efficiently solved using SDP. The constraint $v_0 = 1$ in (3.12) can be relaxed to $v_0 \in \{-1, 1\}$ without modifying the problem because $-\mathbf{v}$ is feasible for the other constraints whenever $\mathbf{v}$ is, and $\mathbf{H} \bullet \mathbf{v}\mathbf{v}^T = \mathbf{H} \bullet (-\mathbf{v})(-\mathbf{v})^T$. Furthermore, the constraints $v_i \in \{-1, 1\}$ can be rewritten in the form $\text{diag}(\mathbf{v}\mathbf{v}^T) = \mathbf{e}$

$$\begin{array}{ll} \min\limits_{\mathbf{v}} & \mathbf{H} \bullet \mathbf{v}\mathbf{v}^T \\ \text{s.t.} & \mathbf{D} \bullet \mathbf{v}\mathbf{v}^T = 4u, \\ & \text{diag}(\mathbf{v}\mathbf{v}^T) = \mathbf{e}. \end{array} \tag{3.13}$$

The problem can now be written in terms of a positive semidefinite matrix $\mathbf{V}$ of rank one

$$\begin{array}{ll} \min\limits_{\mathbf{V}} & \mathbf{H} \bullet \mathbf{V} \\ \text{s.t.} & \mathbf{D} \bullet \mathbf{V} = 4u, \\ & \text{diag}(\mathbf{V}) = \mathbf{e}, \\ & \mathbf{V} \succeq 0. \\ & \text{rank}(\mathbf{V}) = 1. \end{array} \tag{3.14}$$

This reformulation is possible because $\mathbf{V} = \mathbf{v}\mathbf{v}^T$ if and only if $\mathbf{V} \succeq 0$ and $\text{rank}(\mathbf{V}) = 1$. A convex optimization problem that can be solved by SDP is obtained from (3.14) by dropping the rank constraint

$$\begin{array}{ll} \min\limits_{\mathbf{V}} & \mathbf{H} \bullet \mathbf{V} \\ \text{s.t.} & \mathbf{D} \bullet \mathbf{V} = 4u, \\ & \text{diag}(\mathbf{V}) = \mathbf{e}, \\ & \mathbf{V} \succeq 0. \end{array} \tag{3.15}$$

This SDP problem can be efficiently solved in polynomial time with a suitable optimizer, such as the one designed in (Borchers, 1999). Following (Zhang et al., 2006), from a solution matrix $\mathbf{V}$ of problem (3.15), an approximate solution vector $\mathbf{u}$ of problem (3.9) can be obtained by a randomized approximate algorithm (Goemans and Williamson, 1995; Han et al., 2002). In particular, the components of $\mathbf{u}$ are determined by sampling $\mathbf{v}$ from a Gaussian distribution with zero mean and covariance matrix $\mathbf{V}$ and then applying the rule

$$u_i = \begin{cases} 1 & \text{if } \text{sign}(v_i) = \text{sign}(v_0)\,, \\ 0 & \text{if } \text{sign}(v_i) \neq \text{sign}(v_0)\,. \end{cases} \tag{3.16}$$

If the targeted number of predictors in the subensemble is not correctly determined by the application of this rule, we use a greedy algorithm that incorporates or removes elements in $\mathbf{u}$, as needed, causing the least deterioration in the cost function (3.9). This procedure is repeated ten times from which only the best solution is retained.

Even though (3.15) can be cast in a form that is similar to the MC-u problem, the approximation bounds that hold for the relaxed version of this problem (Goemans and Williamson, 1995; Han et al., 2002) are not applicable in the relaxed version of subset selection. Subset selection and the MC-u problem share optimal solutions but not optimal values. The reason is that the objective function in (3.15) does not exactly match the objective in the MC-u problem (Zhang et al., 2006). Despite this lack of guarantees for the quality of the approximation, the procedure described is very effective in selecting near-optimal ensembles in classification tasks (Martínez-Muñoz et al., 2009; Zhang et al., 2006). Similarly, SDP-pruning is expected to perform well also in regression ensembles.

The cost of selecting a subensemble of size u by solving problem (3.15) is $\mathcal{O}(M^3)$, where M is the size of the initial ensemble. The computation of $\boldsymbol{\mathcal{C}}^{(tr)}$ is $\mathcal{O}(M^2 \cdot N)$. Therefore, the total cost is $\mathcal{O}(M^3 + M^2 \cdot N)$, where N is the size of the training set $\mathcal{D}$. Extracting all near-optimal subensembles of sizes $u = 1, 2, \ldots, M$, implies solving (3.15) M times, with an overall cost $\mathcal{O}(M^4 + M^3 \cdot N)$.

3.2.2.2 Ordered Aggregation

Another method that can be used to tackle the subensemble selection problem is ordered aggregation (Hernández-Lobato et al., 2006b). Modifying the order in which predictors are aggregated has been successfully used to prune ensembles in classification tasks (Caruana and Niculescu-Mizil, 2004; Margineantu and Dietterich, 1997; Martínez-Muñoz and Suárez, 2004; Martínez-Muñoz and Suárez, 2006, 2007). In this section we propose to apply this ensemble pruning technique to regression bagging ensembles.

From the initial pool of M predictors generated by bagging, ordered aggregation builds a sequence of nested subensembles, in which the subensemble of size u contains the subensemble of size $u - 1$. The algorithm starts with an empty subensemble that

grows by incorporating at each iteration the predictor that reduces the training error of the enlarged subensemble the most. In particular, the predictor selected in the u-th iteration is the one that minimizes the expression

$$s_u = \arg\min_k \; u^{-2}\Big(\sum_{i=1}^{u-1}\sum_{j=1}^{u-1} \mathcal{C}^{(tr)}_{s_i s_j} + 2\sum_{i=1}^{u-1} \mathcal{C}^{(tr)}_{s_i k} + \mathcal{C}^{(tr)}_{kk}\Big)\,, \tag{3.17}$$

where $k \in \{1,..,N\}\backslash\{s_1, s_2..., s_{u-1}\}$ and the indices $\{s_1, s_2..., s_{u-1}\}$ label the predictors that have been incorporated in the pruned ensemble at iteration $u-1$. Figure 3.1 displays the pseudo-code for ordered aggregation.

Input: Training data $\mathcal{D} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$ and vector of predictors $\mathcal{R} = \{\hat{f}_1(\cdot), \ldots, \hat{f}_M(\cdot)\}$.

Output: Ordered vector of predictors $\mathcal{R} = \{\hat{f}_{s_1}(\cdot), \ldots, \hat{f}_{s_M}(\cdot)\}$.

1. For $i = 1, \ldots, M$
   (a) For $j = 1, \ldots, M$
      i. $\mathcal{C}_{ij} \leftarrow N^{-1}\sum_{n=1}^{N}\left[\left(\hat{f}_i(\mathbf{x}_n) - y_n\right)\left(\hat{f}_j(\mathbf{x}_n) - y_n\right)\right]$
2. $\mathbf{s} \leftarrow$ empty vector
3. For $u = 1, \ldots, M$
   (a) min $\leftarrow +\infty$
   (b) For $k \in \{1, \ldots, M\}\backslash\{s_1, \ldots, s_u\}$
      i. value $\leftarrow u^{-2}(\sum_{i=1}^{u-1}\sum_{j=1}^{u-1}\mathcal{C}_{s_i s_j} + 2\sum_{i=1}^{u-1}\mathcal{C}_{s_i k} + \mathcal{C}_{kk})$
      ii. if value $<$ min
         A. $s_u \leftarrow k$
         B. min $\leftarrow$ value
4. return $\mathbf{s}$

FIGURE 3.1: Algorithm that implements ordered aggregation.

This process can be seen as an ordering of the predictors of the complete ensemble because the subensemble generated at iteration u includes all the predictors of the subensemble generated at iteration $u-1$. The subensemble of size u with $1 \leq u \leq M$ is obtained by selecting the first u predictors from the ordered sequence. Note that subensembles identified by ordered aggregation need not be optimal. In particular, the optimal subensemble of size u (the one with the lowest mean squared error on the training data) need not include all the predictors of the optimal ensemble of size $u-1$. Nevertheless, ordered aggregation is expected to identify near-optimal solutions of the subensemble selection problem.

The time-complexity of this algorithm, as a function of the number of elements in the bagging ensemble, can be readily estimated. Each of the M iterations requires the extraction of the predictor that minimizes (3.17) from the remaining pool of predictors. This task has a cost $\mathcal{O}(((M+1)-u)\cdot u)$, where $1 \leq u \leq M$ is the current iteration. Therefore, the total complexity of the ordering algorithm is $\mathcal{O}(M^3)$. Finally, because computing $\boldsymbol{\mathcal{C}}^{(tr)}$ takes $\mathcal{O}(M^2 \cdot N)$ steps the final cost is $\mathcal{O}(M^3 + M^2 \cdot N)$. In contrast to SDP-pruning, where selecting subensembles of different sizes requires separate executions

of the algorithm, ordered aggregation generates a sequence of near-optimal subensembles of increasing size at no additional cost.

3.2.2.3 Comparison with Optimal Subensembles

Both SDP-pruning and ordered aggregation generate different subensembles of size u that could be suboptimal. To determine how close they are to the optimal subensembles, these approximate solutions are compared with the exact ones, obtained by exhaustive search. These experiments involve building 100 different bagging ensembles of $M = 32$ neural networks following the experimental procedure described in Section 3.2.3. For each ensemble, optimal subensembles of sizes $u = 1, \ldots, M$ are identified by exhaustive search. SDP-pruning and ordered aggregation are then used to generate near-optimal subensembles of different sizes. The experiments are carried out in the regression problem *Servo*. Figure 3.2 displays the percentage of predictors that appear both in the optimal subensemble and in the approximate one (either the one identified by ordered aggregation or by SDP-pruning) as a function of the subensemble size $1 \leq u \leq 32$. The curves show that the subensembles identified by SDP-pruning are almost identical to those obtained by exhaustive search. Ordered aggregation identifies subensembles that share on average at least 85% of their elements with the optimal ones.

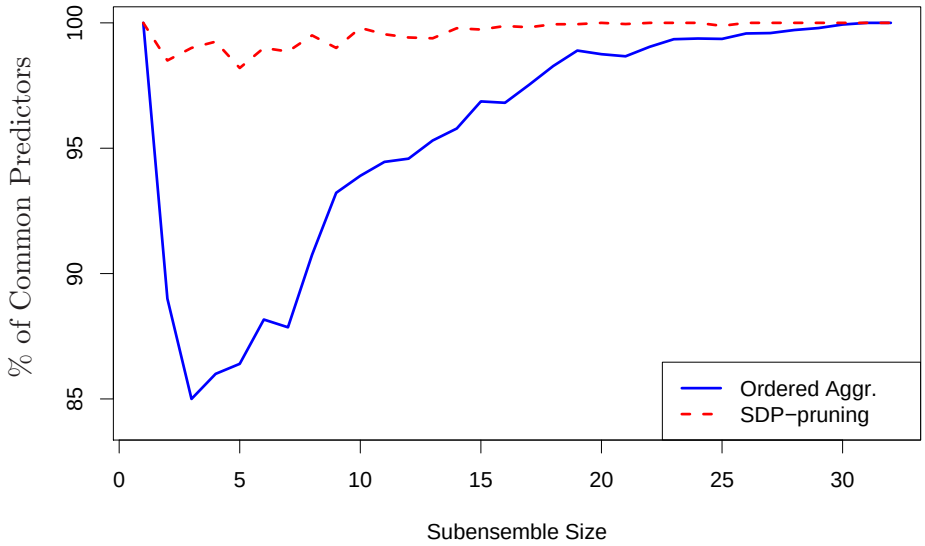

FIGURE 3.2: Average fraction of common predictors between the optimal subensemble and subensembles obtained by SDP-pruning (top curve), or ordered aggregation (bottom curve) as a function of the subensemble size. The regression problem is *Servo*.

Next, the prediction error of these subensembles is estimated in the training set and in an independent test set. Optimality is defined in terms of the prediction accuracy in the training set only. Therefore, the performance of subensembles that are optimal in the training set need not be optimal in the test set. Figure 3.3 displays the evolution of the average train and test error of the subensembles obtained by ordered aggregation, SDP-pruning and exhaustive search as a function of the subensemble size. The error of the original bagging ensemble is also displayed for reference. In bagging, the error decreases almost monotonically as more predictors are incorporated into the ensemble. This behavior is what should be expected from the random order in the aggregation of predictors. The error curves of the near-optimal subensembles also show an initial

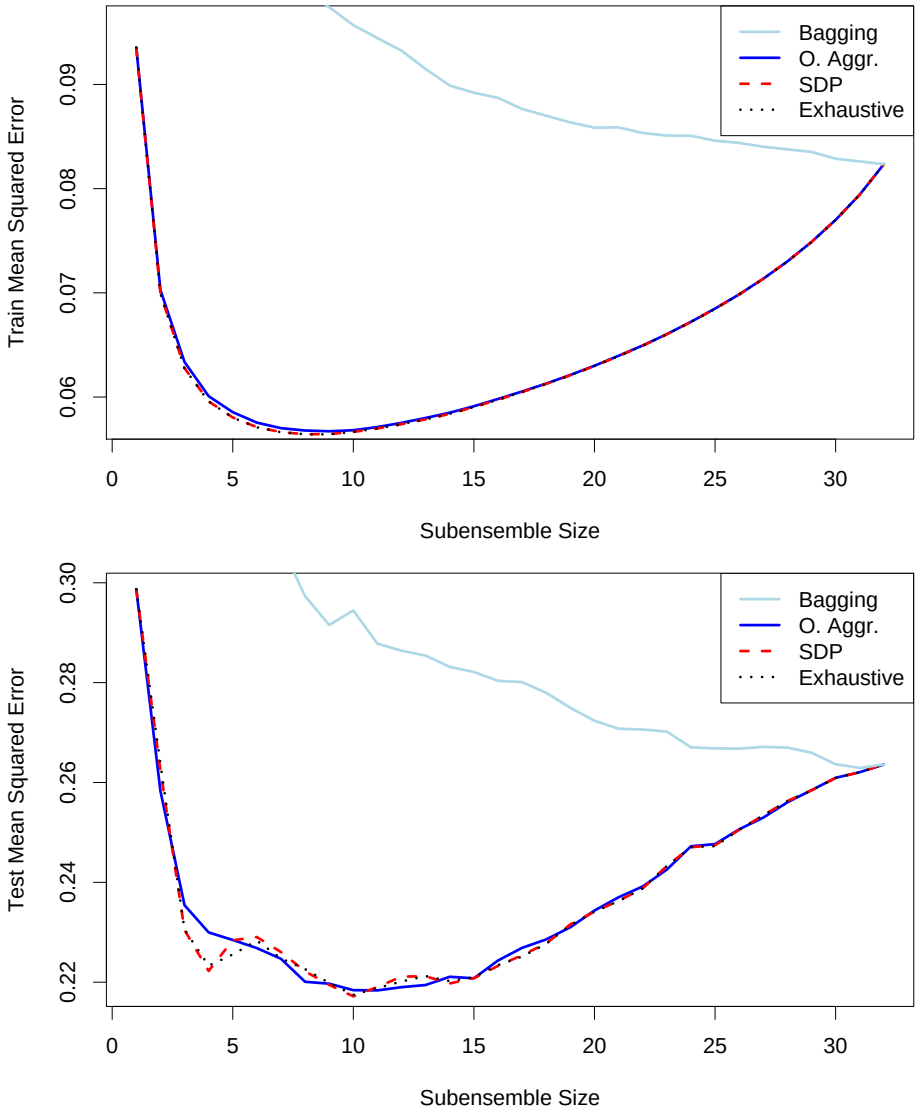

FIGURE 3.3: Average train (top) and test (bottom) errors of the subensembles obtained by means of ordered aggregation, SDP-pruning and the optimal subensembles found by the exhaustive search algorithm. Optimality is defined in terms of the error in the training set. The regression problem is *Servo*.

decrease. However, this decrease is steeper than in the standard bagging ensemble. At intermediate subensemble sizes the error curves display a fairly broad minimum. Beyond this minimum the error increases slowly and eventually approaches the error level of the complete bagging ensemble from below. The curves for the training error of the optimal subensembles identified by exhaustive search are a lower bound to the error curves of subensembles obtained by means of SDP-pruning and ordered aggregation. The training error curve for SDP-pruning is almost identical to the one corresponding to optimal pruning. The curves for ordered aggregation are only slightly above the optimal one. This means that even though some of the predictors included by ordered aggregation replace others that appear in the optimal solution, their aggregate performance is still close to being optimal.

We note that the features observed in the error curves for the training set are also present in the test error curves, albeit with some differences. Similarly as in training, the minimum in the test error curves appears at intermediate subensemble sizes. The value of this minimum error for the pruned subensembles is also significantly lower than the best error of a standard bagging ensemble. In contrast to the training error curves, there are no clear differences in generalization performance among the different subensembles. Another notable difference is that the minimum appears earlier in the aggregation process in the training error curve than in the error curve for the test set. This means that it is difficult to exactly estimate from the training data where the global minimum in the test error curve is. Nevertheless, the error curves are rather flat around the minimum, so that small errors in the estimation of the size of the ensemble that corresponds to the minimum do not have a large impact on the generalization performance. The features displayed by the error curves shown are representative of most regression problems investigated in this chapter and also appear in classification problems (Martínez-Muñoz et al., 2009).

3.2.3 Experiments

To assess the performance of SDP-pruning and ordered aggregation, experiments are carried out on 20 regression problems from the UCI-Repository (Asuncion and Newman, 2007), from the Weka Data Mining Tool (Witten and Frank, 2005) and from other sources (Breiman, 1999b; Chambers and Hastie, 1992; Friedman, 1991; Kung, 1986; R Development Core Team, 2005). They include synthetic and real-world problems from different fields of application. Table 3.1 displays some characteristics of the regression tasks considered: the number of instances, the number of attributes and the source of the different datasets. As base models we employ feed-forward neural networks with a single hidden layer of sigmoidal neurons and linear units in the output layer. The networks are trained during 1,000 epochs using the quasi-Newton optimization method BFGS (Nocedal and Wright, 1999). A weight decay strategy is used in the training process to reduce the amount of over-fitting (Krogh and Hertz, 1992). All computations are performed using the neural networks package (Venables and Ripley, 2002) of the R statistics software (R Development Core Team, 2005).

For each of the real-world datasets, 10-fold cross-validation is used to estimate the squared regression error. This 10-fold cross-validation process is repeated 10 times for different random partitions of the data. The values reported are averages of the cross-validation errors over these different partitions of the data. The computation of each 10-fold cross-validation estimate involves the following steps: (i) Generate a random partition of the original data into 10 disjoint sets to carry out 10-fold cross-validation. In each of the folds, nine sets are used for training and one for validation. The attributes of the data instances are normalized so that they have zero mean and unit variance in the training set. (ii) For each of the ten partitions into training and test, generate a bagging ensemble of 100 neural networks using bootstrap sampling from the training set. For these neural networks, different architectures (3, 5, and 7 hidden units) are explored. For the natural logarithm of the weight decay constant ten evenly-spaced values in the interval $[-6, 2]$ are considered. All possible combinations of the number of hidden units and of the values of the weight decay constant are tried to determine which set of parameters provides the best regression fit based on a separate 10-fold cross-validation estimate on the training data. Once the optimal combination of parameters has been found, the neural networks are trained over 1,000 epochs. (iii) The error

TABLE 3.1: Characteristics of the datasets used in the experiments.

Dataset	Cases	Attr.	Train/Test	Source
AutoPrice	159	15	10-fold-cv	Weka
Bodyfat	252	14	10-fold-cv	Weka
Bolts	40	7	10-fold-cv	Weka
Boston	506	13	10-fold-cv	UCI-Repository
Chick	578	3	10-fold-cv	R *mlbench* package
Fires	517	13	10-fold-cv	UCI-Repository
Friedman1	-	10	200 / 2,000	(Friedman, 1991)
Friedman2	-	4	200 / 2,000	”
Friedman3	-	4	200 / 2,000	”
Loblolly	84	2	10-fold-cv	(Kung, 1986)
Ozone	330	8	10-fold-cv	UCI-Repository
Peak	-	20	200 / 2,000	(Breiman, 1999b)
Pollution	60	15	10-fold-cv	Weka
Sensory	576	11	10-fold-cv	Weka
Servo	167	4	10-fold-cv	UCI-Repository
Solder	720	5	10-fold-cv	(Chambers and Hastie, 1992)
Theoph	132	4	10-fold-cv	R *mlbench* package
Tooth	60	2	10-fold-cv	”
Wisconsin	198	35	10-fold-cv	UCI-Repository

of each ensemble is estimated on the corresponding unseen data and these values are averaged over the different folds to compute the cross-validation error. (iv) A single neural network and a boosting ensemble of 100 members are also built for comparison using the same network configuration as in the bagging ensemble. The boosting ensemble is generated by the Adaboost.R2 algorithm using a linear loss function (Drucker, 1997). (v) A subensemble of 20 neural networks (i.e. $u = 20$) is extracted from the original pool of classifiers generated by bagging using SDP-pruning, as described in Section 3.2.2.1. The bagging ensemble is then ordered according to the algorithm described in Section 3.2.2.2. Then, the first 20 networks of the ordered sequence are selected and aggregated in a pruned subensemble. (vi) The error of each pruned ensemble, the single neural network and the Adaboost.R2 ensemble is estimated on the corresponding unseen data. The cross-validation errors over the different partitions considered are averaged to compute the cross-validation error estimate. For the synthetic datasets (*Friedman1, Friedman2, Friedman3*, and *Peak*) the individual neural networks and the ensembles are generated and ordered using the procedure described above. However, instead of averages over 10 different 10-fold cross-validation error estimates, the values reported are averages over 100 independent realizations of the training and test datasets.

Table 3.2 shows the results for the average mean squared error estimated on the test set for the different regression problems. The values in the first column correspond to the error of a single neural network. The second column displays the error of the complete bagging ensemble. The third column displays the error of the boosting ensemble. Finally, the fourth column displays the error of the ensemble selected by SDP-pruning and the fifth column the corresponding error for ordered aggregation. These results show that bagging ensembles often perform better than a single neural network and that pruning

improves the accuracy of bagging in most of the problems investigated. In particular, pruned subensembles that contain only 20% of the initial predictors exhibit a consistently good generalization performance in the regression problems investigated. As a matter of fact, because the test error curves are rather flat around the minimum, any value between 15% and 25% percent of the original pool of predictors gives similar results. Pruned ensembles also outperform single networks and boosting ensembles in most of the problems investigated.

To determine whether these improvements in accuracy are statistically significant a paired *Wilcoxon-test* (Wilcoxon, 1968) is performed, as suggested by Demšar (2006). Error values that are significantly better than bagging according to the Wilcoxon test at a significance level of 5% are highlighted in boldface. Error values that are significantly worse than bagging are underlined. In a similar way, error values that are significantly better than boosting are marked with the symbol ◀. Values that are significantly worse than boosting are marked with the symbol ◁.

An overall comparison of the performance of the different methods in the collection of problems investigated can be made using the framework proposed by Demšar (2006). A Friedman test on the average ranks of each algorithm in the problems investigated rejects the hypothesis that their performance in the different problems is equivalent with a significance level of 5%. In consequence, a Nemenyi test is used to determine whether the differences in average rank among the different algorithms are significant. Figure 3.4 displays the results of this test. In this figure, algorithms whose differences in performance are not statistically significant at a significance level of 5% are connected with a solid black line. The differences in performance between algorithms whose average ranks are further than the critical difference (CD) are statistically significant. In the collection of regression problems investigated ensembles pruned using ordered aggregation have a better overall performance than the corresponding complete bagging ensembles, boosting ensembles and single neural networks. The differences in average ranks are statistically significant. Ensembles pruned by means of SDP-pruning also perform well. However, the differences in performance with respect to complete bagging are not statistically significant. Subensembles selected by ordered aggregation generally have a slightly better generalization performance than subensembles selected by SDP-pruning, despite the fact that the latter have lower errors in the training set.

The error curves in Figure 3.5 trace the dependence of the average test error curves for bagging, ordered aggregation and boosting as a function of the ensemble size for a representative subset of the regression problems investigated. For SDP-pruning, because of its high computational cost, only the average error of a single subensemble of size $u = 20$ is displayed. The average test error of a single neural network is marked as an horizontal line. The curves for bagging and ordered aggregation exhibit a similar qualitative behavior as those depicted in Figure 3.3. The decrease of the test error is almost monotonous as the size of a randomly ordered bagging ensemble increases. Modifying the order of the aggregation according to the procedure described leads to an initially steeper descent of the error curves, which, with some exceptions (e.g. *Solder*, *Tooth*) is followed by a broad minimum and a final rise to the error level of the complete bagging ensemble. The performance of subensembles selected by SDP-pruning is similar to those subensembles obtained by ordered aggregation for size $u = 20$. However, in some cases SDP-pruning leads to slightly higher error rates (e.g. *Tooth*).

TABLE 3.2: Average mean squared test error for a single neural network, for complete bagging ensembles, Adaboost.R2 ensembles, and pruned ensembles, selected by SDP-pruning or by ordered aggregation.

Problem	**Neural Network**	**Bagging**	**Adaboost.R2**	**Pruned Bagging**	
				Ordered Aggregation	**SDP-pruning**
AutoPrice	$1.17 \cdot 10^7 \pm 8.61 \cdot 10^6$ ◁	$7.37 \cdot 10^6 \pm 5.88 \cdot 10^6$ ◁	$\mathbf{5.56 \cdot 10^6 \pm 4.02 \cdot 10^6}$	$\mathbf{6.32 \cdot 10^6 \pm 4.97 \cdot 10^6}$ ◁	$\mathbf{6.32 \cdot 10^6 \pm 4.96 \cdot 10^6}$ ◁
Bodyfat	2.96 ± 3.76	1.89 ± 2.76 ◀	2.55 ± 2.00	2.10 ± 2.78 ◀	2.12 ± 2.78 ◀
Bolts	$\mathbf{5.31 \cdot 10^1 \pm 7.70 \cdot 10^1}$ ◀	$6.56 \cdot 10^1 \pm 7.69 \cdot 10^1$ ◀	$9.72 \cdot 10^1 \pm 1.33 \cdot 10^2$	$\mathbf{4.50 \cdot 10^1 \pm 7.01 \cdot 10^1}$ ◀	$\mathbf{4.57 \cdot 10^1 \pm 7.17 \cdot 10^1}$ ◀
Boston	$1.27 \cdot 10^1 \pm 6.43$ ◁	$1.15 \cdot 10^1 \pm 6.40$ ◁	$\mathbf{1.00 \cdot 10^1 \pm 4.29}$	$\mathbf{1.07 \cdot 10^1 \pm 5.49}$ ◁	$\mathbf{1.07 \cdot 10^1 \pm 5.50}$ ◁
Chick	$\mathbf{5.44 \cdot 10^1 \pm 3.66 \cdot 10^1}$ ◁	$6.57 \cdot 10^1 \pm 3.58 \cdot 10^1$ ◁	$\mathbf{3.89 \cdot 10^1 \pm 2.74 \cdot 10^1}$	$\mathbf{4.88 \cdot 10^1 \pm 2.60 \cdot 10^1}$ ◁	$\mathbf{4.82 \cdot 10^1 \pm 2.43 \cdot 10^1}$ ◁
Fires	$1.82 \cdot 10^3 \pm 4.04 \cdot 10^3$	$1.55 \cdot 10^3 \pm 3.49 \cdot 10^3$ ◁	$\mathbf{1.15 \cdot 10^3 \pm 2.59 \cdot 10^3}$	$1.31 \cdot 10^3 \pm 3.05 \cdot 10^3$ ◁	$1.32 \cdot 10^3 \pm 3.06 \cdot 10^3$ ◁
Friedman1	$\mathbf{4.82 \pm 1.29}$ ◀	$4.85 \pm 4.43 \cdot 10^{-1}$ ◀	$5.05 \pm 7.61 \cdot 10^{-1}$	$\mathbf{4.45 \pm 4.57 \cdot 10^{-1}}$ ◀	$\mathbf{4.45 \pm 4.61 \cdot 10^{-1}}$ ◀
Friedman2	$2.68 \cdot 10^4 \pm 8.93 \cdot 10^3$ ◁	$2.19 \cdot 10^4 \pm 1.55 \cdot 10^3$	$2.17 \cdot 10^4 \pm 1.34 \cdot 10^3$	$\mathbf{2.06 \cdot 10^4 \pm 1.47 \cdot 10^3}$ ◀	$\mathbf{2.06 \cdot 10^4 \pm 1.45 \cdot 10^3}$ ◀
Friedman3	$1.83 \cdot 10^{-2} \pm 2.46 \cdot 10^{-3}$ ◁	$1.70 \cdot 10^{-2} \pm 2.12 \cdot 10^{-3}$ ◀	$1.74 \cdot 10^{-2} \pm 2.24 \cdot 10^{-3}$	$\mathbf{1.67 \cdot 10^{-2} \pm 2.17 \cdot 10^{-3}}$ ◀	$\mathbf{1.67 \cdot 10^{-2} \pm 2.16 \cdot 10^{-3}}$ ◀
Loblolly	$\mathbf{4.45 \pm 8.93}$ ◀	6.39 ± 6.36 ◀	$1.09 \cdot 10^1 \pm 9.60$	$\mathbf{3.93 \pm 6.22}$ ◀	$\mathbf{3.96 \pm 6.24}$ ◀
Orange	$2.21 \cdot 10^2 \pm 1.80 \cdot 10^2$	$1.85 \cdot 10^2 \pm 1.32 \cdot 10^2$	$1.97 \cdot 10^2 \pm 1.44 \cdot 10^2$	$\mathbf{1.66 \cdot 10^2 \pm 1.06 \cdot 10^2}$ ◀	$1.68 \cdot 10^2 \pm 1.09 \cdot 10^2$ ◀
Ozone	$1.69 \cdot 10^1 \pm 4.42$ ◀	$1.65 \cdot 10^1 \pm 4.28$ ◀	$1.86 \cdot 10^1 \pm 4.83$	$1.65 \cdot 10^1 \pm 4.37$ ◀	$1.65 \cdot 10^1 \pm 4.37$ ◀
Peak	$3.16 \cdot 10^1 \pm 5.29$ ◁	$2.63 \cdot 10^1 \pm 3.37$ ◁	$\mathbf{2.47 \cdot 10^1 \pm 2.34}$	$\mathbf{2.37 \cdot 10^1 \pm 3.37}$ ◀	$\mathbf{2.37 \cdot 10^1 \pm 3.35}$ ◀
Pollution	$5.90 \cdot 10^3 \pm 6.54 \cdot 10^3$	$3.94 \cdot 10^3 \pm 3.25 \cdot 10^3$	$7.50 \cdot 10^3 \pm 1.12 \cdot 10^4$	$\mathbf{3.20 \cdot 10^3 \pm 2.47 \cdot 10^3}$ ◀	$\mathbf{3.25 \cdot 10^3 \pm 2.49 \cdot 10^3}$ ◀
Sensory	$5.34 \cdot 10^{-1} \pm 9.46 \cdot 10^{-2}$ ◀	$5.35 \cdot 10^{-1} \pm 9.23 \cdot 10^{-2}$ ◀	$5.63 \cdot 10^{-1} \pm 9.84 \cdot 10^{-2}$	$\mathbf{5.20 \cdot 10^{-1} \pm 9.18 \cdot 10^{-2}}$ ◀	$\mathbf{5.19 \cdot 10^{-1} \pm 9.11 \cdot 10^{-2}}$ ◀
Servo	$3.27 \cdot 10^{-1} \pm 3.28 \cdot 10^{-1}$ ◁	$2.63 \cdot 10^{-1} \pm 2.35 \cdot 10^{-1}$ ◁	$\mathbf{1.70 \cdot 10^{-1} \pm 1.78 \cdot 10^{-1}}$	$\mathbf{2.05 \cdot 10^{-1} \pm 1.81 \cdot 10^{-1}}$ ◁	$\mathbf{2.03 \cdot 10^{-1} \pm 1.77 \cdot 10^{-1}}$ ◁
Solder	9.04 ± 3.37 ◀	8.36 ± 3.13 ◀	9.44 ± 3.53	8.41 ± 3.20 ◀	8.43 ± 3.23 ◀
Theoph	5.30 ± 3.77 ◁	4.43 ± 2.27 ◁	$\mathbf{3.04 \pm 2.15}$	$\mathbf{3.41 \pm 2.01}$ ◁	$\mathbf{3.42 \pm 2.00}$ ◁
Tooth	$\mathbf{1.40 \cdot 10^1 \pm 6.89}$ ◀	$1.43 \cdot 10^1 \pm 7.08$ ◀	$1.52 \cdot 10^1 \pm 7.92$	$1.44 \cdot 10^1 \pm 7.20$ ◀	$1.44 \cdot 10^1 \pm 7.22$ ◀
Wisconsin	$9.64 \cdot 10^2 \pm 2.66 \cdot 10^2$	$9.18 \cdot 10^2 \pm 2.48 \cdot 10^2$ ◀	$9.70 \cdot 10^2 \pm 2.31 \cdot 10^2$	$9.13 \cdot 10^2 \pm 2.40 \cdot 10^2$ ◀	$9.13 \cdot 10^2 \pm 2.39 \cdot 10^2$ ◀

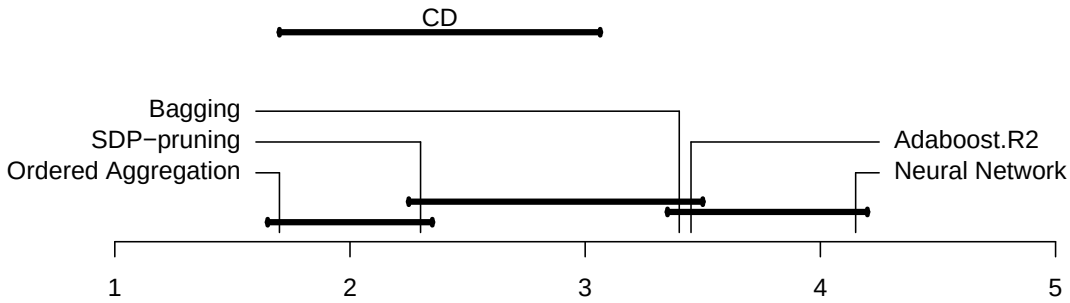

FIGURE 3.4: Results of a Nemenyi test on the average ranks of the different regression systems: a single neural network, bagging, boosting (Adaboost.R2) and pruned subensembles selected by SDP-pruning or by ordered aggregation. The critical difference (CD) between average ranks is displayed at the top of the figure.

The behavior of boosting ensembles is more heterogeneous, reflecting the lack of robustness of this method. Boosting outperforms bagging, ordered aggregation and SDP-pruning in a few regression problems (e.g. *Boston*, *Peak*). However, boosting has a detrimental effect in others (e.g. *Tooth, Pollution, Wisconsin*). A similar deterioration of performance of boosting has been detected in noisy classification problems (Dietterich, 2000b; Martínez-Muñoz and Suárez, 2007). The origin of this behavior is that boosting tends to assign larger weights to incorrectly labeled training instances, which can confuse the learning algorithm. Reducing the error rate of those instances leads to over-fitting and hence, to a poor generalization performance.

3.2.3.1 Bias-variance-covariance Analysis

In this section we carry out a bias-variance-covariance decomposition to analyze the dependence of the error on the size of the ensemble and to investigate the mechanisms by which ensemble learning and ensemble pruning can improve the generalization performance. Since the predictors that make up a bagging ensemble are generated from bootstrap samples of the same original training data $\mathcal{D}$ and they use the same learning algorithm (e.g. neural networks with a fixed architecture), they can be seen as dependent realizations of a random variable drawn from a probability distribution $\mathcal{P}\left(\hat{f}_i(\cdot)\right)$. As shown by Ueda and Nakano (1996), the mean squared error of a regression ensemble of size u is composed of three terms: the average bias, the average variance and the average covariance of the individual predictors of the ensemble

$$\mathcal{L}^{(u)} = u^{-1}\overline{\text{Var}} + (1-u^{-1})\overline{\text{Cov}} + \overline{\text{Bias}}^2\,, \tag{3.18}$$

with the definitions

$$\overline{\text{Bias}} = u^{-1}\sum_{i=1}^{u}\text{Bias}(\hat{f}_i), \qquad \overline{\text{Var}} = u^{-1}\sum_{i=1}^{u}\text{Var}(\hat{f}_i),$$
$$\overline{\text{Cov}} = (u-1)^{-1}u^{-1}\sum_{j\neq i}\text{Cov}(\hat{f}_i,\hat{f}_j)\,. \tag{3.19}$$

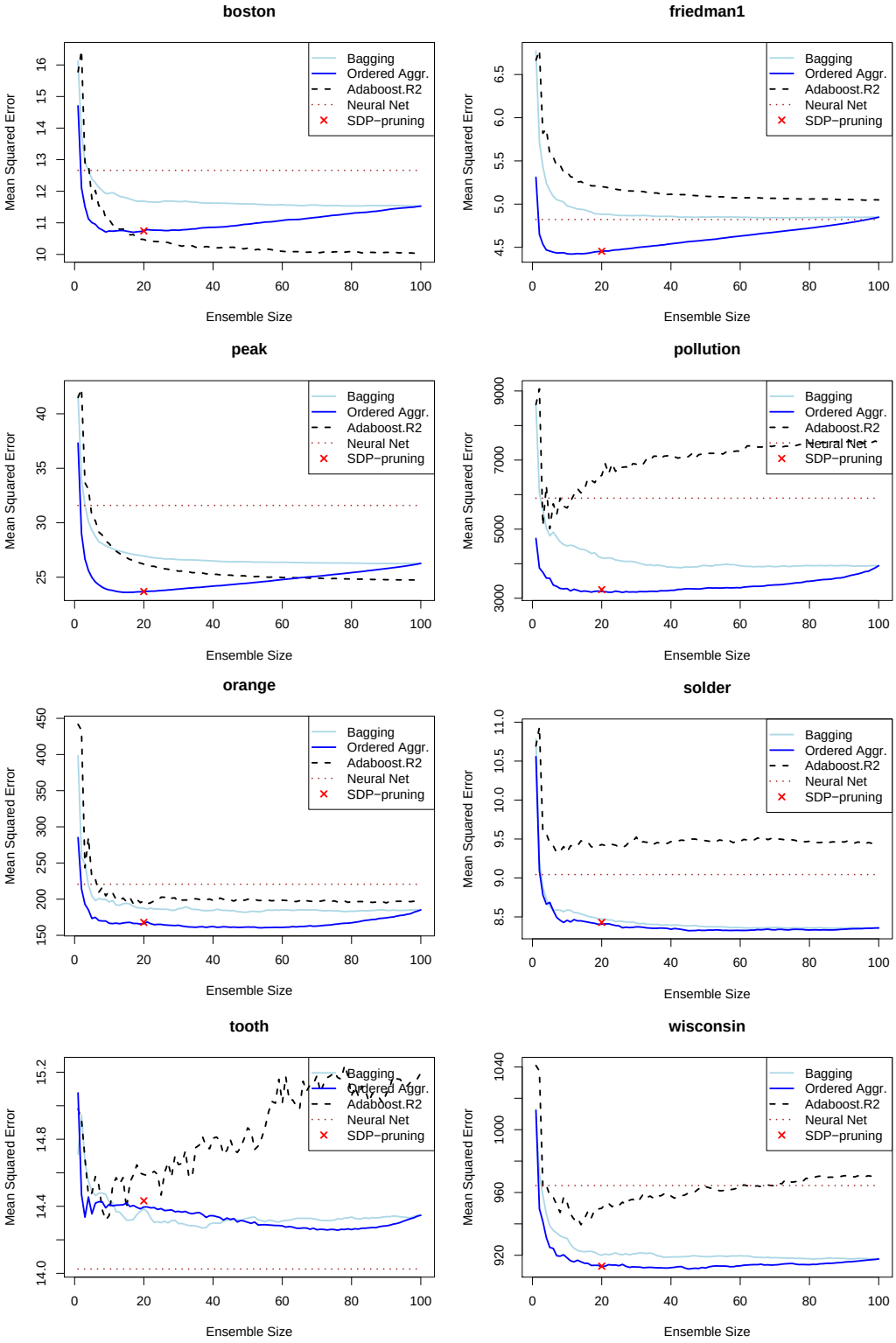

FIGURE 3.5: Test error curves for standard (randomly ordered) bagging, ordered aggregation, and for Adaboost.R2 ensembles as a function of the ensemble size for a variety of regression problems. The error of a single neural network is displayed as an horizontal line for reference. The average test error for a subensemble of size $u = 20$ selected with SDP-pruning is marked with a cross.

In standard bagging the expected variances and biases of all predictors are equal. Therefore, the expected squared error of a standard bagging ensemble of size u is simply

$$\mathcal{L}^{(u)} = u^{-1}\text{Var} + (1 - u^{-1})\overline{\text{Cov}} + \text{Bias}^2\,, \tag{3.20}$$

where Var and Bias are the expected variance and bias of a predictor drawn from the distribution $\mathcal{P}\left(\hat{f}_i(\cdot)\right)$ and $\overline{\text{Cov}}$ is their average covariance.

Given an original bagging ensemble of size M, the selection strategies used in ordered aggregation and in SDP-pruning have the effect of modifying the distribution of the predictors that are part of the near-optimal subensembles of size $u \leq M$. Instead of being independent of the subensemble size, this distribution changes as new predictors are incorporated into the partial subensemble. Let $\mathcal{P}\left(\hat{f}_i(\cdot); u\right)$ denote the distribution of the predictors that are part of the near-optimal subensemble of size u. The bias-variance-covariance error decomposition for the near-optimal subensembles of size u is

$$\mathcal{L}^{(u)} = u^{-1}\text{Var}(u) + (1 - u^{-1})\overline{\text{Cov}}(u) + \text{Bias}(u)^2\,, \tag{3.21}$$

where $\text{Var}(u)$ and $\text{Bia}s(u)$ are the expected variance and bias of a predictor drawn from the distribution $\mathcal{P}\left(\hat{f}_i(\cdot); u\right)$ and $\overline{\text{Cov}}(u)$ is the average covariance between the members of a near-optimal subensemble of size u. As a result of the selection strategies, the values of $\text{Bias}(u)$ and $\text{Var}(u)$ (the average bias and variance of a predictor in a near-optimal subensemble of size u) are expected to be lower than Bias and Var (the average bias and variance bias and variance of a predictor in a standard bagging ensemble) at least up to intermediate subensemble sizes. A similar behavior is expected for $\overline{\text{Cov}}(u)$. In this manner, near-optimal subensembles achieve lower errors than standard bagging subensembles for $u = 1, 2, \ldots, (M-1)$. Note that when $u = M$, the original ensemble of size M is recovered, $\mathcal{P}\left(\hat{f}_i(\cdot); u\right) = \mathcal{P}\left(\hat{f}_i(\cdot)\right)$ and $\overline{\text{Cov}}(M) = \overline{\text{Cov}}$.

The lowest error that standard bagging achieves is $\lim_{u\to\infty}\mathcal{L}^{(u)} = \text{Bias}^2 + \overline{\text{Cov}} \geq 0$. Hence, it is possible for the subensemble at iteration u to have a lower error than this asymptotic limit if the inequality

$$u^{-1}\text{Var}(u) + (1 - u^{-1})\overline{\text{Cov}}(u) + \text{Bias}(u)^2 < \text{Bias}^2 + \overline{\text{Cov}} \tag{3.22}$$

is satisfied. This inequality is fulfilled if the approximate strategy selects from the complete ensemble a set of predictors with low bias, low variance and small correlations as well. The experiments carried out show that the inequality is fulfilled for a large range of subensemble sizes.

Figure 3.6 displays the value of the average test error, and the values of its components (average squared bias, variance and covariance) as a function of ensemble size for standard bagging, ordered aggregation and SDP-pruning in the synthetic regression problem *Peak* (Breiman, 2001). The original bagging ensemble is composed of 100 neural networks with 5 units in the hidden layer. These networks are trained on independent bootstrap samples generated from a training set of 200 instances. The test set is independent of the training data and consists of $2,000$ elements. Errors are estimated by averaging over 1,000 realizations of the training set. The test error curves exhibit similar features to the curves displayed in Figure 3.5. For standard bagging, the values of the average squared bias, variance and covariance remain approximately constant as the number of predictors included in the ensemble varies. In contrast with standard bagging, the average squared bias, variance and covariance typically increase with size

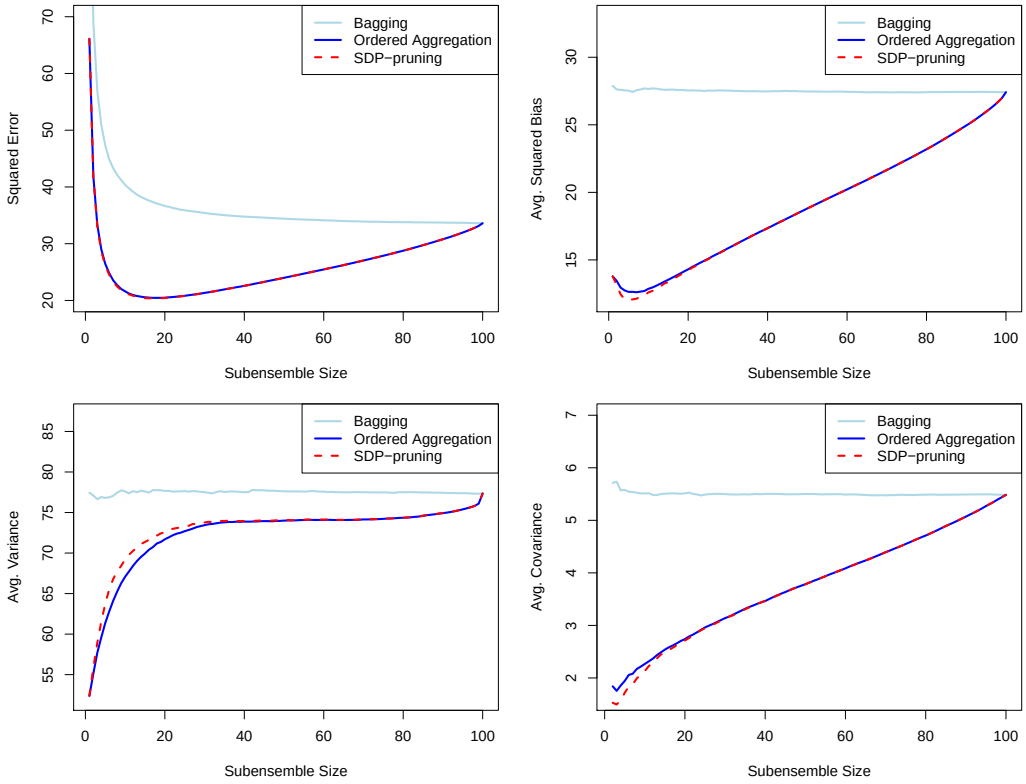

FIGURE 3.6: From top-left to bottom-right: average test error, squared bias, variance and covariance as a function of ensemble size for bagging, ordered aggregation and SDP-pruning for the synthetic regression problem *Peak*.

of the subensembles selected using ordered aggregation and SDP-pruning. A lower covariance among the ensemble members was also observed in (Liu and Yao, 1999) and is a consequence of minimizing (3.5) (Brown et al., 2005b; Liu and Yao, 1999).

According to (3.20) and (3.21), as the size of the ensemble grows, the squared bias and the covariance become the most important terms in the error decomposition. In standard bagging the test error of the complete bagging ensemble is approximately equal to the sum of the squared bias and the covariance term, in agreement with the infinite-size limit of (3.20). The improvements in performance of both ordered aggregation and SDP-pruning are based on incorporating first predictors with a low bias that also have small correlations.

3.3 Pruning Parallel Classification Ensembles

There is a large body of theoretical and empirical research showing that the combination of the predictions of complementary classifiers in ensemble methods can be an effective strategy to improve the generalization performance of a single classifier (Breiman, 1996a, 2001; Bühlmann, 2003; Dietterich, 2000b; Martínez-Muñoz and Suárez, 2005; Rodríguez et al., 2006). In these methods the final ensemble decision for an unlabeled instance is typically obtained by combining the individual decisions of all the classifiers in the ensemble. In this section we seek to reduce the number of classifiers that need to be queried to compute this decision. Specifically, we focus on homogeneous ensembles composed of classifiers that are generated by applying a fixed learning algorithm to

modified versions of the training data or by applying a randomized learning algorithm to the same training data. Furthermore, we assume that the aggregated prediction of the ensemble is computed by simple majority voting. If the classifiers are generated independently given the training data, we show that is possible to estimate the outcome of this voting process with a specified confidence level by polling only a subset of the total classifiers in the ensemble (Hernández-Lobato et al., 2009). The procedure is based on estimating the probability distribution of the remaining class label predictions based on the decisions made by the classifiers that have already been queried. This process corresponds to a Polya urn model with $c = 1$, for which an explicit form for the probability mass function can be given (Johnson et al., 1997). This probability mass function can then be used to estimate the probability that the majority class will not be modified when the remaining votes of the ensemble members are taken into account. If there is some tolerance to disagreements between the predictions of the partially polled ensemble and of the complete ensemble, the voting process can be stopped when this probability exceeds the specified confidence level. The final classification is the combined decision of the polled classifiers only.

This dynamical instance-based (IB) pruning mechanism can be used to improve the prediction efficiency of any classification ensemble provided that its members are generated independently when conditioned to the training data and that majority voting is used for combining the individual classifier predictions. Examples of ensembles of this kind be found in (Breiman, 1996a, 2001; Bühlmann, 2003; Martínez-Muñoz and Suárez, 2005; Rodríguez et al., 2006). In this section we present results for bagging (Breiman, 1996a) and random forests (Breiman, 2001), which are two representative ensemble learning algorithms. Experiments on a variety of classification problems show that the rate of disagreement in the class label predictions of complete ensembles and partially polled ensembles are in the vicinity of the specified confidence level. Furthermore, only small differences in generalization performance are found. When IB-pruning is used only a subset of the predictors of the ensemble need to be queried. As a consequence, the classification process can be significantly sped-up.

The method presented in this section uses a framework similar to the statistical description of the evolution of the prediction error in majority voting algorithms as the number of classifiers in the ensemble increases (Esposito and Saitta, 2003; Hansen and Salamon, 1990; Lam and Suen, 1997; Martínez-Muñoz and Suárez, 2005). However, to our knowledge, the statistical properties of the aggregation process implemented in majority voting have not been used as a basis for ensemble pruning.

3.3.1 Related Work

Before introducing IB-pruning, in this section we review some methods that have been proposed to dynamically select, for each instance that has to be labeled, a small subset of predictors from the original ensemble.

Fan et al. (2002) and Wang et al. (2003) designed a procedure to speed up classification in ensembles based on ideas similar to the ones used in IB-pruning. In these articles, the goal is to minimize the number of classifiers needed for prediction in cost-sensitive applications. For this purpose, the distribution of the discrepancies between the predictions of the subensemble and of the complete ensemble is assumed to be Gaussian. The parameters of this distribution are then estimated on the training data. The method proposed by Fan et al. (2002) assumes that the individual classifiers output continuous probabilities and therefore cannot be applied without modifications to majority voting.

A more serious difficulty is that in the classification problems investigated in Section 3.3.3 the discrepancies between the partial and the complete ensemble class label predictions tend to be smaller for the training set than for an independent test set. This means that, in general, the estimates on the training data are biased. On average, for the same confidence level, fewer classifiers need to be queried for training instances than for instances not included in the training set.

There are other instance-based (dynamic) techniques for classifier selection based on measures of local accuracy (Giacinto and Roli, 2001b; Ho et al., 1994; Tsymbal et al., 2003; Tsymbal and Puuronen, 2000; Woods et al., 1997). The main goal of these methods is to improve the overall performance of the ensemble by selecting and/or giving more weight to the classifiers that perform best in instances that are similar to the one that is being classified. Typically, a meta-level classifier that estimates the local errors of the base classifiers for each new instance is employed in the classifier selection process. An evaluation of different dynamic classifier selection methods for different types of ensemble members is provided in (Canuto et al., 2007). As in IB-pruning, these methods do not reduce the storage needs of the ensemble. All ensemble members need to be retained in memory to resolve potential queries.

3.3.2 Statistical Instance-Based Ensemble Pruning

Consider an ensemble of T classifiers $\hat{f}_1, \ldots, \hat{f}_T$. Assuming that majority voting is used to combine the decisions of these elements, the class predicted by the ensemble for an unlabeled instance, characterized by the vector of attributes $\mathbf{x}$, is

$$\arg\max_{y} \sum_{i=1}^{T} \mathbb{I}\left(\hat{f}_i(\mathbf{x}) = y\right), y \in \mathcal{Y}, \tag{3.23}$$

where $\hat{f}_i(\mathbf{x})$ is the class prediction of the i-th member of the ensemble, $\mathbb{I}(z)$ is an indicator function that takes value one when z is satisfied and zero otherwise, and $\mathcal{Y} = \{y_1, \ldots, y_l\}$ is the set of possible class labels.

This voting scheme is usually implemented by first computing the outputs of all the classifiers in the ensemble and then finding the majority class. However, for most instances, polling all classifiers is not necessary to determine the final prediction. If in the first t queries ($t < T$) the difference of votes between the majority class and the second most voted class is larger than $T - t$ then the class prediction cannot be changed by the remaining (still unknown) votes. In particular, for binary classification problems the querying process can be stopped when more than 50% of the ensemble members agree in their prediction. The querying process can actually be stopped earlier if one is willing to accept that a specified (small) fraction of the class labels predicted by the polled subensemble are in disagreement with the predictions of the complete ensemble. With regard to the generalization performance, the disagreement rate is an upper bound for the differences between the error rate of the polled subensemble and of the complete ensemble. In practice, if the changes in class labels affect both correctly labeled examples and misclassified examples in approximately equal numbers, the differences in error rates should be much smaller than this upper bound.

Consider a parallel ensemble in which the individual classifiers are generated by independent applications of a fixed learning algorithm to different modified versions of the initial training data or by independent applications of a randomized learning algorithm to the initial training data. That is, the ensemble classifiers are generated

independently when conditioned to these training data. Note that this is different from assuming that the ensemble members are unconditionally independent, which is typically not the case. Under these conditions, the classification process of an arbitrary instance $\mathbf{x}$ can be described in terms of the probability vector

$$\boldsymbol{\pi}(\mathbf{x}) = \{\pi_1(\mathbf{x}), \pi_2(\mathbf{x}), \ldots, \pi_l(\mathbf{x})\}, \quad \sum_{i=1}^{l} \pi_i(\mathbf{x}) = 1, \tag{3.24}$$

where $\pi_i(\mathbf{x}) \geq 0$ is the probability that instance $\mathbf{x}$ is assigned class label y_i by an individual classifier in the ensemble. The values of these probabilities are unknown and depend on the ensemble learning algorithm, on the particular classification problem and on the current instance $\mathbf{x}$. The classification of this instance by an ensemble of arbitrary size t can be seen as a sequence of t independent trials, where each trial has a fixed number of possible outcomes in the set $\mathcal{Y}$. Given $\boldsymbol{\pi}(\mathbf{x})$, the probability distribution of the possible outcomes of these t experiments is a multinomial distribution with probability mass function

$$\mathcal{P}(\mathbf{t}|\boldsymbol{\pi}) = \frac{t!}{t_1!t_2!\ldots t_l!}\, \pi_1^{t_1}\pi_2^{t_2}\ldots\pi_l^{t_l}\,, \tag{3.25}$$

where $\mathbf{t} = \{t_1, t_2, \ldots, t_l; \sum_{i=1}^{l} t_i = t\}$, and t_i is the number of classifiers that predict class y_i for the current instance $\mathbf{x}$. The class predicted by the ensemble of size t for instance $\mathbf{x}$ is hence $y_{\hat{k}_\mathbf{t}}$ where

$$\hat{k}_\mathbf{t} = \arg\max_i \{t_i; i = 1, 2, \ldots, l\}\,. \tag{3.26}$$

To simplify the notation, the dependence on $\mathbf{x}$ of $\boldsymbol{\pi}$ is dropped from (3.25), (3.26) and the following expressions of this section.

Under the assumption that there is no information about the prior distribution of classes and that all possible values of the probability vector in (3.25) are equally likely, the prior for $\boldsymbol{\pi}$, $\mathcal{P}(\boldsymbol{\pi})$, is a uniform distribution. This prior can be used in Bayes' theorem to compute the posterior probability distribution of the probability vector $\boldsymbol{\pi}$, given the observed values of $\mathbf{t}$

$$\begin{aligned}
\mathcal{P}(\boldsymbol{\pi}|\mathbf{t}) &= \frac{\mathcal{P}(\mathbf{t}|\boldsymbol{\pi})\mathcal{P}(\boldsymbol{\pi})}{\mathcal{P}(\mathbf{t})} \\
&= \frac{\pi_1^{t_1}\pi_2^{t_2}\ldots\pi_l^{t_l}}{\int_\mathcal{S} \pi_1^{t_1}\pi_2^{t_2}\ldots\pi_l^{t_l}\, d\pi_1 d\pi_2 \ldots d\pi_l} \\
&= \frac{\Gamma(\sum_{i=1}^{l} t_i + l)}{\prod_{i=1}^{l}\Gamma(t_i+1)}\, \pi_1^{t_1}\pi_2^{t_2}\ldots\pi_l^{t_l}\,,
\end{aligned} \tag{3.27}$$

where the region of integration, $\mathcal{S}$, is the space of l-dimensional probability vectors and $\Gamma(z)$ is the gamma function (Abramowitz and Stegun, 1964). The posterior for $\boldsymbol{\pi}$ is a Dirichlet distribution of order l with parameters $(t_1+1, t_2+1, \ldots, t_l+1)$ (Bishop, 2006).

Our goal is to determine how the prediction of an ensemble of size T can be estimated with a certain level of confidence after only $t < T$ of its classifiers have been queried. Consider the vector of integers that summarizes the predictions of the complete ensemble of size T

$$\mathbf{T} \equiv \{T_1, T_2, \ldots, T_l; \sum_{i=1}^{l} T_i = T\}, \tag{3.28}$$

where T_i is the number of members in the ensemble that predict class i for the instance to be classified. Assuming that $\mathbf{t} \equiv \{t_1, t_2, \ldots, t_l; \sum_{i=1}^{l} t_i = t;\, t_i \leq T_i, i = 1, 2, \ldots, l\}$, the vector that stores the predictions of the first $t \leq T$ classifiers in the ensemble, is known, the probability distribution for $\mathbf{T}$ is

$$\begin{aligned} \mathcal{P}(\mathbf{T}|\mathbf{t}) &= \int_{\mathcal{S}} \mathcal{P}(\mathbf{T} - \mathbf{t}|\boldsymbol{\pi}) \mathcal{P}(\boldsymbol{\pi}|\mathbf{t})\, d\boldsymbol{\pi} \\ &= \frac{(T-t)!}{\prod_{i=1}^{l}(T_i - t_i)!} \frac{\prod_{i=1}^{l}(t_i+1)_{T_i - t_i}}{(t+l)_{T-t}}\,, \end{aligned} \tag{3.29}$$

where $\{T_i \geq t_i,\, i = 1, 2, \ldots, l\}$, and $(a)_n = a(a+1)\cdots(a+n-1)$ is the Pocchammer symbol, or rising factorial, with a and n nonnegative integers (Abramowitz and Stegun, 1964). Note that $\mathbf{T} - \mathbf{t}$ is independent of $\mathbf{t}$ given $\boldsymbol{\pi}$ because classifiers are built on independent executions of the learning algorithm.

Expression (3.29) describes a Polya urn model with $t_i + 1$ balls of color i, $i = 1, 2, \ldots, l$ and $c = 1$. Hence, there are a total of $t + l$ balls of l different colors. An experiment with repeated extractions from the urn is made. In each extraction a ball from the urn is selected at random and then returned to the urn together with c other balls of the same color. For $c = 1$ the probability of extracting $\mathbf{T} - \mathbf{t}$ balls of each color after $T - t$ extractions is precisely (3.29) (Johnson et al., 1997).

The probability that the class labels predicted by the subensemble of size $t < T$ and by the complete ensemble of size T coincide is

$$\tilde{\mathcal{P}}(\mathbf{t}, T) = {\sum_{\mathbf{T}}}^{*} \frac{(T-t)!}{\prod_{i=1}^{l}(T_i - t_i)!} \frac{\prod_{i=1}^{l}(t_i+1)_{T_i - t_i}}{(t+l)_{T-t}}\,, \tag{3.30}$$

where the asterisk indicates that the summation runs over all values of $\mathbf{T}$ such that $\sum_{i=1}^{l} T_i = T$, $\{T_i \geq t_i,\, i = 1, 2, \ldots, l\}$, and $T_{\hat{k}_\mathbf{t}} > T_j$ for $j \neq \hat{k}_\mathbf{t}$, where $y_{\hat{k}_\mathbf{t}}$ is the majority class after querying the first t classifiers. If we are willing to accept that the coincidence between these two predictions is not necessarily certain, but occurs with a high confidence level α, (3.30) can be used to determine the number of classifiers needed to estimate the prediction of the complete ensemble. In particular, the querying process can be halted after $t^\star$ classifiers have been queried, if the vector of class predictions of the current subensemble $\mathbf{t}^\star$ is such that $\tilde{\mathcal{P}}(\mathbf{t}^\star, T) \geq \alpha$.

For $T > l$ the cost of computing (3.30) is exponential in the number of different class labels l. In consequence, for typical ensemble sizes ($T \approx 100$), the evaluation becomes costly for problems with a large number of classes. However, in binary classification problems, expression (3.30) can be readily computed. For a fixed value of T the table for $\tilde{\mathcal{P}}(\mathbf{t}, T)$ has $\binom{T+3-1}{T} = (T+1)(T+2)/2$ elements. Namely, the number of different ways in which T objects can be assigned to 3 classes with repetitions allowed. The extra class corresponds to the predictions of the classifiers that have not been queried which are unknown at this point. For a fixed value of α, this table can be replaced by an equivalent table of size T, where the t-th entry of this table is $t^\star(t; T, \alpha)$, the minimum number of observations of the majority class label required to obtain the same prediction as the complete ensemble with a confidence level α, when t classifiers have been queried. Thus, when classifying an instance, polling can be stopped when t_{prune} classifiers have been queried and the number of classifiers that predict the majority class is $t^\star(t_{\text{prune}}; T, \alpha)$. Figure 3.7 displays the critical values $t^\star(t; T, \alpha)/t$ for an ensemble of $T = 101$ classifiers and a confidence level $\alpha = 99\%$. An ensemble with an odd number

of classifiers is used to avoid ties in the majority voting algorithm. In particular, it is necessary that the first 6 classifiers queried agree to have a probability above 99% that this classification is the same as the one given by the complete ensemble. The first few values of $t^\star(t;T,\alpha)/t$ are

$$6/6, 7/7, 8/8, 8/9, 9/10, 10/11, 10/12, 11/13, \ldots \tag{3.31}$$

The non-monotonic behavior observed in the graph arises from the fact that both $t^\star(t;T,\alpha)$ and t are integers. As the number of available observations t increases, the value of $t^\star(t;T,\alpha)$ approaches from above the limit for the complete ensemble $t^\star(t=101;T=101,\alpha)=51$.

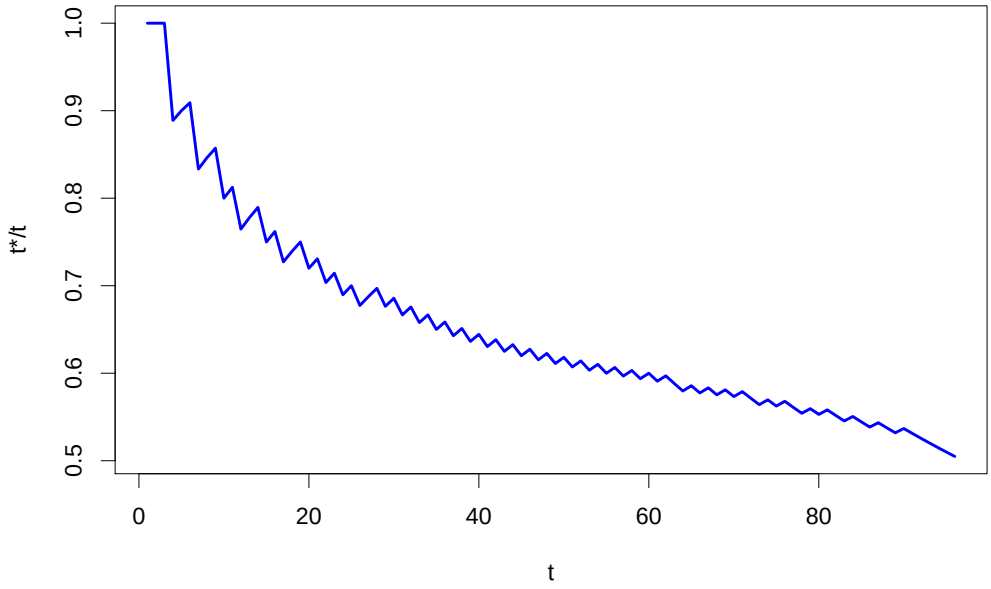

FIGURE 3.7: Minimum value of the fraction of classifiers that need to agree in their class predictions to guarantee that, for a binary classification problem, the majority class after polling t classifiers is the same as the one predicted by the complete ensemble of size $T=101$, with probability $\alpha=99\%$, as a function of t, the number of classifiers whose prediction is known.

Despite the fact that the computation of (3.30) can be costly for problems with a large number of class labels, it is a one-time computation. The calculation of $\tilde{\mathcal{P}}(\mathbf{t},T)$ relies solely on the statistical properties of the majority voting process. These probabilities do not depend on the details of the classification task to be solved or on the base classifiers considered. Given a value for T, they need to be calculated only once and then stored in a look-up table of size $\binom{T+l}{T}$. Then, they can be used for *any* classification task and any type of ensemble of independent classifiers. Similarly, for fixed values T and α, the table $t^\star(t;T,\alpha)$ is valid for any binary classification problem and ensemble and only has to be computed once. This means that the overhead of the instance-based ensemble pruning procedure is negligible because it requires only retrieving values stored in a precomputed look-up table. In consequence, the pruning rates obtained in the experiments directly translate into a speed-up of the classification process.

3.3.3 Experiments

A series of experiments is carried out to illustrate the theoretical analysis presented in the previous section. These experiments are also used to assess the efficiency of instance-based pruning in reducing the expected prediction time in a collection of benchmark classification problems.

A first batch of experiments is carried out in a simple binary classification task that can be analyzed in detail. The problem consists in predicting which points within the square $[-1,1]^2$ are also inside a circumference of diameter $\sqrt{2}$, centered at the origin. The Cartesian coordinates of the points $\mathbf{x} = (x_1, x_2)$ are used as attributes for prediction. Instances are sampled uniformly in the square $[-1,1]^2$. The class assigned to each instance is 1 if $x_1^2 + x_2^2 \geq 0.5$, 0 otherwise. To make the problem more realistic, the class labels in 5% of the training instances, selected at random, are changed. A random forest is built by learning $T = 101$ trees from a fixed set of 100 labeled instances (Breiman, 2001). The ensemble is tested on a set of $10,000$ points located on a regular grid within the square $[-1,1]^2$. For each instance, querying is stopped when the prediction of the subensemble of polled classifiers is expected to coincide with the prediction of the complete ensemble at a confidence level $\alpha = 99\%$. Figure 3.8 displays the results of these experiments, averaged over 200 independent realizations of the training set. In the graph on top, different shades of blue are used to denote the average fraction of trees that are needed to output a decision with the specified level of confidence for instances in that particular region of attribute space. Lighter shades of blue correspond to smaller subensembles. Regions where it is necessary to query all the classifiers in the original ensemble are drawn in blue. Contour curves are also displayed for reference. The optimal decision boundary of the classification problem is marked as a green circumference centered at the origin. We note that data instances that are far from the optimal decision boundary generally require querying fewer classifiers. By contrast, the amount of disagreement among classifiers is larger for data instances close to the decision boundary. In consequence, more queries are needed to produce a decision at the same confidence level. Nevertheless, even in this region, the classification of most of the instances requires knowing the prediction of only a fraction of the total ensemble members. The histogram on the bottom displays the average fraction of test examples whose class label can be predicted by querying a given number of classifiers in the ensemble. We note the bimodal form of the histogram, where the mode around 30 corresponds to instances close to the classification boundary. For this problem, on average, only 15% of the original classifiers need be polled to output a decision that coincides with the prediction of the complete ensemble with a probability above 99%.

The performance of instance-based pruning is further evaluated in a set of benchmark binary classification problems, including datasets from the UCI Repository (Asuncion and Newman, 2007), the synthetic problems *Twonorm*, *Ringnorm* and *Threenorm* from (Breiman, 1996b), *Task 1* from the 2008 UC San Diego Data Mining Contest (see http://mill.ucsd.edu/), which is a binary classification problem with a very unbalanced class distribution, and the *microarray* dataset described in (Hedenfalk et al., 2001), where the goal is to determine whether a particular patient carries the BRCA1 mutation or not. The Boston problem from UCI is transformed into a binary classification task by discriminating between houses worth more than $\$21,000$ and the rest. In the problem *Gisette* the 20 features that are correlated the most with the class label are used for prediction. The main features of the these datasets are displayed in Table 3.3. The results reported are averages over 100 realizations of each problem. These correspond to

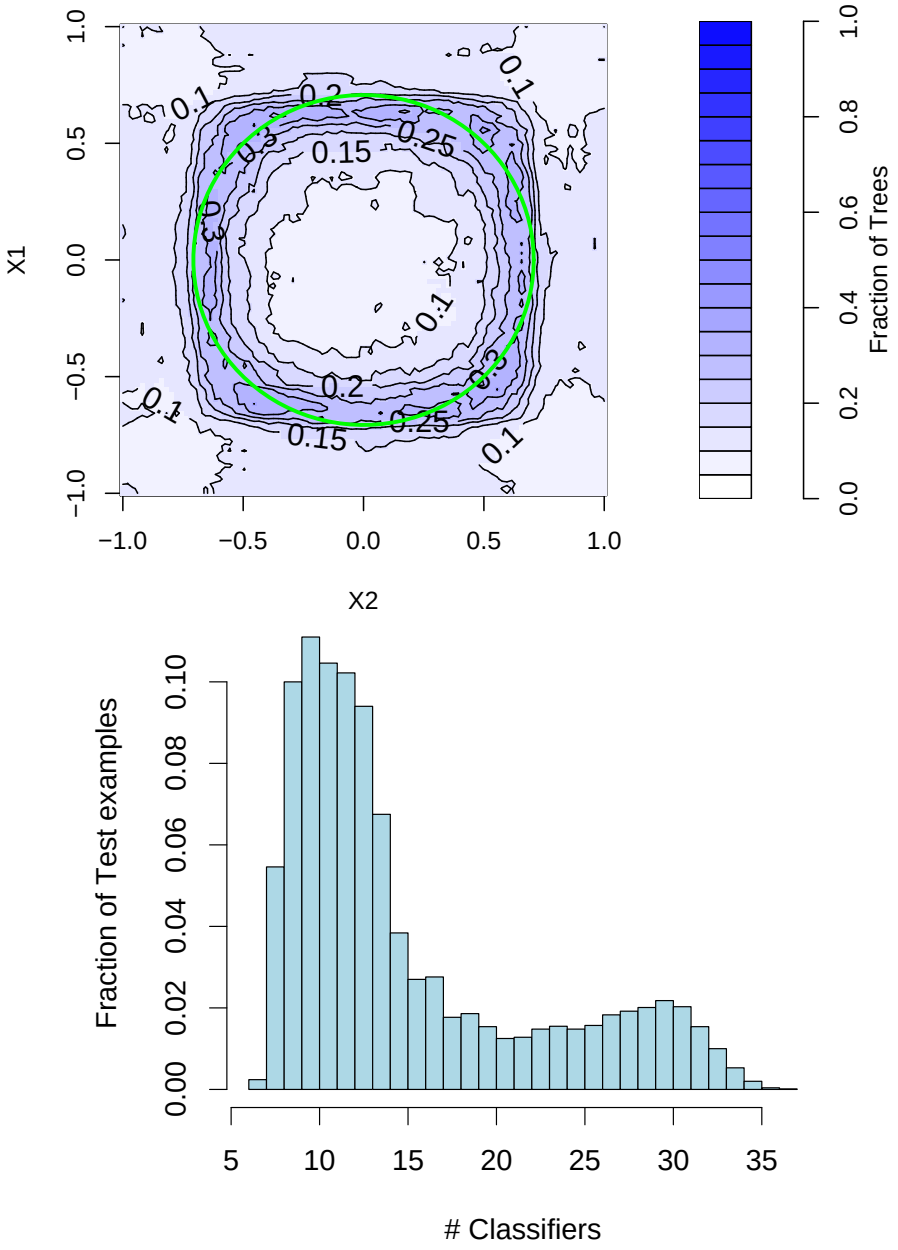

FIGURE 3.8: Average numbers of trees used by bagging with instance-based pruning ($\alpha = 99\%$) to estimate the class labels of test instances in the square $[-1, 1]^2$. In the graph at the top, different shades of blue are used to indicate the fraction of trees that are needed on average to predict an instance located in that region of attribute space. The histogram at the bottom displays the fraction of test instances whose class can be predicted querying a given number of classifiers in the ensemble.

either different simulations of the data in the problems in which a method to generate labeled instances is available (300 instances for training, 1,000 for testing) or to random partitions into train (2/3 of the available data) and test sets (the remaining 1/3) in the remaining sets. In each experiment a bagging ensemble of pruned CART trees (Breiman et al., 1984) and a random forest of 101 elements are built. The predictions of the complete ensembles are compared with the predictions of the corresponding ensembles pruned using instance-based (IB) pruning with $\alpha = 99\%$. Note that it is not necessary to query all the classifiers to determine the class prediction of the complete ensemble for a given instance with certainty. No further queries are required when the count for the majority class is larger than 50% of the number of classifiers in the original ensemble. Therefore, in an ensemble of size $T = 101$ polling can be stopped without loss of relevant information when 51 classifiers agree in their class prediction. This corresponds to IB-pruning with $\alpha = 100\%$, and provides a lower bound for the pruning rates that can be achieved.

TABLE 3.3: Datasets used in the experiments.

Problem	**Attributes**	**Instances**
Australian	14	690
Boston	13	506
Breast	9	699
Gisette	20	6,000
Heart	13	270
Hepatitis	19	155
Ionosphere	34	351
Liver	6	345
Magic	10	19,020
Microarray	3,226	22
Mushroom	20	8,124
Pima	8	768
Ringnorm	20	-
Sonar	60	208
Spam	57	4,601
Threenorm	20	-
Tic-tac-toe	9	958
Twonorm	20	-
Task 1	20	40,000
Votes	16	435

One can ask whether it is actually necessary to use a different stopping point for each instance or whether a fixed pruning rate is sufficient to achieve predictions that are close to those of the complete ensemble. To elucidate this question, the results obtained by an ensemble that queries a subset of fixed size, independently of the instance that is being classified, are reported as well. The number of classifiers queried in fixed-rate (FR) pruning is the ceiling of the average number of classifiers obtained by IB-pruning with $\alpha = 99\%$.

Table 3.4 shows the percentage of disagreement in the test set between the class prediction of the original complete ensembles and the pruned ones for bagging (columns 2 and 3) and for random forests (columns 4 and 5). The results are given in the format

average ± *standard deviation*. For IB-pruning, the average disagreement rates are in the vicinity of the expected value $100\% - \alpha = 1\%$, and generally below it. In contrast, when a fixed ensemble size is used (i.e., in FR-Bag and FR-RF), these rates need not be close to this target and can be rather large. The disagreement rates are in fact an upper bound for the differences in classification error. Even when the disagreement rates are large, similar error rates can be found if the changes in class label affect approximately the same numbers of correctly and incorrectly classified instances.

TABLE 3.4: Disagreement rates in % between complete and pruned ensembles.

	Bagging		**Random Forests**	
Problem	**IB-Bag**	**FR-Bag**	**IB-RF**	**FR-RF**
Australian	0.0 ± 0.0	0.4 ± 1.4	0.4 ± 0.4	4.3 ± 1.5
Boston	0.2 ± 0.4	3.0 ± 2.4	0.4 ± 0.5	4.9 ± 1.6
Breast	0.1 ± 0.2	1.2 ± 0.8	0.1 ± 0.2	1.3 ± 0.7
Gisette	0.1 ± 0.1	1.3 ± 0.4	0.1 ± 0.1	2.0 ± 0.4
Heart	0.3 ± 1.0	4.2 ± 3.1	0.6 ± 0.8	5.3 ± 2.3
Hepatitis	0.0 ± 0.2	1.2 ± 2.9	0.4 ± 0.8	5.2 ± 3.1
Ionosphere	0.1 ± 0.5	2.2 ± 2.0	0.1 ± 0.3	2.5 ± 1.4
Liver	0.7 ± 1.4	6.0 ± 4.0	1.0 ± 1.0	7.9 ± 2.8
Magic	0.1 ± 0.1	2.2 ± 0.3	0.3 ± 0.1	3.9 ± 0.3
Microarray	0.2 ± 1.8	2.1 ± 8.0	0.8 ± 3.0	6.6 ± 9.1
Mushroom	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Pima	0.3 ± 0.6	3.3 ± 2.1	0.6 ± 0.5	6.0 ± 1.7
Ringnorm	0.5 ± 0.3	5.7 ± 1.2	0.4 ± 0.2	4.9 ± 0.9
Sonar	0.2 ± 0.6	4.3 ± 3.4	0.9 ± 1.3	7.9 ± 3.2
Spam	0.1 ± 0.1	1.7 ± 0.4	0.2 ± 0.1	2.3 ± 0.4
Threenorm	1.0 ± 0.5	7.6 ± 1.5	1.0 ± 0.3	7.9 ± 1.0
Tic-tac-toe	0.1 ± 0.3	2.2 ± 0.9	0.3 ± 0.3	3.3 ± 1.2
Twonorm	0.6 ± 0.3	5.4 ± 1.0	0.4 ± 0.2	4.3 ± 0.7
Task 1	0.0 ± 0.0	1.1 ± 0.2	0.1 ± 0.0	1.9 ± 0.1
Votes	0.0 ± 0.1	0.3 ± 0.7	0.1 ± 0.2	1.5 ± 1.2

Tables 3.5 and 3.6 display the results of experiments for the error rate and size of the pruned ensembles in bagging and random forests, respectively. Columns 2 to 4 of these tables display the test error of the complete ensembles, of ensembles pruned with IB-pruning using $\alpha = 99\%$ (*IB-Bag* in Table 3.5 and *IB-RF* in Table 3.6) and of ensembles pruned with a fixed rate (*FR-Bag* in Table 3.5 and *FR-RF* in Table 3.6), respectively. These results confirm that random forests generally outperform bagging (Geurts et al., 2006; Rodríguez et al., 2006). The differences in performance with respect to the original ensemble when instance-based pruning is used are fairly small. The error rates for fixed-rate pruning are generally larger, especially in random forests.

The average number of trees that need to be queried to determine the complete ensemble prediction at a confidence level α is tabulated in the fifth and sixth columns of Tables 3.5 and 3.6. Column 5 corresponds to IB-pruned ensembles with $\alpha = 100\%$, which means that these predictions coincide with those of the complete ensemble. Column 6 corresponds to instance-based pruning with $\alpha = 99\%$ (IB-Bag and IB-FR). Since the overhead to determine when to stop querying is negligible (one simply needs to compare the majority class count with a critical value, stored in a precalculated look-up table),

TABLE 3.5: Error rates and number of trees in pruned ensembles for bagging.

	Test error			Number of trees		speed-up
Problem	**Bagging**	**IB-Bag**	**FR-Bag**	$\alpha = 100\%$	$\alpha = 99\%$	**factor**
Australian	14.5±1.9	14.5±1.9	14.5±2.0	52.5±1.1	7.0±1.5	7.7±1.0
Boston	17.1±3.0	17.1±3.0	17.1±2.9	57.4±1.5	12.4±2.3	4.8±0.8
Breast	5.0±1.5	5.0±1.5	5.1±1.6	53.9±0.5	8.6±0.8	6.3±0.5
Gisette	9.1±0.6	9.1±0.6	9.2±0.7	53.5±0.2	8.4±0.4	6.4±0.3
Heart	20.8±4.7	20.8±4.6	21.2±4.5	63.6±2.3	19.1±3.9	3.5±0.7
Hepatitis	21.2±4.4	21.3±4.4	21.1±4.6	55.2±2.0	9.1±3.0	6.5±1.5
Ionosphere	9.0±2.8	9.0±2.9	9.5±3.0	56.1±1.4	10.5±2.1	5.5±1.0
Liver	33.9±5.6	34.0±5.7	34.6±5.9	68.4±3.1	23.2±5.6	3.1±0.8
Magic	14.3±0.5	14.3±0.5	14.5±0.5	55.6±0.2	10.5±0.3	5.3±0.1
Microarray	30.2±11.3	30.2±11.4	30.0±12.0	60.0±6.4	13.2±9.4	6.0±2.4
Mushroom	0.0±0.1	0.0±0.1	0.0±0.1	51.0±0.0	6.0±0.0	8.5±0.0
Pima	25.4±2.6	25.4±2.7	25.7±2.6	58.9±1.6	13.5±2.4	4.5±0.9
Ringnorm	12.2±3.0	12.3±3.0	13.3±2.8	68.3±1.3	23.7±1.8	2.9±0.2
Sonar	25.5±4.4	25.5±4.4	25.8±4.2	65.2±3.9	19.2±5.5	3.6±1.0
Spam	7.6±0.7	7.6±0.7	7.9±0.7	54.6±0.2	9.4±0.4	5.8±0.2
Threenorm	23.9±3.4	24.1±3.3	24.9±3.2	75.1±1.6	32.9±2.9	2.3±0.2
Tic-tac-toe	2.6±1.1	2.7±1.1	3.9±1.3	57.8±0.6	11.6±1.0	5.0±0.4
Twonorm	9.8±2.9	10.0±2.9	11.3±2.7	69.0±0.6	24.2±1.2	2.9±0.1
Task 1	6.1±0.2	6.1±0.2	6.2±0.2	52.8±0.1	7.7±0.1	6.8±0.1
Votes	4.6±1.3	4.6±1.3	4.7±1.4	51.7±0.3	6.5±0.5	8.0±0.5

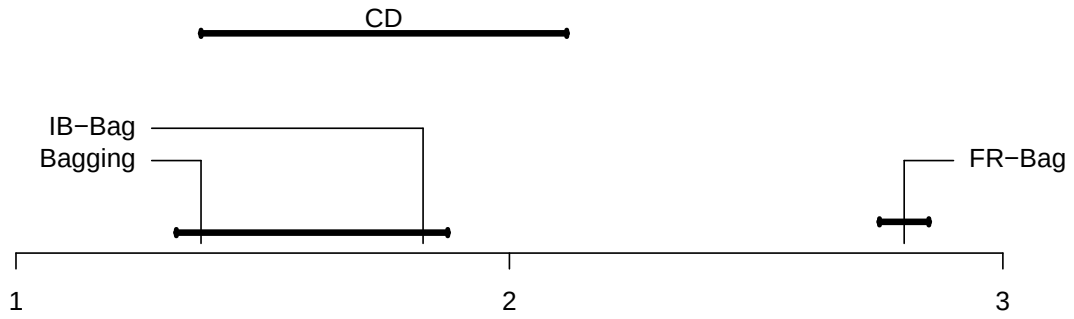

FIGURE 3.9: Average ranks in classification performance for complete bagging, bagging ensembles pruned using IB-pruning with $\alpha = 99\%$ (IB-Bag), and bagging ensembles pruned with a fixed rate (FR-Bag). The significance of the differences in average rank of the different ensembles is determined using a Nemenyi test. The critical value for the differences (CD) is given by the length of the segment plotted at the top.

TABLE 3.6: Error rates and number of trees in pruned ensembles for random forests.

	Test error			Number of trees		speed-up
Problem	**RF**	**IB-RF**	**FR-RF**	$\alpha = 100\%$	$\alpha = 99\%$	**factor**
Australian	13.2±1.9	13.3±1.9	13.7±1.7	63.1±0.9	17.4±1.6	3.7±0.3
Boston	13.4±2.2	13.4±2.2	14.0±2.5	62.6±1.0	17.9±1.7	3.5±0.3
Breast	3.2±0.9	3.2±0.9	3.9±1.1	54.3±0.5	8.9±0.7	6.1±0.4
Gisette	7.8±0.5	7.8±0.5	8.2±0.5	55.8±0.2	10.3±0.3	5.4±0.2
Heart	17.7±3.5	17.7±3.6	18.5±3.7	67.5±1.4	22.5±2.6	3.0±0.3
Hepatitis	15.9±4.3	16.0±4.3	16.2±4.2	64.0±2.2	19.6±3.6	3.4±0.6
Ionosphere	6.8±2.1	6.8±2.1	7.5±2.3	58.8±1.0	12.6±1.7	4.7±0.6
Liver	28.7±3.7	28.8±3.6	30.2±3.7	76.0±1.5	34.6±3.0	2.2±0.2
Magic	12.2±0.4	12.2±0.4	12.9±0.4	61.7±0.2	15.9±0.3	3.9±0.1
Microarray	26.8±13.6	26.5±14.1	26.9±14.4	74.6±4.8	31.1±10.7	2.7±0.9
Mushroom	0.0±0.0	0.0±0.0	0.0±0.0	51.1±0.0	6.1±0.0	8.4±0.0
Pima	24.3±2.0	24.3±2.1	25.0±2.2	69.5±0.9	25.4±1.9	2.7±0.2
Ringnorm	6.7±1.1	6.8±1.1	8.5±1.2	68.7±0.9	22.9±1.2	3.0±0.1
Sonar	18.8±4.5	19.0±4.5	19.6±4.8	74.4±1.4	32.7±3.3	2.3±0.2
Spam	5.1±0.5	5.1±0.5	5.7±0.6	57.4±0.3	11.5±0.4	5.0±0.2
Threenorm	17.2±1.5	17.3±1.5	18.9±1.6	76.9±0.7	35.4±1.3	2.2±0.1
Tic-tac-toe	2.4±0.9	2.6±0.9	4.5±1.4	64.9±0.6	18.0±1.3	3.6±0.2
Twonorm	4.4±0.8	4.5±0.8	6.4±0.8	67.2±0.4	20.9±0.7	3.2±0.1
Task 1	5.3±0.2	5.3±0.2	5.6±0.2	55.6±0.1	10.1±0.1	5.5±0.1
Votes	3.8±1.4	3.9±1.3	4.4±1.5	55.1±0.8	9.3±1.1	6.0±0.6

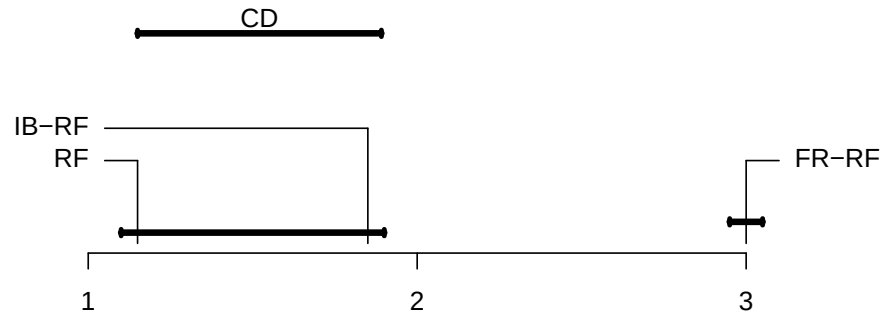

FIGURE 3.10: Average ranks in classification performance for random forests (RF), random forests pruned using IB-pruning with $\alpha = 99\%$ (IB-RF), and random forests pruned with a fixed rate (FR-RF). The significance of the differences in average rank of the different ensembles is determined using a Nemenyi test. The critical value for the differences (CD) is given by the length of the segment plotted at the top.

the reduction in ensemble size by pruning directly entails a reduction in the time needed to classify a given instance. Column 7 displays the factor by which the classification time is reduced when statistical instance-based pruning is used. This speed-up factor in classification varies between ≈ 2 and ≈ 8.5, depending on the problem considered. Problems that are simple for classification trees, such as the *mushroom* problem, need on average very few classifiers to achieve good predictions. In contrast, for *threenorm*, which is a difficult task for classification trees, a large number of trees is required to reach a decision. In general, random forests need to query more trees than bagging. This is a consequence of the larger variability introduced by the random forests algorithm when building the individual classifiers. In consequence, the ensemble members in random forests tend to be more diverse than those generated in bagging and typically agree less frequently in their predictions.

The performance of the different ensembles is compared using the statistical methodology proposed by Demšar (2006). A Friedman test provides evidence of statistically significant differences in performance with a significance level of 5% both for bagging and for random forests. Then, a Nemenyi test is carried out to compare the performance of the different ensembles in terms of their average ranks in the classification problems investigated. The results of this test for a significance level of 5% are displayed in Figure 3.9 for bagging and on Figure 3.10 for random forests. These figures show that ensembles pruned using a fixed rate perform significantly worse than those obtained by instance-based pruning. By contrast, the empirical evidence is insufficient to determine whether the differences in performance between the complete ensembles and the corresponding subensembles selected with instance-based pruning are statistically significant. These results underscore the importance of determining the appropriate pruning rate for each specific instance.

3.4 Conclusions

Ensemble methods have the drawback that they generally require the combination of a large number of predictors to guarantee convergence of the ensemble error. Therefore, these methods demand large memory resources to store the ensemble members. They can also take long time to yield class predictions for unlabeled instances. A possible solution to these problems is to use ensemble pruning methods. These techniques replace the original ensemble by a representative subset of predictors whose combined performance is equivalent (and sometimes better) than the performance of the complete original ensemble.

In this chapter different ensemble pruning methods have been analyzed. The first two methods can be used in regression ensembles: SDP-pruning and ordered aggregation. Both SDP-pruning and ordered aggregation give approximate solutions for the problem of selecting an optimal subensemble from a regression bagging ensemble. This is a difficult problem that is shown to be NP-hard. The approximation obtained by SDP-pruning is based on a semidefinite programming relaxation of the original subensemble selection problem. This method is an extension to regression ensembles of the work of Zhang et al. (2006), focused on classification ensembles. On the other hand, ordered aggregation follows a greedy approach. Starting with an empty subensemble, ordered aggregation iteratively incorporates into the current subensemble the predictor that reduces the training error of the enlarged subensemble the most. Even though these approximate strategies need not give optimal solutions, a detailed analysis in ensembles of intermediate size shows that the subensembles selected have a near-optimal

performance. Furthermore, they share a large fraction of predictors with the optimal subensembles found by exhaustive search.

The error of standard bagging ensembles typically decreases monotonically as the size of the ensemble increases. By contrast, for the subensembles selected by SDP-pruning or ordered aggregation, the curves that trace the dependence of the subensemble error as a function of its size show that the minimum error is attained in subensembles of intermediate size. The features of these curves are qualitatively similar for both the training and testing errors. However, the minimum in the test error curves appears at larger sizes than in the curves for the training error. This minimum is generally below the asymptotic error of the complete bagging ensemble. This means that generalization performance of the ensemble can be improved by retaining only a subset of the predictors generated.

An extensive empirical investigation shows that pruning a regression bagging ensemble by retaining only 20% of the original networks in the ensemble, either by SDP-pruning or ordered aggregation, generally leads to improvements in the prediction accuracy. Pruned ensembles also have a better overall performance than a single neural network or ensembles generated with the Adaboost.R2 algorithm, an extension of boosting for regression (Drucker, 1997). A bias-variance-covariance decomposition of the test error shows that the key to the improvements in generalization performance is the selection of subsets of predictors whose bias is low and whose correlations are small.

The third ensemble pruning method analyzed in this chapter is instance-based (IB) pruning. This method can be used to speed-up the prediction process of parallel classification ensembles that use majority voting for output combination and whose elements are generated independently when conditioned to the training data. At each intermediate step of the majority voting process, IB-pruning uses Bayes' theorem to update an estiamte of the probability that the complete ensemble eventually predicts a given class label. For this, the evidence available from the ouputs of the classifiers that have already been queried is used. The voting process is stopped when the evidence that the final ensemble prediction will coincide with the current majority class is sufficiently large. The number of queries required to stop depends on the instance that is being classified. For some instances for which the ensemble members tend to agree, only a few classifiers need to be queried to gather sufficient evidence about the final prediction. Other instances, typically those that are close to the classification boundary, require a larger number of classifiers. Given that the time needed to determine when to stop querying is negligible in IB-pruning, stopping this process before all classifier outputs are computed directly translates into an improvement of the classification time.

Unlike ordered aggregation or SDP-pruning, IB-pruning is not designed to improve the generalization performance of the original ensemble. However, an empirical investigation using two parallel ensemble methods, bagging and random forests, demonstrates that IB-pruning reduces the number of classifiers that need to be queried without a significant deterioration in the generalization performance of the ensemble. In particular, the average number of classifiers needed for classification is substantially reduced in all the problems investigated. Additionaly, the generalization performance of IB-pruned ensembles is comparable to the corresponding complete ensembles.

In the next chapter the statistical analysis of majority voting used in IB-pruning will be used to estimate the appropiate size of parallel classification ensembles.

Chapter 4

Optimal Size of Parallel Ensembles

We propose to determine the optimal size of a parallel ensemble by estimating the minimum number of classifiers that are required to obtain stable aggregate predictions. Assuming that majority voting is used, a statistical description of the convergence of the ensemble prediction to its asymptotic (infinite size) limit is given. The analysis of the voting process shows that for most test instances the ensemble prediction stabilizes after only a few classifiers are polled. By contrast, a small but non-negligible fraction of these instances requires a large number of classifier queries to reach a stable prediction. The optimal ensemble size is determined as the minimum number of classifiers that are needed, on average, to estimate the infinite ensemble prediction at a given confidence level, α. This approach differs from previous proposals, which are based on determining the size for which the prediction error stabilizes. Specifically, it does not require estimates of the generalization performance of the ensemble, which can be unreliable. Its validity is very general because it is based solely on the statistical convergence of majority voting. Extensive experiments using representative parallel ensembles (bagging and *random forests*) illustrate the application of the proposed framework in a wide range of classification problems.

4.1 Introduction

The use of ensembles in classification problems has been the object of numerous investigations (Banfield et al., 2007; Breiman, 1996a, 2001; Bühlmann, 2003; Dietterich, 2000b; Martínez-Muñoz and Suárez, 2005; Rodríguez et al., 2006). These studies show that combining the decisions of complementary classifiers is an effective mechanism that improves the generalization performance of a single predictor. In this chapter we propose a method to determine the optimal size of parallel ensembles composed of classifiers of the same type. The procedure is valid for all classification tasks and for ensembles composed of any kind of base learners: decision trees, decision stumps, neural networks, support vector machines, etc. The only assumption is that the classifiers that make up the ensemble are generated independently when conditioned to the training data, and that the final prediction is made using simple majority voting. Examples of ensembles of this type are bagging (Breiman, 1996a), variants of bagging (Bühlmann, 2003),

random forests (Breiman, 2001), *class-switching* ensembles (Breiman, 2000; Martínez-Muñoz and Suárez, 2005), *rotation forest* (Rodríguez et al., 2006), *extra-trees* (Geurts et al., 2006), etc.

Extensive empirical evidence shows that in many classification problems the generalization error of parallel ensembles decreases monotonically as the size of the ensemble increases (Martínez-Muñoz and Suárez, 2005; Opitz and Maclin, 1999; Schapire et al., 1998). However, the gains that can be achieved by incorporating additional classifiers become progressively smaller as the ensemble grows. Therefore, it is reasonable to stop aggregating classifiers when the probability of changes in the ensemble output that arise from considering additional predictions is below a given threshold. In this work we use the statistical description of the evolution of the class prediction by majority voting as the number of classifiers in the ensemble increases to determine when a sufficient number of classifiers have been included in the ensemble (Esposito and Saitta, 2003; Hansen and Salamon, 1990; Kuncheva et al., 2003; Lam and Suen, 1997; Narasimhamurthy, 2005; Ruta and Gabrys, 2002). This analysis shows that for most test instances only a small number of queries are needed to obtain a class label prediction that coincides with the asymptotic classification with a high confidence level. By contrast, a small number of these instances require polling exceedingly large numbers of classifiers to reach a stable prediction. Therefore, it is not possible to determine a fixed size for the ensemble so that the finite ensemble prediction coincides with the asymptotic (infinite ensemble) prediction for *every* instance with a specified probability.

Instead of enforcing convergence guarantees for every test instance, we propose to determine the optimal ensemble size by requiring that on average, the predictions coincide with the infinite ensemble classification with a probability of at least α. The value of α is fixed by the user depending on the desired level of confidence in the stability of the predictions. Larger values of α require larger ensembles. In general, this value should be close to but smaller than 1, e.g. $\alpha = 99\%$. Thus, the optimal ensemble size is determined as the minimum size required for convergence to the asymptotic limit with the specified confidence level α. The differences in error between the finite and the infinite ensemble are bounded from above by $100\% - \alpha$, which is the probability that the predictions of the finite and of the infinite ensembles differ. In practice, these discrepancies occur with approximately equal frequency in correctly and incorrectly classified instances. Therefore, the differences in performance between the finite and the infinite ensembles are generally much smaller than this bound.

The main contribution of this research is the use of the convergence properties of majority voting to address the problem of determining the optimal size of the ensemble: The aggregation of classifiers in the ensemble is stopped when the class label predicted is not expected to change by performing further queries. Once the predictions of the ensemble on the test instances stabilize, relevant measures of performance, such as generalization error, the confusion matrix or the margin, also become stable. The advantage of the strategy proposed is its general validity, because it relies solely on the statistical properties of majority voting. Specifically, to determine the convergence of the ensemble prediction, it is not necessary to know the true class labels of the instances. In most previous investigations the optimal ensemble size is determined by aggregating classifiers until an estimate of the generalization error stabilizes with a specified degree of accuracy (Banfield et al., 2007; Fumera et al., 2008; Latinne et al., 2001). Methods based on the convergence of the error require the design of reliable estimators of the generalization performance and could be affected by over-fitting.

One of the earlier proposals to select the optimal ensemble size is based on estimating minimum number of classifiers that are needed to obtain a prediction accuracy similar to a larger ensemble (Latinne et al., 2001). In this work, the McNemar non-parametric test is used to determine whether the differences between the predictions on a validation set of ensembles of sizes T and t with $t < T$ are statistically significant. The size of the optimal ensemble is set to $t^\star$, which is the minimum size of a subensemble whose performance does not significantly differ from an ensemble of T classifiers, with T sufficiently large. In (Banfield et al., 2007), the out-of-bag data (Breiman, 1996c) are used to estimate the generalization error. First, the dependence of the out-of-bag error estimate on the size of the ensemble is smoothed by averaging over a sliding window of size 5. Then, the algorithm identifies the ensemble that has the best accuracy (i.e., the lowest smoothed value the error estimated on the out-of-bag data) among ensembles of sizes 1 to 20. Progressively larger ensembles are processed in batches of 20, until no improvement in accuracy is found. At this point, the algorithm returns the ensemble with the maximum accuracy. The major advantage of this approach is that it does not require to overproduce and then discard classifiers. Finally, the theoretical analysis of the dependence of the generalization error on the size of the ensemble size performed in (Fumera et al., 2008) can be used to select ensembles of optimal size. Instead of majority voting, this analysis assumes that the individual classifiers output a probability level and that the global prediction is obtained by a linear combination of these probabilities.

The organization of the chapter is as follows: in Section 4.2 we derive the probabilistic framework for the estimation of the optimal ensemble size. Then, in Section 4.3 we carry out some experiments for the validation of the proposed method. Finally, Section 4.4 summarizes the conclusions of this chapter.

4.2 Estimation of the Optimal Ensemble Size

Consider an ensemble composed of T classifiers $\hat{f}_1, \ldots, \hat{f}_T$. As described in Section 3.3.2, when majority voting is used to combine the decisions of the individual classifiers, the global ensemble prediction for an unlabeled instance $\mathbf{x}$ is

$$\hat{y}^T = \arg\max_y \sum_{i=1}^{T} \mathbb{I}(\hat{f}_i(\mathbf{x}) = y),\ y \in \mathcal{Y}\,, \tag{4.1}$$

where $\mathbb{I}(z)$ is an indicator function that takes value one when z is satisfied and zero otherwise, and $\mathcal{Y} = \{y_1, \ldots, y_l\}$ is the set of possible class labels.

If the individual classifiers in the ensemble are built independently when conditioned to the training data [1], the polling process defined in (4.1) can be described as a sequence of T independent trials, where the outcome of each trial is in the set $\mathcal{Y}$. Under these conditions, the distribution of class votes is multinomial

$$\mathcal{P}(\mathbf{t}|T, \boldsymbol{\pi}(\mathbf{x})) = \frac{T!}{t_1!, \ldots t_l!} \pi(\mathbf{x})_1^{t_1} \ldots \pi(\mathbf{x})_l^{t_l}\,, \tag{4.2}$$

where t_i is the number of classifiers that predict class label y_i, $\mathbf{t} = (t_1, t_2, \ldots, t_l)$, with $\sum_{i=1}^{l} t_i = T$, and $\boldsymbol{\pi}(\mathbf{x})$ is a probability vector that determines the probability that an ensemble classifier assigns each different class label to the instance characterized by the

[1] Note that this is different from assuming that the classifiers are unconditionally independent, which is not typically the case in classification ensembles.

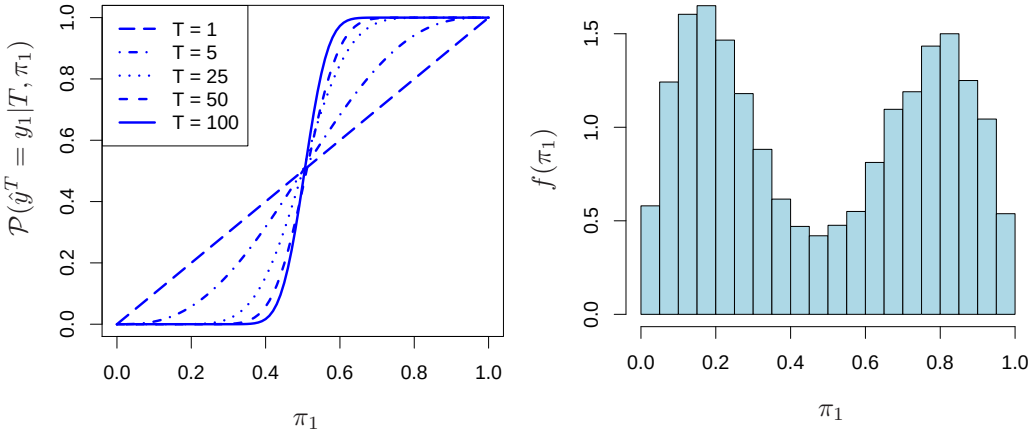

FIGURE 4.1: (left) Probability that an individual ensemble classifier predicts class y_1 as a function of π_1 for different values of T. (right) Histogram of 10,000 samples from the probability distribution of π_1, denoted $f(\pi_1)$, for the *Twonorm* binary classification problem. The estimates are obtained using a random forest (RF) composed of 10,000 trees. This ensemble is built on a fixed training set with 300 labeled instances.

attribute vector $\mathbf{x}$. Recall that the values of these probabilities are in general unknown and they depend on the algorithm used to build the base learners, on the particular classification problem and on the current instance $\mathbf{x}$. To simplify the notation, the dependence on $\mathbf{x}$ of the probability vector $\boldsymbol{\pi}$ and of the vector of votes $\boldsymbol{t}$ is removed from these and the following expressions.

Given $\boldsymbol{\pi}$, the probability that an ensemble of size T assigns class label y_i to instance $\mathbf{x}$ is the sum of (4.2) over all ensemble predictions in which class y_i receives more votes than any other class. For binary classification problems (i.e. $l = 2$) this probability is

$$\mathcal{P}(\hat{y}^T = y_1|T, \pi_1) = \sum_{t_1=\lceil \frac{T}{2} \rceil}^{T} \binom{T}{t_1} \pi_1^{t_1} (1-\pi_1)^{T-t_1} = I_{\pi_1}\left(\lfloor \frac{T}{2} \rfloor + 1, T - \lfloor \frac{T}{2} \rfloor\right), \quad (4.3)$$

where $I_x(a, b)$ is the regularized incomplete beta function (Abramowitz and Stegun, 1964).

Figure 4.1 (left) displays, for different values of the ensemble size T, the dependence of the probability that the ensemble predicts class y_1 (4.3) as a function of π_1, the probability that an individual classifier predicts class y_1. Note that for $T = 1$, (4.3) is simply the identity function. As T grows, (4.3) asymptotically approaches a step function

$$\lim_{T\to\infty} \mathcal{P}(\hat{y}^T = y_1|T, \pi_1) = \begin{cases} 1 & \text{if } \pi_1 > 1/2, \\ 1/2 & \text{if } \pi_1 = 1/2, \\ 0 & \text{if } \pi_1 < 1/2. \end{cases} \quad (4.4)$$

Figure 4.1 (right) displays a histogram of 10,000 samples from the probability density function $f(\pi_1)$ of the π_1 probability values for the *Twonorm* classification problem (Breiman, 1998). The estimations are performed using random forests (RF) of 10,000 trees (Breiman, 2001). The individual decision trees are built using a fixed training set of 300 instances. The estimations of π_1 are made on an independent test set of 10,000 instances. For each test instance the value of π_1 is estimated as the fraction of classifiers that predict class label y_1. The probability density function estimated is bimodal.

This means for some instances the classifiers tend to predict class y_1 more often and for other instances the prediction class y_2 is more frequent. This bimodality should be expected, because there are instances of both classes in approximately equal numbers. There are also some data instances for which approximately half of the predictions are class y_1 and the rest class y_2. These instances are located near the decision boundaries of the classifiers. The corresponding values of π_1 for these instances are in the vicinity of $1/2$. Therefore, more classifiers need to be queried for convergence to a stable ensemble prediction. It is important that these estimations be made on a set that is independent of the training data: the estimate of the probability density using the training set, $f_{\text{train}}(\pi_1)$, is biased because classifiers tend to agree more frequently on these instances. Therefore, a smaller fraction of training instances whose probability π_1 is close to $1/2$ is expected. Furthermore, the modes of $f_{\text{train}}(\pi_1)$ are closer to the extreme values $\pi_1 = 0$ and $\pi_1 = 1$. As a result of this bias, the size of the ensemble required to obtain stable predictions is smaller for the training set than for an independent test set.

In practice, it is not possible to query an infinite number of classifiers to obtain the asymptotic ensemble prediction. However, assuming that the value of π_1 for the instance to be classified is known or can be estimated in some way, the probability that an ensemble of size T assigns the same class label as the infinite ensemble is

$$\mathcal{P}(\hat{y}^T = \hat{y}^\infty | T, \pi_1) = I_{\max\{\pi_1, 1-\pi_1\}} \left(\lfloor \frac{T}{2} \rfloor + 1, T - \lfloor \frac{T}{2} \rfloor \right) . \tag{4.5}$$

Using this expression we can compute $T^\star(\alpha, \pi_1)$, the minimum ensemble size whose prediction for the instance characterized by the probability π_1 coincides with the infinite ensemble prediction with a confidence level α, by finding the minimum value of T that satisfies the inequality

$$\alpha \leq I_{\max\{\pi_1, 1-\pi_1\}} \left(\lfloor \frac{T}{2} \rfloor + 1, T - \lfloor \frac{T}{2} \rfloor \right) . \tag{4.6}$$

It is not possible to derive an explicit exact formula for $T^\star(\alpha, \pi_1)$. Nevertheless, this quantity can be readily calculated using numerical algorithms. If only odd values of T are considered, the right-hand size of (4.6) grows monotonically with T. The analysis is restricted to odd sizes to avoid ties in the ensemble prediction. Therefore, a simple binary search can be used to compute $T^\star(\alpha, \pi_1)$, given α and π_1.

For values of π_1 close to $1/2$ a closed-form approximation for $T^\star(\alpha, \pi_1)$ can be obtained. In this regime, $T^\star(\alpha, \pi_1)$ is a large number. Therefore, the binomial distribution in (4.3) can be approximated by a Gaussian distribution with the same mean and variance

$$\mathcal{P}(\hat{y}^T = \hat{y}^\infty | T, \pi_1) \approx \Phi \left(\frac{T/2 - T\max\{\pi_1, 1-\pi_1\}}{\sqrt{T\pi_1(1-\pi_1)}} \right) , \tag{4.7}$$

where $\Phi(\cdot)$ is the cumulative distribution function of a standard Gaussian distribution. With this replacement (4.6) can be approximated by

$$T^\star(\alpha, \pi_1) \approx \frac{\Phi^{-1}(\alpha)^2 \pi_1(1-\pi_1)}{(\pi_1 - 1/2)^2} , \tag{4.8}$$

where $\Phi^{-1}(\cdot)$ is the quantile function of a standard Gaussian distribution. This approximation to $T^\star(\alpha, \pi_1)$ is accurate for values of π_1 near $1/2$. For a fixed value of α, expression (4.8) shows that $T^\star(\alpha, \pi_1)$ becomes infinite in the limit $\pi_1 \to 1/2$. Therefore,

the factor that determines the ensemble size is the presence of instances for which the classification probability by a single ensemble member π_1 is close to $1/2$. For these instances, a very large number of classifiers needs to be queried to produce a reliable estimate of the infinite ensemble prediction at the specified confidence level (α).

Since different examples have different values of π_1, this quantity is a random variable whose probability density function is $f(\pi_1)$ (see the right-hand-side of Figure 4.1). Thus, $T^\star(\alpha, \pi_1)$ is also a random variable because it depends on π_1. Let $\mathcal{P}(T^\star(\alpha, \pi_1) > T)$ be the probability that the minimum number of queries required for convergence of the ensemble prediction is above threshold T when π_1 follows a distribution $f(\pi_1)$. In the limit $T \to \infty$, this probability tends to

$$\mathcal{P}(T^\star(\alpha, \pi_1) > T) \sim \frac{f(\frac{1}{2})\Phi^{-1}(\alpha)}{\sqrt{T}}\,. \tag{4.9}$$

where the density function of π_1 evaluated at $\pi_1 = 1/2$ is assumed to be positive $f(\pi_1 = 1/2) > 0$. To see this, let $F(\pi_1)$ be the cumulative distribution function of π_1 (see the right-hand-side of Figure 4.1). In terms of this distribution, $\mathcal{P}(T^\star(\alpha, \pi_1) > T)$ can be estimated as the fraction of the instances whose value of π_1 is in the interval

$$\left[I^{-1}_{1-\alpha}\left(\lfloor\frac{T}{2}\rfloor + 1, T - \lfloor\frac{T}{2}\rfloor\right), I^{-1}_{\alpha}\left(\lfloor\frac{T}{2}\rfloor + 1, T - \lfloor\frac{T}{2}\rfloor\right)\right], \tag{4.10}$$

where $I^{-1}_x(a, b)$ is the inverse of the regularized incomplete beta function. That is,

$$\mathcal{P}(T^\star(\alpha, \pi_1) > T) = F\left(I^{-1}_{\alpha}(\lfloor\frac{T}{2}\rfloor + 1, T - \lfloor\frac{T}{2}\rfloor)\right) - F\left(I^{-1}_{1-\alpha}(\lfloor\frac{T}{2}\rfloor + 1, T - \lfloor\frac{T}{2}\rfloor)\right). \tag{4.11}$$

Taking the limit $T \to \infty$

$$\begin{aligned}
\mathcal{P}(T^\star(\alpha, \pi_1) > T) &\approx f(\frac{1}{2})\left(I^{-1}_{\alpha}(\lfloor\frac{T}{2}\rfloor + 1, T - \lfloor\frac{T}{2}\rfloor) - I^{-1}_{1-\alpha}(\lfloor\frac{T}{2}\rfloor + 1, T - \lfloor\frac{T}{2}\rfloor)\right) \\
&\approx f(\frac{1}{2})\left(2I^{-1}_{\alpha}(\lfloor\frac{T}{2}\rfloor + 1, T - \lfloor\frac{T}{2}\rfloor) - 1\right) \\
&\approx f(\frac{1}{2})\left(\frac{1}{\sqrt{1 + T/\left(\Phi^{-1}(\alpha)\right)^2}}\right) \\
&\approx \frac{f(\frac{1}{2})\Phi^{-1}(\alpha)}{\sqrt{T}}\,,
\end{aligned} \tag{4.12}$$

where we have used the same approximation of the incomplete beta function as in (4.7) and $f(\pi_1 = 1/2) > 0$ has been assumed.

This is an important result showing that for a fixed value of α the asymptotic decay of the probability is algebraic with a universal behavior $\propto T^{-1/2}$. The only dependence on the classification problem considered and on the ensemble method used is through the proportionality constant $f(\pi_1 = 1/2) > 0$. The heavy-tailed form of $\mathcal{P}(T^\star(\alpha, \pi_1) > T)$ implies that the probability of encountering instances that require a very large number of classifiers to converge to the asymptotic prediction with a level of confidence α is not negligible. Figure 4.2 illustrates this effect. The left-hand side of this figure displays the histogram for the minimum number of classifiers required to estimate the asymptotic class label of test instances with a confidence level of at least $\alpha = 99\%$ for

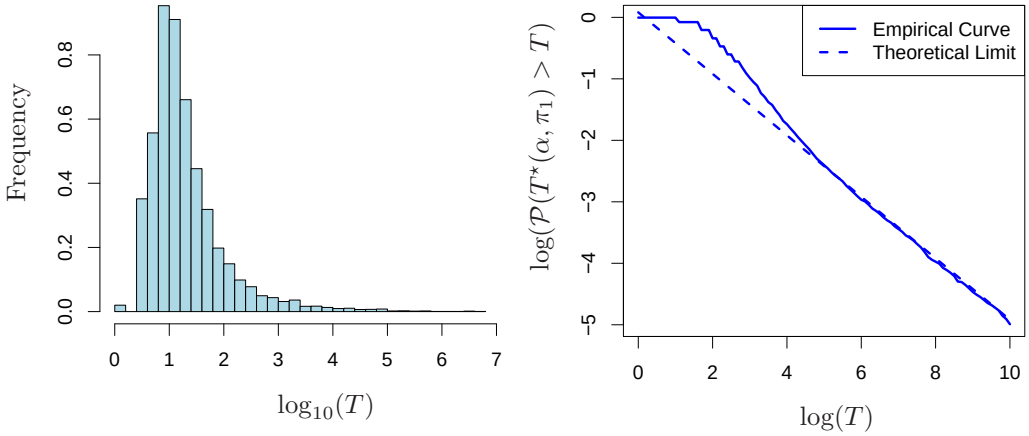

FIGURE 4.2: (left) Histogram for the values of $T^\star(\alpha, \pi_1)$ for the classification problem *Twonorm* and $\alpha = 99\%$. (right) Empirical and theoretical estimations of the distribution $\mathcal{P}(T^\star(\alpha, \pi_1) > T)$ as $T \to \infty$ displayed in double logarithmic axes. The slope of the straight line is $-1/2$.

the *Twonorm* problem. The distribution of the probabilities π_1 is estimated under the same conditions as the experiments whose results are displayed on in Figure 4.1 (right). The histogram shows that the right tail of the distribution has a very slow decay. The origin of this heavy-tailedness is the presence of instances close to the decision border ($\pi_1 \approx 1/2$), whose stable prediction requires querying a large number of classifiers. The right-hand side of this figure displays, in double logarithmic axes, an empirical estimate of $\mathcal{P}(T^\star(\alpha, \pi_1) > T)$ and the asymptotic approximation of $\mathcal{P}(T^\star(\alpha, \pi_1) > T)$ when $T \to \infty$ given by (4.9). This figure shows that the empirical estimate and the theoretical approximation coincide for sufficiently large values of T.

In summary, most of the data instances require querying a fairly small number of classifiers to produce an estimate of the asymptotic prediction that is correct with a high probability. By contrast, a small but not negligible fraction of test instances require querying an extremely large number of classifiers for the ensemble prediction to stabilize. The asymptotic convergence of the ensemble predictions on the test set is dominated by these borderline instances. In consequence, it is not possible to fix a finite ensemble size T so that the asymptotic prediction is reached for every test instance with a level of confidence $\alpha > 0$. This is a general result that applies to any binary classification problems and any ensemble learning algorithm provided that (i) the individual classifiers are built independently when conditioned to the training data; (ii) majority voting is used to combine the outputs of the ensemble classifiers; and (iii) $f(1/2) > 0$.

As a consequence of these findings, in practice it may be more reasonable to determine the size of the ensemble by requiring that the *average* confidence in the asymptotic prediction be larger or equal to α. With this less restrictive condition, the optimal ensemble size $T^\star(\alpha)$ is the minimum value of T that satisfies

$$\begin{aligned} \alpha &\leq \mathcal{P}(\hat{y}^T = \hat{y}^\infty | T) \\ &= \int_0^1 \mathcal{P}(\hat{y}^T = \hat{y}^\infty | \pi_1, T) f(\pi_1) d\pi_1 \\ &= \int_0^1 I_{\max\{\pi_1, 1-\pi_1\}} \left(\lfloor \frac{T}{2} \rfloor + 1, T - \lfloor \frac{T}{2} \rfloor \right) f(\pi_1) d\pi_1 \,. \end{aligned} \tag{4.13}$$

To calculate $T^\star(\alpha)$ we need an estimate of the probability distribution $f(\pi_1)$. This estimate can be obtained by cross-validation or using out-of-bag data (Banfield et al., 2007; Breiman, 1996c). If a test set composed of unlabeled instances is available, it can also be used to estimate $f(\pi_1)$ because the class labels are not required for this purpose. The training data cannot be used for the estimation because, as discussed earlier, it would lead to biased estimates of the distribution. Two different techniques are used in this work for this estimation. The first one computes a cross-validation estimate of $f(\pi_1)$. The training set is split into ten folds. Then, an ensemble of 100 classifiers is built using nine of the folds for training. The values of π_1 are estimated for each instance in the fold that has not been used for training. The process is repeated 10 times, interchanging the roles of the folds. As a result of this process, estimates of π_1 for every instance in the training set are obtained. The second strategy employs unlabeled test data to estimate $f(\pi_1)$. An ensemble of 100 classifiers is built using the training set. Then, the values of π_1 associated with each instance in the test set are computed. In both methods the integral in (4.13) is approximated by an average over the instances considered

$$\frac{1}{N}\sum_{i=1}^{N} I_{\max\{\hat{\pi}_1^{(i)},1-\hat{\pi}_1^{(i)}\}}\left(\lfloor\frac{T}{2}\rfloor+1, T-\lfloor\frac{T}{2}\rfloor\right), \tag{4.14}$$

where N is the number of instances ($N = N_{\text{train}}$ in the cross-validation estimate and $N = N_{\text{test}}$ in the test estimate) and $\{\hat{\pi}_1^{(i)}\}_{i=1}^{n}$ are the estimates of the values of π_1 for the instances in the training set, in the case of the cross validation estimate, and for the instances in the test set, when the test set is employed for the estimation. Once the value of the integral on the right-hand side of (4.13) has been approximated by (4.14), $T^\star(\alpha)$ can be obtained using numerical techniques. Specifically, since (4.14) grows monotonically with increasing values of T, assuming T odd, binary search can be used to find $T^\star(\alpha)$.

The rate of disagreement between the finite ensemble of size $T^\star(\alpha)$ and the infinite-size ensemble should be $\approx 1-\alpha$. Since the disagreement rate is the fraction of instances for which the predictions of these two ensembles differ, this value should be an *upper bound* of the differences between the generalization error rates of these two ensembles. In practice, if the changes in class label predictions affect approximately the same numbers of correctly and incorrectly classified instances the differences in error rate should actually be smaller than this upper bound. This observation is confirmed by the experiments presented in the following section.

4.3 Experiments

In this section we illustrate the application of the proposed framework to determine the optimal size of parallel classification ensembles. For this purpose experiments are carried out in several binary classification problems from the UCI repository (Asuncion and Newman, 2007) and from the *mlbench* package of the R statistics software (Leisch and Dimitriadou, 2007; R Development Core Team, 2005). Table 4.1 displays the number of attributes and instances for each problem. Non-binary classification problems are binarized as follows: in *Abalone* we discriminate between *adult* and *infant*; in *Balance* we discriminate between class label L and class label R; in *Boston* we discriminate between houses worth more that $\$21,000$ and houses worth less than this quantity; in *DNA* we discriminate between class label *IE* and class label *EI*; finally, in *Yeast* we discriminate between class label *cytosolic* and the other class labels. For the synthetic classification

problems (i.e. *Twonorm*, *Ringnorm* and *Circle*) we use 100 training and test sets of 300 instances and 1,000 instances respectively. For the non-synthetic classification problems we randomly split 100 times each dataset into a training and a test set using 2/3 and 1/3 of the total available data, respectively. To build the classification ensembles we use two representative ensemble learning algorithms: bagging (Breiman, 1996a) with un-pruned CART trees (Breiman et al., 1984) and random forests (RF) (Breiman, 2001).

TABLE 4.1: Datasets used in the experiments.

Problem	**Attributes**	**Instances**
Abalone	8	4,177
Australian	14	690
Balance	4	576
Boston	13	506
Breast	9	699
Circle	2	-
DNA	180	1,532
German	20	1,000
Ionosphere	34	351
Pima	8	768
Ringnorm	20	-
Spam	57	4,601
Tic-tac-toe	9	958
Twonorm	20	-
Vehicle	18	846
Votes	16	435
Yeast	8	1,484

For each problem, ensemble method, and train and test partition, we estimate the size $T^\star(\alpha)$ of the classification ensemble for $\alpha = 99\%$ using the procedure described in the previous section. We also compare two strategies for estimating $T^\star(\alpha)$ that differ in how the distribution $f(\pi_1)$ is approximated: either using ten-fold cross validation on the training set or using the unlabeled instances from the test set. Once the estimate of $f(\pi_1)$ has been computed, a binary search procedure is used to find $T^\star(\alpha)$, the minimum value of T that fulfills (4.14). We refer to these as optimal ensembles, based on the fact that they are the smallest ensembles such that on average the ensemble prediction coincides with the asymptotic prediction with a confidence level α. To determine whether, on average, the differences between this ensemble and the asymptotic one are close to $100\% - \alpha$, we compare the optimal ensemble predictions with an ensemble of 10,000 trees, which is used as a proxy of the infinite ensemble. We also estimate the error rate of the different ensembles on the test set. The differences in error rates should be smaller $100\% - \alpha$, which is the average fraction of instances for which the assigned class label changes. The results presented are averages over the different realizations of the training and test data.

Table 4.2 displays for each problem and ensemble method (bagging and RF) the average disagreement rates between the predictions of the optimal ensembles and of the asymptotic ensemble. In this table the suffix *CV* indicates that the value of $T^\star(\alpha)$ is estimated using cross-validation. The suffix *Test* indicates that the value of T has been estimated using the test set. These results show that the optimal ensembles have

TABLE 4.2: Average disagreement rates in % between the test predictions of the optimal and the asymptotic ensemble for bagging and RF.

Problem	RF-CV	RF-Test	Bagging-CV	Bagging-Test
Abalone	1.0±0.3	1.0±0.3	1.0±0.3	1.0±0.3
Australian	1.0±0.7	1.0±0.7	0.9±0.6	1.0±0.6
Balance	1.2±0.8	1.1±0.8	1.1±0.8	1.0±0.8
Boston	0.9±0.8	1.2±0.8	1.0±0.8	1.2±0.7
Breast	1.0±0.7	1.1±0.6	0.9±0.7	1.0±0.5
Circle	1.0±0.5	1.0±0.4	1.0±0.5	1.0±0.3
DNA	1.0±0.4	1.0±0.4	0.9±0.5	0.9±0.4
German	1.0±0.5	1.1±0.6	1.1±0.6	1.0±0.6
Ionosphere	0.8±0.9	0.8±0.8	1.1±1.0	1.0±0.9
Pima	1.0±0.6	1.2±0.7	1.1±0.7	1.1±0.7
Ringnorm	0.9±0.4	1.0±0.3	1.0±0.5	0.9±0.3
Spam	1.0±0.2	1.0±0.2	1.0±0.3	1.0±0.2
Tic-tac-toe	0.8±0.6	0.9±0.5	0.7±0.5	0.8±0.5
Twonorm	1.0±0.5	0.9±0.3	1.0±0.5	1.0±0.4
Vehicle	1.0±0.7	1.0±0.6	0.9±0.5	1.0±0.6
Votes	0.8±0.8	0.9±0.8	1.0±1.0	1.0±1.0
Yeast	1.1±0.5	1.1±0.5	0.9±0.5	1.1±0.5

disagreement rates that are generally around the $100\% - \alpha = 1\%$ threshold level set in the experiments, which confirms the validity of the probabilistic framework introduced in this research.

Tables 4.3 and 4.4 display for each problem and for bagging and RF, respectively, the asymptotic ensemble test error (Bagging∞ and RF∞) and the average test error for optimal ensembles (Bagging-CV, Bagging-Test, RF-CV and RF-Test), averaged over the 100 realizations of the classification problems considered. The standard deviations of these values are given after the ± symbol. As in the previous table, the procedure employed for the estimation of the size of the minimal ensemble is indicated by a suffix attached to the ensemble method (*CV* for cross-validation and *Test* for the method that uses the test set). To determine whether the differences in error rate are statistically significant we perform a Wilcoxon rank test (Wilcoxon, 1968). Error rates that are significantly larger than the corresponding asymptotic ensemble level (a p-value below 5% is obtained in the Wilcoxon test) are highlighted in boldface. The median of the number of trees used in these ensembles for the different realizations of the classification problems are also displayed. The interquartile interval is shown between parentheses. These measures are used instead of the mean and the standard deviation because they are more robust estimates of the center and dispersion of the optimal ensemble sizes. The values obtained for $T^\star(\alpha)$ when $f(\pi_1)$ is estimated using cross-validation or using the test sets are fairly similar.

Regarding the generalization performance, these results confirm that RF typically obtains lower error rates than bagging (Breiman, 2001). The lowest errors correspond to the asymptotic ensembles, as expected. The error rates of the optimal ensembles are only slightly higher and in many cases the differences are not statistically significant. In all cases the increases in error rate are much lower than the upper bound given by $100\% - \alpha$, the fraction of instances whose class label assignment can change.

TABLE 4.3: Average and standard deviation of the test errors for the infinite-size and for the optimal RF ensembles. Median and interquartile interval (between parentheses) of the number of trees for optimal RF ensembles.

Problem	RF∞	RF-CV	RF-Test	# Tree RF-CV		# Tree RF-Test	
Abalone	16.6±0.8	16.6±0.8	16.6±0.8	442	(403, 510)	455	(363, 578)
Australian	13.1±1.9	13.2±2.0	13.2±2.1	286	(206, 390)	334	(204, 567)
Balance	6.4±1.9	6.4±1.9	6.5±1.8	361	(238, 600)	433	(281, 672)
Boston	13.5±2.3	13.3±2.3	13.6±2.3	520	(352, 900)	484	(276, 943)
Breast	3.2±0.9	**3.4±1.1**	**3.6±1.0**	21	(17, 27)	21	(13, 31)
Circle	5.3±1.2	**5.4±1.2**	5.4±1.2	57	(37, 98)	62	(48, 84)
DNA	3.3±0.7	**3.5±0.8**	**3.5±0.7**	135	(120, 177)	134	(104, 184)
German	24.1±1.8	24.2±1.8	24.2±1.8	2213	(1526, 3164)	1811	(1318, 2868)
Ionosphere	6.7±2.0	6.8±2.3	6.8±2.2	85	(53, 129)	67	(42, 114)
Pima	24.0±2.1	23.8±2.0	24.0±2.1	1494	(1072, 2500)	1195	(810, 1919)
Ringnorm	6.2±1.1	6.2±1.1	**6.3±1.1**	724	(502, 1139)	639	(528, 784)
Spam	5.0±0.6	**5.1±0.6**	**5.1±0.5**	65	(57, 69)	61	(53, 72)
Tic-tac-toe	2.0±0.9	**2.2±0.9**	**2.4±0.9**	260	(188, 348)	168	(124, 263)
Twonorm	3.8±0.6	**3.9±0.6**	**3.9±0.7**	403	(270, 608)	398	(311, 532)
Vehicle	5.0±1.3	5.1±1.4	5.0±1.3	129	(97, 173)	138	(90, 196)
Votes	3.8±1.5	**4.1±1.5**	**4.1±1.5**	25	(18, 39)	20	(13, 31)
Yeast	24.5±1.5	24.5±1.5	24.4±1.5	1690	(1291, 2200)	1565	(1122, 2059)

TABLE 4.4: Average and standard deviation of the test errors for the infinite-size and for the optimal bagging ensembles. Median and interquartile interval (between parentheses) of the number of trees for optimal bagging ensembles.

Problem	Bagging$_\infty$	Bagging-CV	Bagging-Test	# Tree Bagging-CV		# Tree Bagging-Test	
Abalone	17.1±0.8	17.1±0.8	17.1±0.8	425	(356, 502)	403	(344, 512)
Australian	13.2±1.7	**13.4±1.7**	13.2±1.8	237	(161, 316)	218	(146, 328)
Balance	7.9±2.0	7.9±2.0	7.9±2.1	270	(168, 414)	321	(173, 462)
Boston	14.5±2.3	**14.7±2.3**	14.5±2.3	411	(269, 587)	362	(211, 696)
Breast	4.0±1.1	4.1±1.2	**4.2±1.1**	23	(17, 31)	21	(15, 36)
Circle	6.0±1.4	**6.1±1.4**	**6.1±1.4**	50	(31, 75)	45	(33, 65)
DNA	4.1±0.9	**4.3±1.0**	**4.3±0.9**	25	(19, 31)	21	(17, 27)
German	24.2±2.0	24.2±2.0	24.1±1.9	2006	(1438, 3219)	2107	(1196, 3768)
Ionosphere	7.9±2.2	7.9±2.3	8.0±2.1	84	(54, 142)	92	(42, 156)
Pima	24.4±2.2	24.4±2.2	24.3±2.2	1215	(856, 1716)	1287	(706, 2038)
Ringnorm	8.9±2.0	8.9±2.0	9.0±2.0	985	(613, 1590)	810	(608, 1084)
Spam	5.9±0.6	**6.1±0.6**	**6.0±0.6**	47	(43, 55)	47	(39, 55)
Tic-tac-toe	2.0±0.8	**2.2±0.9**	**2.3±0.8**	51	(42, 60)	35	(27, 50)
Twonorm	6.2±1.4	**6.3±1.5**	**6.3±1.4**	658	(456, 1040)	593	(452, 738)
Vehicle	6.0±1.6	6.1±1.7	6.1±1.6	127	(97, 174)	126	(86, 208)
Votes	4.6±1.7	4.6±1.9	**4.8±1.7**	21	(15, 33)	25	(13, 44)
Yeast	25.2±1.7	25.3±1.7	**25.3±1.6**	1890	(1455, 2317)	1639	(1112, 2568)

These results show that different classification problems require ensembles of very different sizes. Some classification problems need ensembles of less than 100 trees to reach a stable prediction with a confidence level $\alpha = 99\%$ (e.g. *votes*, *ionosphere* and *breast*). For others, the appropriate number of trees to combine is in the thousands (e.g. *german*, *pima*, *yeast*). This conclusion advices against using the same number of classifiers irrespective of the problem considered, which is the dominant approach in most of the existing literature on ensembles. Furthermore, the ensembles used in previous studies are rarely above 200 classifiers, which is probably too small for some problems.

4.4 Conclusions

In this chapter we have addressed the question of how to determine the size of parallel classification ensembles. The proposed method is based on estimating the number of classifiers that are necessary to yield a prediction that, on average, coincides with a hypothetical ensemble of infinite size at a high confidence level α. In contrast to previous proposals, this method is not based on the availability of accurate estimates of the generalization error. Instead, it relies on the analysis of the convergence of the prediction of parallel classification ensembles as a function of ensemble size in the asymptotic regime, when the number of classifiers in the ensemble tends to infinity. The framework is valid for any classification problem and any parallel ensemble provided that the individual classifiers are generated independently when conditioned to the training data and that their predictions are combined by a simple majority voting.

The analysis performed shows that it is not possible to fix a size for the ensemble so that the prediction of such an ensemble coincides with the asymptotic ensemble prediction for every potential test instance with a fixed confidence level $\alpha > 0$. While most of the instances require only a few classifiers to reach the infinite ensemble prediction with a high confidence, the predictions of a small but not negligible fraction of instances require an extremely large number of queries to converge. Thus, instead of requiring convergence for every instance, we propose to determine the optimal ensemble size by requiring that, *on average*, the finite and the infinite ensemble predictions coincide with a high probability α.

The validity of the probabilistic framework developed and the usefulness of the method is illustrated in two representative parallel ensemble learning algorithms (bagging and RF) for different classification problems. The experiments show that the predictions of the finite classification ensembles constructed tend to agree with the asymptotic ones with a probability close to α, the target confidence level used to determine the ensemble size. Because the differences in error are bound from above by $100\% - \alpha$, the prediction accuracy of the optimal ensembles is only slightly lower than the corresponding infinite-size ensembles.

An important conclusion of this investigation, which agrees with the results of (Banfield et al., 2007), is the need to adapt the ensemble size to the particular classification problem considered. Some problems require ensembles of only a few tens classifiers to obtain, on average, a confidence level α on the asymptotic ensemble prediction. Others require thousands of classifiers for the ensemble prediction to stabilize.

Finally, unlike previous proposals, the method designed in this investigation does not require labeled data to estimate an adequate value for the size of the ensemble.

Part II

Bayesian Techniques

Chapter 5

Bayesian Machine Learning

The uncertainties about the type of model and about the model parameters that arise in automatic induction problems can be described using a Bayesian framework. Assuming a particular form for the model, Bayesian machine learning uses probabilities to quantify different degrees of belief in the values that the model parameters can take. Starting from a prior distribution that represents our initial beliefs, Bayes' theorem is used to compute a posterior distribution for these parameters. This posterior distribution describes how these beliefs should be updated after the training data are observed. The prediction of a Bayesian model for a test instance is computed in terms of the predictive distribution. This distribution is obtained by averaging the product of the posterior distribution of the model parameters and a likelihood function evaluated in the test instance. Besides being used to model the uncertainty about the actual values of the model parameters, Bayesian probabilities can also be employed to assign degrees of belief to different models. Because these probabilities automatically penalize unnecessarily complex models, they provide a robust method to perform model selection. A difficulty of Bayesian machine learning is that carrying out exact Bayesian inference is often intractable. In practice, approximate methods have to be employed. In this chapter we review some of these methods. A first group of methods approximate the posterior distribution by a simple distribution for which the required computations are tractable. A second group includes stochastic algorithms that generate samples from the posterior distribution. These samples can be used to approximate the exact posterior.

5.1 Introduction

The amount of labeled data available for induction is limited in many supervised machine learning problems. Furthermore, these data can be contaminated by noise. Under these circumstances there is uncertainty in the selection of an appropriate model to describe the observed data and in the selection of the model parameters. This uncertainty becomes smaller as the amount of data available to perform the estimations increases. However, for small datasets it is important to take it into account to get reliable estimates. To address these types of problems we need a mechanism to represent, manipulate and update uncertainty when new observations are available. The Bayesian interpretation of probabilities provides a suitable framework that can be used for this

purpose. This interpretation is different from the frequentist/traditional view. In the frequentist interpretation, probabilities are described in terms of the frequencies of random repeatable events, where an event is defined as a particular subset of the possible outcomes of a random experiment. The probability of an event is the limit of its relative frequency in a very large (infinite) number of trials. By contrast, in the Bayesian interpretation, probabilities are used to describe *degrees of belief* in events that do not necessarily involve random variables (Bishop, 2006; MacKay, 2003). Such an event can be, for example, the sea level rising one meter by the end of the century. The use of probabilities to represent uncertainty can be supported by the fact that degrees of belief can be mapped onto probabilities provided that they satisfy simple consistency rules known as the Cox axioms (Cox, 1946). Nevertheless, the Bayesian interpretation of probability is *subjective*, since the final beliefs obtained depend on the assumptions made (MacKay, 2003). These assumptions are expressed in the form of a prior distribution that represents our initial uncertainty about the quantities of interest. Once new evidence has been observed, we update this prior distribution using Bayes' theorem in a procedure called Bayesian inference. Bayes' theorem can be used to go from an initial prior distribution to a posterior distribution that reflects this additional knowledge (typically expressed in the form of new observations) about the quantities of interest (Bishop, 2006; MacKay, 2003).

In supervised machine learning Bayes' theorem is used to compute a posterior probability distribution for the model parameters that reflects our beliefs in their values after having observed the training data (Bishop, 2006; MacKay, 2003). Given the attributes of an unlabeled instance, this posterior probability is used to infer a probability distribution for the unknown target variable y. This distribution is obtained by computing an average with respect to the posterior distribution and it can be used to quantify uncertainty in the predictions of the model. Bayesian probabilities can also be used to describe the uncertainty in the selection of an adequate model to represent the observed data. Because these probabilities automatically penalize unnecessarily complex models, they provide a robust mechanism to perform model selection (MacKay, 2003). A Bayesian approach to supervised machine learning is also useful to incorporate prior information about the learning task that can compensate the limited amount of data available. This information is generally expressed in terms of a prior distribution for the model parameters.

A difficulty of Bayesian machine learning is that using Bayes' theorem to compute the posterior distribution is often infeasible. The application of Bayes' theorem requires computing integrals or summations that are too complex to be solved analytically. Thus, in practical situations approximate techniques have to be employed (Bishop, 2006; MacKay, 2003). These techniques approximate the posterior distribution by a simple distribution for which the required computations are tractable or by samples generated from a Markov chain whose stationary distribution coincides with the posterior distribution of the model parameters.

The organization of the chapter is as follows: Section 5.2 gives an overview of Bayesian machine learning. Section 5.3 describes Bayesian model selection, a robust method to discriminate among different models. Type-II maximum likelihood is reviewed in Section 5.4. This is a technique that can be used to estimate those parameters for which an explicit posterior distribution is not available. Section 5.5 reviews some methods for approximate Bayesian inference. These are grouped in two categories: deterministic methods and sampling techniques. Finally, Section 5.6 summarizes the conclusions of this chapter. Most of the material described in this chapter has been

extracted from (Bishop, 2006) and (MacKay, 2003). This material is included for the purpose of reference in the following chapters of this thesis.

5.2 Bayesian Machine Learning

In the field of machine learning, Bayesian probabilities can be used to account for the lack of knowledge of the correct model or for the uncertainty in the model parameters. Consider a given model with parameters $\boldsymbol{\theta}$. Assuming some prior distribution for these parameters, $\mathcal{P}(\boldsymbol{\theta})$, and given a training set $\mathcal{D}$, Bayes' theorem can be used to compute a posterior probability distribution for $\boldsymbol{\theta}$ given $\mathcal{D}$

$$\mathcal{P}(\boldsymbol{\theta}|\mathcal{D}) = \frac{\mathcal{P}(\mathcal{D}|\boldsymbol{\theta})\mathcal{P}(\boldsymbol{\theta})}{\mathcal{P}(\mathcal{D})}\,, \tag{5.1}$$

where $\mathcal{P}(\mathcal{D}|\boldsymbol{\theta})$ is the likelihood function of the model parameters $\boldsymbol{\theta}$ for the observed data $\mathcal{D}$. This function expresses how probable the observed data are for different values of $\boldsymbol{\theta}$. Note that $\mathcal{P}(\mathcal{D}|\boldsymbol{\theta})$ is not a probability distribution for $\boldsymbol{\theta}$. Therefore, its integral with respect to $\boldsymbol{\theta}$ is in general different from one. The distribution $\mathcal{P}(\boldsymbol{\theta})$ is the prior distribution for the model parameters $\boldsymbol{\theta}$. This distribution reflects our prior beliefs in the values of the model parameters before observing the training data $\mathcal{D}$. Finally, $\mathcal{P}(\mathcal{D})$ is a normalization constant that ensures that the posterior distribution integrates to one.

Note that this is very different from a frequentist approach to machine learning (Hastie et al., 2001). In a frequentist setting, the model parameters $\boldsymbol{\theta}$ are assumed to be fixed, and their values are determined using some estimator. Error bars on this estimate are obtained by considering the distribution of the observed data $\mathcal{D}$. By contrast, in the Bayesian approach there is only a single dataset $\mathcal{D}$ that is fixed, and the uncertainty in the model parameters is calculated using the posterior distribution. However, even though we have computed a probability distribution for $\boldsymbol{\theta}$ in (5.1), this does not imply that $\boldsymbol{\theta}$ is random. The posterior probability distribution is used here to express our beliefs in the possible values of $\boldsymbol{\theta}$ after having observed the data. Besides being used to compute a posterior distribution for the model parameters, Bayes' theorem can also be used to compute a posterior distribution for other values that are not observed. For example, these can be latent variables that are introduced in the model to simplify the computations (Bishop, 2006; MacKay, 2003).

We now illustrate Bayesian machine learning with a simple problem. Consider a dataset $\mathcal{D}$ composed of n independent observations $\{x_1, \ldots, x_n\}$ from a Gaussian distribution with unknown mean μ and unit variance. Assuming that the prior values for μ are given by a standard Gaussian distribution, Bayes' theorem can be used to make inference about μ after having observed $\mathcal{D}$

$$\begin{aligned}
\mathcal{P}(\mu|\mathcal{D}) &= \frac{\mathcal{P}(\mathcal{D}|\mu)\mathcal{P}(\mu)}{\mathcal{P}(\mathcal{D})} \\
&= \frac{\prod_{i=1}^{n}\mathcal{N}(x_i|\mu,1)\mathcal{N}(\mu|0,1)}{\int\prod_{i=1}^{n}\mathcal{N}(x_i|\mu,1)\mathcal{N}(\mu|0,1)d\mu} \\
&= \mathcal{N}\left(\mu|\left(\frac{1}{n+1}\sum_{i=1}^{n}x_i\right),(n+1)^{-1}\right)\,,
\end{aligned} \tag{5.2}$$

where we have used (A.42) to compute (5.2). Thus, the posterior distribution for μ is a Gaussian distribution with mean $(n+1)^{-1}\sum_{i=1}^{n} x_i$ and variance $(n+1)^{-1}$. The assumption of a standard Gaussian distribution for the prior for μ illustrates our belief, before making any empirical observation, that μ should be close to zero.

Expression (5.2) shows that as the number of data points n increases, the mean of the posterior distribution for μ approaches the sample mean. Similarly, the variance of the posterior distribution approaches zero. Note that in the particular case of n equal to zero, we have a standard Gaussian distribution for μ. This is just the prior distribution for μ. These results are illustrated in Figure 5.1, where the posterior distribution for μ is displayed for different values of n. In this case, the data are generated form a Gaussian distribution with unit variance and mean $\mu_0 = -1$. The picture shows that as more data are observed, the posterior distribution for μ becomes more and more peaked around true value of μ. However, for small values of n, the uncertainty about the true value of μ is still very large as many different values of μ could have generated the observed data. This uncertainty leads to a large variance in the posterior distribution for μ.

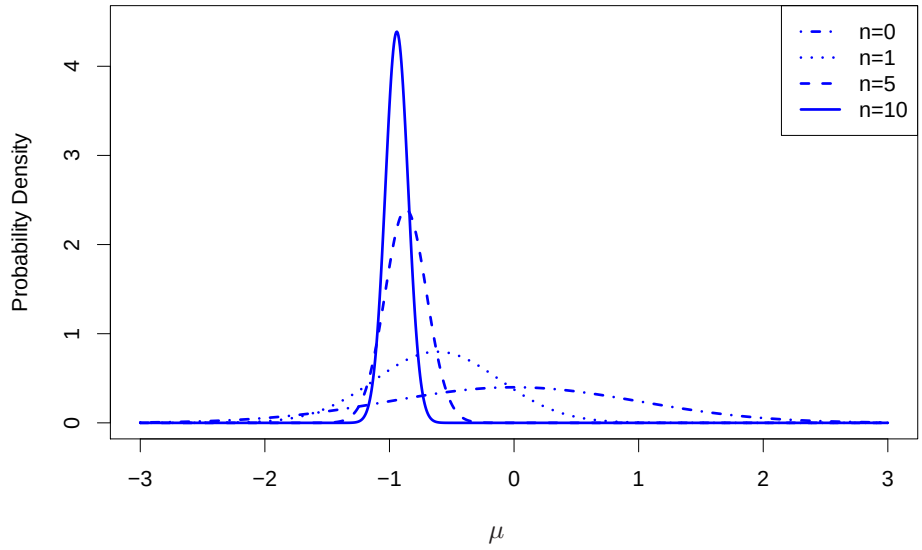

FIGURE 5.1: Posterior probability distribution for μ for different sample sizes n. The data are generated from a Gaussian distribution with unit variance and mean equal to -1. Note that the posterior distribution is more and more peaked around -1 as the sample size n increases.

From a frequentist perspective we can also compute a confidence interval for μ_0. Within this approach, μ_0 is estimated by the sample mean

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i \,. \tag{5.3}$$

Then, error bars can be computed for this estimate, and from these, a confidence interval for μ_0 is given with probability $(1-2\alpha)$ by

$$\left[\hat{\mu} - z_{1-\alpha}\sqrt{1/n}, \hat{\mu} + z_{1-\alpha}\sqrt{1/n}\right] \,. \tag{5.4}$$

We note that as $n \to +\infty$ both perspectives give similar results because the posterior distribution (5.2) approaches a delta function centered at the sample mean.

In a Bayesian approach, once the posterior distribution for μ has been computed, we can calculate a *predictive distribution* for a new instance x_{new} that takes into account the uncertainty in the estimation of this parameter

$$\begin{aligned}\mathcal{P}(x^{\text{new}}|\mathcal{D}) &= \int \mathcal{P}(x^{\text{new}}|\mu)\mathcal{P}(\mu|\mathcal{D})d\mu \\ &= \int \mathcal{N}(x^{\text{new}}|\mu,1)\mathcal{N}\left(\mu|\left(\frac{1}{n+1}\sum_{i=1}^{n}x_i\right),(n+1)^{-1}\right)d\mu \\ &= \mathcal{N}\left(x^{\text{new}}|\left(\frac{1}{n+1}\sum_{i=1}^{n}x_i\right),(n+1)^{-1}+1\right). \end{aligned} \tag{5.5}$$

To compute (5.5) we have used (A.42). Interestingly, the variance of the predictive distribution is larger than the variance of the actual generating model, which is 1. This is a consequence of the uncertainty in the value of the μ parameter, which is described by a distribution instead of a single point estimate. This posterior distribution accounts for the fact that the observed data could have been generated from a Gaussian distribution with different values of μ. The predictive distribution is consistent in the sense that when $n \to +\infty$, (5.5) tends to the actual generating model. Finally, we note that (5.5) is computed by eliminating the dependence of the model on the particular values of the parameters (μ in this case). This is a standard procedure in Bayesian machine learning. Predictive distributions are computed by marginalizing over the model parameters in the product of a likelihood function for the test instance and the posterior distribution for the model parameters.

The frequentist approach for predicting the value of a new instance x^{new} consists in replacing the unknown parameter μ with the estimate $\hat{\mu}$ (Hastie et al., 2001). In consequence, the expected value for x^{new} is just the sample mean $\hat{\mu}$. A confidence interval for x^{new} can then be computed with probability $1-2\alpha$ using

$$[\hat{\mu} - z_{1-\alpha}, \hat{\mu} + z_{1-\alpha}]\,. \tag{5.6}$$

Unlike in the Bayesian approach, this confidence interval does not take into account the uncertainty about the true value of μ. However, we note again that as $n \to +\infty$ the confidence intervals for x^{new} obtained by the two different perspectives converge to the same limit.

An advantage of the Bayesian viewpoint over the frequentist one is that the inclusion of prior knowledge about the model parameters arises quite naturally. This prior knowledge can compensate the limited amount of data available to perform the estimations. Nevertheless, a common criticism is that prior distributions are often chosen for computational convenience rather than as a reflection of any specific prior beliefs (Bishop, 2006). Furthermore, the dependence of the conclusions on the choice of the prior is seen as a source of difficulty by many researchers. The practical application of Bayesian machine learning also faces the difficulty of performing the required computations. In particular, the exact marginalization of all the parameters of the model to compute the posterior distribution or the predictive distribution is often infeasible. Thus, approximate techniques are employed in practice. A standard method to approximate the posterior distribution is by using a finite number of samples generated from a Markov chain whose stationary distribution coincides with the posterior distribution of the model. More recently, efficient deterministic approximation algorithms such as variational inference (Jaakkola, 2001) and expectation propagation (EP) (Minka, 2001b)

have been developed. In this thesis, EP is systematically employed in different contexts to carry out approximate Bayesian inference.

5.3 Bayesian Model Selection

The Bayesian principle of model selection uses probabilities to describe the uncertainty in the selection of the model (Bishop, 2006; MacKay, 2003). Consider a set of candidate models for the learning task $\mathcal{M} = \{m_1, \ldots, m_l\}$. Assume that our initial uncertainty is expressed through a prior probability function $\mathcal{P}(m_i)$, with $i = 1, \ldots, l$. We can evaluate the posterior distribution of each model given $\mathcal{D}$

$$\mathcal{P}(m_i|\mathcal{D}) = \frac{\mathcal{P}(\mathcal{D}|m_i)\mathcal{P}(m_i)}{\mathcal{P}(\mathcal{D})}\,, \tag{5.7}$$

where $\mathcal{P}(\mathcal{D})$ is a normalization constant that guarantees that $\{\mathcal{P}(m_i|\mathcal{D}), i = 1, \ldots, l\}$ adds up to one. Assume that our prior belief is that all models are equally likely. In this situation, the term that dominates (5.7) is $\mathcal{P}(\mathcal{D}|m_i)$, which determines the probability of the observed data given the model m_i. This term is also known as the model evidence for the observed data.

Ideally, in a full Bayesian approach we should compute the predictive distribution for a new instance by averaging over all the different models using the posterior in (5.7)

$$\mathcal{P}(y^{\text{new}}|\mathbf{x}^{\text{new}}, \mathcal{D}) = \sum_{i=1}^{l} \mathcal{P}(y^{\text{new}}|\mathbf{x}^{\text{new}}, m_i, \mathcal{D})\mathcal{P}(m_i|\mathcal{D})\,. \tag{5.8}$$

Nevertheless, computing this last summation can be costly. Thus, it is often considered an approximation to this average, based on selecting the single most probable model alone to make predictions. That is, the model whose associated value $\mathcal{P}(\mathcal{D}|m_i)$ is maximum, as we are considering a uniform prior $\mathcal{P}(m_i)$.

Note that the value of $\mathcal{P}(\mathcal{D}|m_i)$ is simply the normalization constant that appears in Bayes' theorem in the previous section. For a model with parameters $\boldsymbol{\theta}$ this constant is

$$\mathcal{P}(\mathcal{D}|m_i) = \int \mathcal{P}(\mathcal{D}|m_i, \boldsymbol{\theta})\mathcal{P}(\boldsymbol{\theta}|m_i)d\boldsymbol{\theta}\,, \tag{5.9}$$

which can be seen as the probability that the observed data $\mathcal{D}$ has been generated by m_i when the model parameters are sampled at random from the prior distribution.

For the sake of simplicity, consider that the model has only one parameter θ. Assume that the posterior distribution for θ is sharply peaked around the most probable value $\hat{\theta}$ and that the prior for θ is roughly flat. Under these conditions, $\mathcal{P}(\mathcal{D}|m_i)$ can be approximated as

$$\mathcal{P}(\mathcal{D}|m_i) \approx \mathcal{P}(\mathcal{D}|m_i, \hat{\theta})\frac{\Delta_{\text{posterior}}}{\Delta_{\text{prior}}}\,, \tag{5.10}$$

where $\Delta_{\text{posterior}}$ is the width of the posterior, Δ_{prior} is the width of the prior and $\Delta_{\text{posterior}} < \Delta_{\text{prior}}$. (MacKay, 2003). Taking logarithms we get

$$\log(\mathcal{P}(\mathcal{D}|m_i)) \approx \log(\mathcal{P}(\mathcal{D}|m_i, \hat{\theta})) + \log\left(\frac{\Delta_{\text{posterior}}}{\Delta_{\text{prior}}}\right)\,. \tag{5.11}$$

The right-hand side of this last expression shows a trade-off between how well the model explains the observed data, measured by $\mathcal{P}(\mathcal{D}|m_i, \hat{\theta})$, and the model complexity, measured by $\log(\Delta_{\text{posterior}}/\Delta_{\text{prior}})$. This last term is negative, and its magnitude increases as the ratio $\Delta_{\text{posterior}}/\Delta_{\text{prior}}$ becomes smaller. If the expressive capacity of the model is large, $\Delta_{\text{posterior}}$ is much smaller than Δ_{prior}. In consequence, the penalty of this term in $\mathcal{P}(\mathcal{D}|m_i)$ is large. The optimal model complexity is determined by a trade-off between these two competing terms. Thus, $\mathcal{P}(\mathcal{D}|m_i)$ automatically penalizes models that are unnecessarily complex to explain the observed data.

To further illustrate Bayesian model selection, consider its application in the problem described in the previous section about how to estimate the mean of a Gaussian distribution with known variance. For simplicity, let us assume that the possible observed data $\mathcal{D}$ consist only of one single instance x. We want to discriminate between two models, m_1 and m_2, that only differ in the width of the prior distribution for μ. Let $\nu_1 = 1$ be the variance of the prior of first model and $\nu_2 = 50$ the variance of the prior of the second one. Thus, m_1 is a simple model with a limited expressive capacity. By contrast, the larger variance of the prior for μ in m_2 allows to explain a wider variety of datasets. This corresponds to a more complex model. The model evidence for an observed dataset $\mathcal{D} = \{x\}$ is

$$\begin{aligned}\mathcal{P}(\mathcal{D}|m_i) &= \int \mathcal{N}(x|\mu, 1)\mathcal{N}(\mu|0, \nu_i)d\mu \\ &= \mathcal{N}(x|0, \nu_i + 1)\,. \end{aligned} \tag{5.12}$$

Figure 5.2 displays the value of $\mathcal{P}(\mathcal{D}|m_i)$ for each model and for different values of the observed data $\mathcal{D}$. The figure shows that model m_1 provides larger values for $\mathcal{P}(\mathcal{D}|m_i)$ in the vicinity of zero as a consequence of the prominent peak in the prior for μ at zero. This means that m_1 is preferred when the observed data are in this region. On the other hand, because of the flat prior, model m_2 provides larger values for $\mathcal{P}(\mathcal{D}|m_i)$ when $\mathcal{D}$ is far away from the origin. Thus, m_2 is preferred in this case. We note that even though the second model can also explain datasets in the vicinity of zero, the first model is preferred in that region because it is simpler. In regions far away from zero, m_1 fails to explain the observed data because its expressive capacity is too low. Model m_2 has a larger expressive capacity and hence, it is preferred in these regions.

5.4 Type-II Maximum Likelihood Estimation

In many Bayesian models there can be some parameters $\boldsymbol{\vartheta}$ for which it is not possible to compute a posterior distribution. For example, because we can not integrate analytically over all the parameters of the model. In this situation, we can set these parameters to specific values determined by type-II maximum likelihood estimation (Bishop, 2006). This procedure approximates the posterior distribution of these parameters $\mathcal{P}(\boldsymbol{\vartheta}|\mathcal{D})$ by a delta function centered at the most probable value $\hat{\boldsymbol{\vartheta}}$. Note that this is equivalent to selecting a single value of $\boldsymbol{\vartheta}$ equal to $\hat{\boldsymbol{\vartheta}}$. Consider a Bayesian model with parameters $\boldsymbol{\vartheta}$, for which marginalization is not possible, and parameters $\boldsymbol{\theta}$, for which marginalization is possible. From Bayes' theorem we have

$$\mathcal{P}(\boldsymbol{\vartheta}|\mathcal{D}) \propto \mathcal{P}(\mathcal{D}|\boldsymbol{\vartheta})\mathcal{P}(\boldsymbol{\vartheta})\,, \tag{5.13}$$

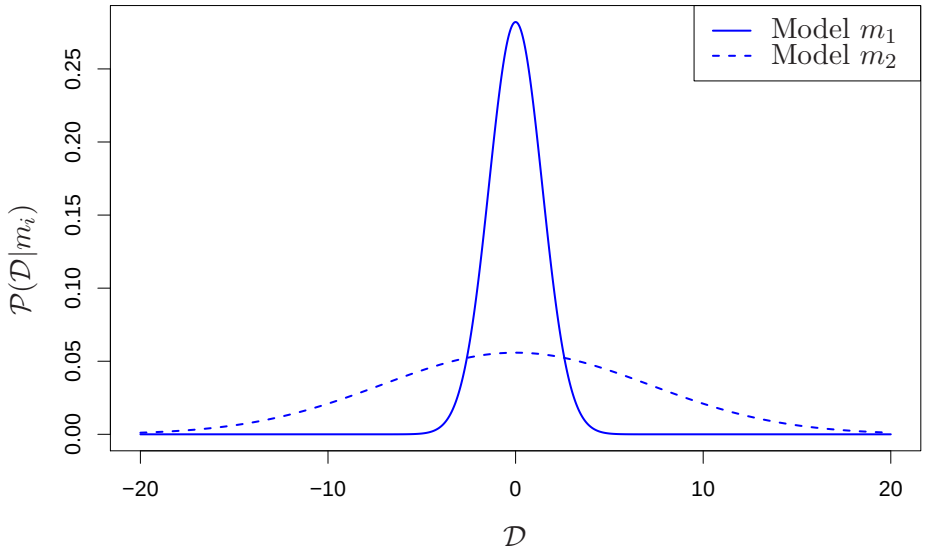

FIGURE 5.2: Model evidence $\mathcal{P}(\mathcal{D}|m_i)$ for the two different models considered, m_1 and m_2, and for different values of the observed data $\mathcal{D}$.

where $\mathcal{P}(\boldsymbol{\vartheta})$ is some prior distribution for $\boldsymbol{\vartheta}$ and

$$\mathcal{P}(\mathcal{D}|\boldsymbol{\vartheta}) = \int \mathcal{P}(\mathcal{D}|\boldsymbol{\vartheta}, \boldsymbol{\theta})\mathcal{P}(\boldsymbol{\theta})d\boldsymbol{\theta}\,. \tag{5.14}$$

Assuming that the prior distribution $\mathcal{P}(\boldsymbol{\vartheta})$ is approximately flat, we can find $\hat{\boldsymbol{\vartheta}}$ by maximizing $\mathcal{P}(\mathcal{D}|\boldsymbol{\vartheta})$ with respect to $\boldsymbol{\vartheta}$.

In many practical applications the analytical evaluation of $\mathcal{P}(\mathcal{D}|\boldsymbol{\vartheta})$ is not possible and its value has to be approximated by some algorithm. In this situation, maximizing $\mathcal{P}(\mathcal{D}|\boldsymbol{\vartheta})$ with respect to $\boldsymbol{\vartheta}$ requires running this algorithm repeatedly.

5.5 Approximate Bayesian Inference

The principal task in the application of Bayesian inference is the computation of the posterior distribution of the model parameters $\boldsymbol{\theta}$ given the observed data $\mathcal{D}$

$$\mathcal{P}(\boldsymbol{\theta}|\mathcal{D}) = \frac{\mathcal{P}(\mathcal{D}|\boldsymbol{\theta})\mathcal{P}(\boldsymbol{\theta})}{\mathcal{P}(\mathcal{D})}\,. \tag{5.15}$$

This posterior distribution is then used to compute the predictive distribution, which is required for making predictions. In particular, in supervised machine learning we are interested in computing

$$\mathcal{P}(\mathcal{D}) = \int \mathcal{P}(\mathcal{D}|\boldsymbol{\theta})\mathcal{P}(\boldsymbol{\theta})d\boldsymbol{\theta}\,, \tag{5.16}$$

$$\mathcal{P}(y_{\text{new}}|\mathbf{x}_{\text{new}}, \mathcal{D}) = \int \mathcal{P}(y_{\text{new}}|\mathbf{x}_{\text{new}}, \boldsymbol{\theta})\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta}\,, \tag{5.17}$$

where (5.16) is the normalization constant of the posterior distribution, $\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})$, and (5.17) is the predictive distribution for an unseen instance. Besides being required for

computing $\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})$, (5.16) is also useful for model selection or for type-II maximum likelihood estimation. Typically, computing (5.16) and (5.17) is intractable. The intractability of (5.16) arises because the dimensionality of the model parameters $\boldsymbol{\theta}$ can be very large, or because the combination of the likelihood function with the prior distribution results in a very complicated function of $\boldsymbol{\theta}$ for which integration in closed form is not possible (Bishop, 2006). If (5.16) is intractable, the computation of (5.17) is also intractable as it requires knowing the posterior distribution. Thus, in many practical situations we have to resort to approximation techniques to compute these integrals.

If analytical integration is not possible, different techniques, either deterministic or stochastic, can be used to approximate (5.16) and (5.17) (Bishop, 2006; MacKay, 2003). In particular, numerical quadrature can be used for this purpose (Press et al., 1992). The disadvantage is that the computation cost grows exponentially with the dimension of the vector $\boldsymbol{\theta}$. Thus, numerical quadrature can only be used in very simple inference problems with a small number of parameters. Another deterministic approximation is to replace the exact posterior distribution by a simple distribution for which expectations can be easily computed. Typical choices are parametric families of distributions such as the Gaussian family. The parameters of the approximation are estimated within the parametric family considered in such a way that the resulting distribution is as close as possible to the exact posterior (Bishop, 2006; MacKay, 2003). Given an approximate posterior distribution $\mathcal{Q}(\boldsymbol{\theta})$, the predictive distribution for $\mathbf{x}_{\text{new}}$ is

$$\mathcal{P}(y_{\text{new}}|\mathbf{x}_{\text{new}}, \mathcal{D}) \approx \int \mathcal{P}(y_{\text{new}}|\mathbf{x}_{\text{new}}, \boldsymbol{\theta})\mathcal{Q}(\boldsymbol{\theta})d\boldsymbol{\theta}\,, \tag{5.18}$$

and the normalization constant of $\mathcal{Q}$ can be used to approximate (5.16). These deterministic techniques are very efficient, but their accuracy is limited by the quality of the approximation of $\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})$ by $\mathcal{Q}(\boldsymbol{\theta})$.

Stochastic techniques based on Markov chain Monte Carlo (MCMC) approximate the posterior distribution by a set of samples $\mathcal{S} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_m\}$ from the posterior distribution (Bishop, 2006; MacKay, 2003). Using these samples, the value of (5.17) can be approximated using the law of large numbers

$$\mathcal{P}(y_{\text{new}}|\mathbf{x}_{\text{new}}, \mathcal{D}) \approx \frac{1}{m}\sum_{i=1}^{m} \mathcal{P}(y_{\text{new}}|\mathbf{x}_{\text{new}}, \boldsymbol{\theta}_i)\,. \tag{5.19}$$

These samples are typically generated by running a Markov chain whose stationary distribution coincides with the posterior distribution $\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})$. Interestingly, it is possible to build such a Markov chain without knowing the exact value of of $\mathcal{P}(\mathcal{D})$. However, unlike the deterministic methods described above, sampling methods do not provide a direct way to approximate $\mathcal{P}(\mathcal{D})$. For this purpose, importance sampling can be used (Bishop, 2006; MacKay, 2003). In particular,

$$\mathcal{P}(\mathcal{D}) = \int \mathcal{P}(\mathcal{D}|\boldsymbol{\theta})\mathcal{P}(\boldsymbol{\theta})d\boldsymbol{\theta} = \int \mathcal{Q}(\boldsymbol{\theta})\frac{\mathcal{P}(\mathcal{D}|\boldsymbol{\theta})\mathcal{P}(\boldsymbol{\theta})}{\mathcal{Q}(\boldsymbol{\theta})}d\boldsymbol{\theta} \approx \sum_{\boldsymbol{\theta}\in\mathcal{S}_{\mathcal{Q}}} \frac{\mathcal{P}(\mathcal{D}|\boldsymbol{\theta})\mathcal{P}(\boldsymbol{\theta})}{\mathcal{Q}(\boldsymbol{\theta})}\,, \tag{5.20}$$

where $\mathcal{Q}$ is a known arbitrary distribution with the same support as the posterior distribution and $\mathcal{S}_{\mathcal{Q}}$ is a set of independent samples generated from $\mathcal{Q}$. A limitation of this method is that (5.20) can have a large variance if $\mathcal{Q}$ is very different from $\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})$ (Bishop, 2006). Sampling methods converge to the exact values as the number of samples generated increases. However, they can be computationally demanding. In practice

obtaining independent samples from the posterior distribution often requires running very long chains.

5.5.1 Deterministic Methods

In this section we describe a series of deterministic methods that can be used to perform approximate Bayesian inference. In general, the methods described in this section approximate the joint distribution of the data and the model parameters $\mathcal{P}(\mathcal{D}, \boldsymbol{\theta})$ by a simple function for which integrals can be evaluated analytically.

5.5.1.1 The Laplace Approximation

This is a simple approximate inference method in which the posterior distribution of the model parameters is replaced by a Gaussian distribution $\mathcal{Q}$ (Bishop, 2006; MacKay, 2003). To illustrate how it works, assume first that the model has a single continuous parameter θ. Consider the joint probability function of the observed data $\mathcal{D}$ and θ, $\mathcal{P}(\theta, \mathcal{D})$. This probability function can be computed as the product of the likelihood and the prior, i.e. $\mathcal{P}(\theta, \mathcal{D}) = \mathcal{P}(\mathcal{D}|\theta)\mathcal{P}(\theta)$. Normalized with respect to θ it becomes the posterior distribution of the model parameter $\mathcal{P}(\theta|\mathcal{D})$. The normalizing constant is the model evidence $\mathcal{P}(\mathcal{D})$. Thus, using a Gaussian distribution $\mathcal{Q}$, we are interested in approximating

$$\mathcal{P}(\theta|\mathcal{D}) \propto \mathcal{P}(\theta, \mathcal{D}) \,, \tag{5.21}$$

where the normalization constant $\mathcal{P}(\mathcal{D}) = \int \mathcal{P}(\theta, \mathcal{D}) d\theta$ is unknown. For simplicity, consider the notation $\tilde{\mathcal{P}}(\theta) = \mathcal{P}(\theta, \mathcal{D})$. To find the parameters of $\mathcal{Q}$, the Laplace method assumes that $\mathcal{P}(\theta|\mathcal{D})$ has a mode at some point θ_0. This mode can be found using any optimization technique (Press et al., 1992). In particular, since $\mathcal{P}(\theta|\mathcal{D})$ and $\tilde{\mathcal{P}}(\theta_0)$ only differ by a normalization constant, we have to find θ_0 such that $d\tilde{\mathcal{P}}(\theta_0)/d\theta_0 = 0$. Consider a Taylor expansion of the logarithm of $\tilde{\mathcal{P}}$ around θ_0

$$\log \tilde{\mathcal{P}}(\theta) = \log \tilde{\mathcal{P}}(\theta_0) - \frac{1}{2} A(\theta - \theta_0)^2 + \dots \,, \tag{5.22}$$

where

$$A = -\frac{d^2}{d\theta^2} \log \tilde{\mathcal{P}}(\theta) \bigg|_{\theta=\theta_0} \,. \tag{5.23}$$

Note that the first term of the Taylor expansion does not appear in (5.22) because $d\tilde{\mathcal{P}}(\theta_0)/d\theta_0 = 0$. Considering only up the second order term in (5.22)

$$\tilde{\mathcal{P}}(\theta) \approx \tilde{\mathcal{P}}(\theta_0) \exp\left(-\frac{A}{2}(\theta - \theta_0)^2\right) = \tilde{\mathcal{Q}}(\theta) \,, \tag{5.24}$$

which is an unnormalized Gaussian distribution (see Appendix A.3). The normalization constant of $\tilde{\mathcal{Q}}$, $Z_{\mathcal{Q}}$, can be computed using the formula for the normalization constant of the Gaussian distribution

$$Z_{\mathcal{Q}} = \left(\frac{2\pi}{A}\right)^{1/2} \tilde{\mathcal{P}}(\theta_0) \,, \tag{5.25}$$

and in consequence, the final approximation of the posterior distribution is a Gaussian with mean θ_0 and variance A^{-1}

$$\mathcal{Q}(\theta) = \left(\frac{A}{2\pi}\right)^{1/2} \exp\left(-\frac{A}{2}(\theta - \theta_0)^2\right) . \tag{5.26}$$

Note that this approximation is only well defined if $A > 0$, or similarly, if θ_0 is a maximum of $\tilde{\mathcal{P}}$. Finally, the normalization constant $Z_{\mathcal{Q}}$ can be used to approximate $\mathcal{P}(\mathcal{D})$, the normalization constant of $\tilde{\mathcal{P}}(\theta)$.

The previous approximation can be extended to a multivariate posterior distribution $\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})$. As in the single variable case, use the notation $\tilde{\mathcal{P}}(\boldsymbol{\theta}) = \mathcal{P}(\boldsymbol{\theta}, \mathcal{D})$ and assume that $\tilde{\mathcal{P}}$ has a maximum at some point $\boldsymbol{\theta}_0$. Again, expanding the logarithm of $\tilde{\mathcal{P}}$ around $\boldsymbol{\theta}_0$ we get

$$\log \tilde{\mathcal{P}}(\boldsymbol{\theta}) = \log \tilde{\mathcal{P}}(\boldsymbol{\theta}_0) - \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \mathbf{A} (\boldsymbol{\theta} - \boldsymbol{\theta}_0) + \dots , \tag{5.27}$$

where in this case $\mathbf{A}$ is the Hessian matrix of $\tilde{\mathcal{P}}$ evaluated at $\boldsymbol{\theta}_0$. Namely,

$$\mathbf{A} = -\nabla\nabla \log \tilde{\mathcal{P}}(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0} , \tag{5.28}$$

where ∇ is the gradient operator. Truncating the Taylor expansion in (5.27) at second order

$$\tilde{\mathcal{P}}(\boldsymbol{\theta}) \approx \tilde{\mathcal{P}}(\boldsymbol{\theta}_0) \exp\left(-\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \mathbf{A} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)\right) = \tilde{\mathcal{Q}}(\boldsymbol{\theta}) , \tag{5.29}$$

which is an unnormalized multivariate Gaussian distribution (see Appendix A.3). As in the univariate case, the normalization constant $Z_{\mathcal{Q}}$ of $\tilde{\mathcal{Q}}(\boldsymbol{\theta})$ can be computed using the formula for the normalization constant of a multivariate Gaussian distribution

$$Z_{\mathcal{Q}} = \tilde{\mathcal{P}}(\boldsymbol{\theta}_0) \left(\frac{(2\pi)^d}{|\mathbf{A}|}\right)^{1/2} . \tag{5.30}$$

In consequence, the final approximation of $\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})$ is

$$\mathcal{Q}(\boldsymbol{\theta}) = \left(\frac{|\mathbf{A}|}{(2\pi)^d}\right)^{1/2} \exp\left(-\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \mathbf{A} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)\right) , \tag{5.31}$$

a multivariate Gaussian distribution with mean vector $\boldsymbol{\theta}_0$ and covariance matrix $\mathbf{A}^{-1}$. In this case the restriction is that the matrix $\mathbf{A}$ has to be positive definite for the approximation to be valid. Again, the normalization constant $Z_{\mathcal{Q}}$ can be used to approximate $\mathcal{P}(\mathcal{D})$.

In the Laplace approximation the mode of the posterior distribution $\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})$ is typically found by numerical optimization. Once this maximum has been found, the Hessian matrix $\mathbf{A}$ needs to be computed. This can be implemented exactly or using approximation techniques (Bishop, 1996). The most expensive task consists in inverting the Hessian, which has a cost that is cubic in the dimension of the vector $\boldsymbol{\theta}$. The Laplace approximation has problems when the distribution to be approximated is multi-modal (Bishop, 2006). In this situation, there will be different approximations, one for each different mode. As a result of the central limit theorem, it is expected that the posterior distribution of the model parameters approaches a Gaussian distribution as the number of training instances increases. Thus, the approximation obtained by the Laplace

method should be accurate when the training data are abundant. One of the disadvantages of the Laplace approximation is that it can not be applied to approximate discrete distributions. Furthermore, it is basis dependent (Bishop, 2006; MacKay, 2003). This means that if the vector $\boldsymbol{\theta}$ is transformed by a non-linear function $\mathbf{u}(\boldsymbol{\theta})$, the approximation obtained in this transformed space and also the normalization constant $Z_{\mathcal{Q}}$ will be different. Nevertheless, the most important limitation of the Laplace approximation is that it is based solely on the shape of the exact posterior distribution in the vicinity of the mode. Thus, it can fail to capture important global properties of the posterior distribution, such as the variance.

5.5.1.2 Variational Inference

Variational inference approximates the posterior distribution of the model $\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})$ using a simple distribution $\mathcal{Q}(\boldsymbol{\theta})$ for which the required computations are tractable (Bishop, 2006; Jaakkola, 2001; MacKay, 2003). In variational inference the approximation problem is transformed into an optimization problem. For this purpose, the following decomposition of the logarithm of the marginal likelihood of the observed data is used

$$\log \mathcal{P}(\mathcal{D}) = \mathcal{L}(\mathcal{Q}) + \mathrm{KL}\left(\mathcal{Q}||\mathcal{P}\right) , \tag{5.32}$$

where $\mathcal{Q}$ is some parametric distribution that will be used to approximate $\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})$ and where we have defined

$$\mathcal{L}(\mathcal{Q}) = \int \mathcal{Q}(\boldsymbol{\theta}) \log \left(\frac{\mathcal{P}(\mathcal{D}, \boldsymbol{\theta})}{\mathcal{Q}(\boldsymbol{\theta})} \right) d\boldsymbol{\theta} , \tag{5.33}$$

$$\mathrm{KL}\left(\mathcal{Q}||\mathcal{P}\right) = \int \mathcal{Q}(\boldsymbol{\theta}) \log \left(\frac{\mathcal{Q}(\boldsymbol{\theta})}{\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})} \right) d\boldsymbol{\theta} . \tag{5.34}$$

$\mathrm{KL}\left(\mathcal{Q}||\mathcal{P}\right)$ is the Kullback-Leibler (KL) divergence between the probability distributions $\mathcal{Q}(\boldsymbol{\theta})$ and $\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})$. This divergence takes only non-negative values, being equal to zero if and only if the two probability distributions coincide. In consequence, from the decomposition in (5.32) it follows that $\mathcal{L}(\mathcal{Q}) \leq \log \mathcal{P}(\mathcal{D})$. Thus, $\mathcal{L}(\mathcal{Q})$ is a lower bound on $\log \mathcal{P}(\mathcal{D})$. Note that $\mathcal{L}(\mathcal{Q})$ and $\mathrm{KL}\left(\mathcal{Q}||\mathcal{P}\right)$ have to add up to $\log \mathcal{P}(\mathcal{D})$. Figure 5.3 illustrates this decomposition.

We can now approximate the posterior distribution $\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})$ by optimizing the lower bound $\mathcal{L}(\mathcal{Q})$ with respect to $\mathcal{Q}$. This is equivalent to minimizing the KL divergence between these two probability distributions, as described by (5.32). If the form of $\mathcal{Q}$ is not restricted, the solution to the optimization problem is $\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})$. In practice, $\mathcal{Q}$ is restricted to have a simple parametric form.

Another possibility is to assume that the posterior approximation $\mathcal{Q}$ factorizes with respect to some disjoint groups of parameters $\boldsymbol{\theta}_i$, $i = 1, \ldots, k$. In this situation we can express $\mathcal{Q}$ as

$$\mathcal{Q}(\boldsymbol{\theta}) = \prod_{i=1}^{k} \mathcal{Q}_i(\boldsymbol{\theta}_i) . \tag{5.35}$$

This factorized form of the approximation is known as *variational mean field* (Jaakkola, 2001). We can now optimize the lower bound $\mathcal{L}(\mathcal{Q})$ with respect to each term $\mathcal{Q}_i$ in (5.35). For this purpose, we substitute (5.35) in (5.33) and extract the dependence of

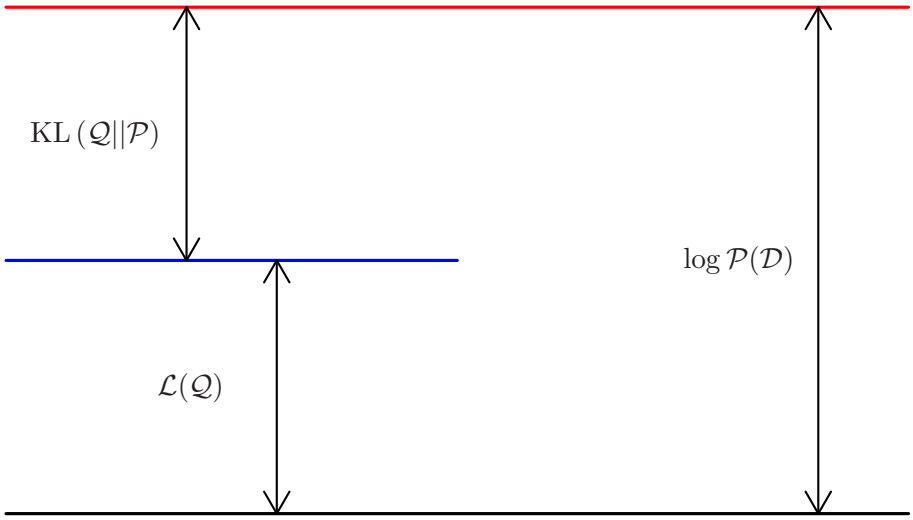

FIGURE 5.3: Decomposition of $\log \mathcal{P}(\mathcal{D})$ into the two terms $\mathcal{L}(\mathcal{Q})$ and $\mathrm{KL}\,(\mathcal{Q}||\mathcal{P})$. Adapted from (Bishop, 2006).

the lower bound with respect to an arbitrary factor $\mathcal{Q}_j$

$$\begin{aligned}
\mathcal{L}(\mathcal{Q}) &= \int \prod_{i=1}^{k} \mathcal{Q}_i(\boldsymbol{\theta}_i) \left[\log \mathcal{P}(\mathcal{D}, \boldsymbol{\theta}) - \sum_{i=1}^{k} \log \mathcal{Q}_i(\boldsymbol{\theta}_i) \right] d\boldsymbol{\theta} \\
&= \int \mathcal{Q}_j(\boldsymbol{\theta}_j) \left[\log \mathcal{P}(\mathcal{D}, \boldsymbol{\theta}) \prod_{i \neq j} \mathcal{Q}(\boldsymbol{\theta}_i) d\boldsymbol{\theta}_i \right] d\boldsymbol{\theta}_j - \int \mathcal{Q}_j(\boldsymbol{\theta}_j) \log \mathcal{Q}_j(\boldsymbol{\theta}_j) d\boldsymbol{\theta}_j \\
&\quad - \sum_{i \neq j} \int \mathcal{Q}_i(\boldsymbol{\theta}_i) \log \mathcal{Q}_i(\boldsymbol{\theta}_i) d\boldsymbol{\theta}_i \\
&= \int \mathcal{Q}_j(\boldsymbol{\theta}_j) \log \hat{\mathcal{P}}(\mathcal{D}, \boldsymbol{\theta}_j) d\boldsymbol{\theta}_j - \int \mathcal{Q}_j(\boldsymbol{\theta}_j) \log \mathcal{Q}_j(\boldsymbol{\theta}_j) d\boldsymbol{\theta}_j + C\,, \qquad (5.36)
\end{aligned}$$

where C is a constant value independent of $\mathcal{Q}_j$ that summarizes the entropies of the terms $\prod_{i \neq j} \mathcal{Q}_i$ plus the logarithm of the normalization constant of $\hat{\mathcal{P}}(\mathcal{D}, \boldsymbol{\theta}_j)$, defined as

$$\begin{aligned}
\log \hat{\mathcal{P}}(\mathcal{D}, \boldsymbol{\theta}_j) &= \int \log \mathcal{P}(\mathcal{D}, \boldsymbol{\theta}) \prod_{i \neq j} \mathcal{Q}_i(\boldsymbol{\theta}_i) d\boldsymbol{\theta}_i + \hat{C} \\
&= \mathbb{E}_{i \neq j} \left[\log \mathcal{P}(\mathcal{D}, \boldsymbol{\theta}) \right] + \hat{C}\,, \qquad (5.37)
\end{aligned}$$

where $\hat{C}$ is the negative of the logarithm of the normalization constant of $\hat{\mathcal{P}}(\mathcal{D}, \boldsymbol{\theta}_j)$. To maximize $\mathcal{L}(\mathcal{Q})$ with respect to $\mathcal{Q}$, we follow a component wise procedure in which we optimize with respect to $\mathcal{Q}_j$ keeping $\mathcal{Q}_i$, with $i \neq j$ fixed. This can be easily done if we note that (5.36) is the negative value of the KL divergence between $\mathcal{Q}_j$ and $\hat{\mathcal{P}}$ plus some constant terms. Thus, $\mathcal{L}(\mathcal{Q})$ is maximized with respect to $\mathcal{Q}_j$ by setting $\mathcal{Q}_j = \hat{\mathcal{P}}$

$$\log \mathcal{Q}_j(\boldsymbol{\theta}_j) = \mathbb{E}_{i \neq j} \left[\log \mathcal{P}(\mathcal{D}, \boldsymbol{\theta}) \right] + \hat{C}\,, \qquad (5.38)$$

That is, the optimal value for $\mathcal{Q}_j$ is obtained by taking the logarithm of the joint probability distribution $\mathcal{P}(\mathcal{D}, \boldsymbol{\theta})$ and then computing the expected values with respect the

other factors $\mathcal{Q}_i$, with $i \neq j$. By taking the exponential at both sides we find that

$$\mathcal{Q}_j(\boldsymbol{\theta}_j) = \frac{1}{Z_{\mathcal{Q}_j}} \exp\left(\mathbb{E}_{i \neq j}\left[\log \mathcal{P}(\mathcal{D}, \boldsymbol{\theta})\right]\right) , \tag{5.39}$$

where $Z_{\mathcal{Q}_j}$ is a normalization constant. By successively updating each term $\mathcal{Q}_j$ in this way we can iteratively maximize the lower bound $\mathcal{L}(\mathcal{Q})$. Unfortunately, even though the optimization of $\mathcal{L}(\mathcal{Q})$ with respect to $\mathcal{Q}_j$ is convex, the lower bound is not jointly convex with respect to $\mathcal{Q}$. This means that the global optimization process can end up in a local maximum. This maximum depends on the initial parameters of the factors $\mathcal{Q}_i$ and the order in which the updates are carried out (Jaakkola, 2001). Once the algorithm has converged, the lower bound $\mathcal{L}(\mathcal{Q})$ can be used to approximate the logarithm of the model evidence $\log \mathcal{P}(\mathcal{D})$, which can be useful for model comparison.

The technique presented in this section is suited to approximate distributions that simplify when taking the logarithm. In particular, we must be able to compute the expectations under the approximate distribution of the logarithm of the unnormalized distribution. Otherwise, the evaluation of the lower bound is not possible. When this is the case, it might still be possible to compute a less stringent lower bound on $\log \mathcal{P}(\mathcal{D})$ by lower bounding some of the terms that do not simplify. This can be achieved by parameterizing these lower bounds and then optimizing these parameters to make the bounds tighter (Jaakkola and Jordan, 2000). Variational inference is independent of the parameterization used. This is so because the KL divergence is invariant with respect to arbitrary transformations of the parameters. Thus, this method avoids the problem of basis dependence of the Laplace approximation (MacKay, 2003). Typically, the approximate distributions obtained by variational inference tend to be more compact that the actual distributions. This is a consequence of minimizing $\mathrm{KL}(\mathcal{Q}|\mathcal{P})$ instead of $\mathrm{KL}(\mathcal{P}|\mathcal{Q})$ (Bishop, 2006). In particular, $\mathrm{KL}(\mathcal{Q}|\mathcal{P})$ takes large values in regions of the space of parameters in which $\mathcal{P}$ is close to zero and $\mathcal{Q}$ is not. The opposite effect is observed in the minimization of $\mathrm{KL}(\mathcal{P}|\mathcal{Q})$. In this case, the resulting approximation $\mathcal{Q}$ tends to have non-zero density values where $\mathcal{P}$ is close to zero (MacKay, 2003). Thus, in a multi-modal setting the approximate distribution $\mathcal{Q}$ that is obtained from minimizing the lower bound is typically placed under one of the modes, like the Laplace approximation (Bishop, 2006). Variational inference has proven useful in a wide range of applications (Attias, 2000; Bishop and Svensen, 2003; Bishop et al., 2002; Blei and Jordan, 2006; Gibbs and MacKay, 2000; Jaakkola and Jordan, 2000; Lawrence, 2000) and often performs better and even faster than the Laplace method or Monte Carlo sampling approaches (Minka, 2001b).

5.5.1.3 Expectation Propagation

Expectation propagation (EP) (Minka, 2001a,b) approximates the posterior distribution of the model parameters using a simpler distribution $\mathcal{Q}$. Assume that $\mathcal{P}$ is a target distribution. EP minimizes the Kullback-Leibler (KL) divergence between $\mathcal{P}$ and $\mathcal{Q}$

$$\mathrm{KL}\left(\mathcal{P}||\mathcal{Q}\right) = \int \mathcal{P}(\boldsymbol{\theta}) \log\left(\frac{\mathcal{P}(\boldsymbol{\theta})}{\mathcal{Q}(\boldsymbol{\theta})}\right) d\boldsymbol{\theta} \, . \tag{5.40}$$

Expression (5.40) takes only non-negative values, and is equal to zero if and only if the two probability distributions $\mathcal{Q}$ and $\mathcal{P}$ coincide.

Assume that the probability distribution $\mathcal{Q}$ belongs to the exponential family $\mathcal{F}$ of probability distributions, i.e. $\mathcal{Q}$ can be written as

$$\mathcal{Q}(\boldsymbol{\theta}) = \exp\left(\boldsymbol{\eta}^T \mathbf{u}(\boldsymbol{\theta}) - g(\boldsymbol{\eta})\right) , \tag{5.41}$$

where the superscript T means transpose, $\boldsymbol{\eta}$ is a vector of *natural parameters*, $\mathbf{u}(\boldsymbol{\theta})$ is some vector function of $\boldsymbol{\theta}$ known as the *sufficient statistics* and $g(\boldsymbol{\eta})$ is a log partition function that guarantees that $\mathcal{Q}$ integrates to one. Note that because of the exponential form of (5.41) the family is closed under product and division. As a function of $\boldsymbol{\eta}$, the KL divergence (5.40) is

$$\mathrm{KL}\left(\mathcal{P}||\mathcal{Q}\right) = g(\boldsymbol{\eta}) - \boldsymbol{\eta}^T \mathbb{E}_{\mathcal{P}}\left[\mathbf{u}(\boldsymbol{\theta})\right] + \mathcal{C} , \tag{5.42}$$

where $\mathcal{C}$ is a constant value independent of the natural parameters $\boldsymbol{\eta}$. In this situation, minimizing $\mathrm{KL}\left(\mathcal{P}||\mathcal{Q}\right)$ with respect to the parameters of $\mathcal{Q}$ is equivalent to setting the gradient of (5.42) with respect to $\boldsymbol{\eta}$ to zero

$$\frac{\mathrm{KL}\left(\mathcal{P}||\mathcal{Q}\right)}{\partial \boldsymbol{\eta}} = 0 \Longleftrightarrow \frac{\partial g(\boldsymbol{\eta})}{\partial \boldsymbol{\eta}} = \mathbb{E}_{\mathcal{P}}\left[\mathbf{u}(\boldsymbol{\theta})\right] . \tag{5.43}$$

It is easy to show that the gradient of $g(\boldsymbol{\eta})$ is also given by the expectation of $\mathbf{u}(\boldsymbol{\theta})$ under the distribution $\mathcal{Q}$. In consequence, (5.43) is equivalent to

$$\mathbb{E}_{\mathcal{Q}}\left[\mathbf{u}(\boldsymbol{\theta})\right] = \mathbb{E}_{\mathcal{P}}\left[\mathbf{u}(\boldsymbol{\theta})\right] . \tag{5.44}$$

Hence, the optimal $\mathcal{Q}$ can be obtained by matching the expected sufficient statistics under $\mathcal{Q}$ and $\mathcal{P}$. This result is systematically used in the EP algorithm to carry out approximate inference in an efficient way.

We now proceed to describe the EP algorithm in its general form. For many probabilistic models, the joint probability function of a set of i.i.d. data instances, $\mathcal{D}$, and the model parameters, $\boldsymbol{\theta}$, can be written as a product of several terms. That is,

$$\mathcal{P}(\mathcal{D}, \boldsymbol{\theta}) = \prod_i t_i(\boldsymbol{\theta}) . \tag{5.45}$$

There are many ways to achieve this factorized form. As an example, consider a linear ridge regression model with n observed data instances. We can associate one term $t_i(\boldsymbol{\theta})$ with each factor $\mathcal{P}(y_i|\mathbf{x}_i, \boldsymbol{\theta})$ of the likelihood and one additional term $t_{n+1}(\boldsymbol{\theta})$ with the prior for $\boldsymbol{\theta}$, $\mathcal{P}(\boldsymbol{\theta})$. In principle, we are interested in computing a posterior distribution $\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})$ for making predictions and the model evidence $\mathcal{P}(\mathcal{D})$ for model comparison. These quantities are computed from (5.45) as follows

$$\mathcal{P}(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{\mathcal{P}(\mathcal{D})} \prod_i t_i(\boldsymbol{\theta}) , \tag{5.46}$$

$$\mathcal{P}(\mathcal{D}) = \int \prod_i t_i(\boldsymbol{\theta}) d\boldsymbol{\theta} . \tag{5.47}$$

EP approximates the joint distribution of $\mathcal{D}$ and $\boldsymbol{\theta}$ by a product of simple terms

$$\mathcal{P}(\mathcal{D}, \boldsymbol{\theta}) \approx \prod_i \tilde{t}_i(\boldsymbol{\theta}) , \tag{5.48}$$

where each term $\tilde{t}_i$ in the approximation corresponds to one term t_i in the true joint distribution and the approximate terms $\tilde{t}_i$ are constrained to have a similar form. In particular, they all have to belong to the same family of exponential distributions, although they do not have to be normalized. The posterior distribution for $\boldsymbol{\theta}$ is then approximated by a simple distribution $\mathcal{Q}$ that is computed as follows

$$\mathcal{Q}(\boldsymbol{\theta}) = \frac{1}{Z} \prod_i \tilde{t}_i(\boldsymbol{\theta}) \,, \tag{5.49}$$

where

$$Z = \int \prod_i \tilde{t}_i(\boldsymbol{\theta}) d\boldsymbol{\theta} \tag{5.50}$$

is a normalization constant that guarantees that $\mathcal{Q}$ integrates to one. Because of the closure property of the exponential family, $\mathcal{Q}$ has a form that is similar to the approximate terms $\tilde{t}_i$. As an example, consider that the approximate terms $\tilde{t}_i$ have a Gaussian form, then, the posterior approximation $\mathcal{Q}$ will also be Gaussian. In practice, the form of $\mathcal{Q}$ is selected first and the approximate terms $\tilde{t}_i$ are constrained to have the same form as $\mathcal{Q}$, although they do not have to be normalized.

Ideally, the EP algorithm should determine the approximate terms $\tilde{t}_i$ by minimizing the KL divergence between the true posterior $\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})$ and the approximation $\mathcal{Q}$. However, this is intractable because as described by (5.44) it involves computing an average over $\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})$, which is infeasible. As an approximation, EP performs a sequential optimization by minimizing the KL divergence between the pairs of terms t_i and $\tilde{t}_i$. Until convergence of the approximate terms $\tilde{t}_i$, at each step of the algorithm EP updates the term $\tilde{t}_i$ so that the product

$$\tilde{t}_i(\boldsymbol{\theta}) \prod_{j \neq i} \tilde{t}_j(\boldsymbol{\theta}) \tag{5.51}$$

is as close as possible to the product

$$t_i(\boldsymbol{\theta}) \prod_{j \neq i} \tilde{t}_j(\boldsymbol{\theta}) \tag{5.52}$$

in terms of the KL divergence. This procedure ensures that the approximate term $\tilde{t}_i$ is accurate in regions of high posterior probability as defined by the remaining terms (Minka, 2001b).

The EP algorithm involves the following steps:

1. Initialize all the approximate terms $\tilde{t}_i$ and $\mathcal{Q}$ to be uniform.
2. Repeat until all $\tilde{t}_i$ converge:
   (a) Select an approximate term $\tilde{t}_i$ to update and remove $\tilde{t}_i$ from $\mathcal{Q}$ by dividing and normalizing:
   $$\mathcal{Q}^{\setminus i}(\boldsymbol{\theta}) \propto \frac{\mathcal{Q}(\boldsymbol{\theta})}{\tilde{t}_i(\boldsymbol{\theta})} \,. \tag{5.53}$$
   (b) Compute an updated posterior distribution $\hat{\mathcal{P}}$:
   $$\hat{\mathcal{P}}(\boldsymbol{\theta}) = \frac{1}{Z_i} t_i(\boldsymbol{\theta}) \mathcal{Q}^{\setminus i}(\boldsymbol{\theta}) \,, \tag{5.54}$$

where Z_i is a normalization constant needed to ensure that $\hat{\mathcal{P}}$ integrates to one.

(c) Update the posterior approximation $\mathcal{Q}$:

$$\mathcal{Q}^{\text{new}} = \arg\min_{\mathcal{Q}} \text{KL}\left(\hat{\mathcal{P}}||\mathcal{Q}\right) . \tag{5.55}$$

(d) Update $\tilde{t}_i$ by setting

$$\tilde{t}_i(\boldsymbol{\theta}) = Z_i \frac{\mathcal{Q}^{\text{new}}(\boldsymbol{\theta})}{\mathcal{Q}^{\setminus i}(\boldsymbol{\theta})} . \tag{5.56}$$

3. Evaluate the approximation to the model evidence:

$$\mathcal{P}(\mathcal{D}) \approx \int \prod_i \tilde{t}_i(\boldsymbol{\theta}) d\boldsymbol{\theta} . \tag{5.57}$$

In the case of a Gaussian distribution, the first step of the algorithm is implemented by setting the mean and the variance of all the approximate terms $\tilde{t}_i$ and the posterior approximation $\mathcal{Q}$ equal to zero and $+\infty$, respectively. Step 2-(a) can be carried out by computing $\prod_{j \neq i} \tilde{t}_j$ and normalizing. However, division is usually faster. Note that $\mathcal{Q}^{\setminus i}$ has the same form as $\mathcal{Q}$ as a consequence of the closure property of the exponential family. Furthermore, because $\mathcal{Q}$ is in the exponential family, the optimization problem of step 2-(c) is performed by matching the sufficient statistics of $\hat{\mathcal{P}}$ and $\mathcal{Q}$, as described by (5.44). Thus, the algorithm only requires these integrals to be tractable. Step 2-(d) guarantees that $\tilde{t}_i \mathcal{Q}^{\setminus i}$ integrates to the same value as $t_i \mathcal{Q}^{\setminus i}$ and also that $\mathcal{Q}^{\text{new}}$ is obtained when we compute the product $\prod_i \tilde{t}_i$ and normalize. The approximate terms $\tilde{t}_i$ are often selected so that they have the same form as the prior for $\boldsymbol{\theta}$, $\mathcal{P}(\boldsymbol{\theta})$. In this situation, the update of the approximate term $\tilde{t}_{n+1}$ corresponding to the prior for $\boldsymbol{\theta}$ is not required because the optimal update is always to set $\tilde{t}_{n+1}$ equal to $\mathcal{P}(\boldsymbol{\theta})$. Furthermore, if the posterior approximation $\mathcal{Q}$ is initialized to the prior, the term t_{n+1} can be ignored in the main loop of the EP algorithm (Minka, 2001b).

The EP algorithm is not guaranteed to converge, although extensive empirical evaluation has shown that most of the times it is convergent (Minka, 2001b). Furthermore, if the approximation $\mathcal{Q}$ is in the exponential family and the algorithm does converge, the resulting solution is a minimum of a particular energy function (Minka, 2001a), although each iteration of EP does not necessarily decrease the value of this energy function. To guarantee convergence, there is a generalized version of EP based on a double loop algorithm (Heskes and Zoeter, 2002). EP has demonstrated to have an excellent performance when compared to other approximate inference techniques such as Monte Carlo methods, variational inference and the Laplace approximation (Minka, 2001b). Nevertheless, because of the moment matching procedure of the algorithm, EP can lead to poor approximations when the posterior is multi-modal and the form assumed for $\mathcal{Q}$ is uni-modal (Bishop, 2006). The typical scenario for such poor behavior is a bimodal posterior distribution that is approximated by a Gaussian distribution. EP places the mean of the Gaussian between the two modes of the true posterior, where the probability can be rather low.

We now illustrate the EP algorithm with a simple problem. Consider a model with a single parameter θ. Assume that the joint probability function for the observed data

and the model parameter $\mathcal{P}(\mathcal{D}, \theta)$ is

$$\mathcal{P}(\mathcal{D}, \theta) = \Phi(5\theta) \exp(-\frac{1}{2}\theta^2) \,, \tag{5.58}$$

where $\Phi(\cdot)$ is the cumulative probability function of a standard Gaussian distribution. To use EP for approximating the posterior distribution of the model parameter $\mathcal{P}(\theta|\mathcal{D})$, we factorize (5.58) as the product of two terms, $t_1(\theta) = \Phi(5\theta)$ and $t_2(\theta) = \exp(-\theta^2/2)$. Next, we restrict the approximation $\mathcal{Q}$ to be a simple distribution from the exponential family. In particular, we assume $\mathcal{Q}$ is a Gaussian distribution with mean μ and variance ν

$$\mathcal{Q}(\theta) = \mathcal{N}(\theta|\mu, \nu) \,. \tag{5.59}$$

Because the Gaussian distribution belongs to the exponential family (see Appendix A.3), the approximate terms $\tilde{t}_1$ and $\tilde{t}_2$ will also have a Gaussian form, although they do not have to integrate to one

$$\tilde{t}_i(\theta) = s_i \exp(-\frac{1}{2v_i}(\theta - m_i)^2) \,, \tag{5.60}$$

where s_i, m_i and v_i are free parameters. We now derive the EP update equations for this problem. The first step of the EP algorithm consists in initializing $\mathcal{Q}$ and all the $\tilde{t}_i$ terms to be uniform. This is achieved by setting the mean and the variance of all $\tilde{t}_i$ and $\mathcal{Q}$ to zero and infinity, respectively. The next step is computing $\mathcal{Q}^{\backslash i}$ from $\mathcal{Q}$. Recall that $\mathcal{Q}^{\backslash i}$ has the same form as $\mathcal{Q}$ (i.e. a Gaussian distribution) because of the closure property of the exponential family. Let $\mu^{\backslash i}$ and $\nu^{\backslash i}$ be the parameters of $\mathcal{Q}^{\backslash i}$. As described by (5.53), $\mathcal{Q}^{\backslash i}$ is the quotient between two Gaussian distributions, i.e. the posterior approximation $\mathcal{Q}$ and $\tilde{t}_i$, although $\tilde{t}_i$ is not normalized. Using (A.46) we find that

$$\nu^{\backslash i} = \left(\nu^{-1} - v_i^{-1}\right)^{-1} \,, \qquad \mu^{\backslash i} = \nu^{\backslash i}\left(\nu^{-1}\mu - v_i^{-1}m_i\right) \,. \tag{5.61}$$

The next step consists in computing Z_i, the normalization constant of $\hat{\mathcal{P}}(\theta)$, as defined in (5.54), for $i = 1, 2$. This constant depends on the term t_i that is being processed. In particular,

$$Z_i = \begin{cases} \int \Phi(5\theta)\mathcal{N}(\theta|\mu^{\backslash i}, \nu^{\backslash i})d\theta = \Phi\left(\frac{\mu^{\backslash i}}{\sqrt{5^{-2}+\nu^{\backslash i}}}\right) & \text{if } i = 1 \,, \\ \int \exp(-\frac{1}{2}\theta^2)\mathcal{N}(\theta|\mu^{\backslash i}, \nu^{\backslash i})d\theta = \frac{1}{\sqrt{\nu^{\backslash i}+1}} \exp\left(-\frac{1}{2}\frac{(\mu^{\backslash i})^2}{\nu^{\backslash i}+1}\right) & \text{if } i = 2 \,. \end{cases} \tag{5.62}$$

To compute $\mathcal{Q}^{\text{new}}$ we have to match the sufficient statistics between $\mathcal{Q}$ and $\hat{\mathcal{P}}(\theta)$. Specifically, because $\mathcal{Q}$ is Gaussian, we have to match the mean and variance between these two distributions (see Appendix A.3). The mean and the variance of $\hat{\mathcal{P}}(\theta)$ can be computed from Z_i using (A.40) and (A.41), respectively. Assume that $z = \mu^{\backslash i}/\sqrt{5^{-2} + \nu^{\backslash i}}$ and that $\alpha = \mathcal{N}(z|0, 1)/(\Phi(z)\sqrt{5^{-2} + \nu^{\backslash i}})$, the parameters of $\mathcal{Q}^{\text{new}}$ are in the case of t_1

$$\mu^{\text{new}} = \mu^{\backslash i} + \nu^{\backslash i}\alpha \,, \tag{5.63}$$

$$\nu^{\text{new}} = \nu^{\backslash i} - \alpha(\nu^{\backslash i})^2\left(\alpha + \frac{\mu^{\backslash i}}{5^{-2} + \nu^{\backslash i}}\right) \,, \tag{5.64}$$

and in the case of t_2

$$\mu^{\text{new}} = \frac{\mu^{\setminus i}}{\nu^{\setminus i} + 1}\,, \tag{5.65}$$

$$\nu^{\text{new}} = \frac{\nu^{\setminus i}}{\nu^{\setminus i} + 1}\,. \tag{5.66}$$

Once we have an updated posterior distribution $\mathcal{Q}^{\text{new}}$, the last step of the algorithm is to update the corresponding approximate term $\tilde{t}_i$ using (5.56). Because $\tilde{t}_i$ is computed as the quotient between the Gaussian distributions $\mathcal{Q}^{\text{new}}$ and $\mathcal{Q}^{\setminus i}$, we can use (A.46) to find that

$$v_i = \left((\nu^{\text{new}})^{-1} - (\nu^{\setminus i})^{-1}\right)^{-1}\,, \tag{5.67}$$

$$m_i = v_i \left((\nu^{\text{new}})^{-1}\mu^{\text{new}} - (\nu^{\setminus i})^{-1}\mu^{\setminus i}\right)\,, \tag{5.68}$$

$$s_i = Z_i \sqrt{\frac{\nu^{\setminus i}}{\nu^{\text{new}}}} \exp\left(-\frac{1}{2}\left(\frac{(\mu^{\text{new}})^2}{\nu^{\text{new}}} - \frac{(\mu^{\setminus i})^2}{\nu^{\setminus i}} - \frac{m_i^2}{v_i}\right)\right)\,. \tag{5.69}$$

Using these update rules we can iterate the EP algorithm until convergence of the approximate terms $\tilde{t}_i$. Figure 5.4 displays the posterior approximation $\mathcal{Q}$ that is obtained by the EP algorithm in this simple problem. The exact posterior distribution $\mathcal{P}(\theta|\mathcal{D})$ is also displayed in the picture as a reference. We note that, as in variational inference, the resulting Gaussian distribution need not be placed at the mode of the true posterior. Once EP has converged, we can compute (5.57) to approximate $\mathcal{P}(\mathcal{D})$. For this task we have to integrate the product of the approximate terms $\tilde{t}_i$, which is the product of several Gaussian functions and hence, is also Gaussian. In particular, (A.42) can be used to give

$$\mathcal{P}(\mathcal{D}) \approx \int \tilde{t}_1(\theta)\tilde{t}_2(\theta)d\theta = s_1 s_2 \sqrt{2\pi\nu} \exp\left(-\frac{1}{2}\left(\frac{m_1^2}{v_1} + \frac{m_2^2}{v_2} - \frac{\mu^2}{\nu}\right)\right)\,. \tag{5.70}$$

In this problem the exact value of $\mathcal{P}(\mathcal{D})$ is 1.253, which is the same value estimated by EP in (5.70).

5.5.2 Sampling Methods

Most of the methods described in this section employ Markov chains whose stationary distribution is the posterior distribution of the model parameters given the observed data $\mathcal{P}(\boldsymbol{\theta}|\mathcal{D})$ (Bishop, 2006; MacKay, 2003). Thus, before describing these techniques in detail, we first introduce Markov chains. Define $\mathcal{S}$ to be a set of states and let $\mathbf{z}^{(n)} \in \mathcal{S}$ be a random variable that represents a fixed state at stage n. The sequence $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \mathbf{z}^{(3)}, \ldots$ is a Markov chain of first order if

$$\mathcal{P}(\mathbf{z}^{(n)}|\mathbf{z}_1, \ldots, \mathbf{z}^{(n-1)}) = \mathcal{P}(\mathbf{z}^{(n)}|\mathbf{z}^{(n-1)}) \tag{5.71}$$

holds for all $n \geq 2$. The Markov chain is hence fully determined by the initial probability distribution $\mathcal{P}(\mathbf{z}^{(1)})$ and the conditional transition probabilities $\mathcal{P}(\mathbf{z}^{(n)}|\mathbf{z}^{(n-1)})$. A Markov chain is said to be *homogeneous* if the transition probabilities do not depend on n, i.e. they only depend on the current state and the new state. The marginal

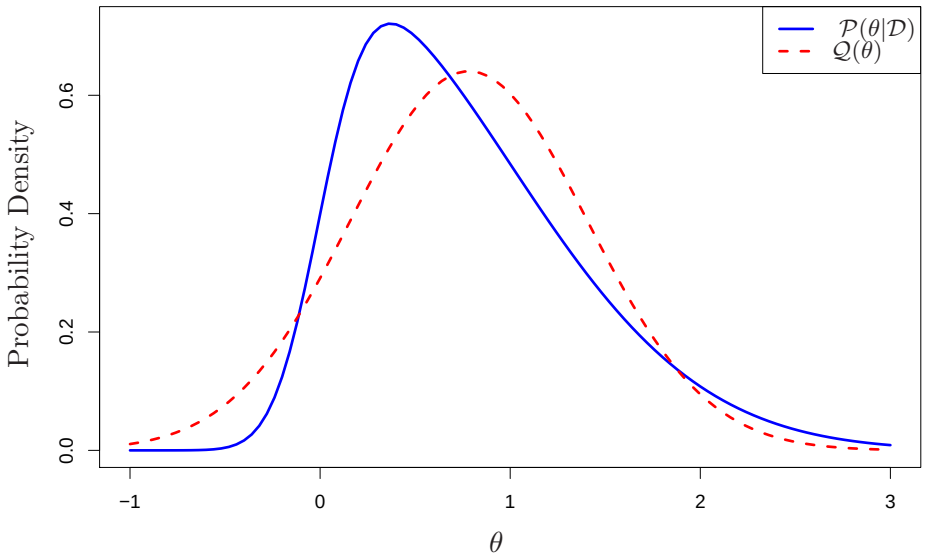

FIGURE 5.4: Exact posterior distribution $\mathcal{P}(\theta|\mathcal{D})$ obtained by normalizing $\mathcal{P}(\mathcal{D},\theta)$ alongside with the approximation $\mathcal{Q}$ obtained by the EP algorithm.

probability at stage n is

$$\mathcal{P}(\mathbf{z}^{(n)}) = \sum_{\mathbf{z}^{(n-1)}} \mathcal{P}(\mathbf{z}^{(n)}|\mathbf{z}^{(n-1)})\mathcal{P}(\mathbf{z}^{(n-1)})\,. \tag{5.72}$$

Let $\mathcal{S}$ be the space of sates of a Markov chain and $\mathcal{P}^\star(\mathbf{z})$, with $\mathbf{z} \in \mathcal{S}$, a probability distribution over $\mathcal{S}$. $\mathcal{P}^\star(\mathbf{z})$ is the stationary distribution of the Markov chain if

$$\mathcal{P}^\star(\mathbf{z}) = \sum_{\mathbf{z}' \in \mathcal{S}} \mathcal{P}^\star(\mathbf{z}')\mathcal{P}(\mathbf{z}|\mathbf{z}')\,. \tag{5.73}$$

A condition that ensures that a probability distribution is the stationary distribution is the *detailed balance* property

$$\mathcal{P}^\star(\mathbf{z})\mathcal{P}(\mathbf{z}'|\mathbf{z}) = \mathcal{P}^\star(\mathbf{z}')\mathcal{P}(\mathbf{z}|\mathbf{z}')\,. \tag{5.74}$$

This can be proved by marginalizing $\mathbf{z}'$ in (5.74). However, note that there can be stationary distributions that do not satisfy (5.74).

Finally, if a Markov chain is homogeneous then it can be proved that a stationary distribution exists and that

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\left(\mathbf{z}^{(i)} = \mathbf{z}\right) \to \mathcal{P}^\star(\mathbf{z}) \quad \text{as } n \to \infty \quad \text{a.s.}\,, \tag{5.75}$$

where $\mathbb{I}(z)$ is an indicator function that takes value one when z is satisfied and zero otherwise, under weak restrictions on the stationary distribution. A similar analysis is possible for a continuous space of states $\mathcal{S}$. See (Neal, 1993) for further details.

The techniques described below employ Markov chains to sample from probability distributions. For this purpose, they build a Markov chain whose stationary distribution is the distribution we are interested in sampling from. Then, expectations under the stationary distribution are approximated by averages computed over the samples

generated from the Markov chain. Typically, the first samples generated from the chain are discarded to give time to the chain to reach the stationary regime. As then number of samples generated from the Markov chain increase, these methods become exact. However, with the computational resources that are commonly available, it is only feasible to draw a few thousand samples from the chain. In consequence, the sampling algorithms described below can only provide approximate solutions. A more serious difficulty is that the samples obtained are correlated, which means that the variance of the estimates obtained by computing the averages can be rather large.

5.5.2.1 Metropolis-Hastings

The Metropolis-Hastings algorithm can be used to approximately sample from an unnormalized probability distribution $\tilde{\mathcal{P}}(\mathbf{z})$ (Bishop, 2006; MacKay, 2003). For this purpose, this algorithm introduces an arbitrary proposal probability distribution $\mathcal{Q}(\mathbf{z}|\mathbf{z}')$, which needs not be similar to the normalized version of $\tilde{\mathcal{P}}(\mathbf{z})$. A typical choice for this probability density is a Gaussian distribution with mean equal to $\mathbf{z}'$. The Metropolis-Hastings algorithm builds a Markov chain whose stationary distribution is the normalized version of $\tilde{\mathcal{P}}(\mathbf{z})$,

$$\mathcal{P}(\mathbf{z}) = \frac{1}{Z}\tilde{\mathcal{P}}(\mathbf{z})\,, \tag{5.76}$$

where Z is the normalization constant. Let n be the current stage of the Markov chain and let $\mathbf{z}^{(n)}$ be the current state. A candidate state $\mathbf{z}^\star$ is generated from $\mathcal{Q}$ conditioned to $\mathbf{z}^{(n)}$, i.e. $\mathbf{z}^\star \sim \mathcal{Q}(\mathbf{z}|\mathbf{z}^{(n)})$. This candidate state is accepted with probability

$$\mathcal{A}(\mathbf{z}^\star|\mathbf{z}^{(n)}) = \min\left(1, \frac{\mathcal{P}(\mathbf{z}^\star)\mathcal{Q}(\mathbf{z}^{(n)}|\mathbf{z}^\star)}{\mathcal{P}(\mathbf{z}^{(n)})\mathcal{Q}(\mathbf{z}^\star|\mathbf{z}^{(n)})}\right). \tag{5.77}$$

If the new state is accepted we set $\mathbf{z}^{(n+1)} = \mathbf{z}^\star$, otherwise we set $\mathbf{z}^{(n+1)} = \mathbf{z}^{(n)}$. Note that a rejection causes the current state to be repeated in the chain. To compute (5.77) the normalization constant of $\tilde{\mathcal{P}}$ is not required since it cancels out when evaluating the quotient. Furthermore, if the proposal density $\mathcal{Q}$ is a simple symmetrical density such as the Gaussian distribution, the ratio $\mathcal{Q}(\mathbf{z}^{(n)}|\mathbf{z}^\star)/\mathcal{Q}(\mathbf{z}^\star|\mathbf{z}^n)$ is one and can also be ignored.

We now show that if a Markov chain is generated as described, the distribution $\mathcal{P}(\mathbf{z})$ is the stationary distribution of the chain. For this purpose we employ the transition probabilities of the Markov chain, which are given by (5.77) times the probability of generating the new state using the proposal distribution $\mathcal{Q}$, and show that $\mathcal{P}(\mathbf{z})$ satisfies the detailed balance property of the stationary distribution, i.e.

$$\begin{aligned}
\mathcal{P}(\mathbf{z})\mathcal{Q}(\mathbf{z}|\mathbf{z}')\mathcal{A}(\mathbf{z}|\mathbf{z}') &= \min\left(\mathcal{P}(\mathbf{z})\mathcal{Q}(\mathbf{z}|\mathbf{z}'), \mathcal{P}(\mathbf{z}')\mathcal{Q}(\mathbf{z}'|\mathbf{z})\right) \\
&= \min\left(\mathcal{P}(\mathbf{z}')\mathcal{Q}(\mathbf{z}'|\mathbf{z}), \mathcal{P}(\mathbf{z})\mathcal{Q}(\mathbf{z}|\mathbf{z}')\right) \\
&= \mathcal{P}(\mathbf{z}')\mathcal{Q}(\mathbf{z}'|\mathbf{z})\mathcal{A}(\mathbf{z}'|\mathbf{z})\,.
\end{aligned} \tag{5.78}$$

The Metropolis-Hastings algorithm is a widely used method for sampling from unnormalized probability distributions in high dimensional spaces. Note that for the acceptance ratio of the algorithm to be large, the value of $\tilde{\mathcal{P}}$ evaluated in the new candidate state $\mathbf{z}^\star$ should be similar or larger than the value of $\tilde{\mathcal{P}}$ evaluated in the current state $\mathbf{z}^{(n)}$. In consequence, the variance of the candidate distribution $\mathcal{Q}$ that generates new states should be relatively small. Otherwise, it is likely that the new candidate state $\mathbf{z}^\star$ ends in a region of the state space with very low probability (Bishop, 2006; MacKay,

2003). However, if the variance of $\mathcal{Q}$ is small, the exploration is made in very small steps, taking very long time to explore the whole state space. Furthermore, the different samples obtained from the chain will be very correlated as the different states obtained will be very similar. Thus, in this algorithm there is an important trade-off between the acceptance ratio and the set of different states that is explored. This trade-off becomes more serious in very high dimensions. In particular, when the dimension is high the Markov chain has to be run for very long periods of time to obtain a few independent samples from the stationary distribution (Bishop, 2006; MacKay, 2003).

5.5.2.2 Gibbs Sampling

Gibbs sampling is a simple Markov chain Monte Carlo method that can be seen as a special case of the Metropolis-Hastings algorithm (Bishop, 2006; MacKay, 2003). Gibbs sampling can be applied to sample from distributions with at last two components in $\mathbf{z}$. Consider that we are interested in sampling from the distribution $\mathcal{P}(\mathbf{z}) = \mathcal{P}(z_1, z_2, \ldots, z_m)$, where $m \geq 2$ is the dimension of the vector $\mathbf{z}$. We shall assume that $\mathcal{P}(\mathbf{z})$ is too complicated to sample from it directly. However, the conditional distributions $\mathcal{P}(z_i|\{z_j\}_{j\neq i})$ are simpler and hence, we can sample from them. Gibbs sampling builds a Markov chain whose stationary distribution is $\mathcal{P}(\mathbf{z})$ by sampling from these conditional distributions. Given $\mathbf{z}^{(n)}$, $\mathbf{z}^{(n+1)}$ is generated in Gibbs sampling by replacing the value of one of the components in $\mathbf{z}^{(n)}$ by a new value drawn from the distribution of that component conditioned to the values of the remaining components. This procedure is then repeated to generate subsequent samples from the chain by cycling over the components in some particular order or by choosing the variable to be updated at each step at random. Figure 5.5 displays the pseudo-code for the Gibbs sampling algorithm.

Input: dimension of the vector $\mathbf{z}$, m, number of samples N and marginal distributions for $\mathbf{z}$.

Output: Samples from the Markov chain $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots, \mathbf{z}^{(N)}$.

1. Compute the initial state of the chain $\mathbf{z}^{(1)}$.
2. For $i = 1, \ldots, N$
   - Sample $z_1^{(i+1)} \sim \mathcal{P}(z_1|z_2^{(i)}, z_3^{(i)}, \ldots, z_m^{(i)})$.
   - Sample $z_2^{(i+1)} \sim \mathcal{P}(z_1|z_1^{(i)}, z_3^{(i)}, \ldots, z_m^{(i)})$.
   - Sample $z_3^{(i+1)} \sim \mathcal{P}(z_1|z_1^{(i)}, z_2^{(i)}, z_4^{(i)}, \ldots, z_m^{(i)})$.
   - $\vdots$
   - Sample $z_m^{(i+1)} \sim \mathcal{P}(z_1|z_1^{(i)}, z_2^{(i)}, \ldots, z_{m-1}^{(i)})$.
3. Return samples from the chain $\{\mathbf{z}^{(n)}, n = 1, \ldots, N\}$.

FIGURE 5.5: Algorithm that implements Gibbs sampling. Extracted from (Bishop, 2006).

The Gibbs sampling algorithm is a particular case of the Metropolis-Hastings algorithm in which the acceptance probability of a new candidate state is always one (Bishop, 2006; MacKay, 2003). In particular, consider a sampling step involving the k-th component z_k in which the remaining components, summarized in the vector $\mathbf{z}_{\setminus k}$, remain unchanged. In this situation, the transition probability from the current state

$\mathbf{z}$, to a new candidate state $\mathbf{z}^\star$ is $\mathcal{Q}(\mathbf{z}^\star|\mathbf{z}) = \mathcal{P}(z_k|\mathbf{z}_{\setminus k})$. Because the components $j \neq k$ remain unchanged, we have that $\mathbf{z}^\star_{\setminus k} = \mathbf{z}_{\setminus k}$. Furthermore, $\mathcal{P}(\mathbf{z}) = \mathcal{P}(z_k|\mathbf{z}_{\setminus k})\mathcal{P}(\mathbf{z}_{\setminus k})$. In consequence, the acceptance probability of an equivalent Metropolis-Hastings step is

$$\begin{aligned}\mathcal{A}(\mathbf{z}^\star, \mathbf{z}) &= \min\left(1, \frac{\mathcal{P}(\mathbf{z}^\star)\mathcal{Q}(\mathbf{z}|\mathbf{z}^\star)}{\mathcal{P}(\mathbf{z})\mathcal{Q}(\mathbf{z}^\star|\mathbf{z})}\right) \\ &= \min\left(1, \frac{\mathcal{P}(z^\star_k|\mathbf{z}^\star_{\setminus k})\mathcal{P}(\mathbf{z}^\star_{\setminus k})\mathcal{P}(z_k|\mathbf{z}^\star_{\setminus k})}{\mathcal{P}(z_k|\mathbf{z}_{\setminus k})\mathcal{P}(\mathbf{z}_{\setminus k})\mathcal{P}(z^\star_k|\mathbf{z}_{\setminus k})}\right) \\ &= 1\,. \end{aligned} \tag{5.79}$$

This means that the steps of the Gibbs sampling algorithm are always accepted.

Because Gibbs sampling is in fact a particular case of the Metropolis-Hastings, it suffers from the same shortcomings. In particular, the Markov chain explores the space of states by a random walk which can take long time to explore the whole state space. The advantage of Gibbs sampling is that the acceptance ratio is always one. Thus, there are no consecutive repeated states in the chain. The random walk behavior of Gibbs sampling can be alleviated by a technique called *over-relaxation* (Adler, 1981). This technique applies to problems with Gaussian conditional distributions and is based on encouraging a directed motion through the space of states when the variables are highly correlated. The feasibility of the Gibbs sampling algorithm depends on whether the conditional distributions can be efficiently computed. In practice, there are many inference problems in which the joint probability distribution is intractable, but conditionals are easy to compute. Gibbs sampling can be useful in these problems. Finally, because Gibbs sampling only considers one variable each time, there are strong correlations between consecutive samples from the Markov chain. These correlations can be reduced by considering groups of variables in each step of the algorithm (Bishop, 2006; MacKay, 2003).

5.5.2.3 Hamilton Monte Carlo

This is an improved sampling technique that aims to solve the slow random walk behavior of the Metropolis-Hastings algorithm (Bishop, 2006; MacKay, 2003). Hamilton Monte Carlo makes use of gradient information to direct the exploration to locations where the sampled distribution has high probability so that the rejection rate of the Metropolis-Hastings algorithm is reduced. This sampling algorithm can be applied to continuous distributions for which the gradient of the logarithm of the probability can be efficiently evaluated.

In Hamilton Monte Carlo, the state vector $\mathbf{z}$ is extended to incorporate some momentum variables $\mathbf{p}$, one for each component in $\mathbf{z}$. Unlike the Metropolis-Hastings algorithm, Hamilton Monte Carlo generates new candidate states alternatively. First, a new momentum variable is generated randomly. Then, both the sate vector $\mathbf{z}$ and the momentum variables $\mathbf{p}$ are updated using simulated Hamiltonian dynamics, in analogy with a dynamical system. Assume that $\mathcal{P}(\mathbf{z})$ is the probability distribution we are interested in sampling from. We can always write $\mathcal{P}(\mathbf{z})$ in terms of an energy function $E(\mathbf{z})$

$$\mathcal{P}(\mathbf{z}) = \frac{1}{Z_E}\exp\left(-E(\mathbf{z})\right)\,, \tag{5.80}$$

where Z_E is just a normalization constant and $E(\mathbf{z})$ is interpreted as the *potential energy* of the system in state $\mathbf{z}$ (Bishop, 2006). To describe this dynamical system, we also

incorporate a *kinetic energy* term that depends on the momentum variables $\mathbf{p}$

$$K(\mathbf{p}) = \frac{1}{2}\mathbf{p}^T\mathbf{p}\,, \tag{5.81}$$

where T indicates transpose. The total energy of the dynamical system is the sum of these two components

$$H(\mathbf{z},\mathbf{p}) = E(\mathbf{z}) + K(\mathbf{p})\,. \tag{5.82}$$

The total energy $H(\mathbf{z},\mathbf{p})$ is also known as the *Hamiltonian* function (Bishop, 2006; MacKay, 2003), by analogy to the Hamiltonian function in mechanics. This energy function can be used to describe the probability of the system being at a given configuration. In particular, the joint probability of the system being at state $\mathbf{z}$ with momentum variables $\mathbf{p}$ is

$$\begin{aligned}\mathcal{P}(\mathbf{z},\mathbf{p}) &= \frac{1}{Z_H}\exp\left(-H(\mathbf{z},\mathbf{p})\right)\\ &= \frac{1}{Z_H}\exp\left(-E(\mathbf{z}) - K(\mathbf{p})\right)\,.\end{aligned} \tag{5.83}$$

This probability distribution factorizes with respect to $\mathbf{z}$ and $\mathbf{p}$. In consequence, to obtain samples from $\mathcal{P}(\mathbf{z})$ we can sample from (5.83) and then discard the momentum variables. Furthermore, the conditional distribution of the momentum variables $\mathbf{p}$ given a state vector $\mathbf{z}$ is simply a standard Gaussian distribution

$$\mathcal{P}(\mathbf{p}|\mathbf{z}) = \exp\left(-\frac{1}{2}\mathbf{p}^T\mathbf{p}\right)\,. \tag{5.84}$$

This dynamical system can be used to describe the evolution of the state variables $\mathbf{z}$ and the momentum variables $\mathbf{p}$ in continuous time t using the Hamilton equations

$$\frac{d\mathbf{z}}{dt} \equiv \frac{\partial H}{\partial \mathbf{p}} = \mathbf{p}\,, \tag{5.85}$$

$$\frac{d\mathbf{p}}{dt} \equiv -\frac{\partial H}{\partial \mathbf{z}} = -\frac{\partial E}{\partial \mathbf{z}}\,. \tag{5.86}$$

Note that in the evolution of the dynamical system, the value of (5.82) remains constant (Bishop, 2006)

$$\begin{aligned}\frac{dH}{dt} &= \sum_i \left(\frac{\partial H}{\partial z_i}\frac{dz_i}{dt} + \frac{\partial H}{\partial p_i}\frac{dp_i}{dt}\right)\\ &= \sum_i \left(\frac{\partial H}{\partial z_i}\frac{\partial H}{\partial p_i} - \frac{\partial H}{\partial p_i}\frac{\partial H}{\partial z_i}\right)\\ &= 0\,.\end{aligned} \tag{5.87}$$

However, even though (5.82) is invariant, the values of $\mathbf{z}$ and $\mathbf{p}$ will vary with time. Thus, by integrating the Hamiltonian dynamics described by (5.85) and (5.86) over a finite amount of time it is possible to make large changes in $\mathbf{z}$, which is distributed according to $\mathcal{P}(\mathbf{z})$.

The Hamilton Monte Carlo algorithm works by first picking a random value for the momentum variables $\mathbf{p}$. This is typically implemented by drawing $\mathbf{p}$ from the conditional Gaussian probability (5.84). Note that this step is always accepted according to the

Metropolis-Hastings acceptance rule because it is equivalent to a Gibbs sampling step. Then, the mechanics of the dynamical system are simulated for a finite amount of time using approximate techniques for integrating the differential equations (5.85) and (5.86). This step produces large changes in $\mathbf{z}$ and $\mathbf{p}$, giving a candidate configuration $(\mathbf{z}^\star, \mathbf{p}^\star)$. If the integration of (5.85) and (5.86) were perfect, the acceptance probability of the Metropolis-Hastings algorithm for this step should always be one, because (5.82) is invariant under the simulated mechanics. However, in practice, due to numerical errors the value of H evaluated at the initial configuration can be different from the value of H evaluated at the candidate configuration. In consequence, we use the Metropolis-Hastings acceptance rule and accept the new configuration of the system with probability

$$\begin{aligned} \mathcal{A}((\mathbf{z}^\star, \mathbf{p}^\star), (\mathbf{z}, \mathbf{p})) &= \min\left(1, \frac{\mathcal{P}(\mathbf{z}^\star, \mathbf{p}^\star)}{\mathcal{P}(\mathbf{z}, \mathbf{p})}\right) \\ &= \min\left(1, \exp\left(-\Delta H\right)\right) , \end{aligned} \tag{5.88}$$

where ΔH measures the change in (5.82). Note that in this last expression there is no proposal distribution $\mathcal{Q}$. This is because the dynamics of the system are reversible. By choosing randomly whether to integrate the dynamics forward or backward in time at the beginning of each step, the transition probabilities given by $\mathcal{Q}$ are symmetric and hence, they do not appear in (5.88) (Bishop, 2006). Figure 5.6 displays the pseudo-code of the Hamilton Monte Carlo algorithm.

Input: number of samples N, number of steps T, step size ϵ, $H(\mathbf{z}, \mathbf{p})$ and $-\partial E(\mathbf{z})/\partial z$.

Output: Samples from the Markov chain $(\mathbf{z}^{(1)}, \mathbf{p}^{(1)}), \ldots, (\mathbf{z}^{(N)}, \mathbf{p}^{(N)})$.

1. Compute the initial state of the chain $(\mathbf{z}^{(1)}, \mathbf{p}^{(1)})$
2. For $i = 1, 2, \ldots, N$
   - Draw $\mathbf{p}$ from $\mathcal{N}(0|\mathbf{0}, \mathbf{I})$.
   - $\mathbf{p}^{(i+1)} \leftarrow \mathbf{p}$.
   - $\mathrm{H}' \leftarrow H(\mathbf{z}^{(i)}, \mathbf{p}^{(i+1)})$
   - $\mathbf{z}^{(i+1)} \leftarrow \mathbf{z}^{(i)}$.
   - Switch randomly the sign of ϵ.
   - For $t = 1, 2, \ldots, T$
     - $\mathbf{p}^{(i+1)} \leftarrow \mathbf{p}^{(i+1)} - \epsilon\frac{1}{2}\left[\frac{\partial E(\mathbf{z})}{\partial \mathbf{z}}\right]_{\mathbf{z}=\mathbf{z}^{(i+1)}}$.
     - $\mathbf{z}^{(i+1)} \leftarrow \mathbf{z}^{(i+1)} + \epsilon\mathbf{p}$.
     - $\mathbf{p}^{(i+1)} \leftarrow \mathbf{p}^{(i+1)} - \epsilon\frac{1}{2}\left[\frac{\partial E(\mathbf{z})}{\partial \mathbf{z}}\right]_{\mathbf{z}=\mathbf{z}^{(i+1)}}$.
   - $\Delta H \leftarrow H(\mathbf{z}^{(i+1)}, \mathbf{p}^{(i+1)}) - \mathrm{H}'$.
   - Pick u uniformly from the interval $[0, 1]$.
   - if $u > \min(1, \exp(-\Delta H))$ then $(\mathbf{z}^{(i+1)}, \mathbf{p}^{(i+1)}) \leftarrow (\mathbf{z}^{(i)}, \mathbf{p})$.
3. Return samples from the chain $(\mathbf{z}^{(1)}, \mathbf{p}^{(1)}), \ldots, (\mathbf{z}^{(N)}, \mathbf{p}^{(N)})$.

FIGURE 5.6: Algorithm that implements the Hamilton Monte Carlo algorithm. Extracted from (Bishop, 2006).

5.6 Conclusions

In this chapter we have given an overview of Bayesian machine learning. This paradigm provides a principled approach to deal with the uncertainties about the selection of the model and the model parameters in automatic induction problems. In particular, when the amount of data available for induction is reduced and the training instances are contaminated by noise, a Bayesian approach has several advantages over standard learning methods. First, the posterior probabilities of the model given the observed data can be used to discriminate among different candidate models. These probabilities automatically penalize models that are unnecessarily complex to solve the learning task. Second, assuming that the correct model has been selected, the posterior probability of the model parameters can be used to account for different values of the parameters, each one weighed by the corresponding value of the posterior distribution. Predictions are then computed in terms of a predictive distribution that considers all these different values for the model parameters. Finally, in this learning paradigm expert knowledge about the learning task can be incorporated in the model by specifying some prior distribution for the model parameters. This prior knowledge can compensate for the limited amount of data available.

Despite these advantages, the practical implementation of Bayesian machine learning is difficult. In particular, the computation of the posterior distribution often requires solving difficult integrals or evaluating summations that involve an exponential number of terms. Thus, this distribution has to be approximated in practice. A standard method for computing this approximation is Markov chain Monte Carlo (MCMC) (Neal, 1993), in which the posterior distribution is approximated by samples from Markov chain. The problem is that MCMC techniques can be very slow because generating independent samples from the posterior distribution requires running the Markov chain for a long period of time. An alternative to MCMC is to use deterministic methods that approximate the posterior distribution by a simple distribution for which computations are tractable. Sometimes it can be difficult to compute an approximate posterior distribution for all the parameters of the model. Under these circumstances, a technique called type-II maximum likelihood can be used to estimate these parameters. However, type-II maximum likelihood demands the running of the approximate inference algorithm multiple times, increasing the total cost of training the Bayesian model.

The following chapters of this thesis describe how the expectation propagation (EP) algorithm can be used to alleviate some of these problems. Specifically, the EP algorithm can be used in Bayesian models as an efficient alternative to MCMC sampling or type-II maximum likelihood estimation to approximate the posterior distribution.

Chapter 6

Bayes Machines for Binary Classification

In this chapter we propose a Bayesian approach to binary classification based on an extension of Bayes point machines. This extension is carried out by taking into account all the hyper-planes that are consistent with the training data, considering the possibility of mislabeled instances. Using Bayes' theorem, a posterior distribution for the model parameters and a predictive distribution for unseen data are computed. The most compelling feature of the proposed model is that the level of noise in the class labels of the training data can be learned at no additional cost. All the computations are carried out using the approximate inference algorithm expectation propagation (EP). This algorithm is written in terms of inner products so that feature expansion using the kernel trick is straightforward. Experimental results show that the proposed model outperforms support vector machines and is competitive with other Bayesian classification algorithms based on Gaussian processes in several of the classification problems studied. The training and prediction time of this model can be reduced using a sparse representation. For this purpose, the sparse representation proposed in the informative vector machine (IVM) is used. However, some modifications are made to provide a better approximation to the posterior distribution. Specifically, we introduce in the training algorithm additional refining iterations over the set of active instances included in the model. These refining iterations can be thought as a back-fitting algorithm that tries to fix some of the mistakes that result from the greedy approach of the IVM. Experiments comparing the performance of this sparse representation with the performances of the complete (non-sparse) machine and the support vector machine confirm that it is competitive with these two classifiers.

6.1 Introduction

Kernel classifiers have shown an excellent performance on numerous classification problems (Gibbs and MacKay, 2000; Herbrich et al., 2001; Kim and Ghahramani, 2006; Minka, 2001b; Vapnik, 1995). In consequence, they have received a lot of attention from the machine learning community. Examples of these types of classifiers are support vector machines (SVMs), Bayes point machines (BPMs), and Gaussian process classifiers (GPCs). The SVM was devised as a classifier that maximizes the margin, which is defined as the minimum distance between data points and the decision boundary (Vapnik, 1995). Inspired by results from the statistical/PAC learning theory, SVMs have

proved quite successful on many learning applications and hence, they have become a standard tool for automatic induction. On the other hand, GPCs were developed within a Bayesian framework. They are kernel classifiers derived from Gaussian process (GP) priors over latent functions. GPs were originally introduced for regression problems. However, they can be readily adapted to address classification tasks by using a latent function whose value at a certain input location is monotonically related to the class-posterior probability at that particular location. A GP can be used as a prior for the latent function and probabilities can be obtained from it by applying a monotonic function bounded in the interval $[0, 1]$ (Bishop, 2006; Kuss and Rasmussen, 2005; Williams and Barber, 1998). Typically, exact inference in GPCs is not tractable and therefore, approximation techniques have to be employed (Barber and Williams, 1997; Gibbs and MacKay, 2000; Kuss and Rasmussen, 2005; Opper and Winther, 2000b). Finally, the BPM is a kernel classifier that tries to find the single parameter combination that mimics best the Bayes optimal classification strategy (Herbrich et al., 2001). This parameter combination is known as the Bayes point. Because computing the Bayes point is typically infeasible, Herbrich et al. (2001) propose to approximate it by the center of mass in *version space*. The version space is defined as the set of all hypotheses space that are consistent with the data in feature space. Computing the Bayes point using this definition is also difficult and hence, approximate algorithms are employed (Herbrich et al., 2001; Minka, 2001b). When expectation propagation is used to approximate the Bayes point, the decision boundary of the resulting classifier is equal to the decision boundary of the GPC (Minka, 2001b). Finally, Herbrich et al. (2001) show that in the noise free scenario, the SVM can be seen as a particular case of the BPM that tries to find the center of the largest ball still inscribable in version space. Because both the SVM and the BPM only consider a single value for the model parameters, they output single class label. By contrast, GPCs output class probabilities.

In this chapter we propose a classification model devised to improve BPMs in two ways (Hernández-Lobato and Hernández-Lobato, 2008). First, we take into account the whole *version space* instead of only its center of mass. Second, we consider the possibility of mislabeled data in the training set. The proposed model is formulated in a Bayesian framework and Bayes' theorem is used to compute a posterior distribution for all the parameters of the model, including a parameter that quantifies the level of noise. Using this posterior distribution we can compute a predictive distribution for new instances. This predictive distribution provides a mechanism to take into account uncertainty in the predictions. Because exact Bayesian inference in this model is infeasible, we use expectation propagation (EP) to approximate the posterior distribution of the model parameters (Minka, 2001b). The most attractive feature of this model is the ability to learn the intrinsic noise in the class labeling of the training data with no additional cost. This is not possible in SVMs, BPMs or GPCs, which have to resort to costly methods.

In SVMs and GPCs noise is typically handled by means of an additional parameter that must be carefully tuned to obtain good generalization properties. In the case of SVMs this parameter is often found by cross-validation. That is, several SVMs are built for different values of the noise parameter and the performance of each machine is measured on a validation set. Finally, the value of the noise parameter that leads to the SVM with the best performance is chosen. In the case of BPMs and GPCs a similar procedure is carried out. However, instead of using a dataset for validation, they follow a type-II maximum likelihood approach (Bishop, 2006) and maximize the marginal likelihood of the data, where the model parameters have been integrated out (Bishop, 2006; Kim and Ghahramani, 2006; MacKay, 2003; Minka, 2001b). The advantage is that all

the available data can be used for training the model, unlike in the SVM. Other techniques devised for GPCs make use of a Markov chain Monte Carlo (MCMC) algorithm to sample from a posterior distribution for the noise parameter (Barber and Williams, 1997; Neal, 1997). Nevertheless, obtaining independent samples in these algorithms is a difficult task. In particular, the Markov chain has to be run for a considerable time, resulting in a large computational cost. Thus, sampling techniques are usually limited to small classification problems (MacKay, 2003).

The major drawback of SVMs, BPMs and GPCs is that they require multiple executions of the underlying learning algorithm to estimate the appropriate values of the hyper-parameters that quantify the level of noise. This increases the computational cost of training the final models. The method proposed in this chapter can learn the level of noise in the training set in a single application of the learning algorithm.

The computational cost of training the proposed model is determined by the cost of EP which is $\mathcal{O}(n^3)$, where n is the number of training instances. This cubic dependence in the training set becomes a problem if we want to use this model in large classification problems. This limitation can be alleviated by obtaining a sparse representation for this model. This sparse representation is based on an approach similar to the informative vector machine (IVM) (Lawrence et al., 2003; Seeger, 2003). It consists in maintaining a set $\mathcal{I}$ of active instances. Instances that are not included in $\mathcal{I}$ are ignored by the model. Because finding the best active set is a difficult task, the IVM employs a greedy algorithm for this purpose. In this chapter we improve this greedy algorithm by performing additional refining iterations that provide a better approximation to the posterior distribution.

The chapter is organized as follows. In Section 6.2 we introduce the proposed binary classification model that extends BPMs. The EP algorithm is used in Section 6.2.1 for making approximate inference in this model. Then, in Section 6.2.2 we carry out experiments using several binary classification problems to assess the performance of the proposed model. Experimental comparison with other kernel classifiers such as SVMs and GPCs is also provided in this section. Later, in Section 6.3.1 we introduce the training algorithm that provides a sparse representation for the proposed model and in Section 6.3.2 we carry out experiments to assess the accuracy of the sparse models obtained by this algorithm. Finally, the conclusions of this chapter are summarized in Section 6.4.

6.2 Bayes Machines

The Bayes machine (BM) is a Bayesian approach to linear binary classification. A linear classifier assigns the class label (-1 or 1) for the instances characterized by the vector of attributes $\mathbf{x}$ using the rule $y = \text{sign}(\mathbf{w}^T\mathbf{x})$, for some parameter $\mathbf{w}$ that defines a hyper-plane $\mathbf{w}$ in $\mathbb{R}^d$, where d is the dimensionality of the data. The hyper-plane $\mathbf{w}$ is required to contain the origin. This can be readily achieved by extending the vector of attributes with one additional component that takes value 1. Given a set of n d-dimensional input examples $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and corresponding target class labels $\mathbf{y} = \{y_1, \ldots, y_n\}$, $y_i \in \{-1, 1\}$, the likelihood for $\mathbf{w}$ (Herbrich et al., 2001) is

$$\mathcal{P}(\mathbf{y}|\mathbf{w}, \mathbf{X}) = \prod_{i=1}^{N} \mathcal{P}(y_i|\mathbf{x}_i, \mathbf{w}) = \prod_{i=1}^{N} \Theta(y_i \mathbf{w}^T \mathbf{x}_i)\,, \tag{6.1}$$

where the function Θ is the step function given by

$$\Theta(y) = \begin{cases} 1 & \text{if } y > 0 \\ 0 & \text{if } y \leq 0 \end{cases}, \tag{6.2}$$

and the superscript T means transposed. Note that we are not seeking to model the distribution of the input variables. For this reason, $\mathbf{X}$ will always appear in the set of conditioning variables. To simplify the notation, from now on, we drop the explicit $\mathbf{X}$ from all the expressions. For instance, $\mathcal{P}(\mathbf{y}|\mathbf{w}, \mathbf{X})$ will be simply written as $\mathcal{P}(\mathbf{y}|\mathbf{w})$.

We note that the likelihood function (6.1) is one if the hyper-plane $\mathbf{w}$ is a perfect separator of the data and zero otherwise. Thus, assuming a uniform prior for $\mathbf{w}$ the posterior distribution for $\mathbf{w}$ is uniform over all the hyper-planes that correctly separate the training data. This means that the approach followed by the BPM, which only considers one single value for $\mathbf{w}$ in version space, will actually produce a decision boundary that, while being similar to the Bayesian average, fails to take into account the uncertainty that arises form the fact that many other hypotheses are equally likely.

We can extend the model given by (6.1) to take into account labeling errors. Following Opper and Winther (2000a)

$$\mathcal{P}(y|\mathbf{x}, \mathbf{w}, \epsilon) = \epsilon(1 - \Theta(y\mathbf{w}^T\mathbf{x})) + (1 - \epsilon)\Theta(y\mathbf{w}^T\mathbf{x}) \tag{6.3}$$

$$= \epsilon + (1 - 2\epsilon)\Theta(y\mathbf{w}^T\mathbf{x}), \tag{6.4}$$

where ϵ is the labeling error rate. For a fixed hyper-plane $\mathbf{w}$, the likelihood for ϵ given $\mathbf{y}$ is

$$\mathcal{P}(\mathbf{y}|\mathbf{w}, \epsilon) = \epsilon^r(1 - \epsilon)^{n-r}, \tag{6.5}$$

where r is the number of errors in the training set. This likelihood is maximum for $\epsilon = r/n$ which is precisely the error rate of the hyper-plane $\mathbf{w}$. Thus, the model only takes into account the number of errors, not how large these errors are. Assuming this likelihood function, outliers do not produce distortions in the resulting classifier because the model does not care about how far the errors are from the decision boundary. An alternative approach to introduce labeling errors in the model which is sensitive to outliers is to penalize errors far from the decision boundary (Bishop, 2006; Minka, 2001b; Opper and Winther, 2000a). This can be achieved by substituting the step function in (6.1) by a smooth activation function, such as the cumulative probability function of a standard Gaussian distribution. Another choice is the sigmoid function whose tails fall like $\exp(-x)$ instead of $\exp(-x^2)$, and is therefore less affected by outliers. Nevertheless, we believe that the likelihood function given in (6.4) is more robust than these alternatives.

To complete the Bayesian formulation of the model we need to select a prior distribution for the model parameters $\mathbf{w}$ and ϵ. We will assume a factorizing prior

$$\mathcal{P}(\mathbf{w}, \epsilon) = \mathcal{P}(\mathbf{w})\mathcal{P}(\epsilon). \tag{6.6}$$

The next step is to choose a parametric family of distributions for $\mathcal{P}(\mathbf{w})$ and $\mathcal{P}(\epsilon)$. These distributions should be uniform for both $\mathbf{w}$ and ϵ so that no particular value of the parameters is favored over other values. As discussed in (Minka, 2001b), a spherical Gaussian distribution

$$\mathcal{P}(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{I}) \tag{6.7}$$

is a uniform prior for $\mathbf{w}$ because it assumes that all directions of the hyper-plane $\mathbf{w}$ are equally likely. In the case of the prior over the noise parameter ϵ, we assume that it follows a beta distribution (see Appendix A.2)

$$\mathcal{P}(\epsilon) = \text{Beta}(\epsilon|a_0, b_0)\,. \tag{6.8}$$

In this case, a uniform distribution for ϵ is obtained by setting $a_0 = 1$ and $b_0 = 1$. However, using this prior actually prevents the machine from learning anything at all. In particular, under a uniform prior for ϵ the data instances have equal probability of being correctly or incorrectly labeled. To extract some knowledge from the training data we have to trust the data to a certain extent. That is, we have to make the assumption that most of the data are correctly labeled. A uniform prior for ϵ does not reflect this. In consequence, we consider an alternative prior distribution by setting $a_0 = 1$ and $b_0 = 10$. These parameter values lead to a prior for ϵ whose average value is close to zero, providing at the same time a sufficiently large variance so that larger values for ϵ are also possible. The performance of the BM is not very sensitive to the values of these parameters provided that the resulting prior distribution has the properties described.

The posterior distribution of the model parameters ϵ and $\mathbf{w}$ given the observed class labels $\mathbf{y}$ can be computed using Bayes' theorem

$$\mathcal{P}(\epsilon, \mathbf{w}|\mathbf{y}) = \frac{\mathcal{P}(\mathbf{y}|\mathbf{w}, \epsilon)\mathcal{P}(\epsilon, \mathbf{w})}{\mathcal{P}(\mathbf{y})}\,, \tag{6.9}$$

where the denominator in (6.9) is a normalization constant which is known as the evidence of the observed labels $\mathbf{y}$ given the current model. In a Bayesian framework, this constant can be used to discriminate among different models (Bishop, 2006; MacKay, 2003).

A new instance $\mathbf{x}_{\text{new}}$ is classified by means of the predictive distribution for its class label y_{new} given the data. This distribution is obtained by computing the expected value of the likelihood of the model parameters for that instance over the posterior distribution

$$\mathcal{P}(y_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{y}) = \int \mathcal{P}(y_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{w}, \epsilon)\mathcal{P}(\mathbf{w}, \epsilon|\mathbf{y})\, d\mathbf{w} d\epsilon\,. \tag{6.10}$$

The model, as described by the previous definitions, only takes into account hypotheses based on hyper-planes and hence, it can handle only linearly separable classification problems. One way to achieve linear separation in non-linear classification problems is to progressively enlarge the set of features until the data are linearly separable in the expanded feature space. Solving the classification problem in this extended space using a linear model has the effect of producing a non-linear decision boundary in the original measurement space. This form of feature expansion can be efficiently implemented using *kernels* (Aizerman et al., 1964).

Because the prior for $\mathbf{w}$ is assumed to be Gaussian, the BM can be understood within the framework of Gaussian processes. In particular, the product $\mathbf{w}^T\mathbf{x}$ follows a Gaussian distribution and, in consequence, can be described by an arbitrary function $f(\mathbf{x})$ whose prior is assumed to be a Gaussian process, meaning that any set of points sampled from f has a multivariate Gaussian distribution. However, unlike GPCs based on similar definitions (Kim and Ghahramani, 2006; Opper and Winther, 2000b), the BM proposed in this chapter can learn a posterior distribution that includes the noise parameter ϵ with no additional costs.

In general, neither (6.10) nor (6.9) are tractable. Therefore, approximate algorithms have to be employed to make Bayesian inference. For this purpose we use the EP algorithm (Minka, 2001b).

6.2.1 Expectation Propagation for Bayes Machines

In this section we describe how EP can be used for making approximate inference in the BM. The algorithm presented here is an extension of previous implementations of EP (Lawrence et al., 2003; Minka, 2001b; Opper and Winther, 2000b). In particular, Minka (2001b) describes the EP algorithm for computing the posterior approximation over the parameter $\mathbf{w}$. In this section we extend the derivations of Minka (2001b) to take into account the posterior distribution over the noise parameter ϵ. The final EP algorithm presented in this section is written in terms of inner products so that the use of kernels for feature expansion is straightforward. This allows the BM to handle non-linear decision boundaries. From now on, we assume that all $\mathbf{x}_i$ are scaled by their corresponding label y_i in order to improve the readability of the different expressions.

To apply the EP algorithm to the proposed model, we write the joint distribution of the observed class labels and the model parameters as the product of $n+1$ terms t_i, $i = 1, \ldots, n+1$, where the first n terms correspond to the likelihood and the last term to the prior. Each of these terms is approximated by a corresponding term $\tilde{t}_i$ that is restricted to belong to the exponential family of probability distributions

$$\mathcal{P}(\mathbf{y}, \mathbf{w}, \epsilon) = \prod_{i=1}^{n} \mathcal{P}(y_i|\mathbf{x}_i, \mathbf{w}, \epsilon)\mathcal{P}(\mathbf{w}, \epsilon) = \prod_{i=1}^{n+1} t_i(\mathbf{w}, \epsilon) \approx \prod_{i=1}^{n+1} \tilde{t}_i(\mathbf{w}, \epsilon)\,. \tag{6.11}$$

Assume that the posterior approximation of (6.9), $\mathcal{Q}$, is the product of two distributions from the exponential family

$$\mathcal{Q}(\mathbf{w}, \epsilon) = \mathcal{N}(\mathbf{w}|\mathbf{m}_\mathbf{w}, \mathbf{V}_\mathbf{w})\mathrm{Beta}(\epsilon|a_\epsilon, b_\epsilon)\,, \tag{6.12}$$

where $\mathcal{N}(\mathbf{w}|\mathbf{m}_\mathbf{w}, \mathbf{V}_\mathbf{w})$ denotes a Gaussian distribution with mean vector $\mathbf{m}_\mathbf{w}$ and covariance matrix $\mathbf{V}_\mathbf{w}$ and $\mathrm{Beta}(\epsilon|a_\epsilon, b_\epsilon)$ denotes a beta distribution with parameters a_ϵ and b_ϵ. The choice of a beta distribution to model the noise makes Bayesian inference simpler because the likelihood (6.5) has a binomial form. Note that (6.12) belongs to the exponential family because it is the product of two distributions that also belong to that family. Thus, the approximate terms $\tilde{t}_i$, with $i = 1, \ldots, n+1$, are restricted to have the same form as (6.12), although they do not have to be normalized

$$\tilde{t}_i(\mathbf{w}, \epsilon) = s_i \exp\left(-\frac{1}{2}(\mathbf{w} - \mathbf{m}_i)^T \mathbf{V}_i^{-1}(\mathbf{w} - \mathbf{m}_i)\right) \epsilon^{a_i - 1}(1 - \epsilon)^{b_i - 1}\,. \tag{6.13}$$

The value of s_i is determined by requiring that $\tilde{t}_i \prod_{j \neq i} \tilde{t}_j$ integrates to the same value as $t_i \prod_{j \neq i} \tilde{t}_j$ and $\mathbf{m}_i$, $\mathbf{V}_i$, a_ϵ and b_ϵ are free parameters. Note that (6.13) has the same form as the prior for $\mathbf{w}$ and ϵ (6.6). In consequence, we only describe the EP updates of the approximate terms $\tilde{t}_i$, with $i = 1, \ldots, n$, corresponding to the likelihood terms. The EP update of the approximate term $\tilde{t}_{n+1}$ corresponding to the prior for $\mathbf{w}$ and ϵ is straightforward and consists in setting $\tilde{t}_{n+1}$ equal to (6.6) using (6.7) and (6.8). Furthermore, if $\mathcal{Q}$ is initialized to (6.6) the term t_{n+1} can be ignored in the main loop of the EP algorithm (see Section 5.5.1.3).

The first step of the EP algorithm initializes all approximate terms $\tilde{t}_i$ corresponding to the likelihood to be uniform. In the case of the Gaussian part of $\tilde{t}_i$, this is obtained by setting the mean equal to zero and the covariance matrix equal to a diagonal matrix with infinities in the diagonal. In the case of the beta part, the parameters a_i and b_i are both set to one. The approximate term $\tilde{t}_{n+1}$ corresponding to the prior for $\mathbf{w}$ and ϵ is initialized to (6.6) and it is never modified. Finally, the posterior approximation $\mathcal{Q}$ is initialized to the prior (6.6).

The next step of the EP algorithm is to remove an approximate term $\tilde{t}_i$ from $\mathcal{Q}$ to compute $\mathcal{Q}^{\backslash i}$. Recall that $\mathcal{Q}^{\backslash i}$ has the same form as $\mathcal{Q}$ because of the closure property of the exponential family

$$\mathcal{Q}^{\backslash i}(\mathbf{w}, \epsilon) = \mathcal{N}(\mathbf{w}|\mathbf{m_w}^{\backslash i}, \mathbf{V_w}^{\backslash i})\text{Beta}(\epsilon|a_\epsilon^{\backslash i}, b_\epsilon^{\backslash i}) \,. \tag{6.14}$$

The parameters $\mathbf{m}_\mathbf{w}^{\backslash i}$, $\mathbf{V}_\mathbf{w}^{\backslash i}$ $a_\epsilon^{\backslash i}$ and $b_\epsilon^{\backslash i}$ of $\mathcal{Q}^{\backslash i}$ are found by computing the ratio between $\mathcal{Q}$ and $\tilde{t}_i$ and normalizing

$$\mathbf{V}_\mathbf{w}^{\backslash i} = \left(\mathbf{V}_\mathbf{w}^{-1} - \mathbf{V}_i^{-1}\right)^{-1} \,, \tag{6.15}$$

$$\mathbf{m}_\mathbf{w}^{\backslash i} = \mathbf{V}_\mathbf{w}^{\backslash i}\left(\mathbf{V}_\mathbf{w}^{-1}\mathbf{m}_\mathbf{w} - \mathbf{V}_i^{-1}\mathbf{m}_i\right) \,, \tag{6.16}$$

$$a_\epsilon^{\backslash i} = a_\epsilon - a_i + 1 \,, \tag{6.17}$$

$$b_\epsilon^{\backslash i} = b_\epsilon - b_i + 1 \,, \tag{6.18}$$

where we have used (A.46) for the computation of (6.15) and (6.16), and (A.30) for the computation of (6.17) and (6.18).

The next step of the algorithm consists in computing an updated posterior distribution $\hat{\mathcal{P}}$ defined as

$$\hat{\mathcal{P}}(\mathbf{w}, \epsilon) = \frac{1}{Z_i} t_i(\mathbf{w}, \epsilon)\mathcal{Q}^{\backslash i}(\mathbf{w}, \epsilon) \,, \tag{6.19}$$

where Z_i is just a normalization constant. In the case of the likelihood terms this constant is

$$\begin{aligned} Z_i &= \int t_i(\mathbf{w}, \epsilon)\mathcal{Q}^{\backslash i}(\mathbf{w}, \epsilon) d\mathbf{w} d\epsilon \\ &= \int \left(\epsilon + (1 - 2\epsilon)\Theta(\mathbf{w}^T\mathbf{x}_i)\right) \mathcal{N}(\mathbf{w}|\mathbf{m_w}^{\backslash i}, \mathbf{V_w}^{\backslash i})\text{Beta}(\epsilon|a_\epsilon^{\backslash i}, b_\epsilon^{\backslash i}) d\mathbf{w} d\epsilon \\ &= \overline{\epsilon}^{\backslash i} + (1 - 2\overline{\epsilon}^{\backslash i})\Phi(z_i) \,, \end{aligned} \tag{6.20}$$

where

$$\overline{\epsilon}^{\backslash i} = \frac{a_\epsilon{}^{\backslash i}}{(a_\epsilon^{\backslash i} + b_\epsilon^{\backslash i})} \,, \qquad\qquad z_i = \frac{(\mathbf{m}_\mathbf{w}^{\backslash i})^T\mathbf{x}_i}{\sqrt{\mathbf{x}_i^T\mathbf{V_w}^{\backslash i}\mathbf{x}_i}} \tag{6.21}$$

and $\Phi(\cdot)$ is the cumulative probability function of a standard Gaussian distribution.

Once $\hat{\mathcal{P}}$ has been normalized, we have to minimize the Kullback-Liebler (KL) divergence between $\hat{\mathcal{P}}$ and $\mathcal{Q}$ to find the parameters of the updated posterior approximation $\mathcal{Q}^{\text{new}}$. Because $\mathcal{Q}$ is in the exponential family and factorizes with respect to $\mathbf{w}$ and ϵ, the parameters of $\mathcal{Q}^{\text{new}}$ are found by matching the expected values of the sufficient statistics of each marginal distribution of the posterior approximation with the expected values of these sufficient statistics under the distribution $\hat{\mathcal{P}}$. In particular, from the minimization of the KL divergence we get the following expectation constraints for the updated

posterior approximation $\mathcal{Q}^{\text{new}}$

$$\mathbb{E}_{\mathcal{Q}^{\text{new}}}[\mathbf{w}] = \mathbb{E}_{\hat{\mathcal{P}}}[\mathbf{w}]\,, \tag{6.22}$$

$$\mathbb{E}_{\mathcal{Q}^{\text{new}}}[\mathbf{w}\mathbf{w}^T] = \mathbb{E}_{\hat{\mathcal{P}}}[\mathbf{w}\mathbf{w}^T]\,, \tag{6.23}$$

$$\mathbb{E}_{\mathcal{Q}^{\text{new}}}[\log(\epsilon)] = \mathbb{E}_{\hat{\mathcal{P}}}[\log(\epsilon)]\,, \tag{6.24}$$

$$\mathbb{E}_{\mathcal{Q}^{\text{new}}}[\log(1-\epsilon)] = \mathbb{E}_{\hat{\mathcal{P}}}[\log(1-\epsilon)]\,. \tag{6.25}$$

See Appendix A.2 and Appendix A.3 for further information about the sufficient statistics of the Gaussian distribution and the beta distribution. To match constraints (6.22) and (6.23) we can use (A.40) and (A.41), respectively. This gives the following equations for the parameters of $\mathcal{Q}^{\text{new}}$

$$\mathbf{m}_{\mathbf{w}}^{\text{new}} = \mathbf{m}_{\mathbf{w}}^{\backslash i} + \mathbf{V}_{\mathbf{w}}^{\backslash i}\alpha_i\mathbf{x}_i\,, \tag{6.26}$$

$$\mathbf{V}_{\mathbf{w}}^{\text{new}} = \mathbf{V}_{\mathbf{w}}^{\backslash i} - (\mathbf{V}_{\mathbf{w}}^{\backslash i}\mathbf{x}_i)\left(\frac{\alpha_i\mathbf{x}_i^T\mathbf{m}_{\mathbf{w}}^{\text{new}}}{\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\backslash i}\mathbf{x}_i}\right)(\mathbf{V}_{\mathbf{w}}^{\backslash i}\mathbf{x}_i)^T\,, \tag{6.27}$$

where

$$\alpha_i = \frac{1}{\sqrt{\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\backslash i}\mathbf{x}_i}}\frac{(1-2\overline{\epsilon}^{\backslash i})\mathcal{N}(z_i|0,1)}{\overline{\epsilon}^{\backslash i} + (1-2\overline{\epsilon}^{\backslash i})\Phi(z_i)}\,. \tag{6.28}$$

Using (A.21), (A.22), (A.25) and (A.26), constraints (6.24) and (6.25) can be shown to be equivalent to

$$\Psi(a_\epsilon^{\text{new}}) - \Psi(a_\epsilon^{\text{new}} + b_\epsilon^{\text{new}}) = \Psi(a_\epsilon^{\backslash i}) - \Psi(a_\epsilon^{\backslash i} + b_\epsilon^{\backslash i}) + \frac{(1-\overline{\epsilon}^{\backslash i})(1-2\Phi(z_i))}{\Phi(z_i)b_\epsilon^{\backslash i} + (1-\Phi(z_i))a_\epsilon^{\backslash i}}\,, \tag{6.29}$$

$$\Psi(b_\epsilon^{\text{new}}) - \Psi(a_\epsilon^{\text{new}} + b_\epsilon^{\text{new}}) = \Psi(b_\epsilon^{\backslash i}) - \Psi(a_\epsilon^{\backslash i} + b_\epsilon^{\backslash i}) + \frac{\overline{\epsilon}^{\backslash i}(1-2\Phi(z_i))}{\Phi(z_i)b_\epsilon^{\backslash i} + (1-\Phi(z_i))a_\epsilon^{\backslash i}}\,, \tag{6.30}$$

where $\Psi(x) = d\log(\Gamma(x))/dx$ and $\Gamma(x)$ is the gamma function (Abramowitz and Stegun, 1964). Because the Ψ function is non-linear, the parameters a_ϵ^{new} and b_ϵ^{new} of the updated posterior approximation $\mathcal{Q}^{\text{new}}$ have to be found by numerical methods. Solving the set of non-linear set of equations (6.24) and (6.25) in each iteration of EP would significantly increase the computational time needed for training the BM. Thus, an alternative method is used in practice. Instead of propagating the expectations of $\log(\epsilon)$ and $\log(1-\epsilon)$, we propagate the expectations of ϵ (mean) and ϵ^2 (variance), as suggested by Cowell et al. (1996). This leads to the following simplified update equations for the parameters a_ϵ^{new} and b_ϵ^{new} of $\mathcal{Q}^{\text{new}}$

$$a_\epsilon^{\text{new}} = \frac{\mathbb{E}_{\hat{\mathcal{P}}}[\epsilon] - \mathbb{E}_{\hat{\mathcal{P}}}[\epsilon^2]}{\mathbb{E}_{\hat{\mathcal{P}}}[\epsilon^2] - \mathbb{E}_{\hat{\mathcal{P}}}[\epsilon]^2}\mathbb{E}_{\hat{\mathcal{P}}}[\epsilon]\,, \tag{6.31}$$

$$b_\epsilon^{\text{new}} = \frac{\mathbb{E}_{\hat{\mathcal{P}}}[\epsilon] - \mathbb{E}_{\hat{\mathcal{P}}}[\epsilon^2]}{\mathbb{E}_{\hat{\mathcal{P}}}[\epsilon^2] - \mathbb{E}_{\hat{\mathcal{P}}}[\epsilon]^2}\left(1 - \mathbb{E}_{\hat{\mathcal{P}}}[\epsilon]\right)\,, \tag{6.32}$$

where we have used (A.19) and (A.20), and where

$$\mathbb{E}_{\hat{\mathcal{P}}}[\epsilon] = \frac{1}{Z_i}\left[\Phi(z_i)(1-\overline{\epsilon}^{\setminus i})\frac{a_\epsilon^{\setminus i}}{a_\epsilon^{\setminus i}+b_\epsilon^{\setminus i}+1} + (1-\Phi(z_i))\overline{\epsilon}^{\setminus i}\frac{a_\epsilon^{\setminus i}+1}{a_\epsilon^{\setminus i}+b_\epsilon^{\setminus i}+1}\right], \tag{6.33}$$

$$\begin{aligned}\mathbb{E}_{\hat{\mathcal{P}}}[\epsilon^2] = \frac{1}{Z_i}\Bigg[&\Phi(z_i)(1-\overline{\epsilon}^{\setminus i})\frac{a_\epsilon^{\setminus i}(a_\epsilon^{\setminus i}+1)}{(a_\epsilon^{\setminus i}+b_\epsilon^{\setminus i}+1)(a_\epsilon^{\setminus i}+b_\epsilon^{\setminus i}+2)} \\ &+(1-\Phi(z_i))\overline{\epsilon}^{\setminus i}\frac{(a_\epsilon^{\setminus i}+1)(a_\epsilon^{\setminus i}+2)}{(a_\epsilon^{\setminus i}+b_\epsilon^{\setminus i}+1)(a_\epsilon^{\setminus i}+b_\epsilon^{\setminus i}+2)}\Bigg].\end{aligned} \tag{6.34}$$

Note that these update equations do not minimize the KL-divergence between $\hat{\mathcal{P}}$ and $\mathcal{Q}$. However, we expect them to provide a good approximation as EP has shown similar results when using these update rules instead of the optimal ones in a related problem on weight estimation in mixtures of Gaussians (Minka, 2001b).

The next step of the EP algorithm consists in updating the corresponding approximate term $\tilde{t}_i$ by setting $\tilde{t}_i = Z_i\mathcal{Q}^{\text{new}}/\mathcal{Q}^{\setminus i}$

$$\mathbf{V}_i = \left((\mathbf{V}_{\mathbf{w}}^{\text{new}})^{-1} - (\mathbf{V}_{\mathbf{w}}^{\setminus i})^{-1}\right)^{-1}, \tag{6.35}$$

$$\mathbf{m}_i = \mathbf{V}_i\left((\mathbf{V}_{\mathbf{w}}^{\text{new}})^{-1}\mathbf{m}_{\mathbf{w}}^{\text{new}} - (\mathbf{V}_{\mathbf{w}}^{\setminus i})^{-1}\mathbf{m}_{\mathbf{w}}^{\setminus i}\right), \tag{6.36}$$

$$a_i = a_\epsilon^{\text{new}} - a_\epsilon^{\setminus i} + 1, \tag{6.37}$$

$$b_i = b_\epsilon^{\text{new}} - b_\epsilon^{\setminus i} + 1, \tag{6.38}$$

$$\begin{aligned}s_i = Z_i\frac{\beta(a_\epsilon^{\setminus i}, b_\epsilon^{\setminus i})}{\beta(a_\epsilon^{\text{new}}, b_\epsilon^{\text{new}})}\sqrt{\frac{|\mathbf{V}_{\mathbf{w}}^{\setminus i}|}{|\mathbf{V}_{\mathbf{w}}^{\text{new}}|}}\exp\Bigg(-\frac{1}{2}\Big((\mathbf{m}_{\mathbf{w}}^{\text{new}})^T(\mathbf{V}_{\mathbf{w}}^{\text{new}})^{-1}\mathbf{m}_{\mathbf{w}}^{\text{new}} - \\ (\mathbf{m}_{\mathbf{w}}^{\setminus i})^T(\mathbf{V}_{\mathbf{w}}^{\setminus i})^{-1}\mathbf{m}_{\mathbf{w}}^{\setminus i} - \mathbf{m}_i^T\mathbf{V}_i^{-1}\mathbf{m}_i\Big)\Bigg),\end{aligned} \tag{6.39}$$

where $\beta(a,b) = \Gamma(a)\Gamma(b)/\Gamma(a+b)$ is the beta function (Abramowitz and Stegun, 1964). We have used (A.46) for the computation of (6.35) and (6.36), and (A.30) for the computation of (6.37) and (6.38).

Once EP has converged, we can approximate the model evidence by

$$\mathcal{P}(\mathbf{y}) \approx \int\prod_{i=1}^{n+1}\tilde{t}_i(\mathbf{w},\epsilon)d\mathbf{w}d\epsilon = \sqrt{|\mathbf{V}_{\mathbf{w}}|}\frac{\beta(a_\epsilon,b_\epsilon)}{\beta(a_0,b_0)}\exp\left(\frac{B}{2}\right)\prod_{i=1}^{n}s_i, \tag{6.40}$$

where (A.27) and (A.42) have been used, and where

$$B = \mathbf{m}_{\mathbf{w}}^T\mathbf{V}_{\mathbf{w}}^{-1}\mathbf{m}_{\mathbf{w}} - \sum_{i=1}^{n}\mathbf{m}_i^T\mathbf{V}_i^{-1}\mathbf{m}_i. \tag{6.41}$$

Furthermore, in the computation of (6.40) we have also used that the approximate term corresponding to the prior, $\tilde{t}_{n+1}$, is initialized to (6.6) and never modified.

The EP algorithm for training the BM can be significantly sped-up by taking into account the special form of the matrix $\mathbf{V}_i^{-1}$ (Minka, 2001b). In particular, using (6.27) and the Woodbury formula (B.1) gives

$$\mathbf{V}_i^{-1} = (\mathbf{V}_{\mathbf{w}}^{\text{new}})^{-1} - (\mathbf{V}_{\mathbf{w}}^{\setminus i})^{-1} = v_i^{-1}\mathbf{x}_i\mathbf{x}_i^T \tag{6.42}$$

where

$$v_i = \mathbf{x}_i^T \mathbf{V}_{\mathbf{w}}^{\backslash i} \mathbf{x}_i \left(\frac{1}{\alpha_i \mathbf{x}_i^T \mathbf{m}_{\mathbf{w}}^{\text{new}}} - 1 \right) . \tag{6.43}$$

According to (6.42), the matrix $\mathbf{V}_i^{-1}$ has only one eigenvector in the direction of $\mathbf{x}_i$ and one eigenvalue. In consequence, $\mathbf{V}_i^{-1}$ can not be inverted to compute $\mathbf{V}_i$. This is a problem when computing the exact value of $\mathbf{m}_i$ in (6.36), since $\mathbf{V}_i$ cannot be evaluated. However, it can be shown that (see Appendix B.2)

$$\mathbf{m}_i = \mathbf{m}_{\mathbf{w}}^{\backslash i} + \alpha_i \mathbf{V}_{\mathbf{w}}^{\backslash i} \mathbf{x}_i + \alpha_i v_i \boldsymbol{\zeta} + K \boldsymbol{\xi} , \tag{6.44}$$

where $\boldsymbol{\zeta}$ is a vector such that $\mathbf{x}_i^T \boldsymbol{\zeta} = 1$, $\boldsymbol{\xi}$ is a vector such that $\mathbf{x}_i^T \boldsymbol{\xi} = 0$ and K is any arbitrary constant.

The special form of $\mathbf{V}_i^{-1}$ given in (6.42) allows to write the approximate terms $\tilde{t}_i$ as the unnormalized product of a univariate Gaussian distribution and a beta distribution. In particular, by substituting (6.42) into (6.13) we get

$$\tilde{t}_i(\mathbf{w}, \epsilon) = s_i \exp\left(\frac{1}{2v_i} \left(\mathbf{w}^T \mathbf{x}_i - m_i \right)^2 \right) \epsilon^{a_i - 1} (1 - \epsilon)^{b_i - 1} , \tag{6.45}$$

where

$$m_i = \mathbf{x}_i^T \mathbf{m}_i . \tag{6.46}$$

This representation has the advantage of significantly reducing the storage requirements for the approximate terms $\tilde{t}_i$ corresponding to the likelihood. The computational cost of updating these terms is also reduced as well.

When (6.45) is used instead of (6.13) to represent the approximate terms corresponding to the likelihood, (6.15) and (6.16) become

$$\begin{aligned} \mathbf{V}_{\mathbf{w}}^{\backslash i} &= \left(\mathbf{V}_{\mathbf{w}}^{-1} - v_i^{-1} \mathbf{x}_i \mathbf{x}_i^T \right)^{-1} \\ &= \mathbf{V}_{\mathbf{w}}^{-1} + (\mathbf{V}_{\mathbf{w}}^{-1} \mathbf{x}_i) \left(v_i - \mathbf{x}_i^T \mathbf{V}_{\mathbf{w}} \mathbf{x}_i \right)^{-1} (\mathbf{V}_{\mathbf{w}}^{-1} \mathbf{x}_i)^T , \end{aligned} \tag{6.47}$$

$$\begin{aligned} \mathbf{m}_{\mathbf{w}}^{\backslash i} &= \mathbf{m}_{\mathbf{w}} + \mathbf{V}_{\mathbf{w}}^{\backslash i} \mathbf{V}_i^{-1} (\mathbf{m}_{\mathbf{w}} - \mathbf{m}_i) \\ &= \mathbf{m}_{\mathbf{w}} + (\mathbf{V}_{\mathbf{w}}^{\backslash i} \mathbf{x}_i) v_i^{-1} (\mathbf{x}_i^T \mathbf{m}_{\mathbf{w}} - m_i) . \end{aligned} \tag{6.48}$$

To compute (6.47) we have used (6.42) and the Woodbury formula (B.1). To compute (6.48) we have used (6.42) and $\mathbf{V}_{\mathbf{w}}^{-1} = (\mathbf{V}_{\mathbf{w}}^{\backslash i})^{-1} + \mathbf{V}_i^{-1}$, which can be derived from (6.15). Similarly, (6.35), (6.36) and (6.39) become

$$v_i = \mathbf{x}_i^T \mathbf{V}_{\mathbf{w}}^{\backslash i} \mathbf{x}_i \left(\frac{1}{\alpha_i \mathbf{x}_i^T \mathbf{m}_{\mathbf{w}}^{\text{new}}} - 1 \right) , \tag{6.49}$$

$$m_i = \mathbf{x}_i^T \mathbf{m}_{\mathbf{w}}^{\backslash i} + \alpha_i (v_i + \mathbf{x}_i^T \mathbf{V}_{\mathbf{w}}^{\backslash i} \mathbf{x}_i) , \tag{6.50}$$

$$s_i = Z_i \frac{\beta(a_\epsilon^{\backslash i}, b_\epsilon^{\backslash i})}{\beta(a_\epsilon, b_\epsilon)} \sqrt{1 + v_i^{-1} \mathbf{x}_i^T \mathbf{V}_{\mathbf{w}}^{\backslash i} \mathbf{x}_i} \exp\left(\frac{1}{2} \frac{\alpha_i \mathbf{x}_i^T \mathbf{V}_{\mathbf{w}}^{\backslash i} \mathbf{x}_i}{\mathbf{x}_i^T \mathbf{m}_{\mathbf{w}}^{\text{new}}} \right) , \tag{6.51}$$

where (6.46) and (6.44) have been used to compute (6.50). The derivation of (6.51) is given in Appendix B.3. Finally, the value of B in (6.40) is

$$B = \mathbf{m}_{\mathbf{w}}^T \mathbf{V}_{\mathbf{w}}^{-1} \mathbf{m}_{\mathbf{w}} - \sum_{i=1}^{n} \frac{m_i^2}{v_i} . \tag{6.52}$$

Consider the parameters of the posterior approximation $\mathcal{Q}$. From the definition of $\mathcal{Q}$ as the normalized product of all the approximate terms $\tilde{t}_i$, these parameters are

$$\mathbf{V}_\mathbf{w} = \left(\mathbf{I} + \sum_{i=1}^{n} v_i^{-1} \mathbf{x}_i \mathbf{x}_i^T \right)^{-1} = \left(\mathbf{I} + \mathbf{X} \boldsymbol{\Lambda}^{-1} \mathbf{X}^T \right)^{-1} , \tag{6.53}$$

$$\mathbf{m}_\mathbf{w} = \mathbf{V}_\mathbf{w} \sum_{i=1}^{n} m_i v_i^{-1} \mathbf{x}_i , \tag{6.54}$$

$$a_\epsilon = \sum_{i=1}^{n+1} a_i - n , \tag{6.55}$$

$$b_\epsilon = \sum_{i=1}^{n+1} b_i - n , \tag{6.56}$$

where $\mathbf{I}$ is the identity matrix and $\boldsymbol{\Lambda} = \mathrm{diag}(v_1, \ldots, v_n)$. We have used (A.42) for the computation of (6.53) and (6.54), and (A.27) for the computation of (6.55) and (6.56).

The EP algorithm can be reformulated in terms of inner products (Minka, 2001b). This is useful to apply the kernel tick for feature expansion. When expressed in terms of inner products, the EP algorithm no longer requires the parameters of the posterior approximation $\mathbf{m}_\mathbf{w}$ and $\mathbf{V}_\mathbf{w}$ to be stored in memory. Instead, all the steps of the algorithm can be written in terms of the quantities

$$\mathbf{A} = \mathbf{X}^T \mathbf{V}_\mathbf{w} \mathbf{X} , \tag{6.57}$$

$$\mathbf{h} = \mathbf{X}^T \mathbf{m}_\mathbf{w} , \tag{6.58}$$

which are updated after each iteration. To describe EP in terms of inner products, let us define

$$\lambda_i^{\backslash i} = \mathbf{x}_i^T \mathbf{V}_\mathbf{w}^{\backslash i} \mathbf{x}_i , \tag{6.59}$$

$$C_{ij} = \mathbf{x}_i^T \mathbf{x}_j , \tag{6.60}$$

$$\boldsymbol{\Lambda} = \mathrm{diag}(v_1, \ldots, v_n) , \tag{6.61}$$

$$h_i = \mathbf{x}_i^T \mathbf{m}_\mathbf{w} , \tag{6.62}$$

$$h_i^{\backslash i} = \mathbf{x}_i^T \mathbf{m}_\mathbf{w}^{\backslash i} . \tag{6.63}$$

In terms of these quantities, the algorithm is:

1. Initialize all $\tilde{t}_i$, with $i = 1, \ldots, n$, to be uniform and $\mathcal{Q}$ to the prior (6.6) by setting

$$v_i = +\infty , \quad m_i = 0 , \quad s_i = 1 , \quad a_i = b_i = 1 , \quad h_i = 0 , \quad \mathbf{A} = \mathbf{C} . \tag{6.64}$$

2. Repeat until all $\tilde{t}_i$ converge:

(a) Choose a $\tilde{t}_i$ to refine and remove $\tilde{t}_i$ from $\mathcal{Q}$ to get $\mathcal{Q}^{\backslash i}$.

$$h_i^{\backslash i} = h_i + \frac{A_{ii}}{v_i - A_{ii}}(h_i - m_i)\,, \tag{6.65}$$

$$\lambda_i^{\backslash i} = A_{ii} + \frac{A_{ii}^2}{v_i - A_{ii}}\,, \tag{6.66}$$

$$a_\epsilon^{\backslash i} = a_\epsilon - a_i + 1\,, \tag{6.67}$$

$$b_\epsilon^{\backslash i} = b_\epsilon - b_i + 1\,. \tag{6.68}$$

The derivation of (6.65) and (6.66) is presented in Appendix B.4.

(b) Update part of the posterior approximation:

$$\overline{\epsilon}^{\backslash i} = \frac{a_\epsilon^{\backslash i}}{a_\epsilon^{\backslash i} + b_\epsilon^{\backslash i}}\,, \tag{6.69}$$

$$z_i = \frac{h_i^{\backslash i}}{\sqrt{\lambda_i^{\backslash i}}}\,, \tag{6.70}$$

$$Z_i = \overline{\epsilon}^{\backslash i} + (1 - 2\overline{\epsilon}^{\backslash i})\Phi(z_i)\,, \tag{6.71}$$

$$\alpha_i = \frac{1}{\sqrt{\lambda_i^{\backslash i}}} \frac{(1 - 2\overline{\epsilon}^{\backslash i})\mathcal{N}(z_i|0,1)}{\overline{\epsilon}^{\backslash i} + (1 - 2\overline{\epsilon}^{\backslash i})\Phi(z_i)}, \tag{6.72}$$

$$h_i^{\text{new}} = \mathbf{x}_i^T \mathbf{m}_{\mathbf{w}}^{\text{new}} = h_i^{\backslash i} + \lambda_i^{\backslash i}\alpha_i\,, \tag{6.73}$$

where we have used (6.26) for the computation of (6.73). Then, a_ϵ^{new} and b_ϵ^{new} are obtained as in (6.31) and (6.32).

(c) Find the corresponding approximate term $\tilde{t}_i$:

$$a_i = a_\epsilon^{\text{new}} - a_\epsilon^{\backslash i} + 1\,, \tag{6.74}$$

$$b_i = b_\epsilon^{\text{new}} - b_\epsilon^{\backslash i} + 1\,, \tag{6.75}$$

$$v_i = \lambda_i^{\backslash i}\left(\frac{1}{\alpha_i h_i^{\text{new}}} - 1\right)\,, \tag{6.76}$$

$$m_i = h_i^{\backslash i} + (v_i + \lambda_i^{\backslash i})\alpha_i = h_i^{\text{new}} + v_i\alpha_i\,, \tag{6.77}$$

$$s_i = Z_i \frac{\beta(a_\epsilon^{\backslash i}, b_\epsilon^{\backslash i})}{\beta(a_\epsilon^{\text{new}}, b_\epsilon^{\text{new}})}\sqrt{1 + v_i^{-1}\lambda_i^{\backslash i}}\exp\left(\frac{\lambda_i^{\backslash i}\alpha_i}{2h_i^{\text{new}}}\right)\,, \tag{6.78}$$

where we have used (6.49), (6.50) and (6.51) for the computation of (6.76), (6.77) and (6.78), respectively.

(d) Update the matrix $\mathbf{A}$ and the vector $\mathbf{h}$:

$$\mathbf{A}^{\text{new}} = \mathbf{A} - \frac{\mathbf{A}_{\cdot,i}\mathbf{A}_{i,\cdot}}{\kappa^{-1} + A_{ii}}\,, \tag{6.79}$$

$$\mathbf{h}^{\text{new}} = \mathbf{A}^{\text{new}}(\mathbf{\Lambda}^{\text{new}})^{-1}\mathbf{m}^{\text{new}}\,, \tag{6.80}$$

where

$$\kappa = \left(\frac{1}{v_i^{\text{new}}} - \frac{1}{v_i^{\text{old}}}\right)\,, \tag{6.81}$$

and $\boldsymbol{\Lambda}^{\text{new}}$ and $\mathbf{m}^{\text{new}}$ are the matrix $\boldsymbol{\Lambda}$ and the vector $\mathbf{m}$ after the approximate term $\tilde{t}_i$ has been updated. In (6.79) $\mathbf{A}_{i,\cdot}$ denotes the i-th row of the matrix $\mathbf{A}$ and $\mathbf{A}_{\cdot,i}$ denotes the i-th column. Finally, in (6.81) v_i^{new} is the new value for the parameter v_i of the approximate term and v_i^{old} is the previous value. The derivation of (6.79) and (6.80) is given in Appendix B.5.

3. Once EP has converged, compute the normalizing constant:

$$\mathcal{P}(\mathbf{y}) \approx \sqrt{\frac{|\boldsymbol{\Lambda}|}{|\mathbf{C}+\boldsymbol{\Lambda}|}} \frac{\beta(a_\epsilon, b_\epsilon)}{\beta(a_0, b_0)} \exp\left(\frac{B}{2}\right) \prod_{i=1}^{n} s_i \,, \tag{6.82}$$

where

$$B = \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}\frac{m_i m_j}{v_i v_j} - \sum_{i=1}^{n}\frac{m_i^2}{v_i} \,. \tag{6.83}$$

To arrive at (6.82) we have used (6.40), (6.52), (6.54) and (6.57). Finally, the value of $|\mathbf{V}_{\mathbf{w}}|$ is derived in Appendix B.6.

Because (6.79) can be computed in $\mathcal{O}(n^2)$ operations using the Woodbury formula, the total cost of the EP algorithm assuming a constant number of iterations is $\mathcal{O}(n^3)$, where n is the size of the training set. After convergence, the posterior approximation $\mathcal{Q}$ can be used to approximate the predictive distribution (6.10) for a new instance $\mathbf{x}_{\text{new}}$

$$\mathcal{P}(y_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{y}) \approx \int \mathcal{P}(y_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{w}, \epsilon)\mathcal{Q}(\mathbf{w}, \epsilon) d\mathbf{w} d\epsilon = \overline{\epsilon} + (1 - 2\overline{\epsilon})\Phi(z) \,, \tag{6.84}$$

where $\overline{\epsilon} = a_\epsilon/(a_\epsilon + b_\epsilon)$ and z can be computed by means of the Woodbury formula

$$z = \frac{\mathbf{m}_{\mathbf{w}}^T \mathbf{x}_{\text{new}}}{\sqrt{(\mathbf{x}_{\text{new}})^T \mathbf{V}_{\mathbf{w}} \mathbf{x}_{\text{new}}}} = \frac{\sum_{i=1}^{n} \alpha_i \mathbf{x}_{\text{new}}^T \mathbf{x}_i}{\sqrt{(\mathbf{x}_{\text{new}})^T \mathbf{x}_{\text{new}} - (\mathbf{x}_{\text{new}})^T \mathbf{X}\mathbf{K}\mathbf{X}^T \mathbf{x}_{\text{new}}}} \,, \tag{6.85}$$

with $\mathbf{K} = \boldsymbol{\Lambda}^{-1}(\boldsymbol{\Lambda} - \mathbf{A})\boldsymbol{\Lambda}^{-1}$. The derivation of (6.85) is presented in Appendix B.7. Unlike standard GPCs, computing (6.84) does not need any direct matrix inversion. The inversion of $\boldsymbol{\Lambda}$ is straightforward because it is a diagonal matrix. The inner product matrix (Gram matrix) $\mathbf{C}$ between vectors can be written in terms of an arbitrary kernel function $C(\mathbf{x}_i, \mathbf{x}_j)$. However, in that case we need to keep y_i and $\mathbf{x}_i$ separated as follows

$$C_{ij} = y_i y_j C(\mathbf{x}_i, \mathbf{x}_j) \,. \tag{6.86}$$

Employing a non-linear kernel function to compute $\mathbf{C}$ effectively expands the feature space, so that the BM has the capacity of producing non-linear decision boundaries. Finally, the approximate evidence (6.82) can be used to determine which kernel is more suited to the training data.

As described in Chapter 5, besides EP there are other methods that could have been employed to carry out approximate inference in this model. An alternative approach is the Laplace approximation, which is based on placing a Gaussian at the peak of the posterior distribution (Barber and Williams, 1997; Kuss and Rasmussen, 2005). However, this technique is not really suited to the model assumed because when conditioned to ϵ, the $\mathbf{w}$ component of the posterior distribution in (6.9) is a Gaussian distribution modulated by a piecewise constant function (Minka et al., 2009). Thus, it is unlikely that the gradient of the posterior distribution can be used to find regions of high posterior

probability. For this same reason, the Hamilton Monte Carlo algorithm (Bishop, 2006; MacKay, 2003) is not expected to be useful in this model. Another alternative is to use variational inference (Gibbs and MacKay, 2000). This method consists in optimizing a lower bound on the data log likelihood which is often computed by means of Jensen's inequality (Jaakkola, 2001). However, computing this lower bound can be difficult in this model. Specifically, even though the numerator of the posterior distribution and particularly the likelihood function (6.4) simplify when taking logarithms, the lower bound can be undefined for $\epsilon = 0$, since the likelihood function is zero for different values of $\mathbf{w}$.

Finally, we would like to remark that even though it is based on similar concepts, we believe that the approach followed to train the BM is more robust than the EM-EP algorithm used by Kim and Ghahramani (2006) for training GPCs. The BM approximates a posterior distribution for the noise parameter ϵ while the EM-EP algorithm maximizes the marginal likelihood of the data with respect to this parameter. Furthermore, in order to perform this maximization, an approximation is made so that the contribution to a lower bound on the marginal likelihood can be evaluated.

6.2.2 Experiments

In this section we assess the performance of the BM in several classification problems. First, we analyze in detail a simple problem to determine whether the BM can learn the intrinsic noise in the class labeling of the data. We generate a simple 2-class 2-dimensional dataset whose input features x_1 and x_2 are uniform random numbers between -1 and 1. The class assigned to each instance, either $+1$ or -1, is determined by the decision rule $x_1^2 + x_2^2 \geq 1/2$. Hence, the decision boundary of this problem is a circumference of radius $\sqrt{1/2}$ centered at the origin. Each training set is composed of 100 points. Labeling errors are introduced to a fraction of randomly selected training instances. In particular, in this classification problem we flip the labels of 0%, 5%, 10% and 20% of the training data. Finally, each test set is composed of $1,000$ noiseless points. We generate 50 training and test sets for each one of the noise values described before. For each training and test set combination, a BM is built on the training set and evaluated on the test set. In a similar way, a BM that assumes no labeling errors (the prior over the noise parameter ϵ is fixed to be a delta function centered at zero, i.e. $b_0 = +\infty$) is also built and evaluated. In both cases a Gaussian kernel is used for feature expansion. The parameter of this kernel is determined by a grid search procedure that will be described later on in this section.

Table 6.1 displays for each value of the noise injected the test error of each method averaged over the 50 realizations of each classification problem. The second column displays the error rate for the BM that learns the intrinsic noise in the class labeling. The third column displays the error rate for the BM model with $\epsilon = 0$ ($\text{BM}_{\epsilon=0}$). Finally, the last column displays the expected value of the ϵ parameter given by the posterior approximation of the BM. We note that when there is no noise in the training set the BM obtains an error rate similar to the one provided by the BM that assumes no labeling errors. This is a very interesting result because in this situation assuming $\epsilon = 0$ is optimal. Furthermore, these experiments show that as the noise level induced in the class labeling of the training set increases, the error of $\text{BM}_{\epsilon=0}$ becomes larger than the error of the standard BM in which the possibility of mislabeled training instances is considered. The table also shows that the expected value of ϵ in the BM model increases with the noise level of the training set. Thus, these results confirm that the BM is able

to capture the intrinsic noise in the class labeling by means of the posterior distribution of the ϵ parameter.

TABLE 6.1: Classification error in the simple problem for each noise value of a BM and a BM that assumes no labeling error ($\epsilon = 0$). The expected value in % of the ϵ parameter of the BM is also displayed in the last column of this table.

Noise	**BM**	**BM**$_{\epsilon=0}$	$\mathbb{E}[\epsilon]$ (in %)
0%	2.8±1.0	2.8±1.0	1.0±0.1
5%	4.3±2.3	9.8±2.0	5.0±0.8
10%	5.7±2.7	14.6±2.9	8.8±1.3
20%	13.4±5.5	24.3±3.2	15.5±3.0

Figure 6.1 (top) shows a realization of the training set for the problem described before and the decision boundary obtained by a BM that assumes no labeling noise. The optimal decision boundary is displayed as a green circumference. This model cannot account for outliers properly. The presence of outliers in the data has a strong influence in the decision boundary obtained, leading to small regions in which the incorrect class is predicted. On the other hand, Figure 6.1 (bottom) shows the training set and the decision boundary generated by a BM that models the noise as described in the previous sections. In this case, the figure shows that the decision boundary of the BM agrees quite well with the optimal one (green circumference).

The performance of BMs trained with the proposed algorithm is investigated more extensively in real and synthetic classification problems. The detailed information of each dataset is displayed in Table 6.2. Most of these classification problems are taken from the UCI Repository (Asuncion and Newman, 2007). The *Boston Housing* problem is transformed into a binary classification problem by discriminating between houses worth more than $\$21,000$ and houses worth less than this threshold. In this problem, the *Charles River* binary attribute (= 1 if the tract bounds the river; 0 otherwise) is removed, so that only 12 attributes are considered. In the *Ionosphere* problem a non-informative attribute (all values are equal to zero) is also removed. The *Digits* problem is transformed into a binary classification problem by considering only two possible digits, three and five. Instances containing missing values are removed in the *Breast Cancer* problem. The *Crabs* problem is described in (Ripley, 1996)[1]. The *Twonorm* and *Threenorm* problems are the synthetic classification problems described in (Breiman, 1996b). In general, the training sets are small because the cost of the learning algorithms compared in the experiments is rather high. They all have a cubic dependence $\mathcal{O}(n^3)$ on the number of training instances n.

We also compare the performance of the BM with other popular kernel classifiers: SVMs (Vapnik, 1995) and GPCs based on the EM-EP algorithm, as described in (Kim and Ghahramani, 2006). We do not compare with BPMs because if EP is used to approximate the Bayes point the resulting classifier is equal to the GPC (Minka, 2001b). Because the kernel function has a strong influence in the final decision boundary of these models, we use a Gaussian kernel function in all the experiments

$$C(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma |\mathbf{x}_i - \mathbf{x}_j|^2\right) , \tag{6.87}$$

[1]The problem is available online at http://www.stats.ox.ac.uk/pub/PRNN

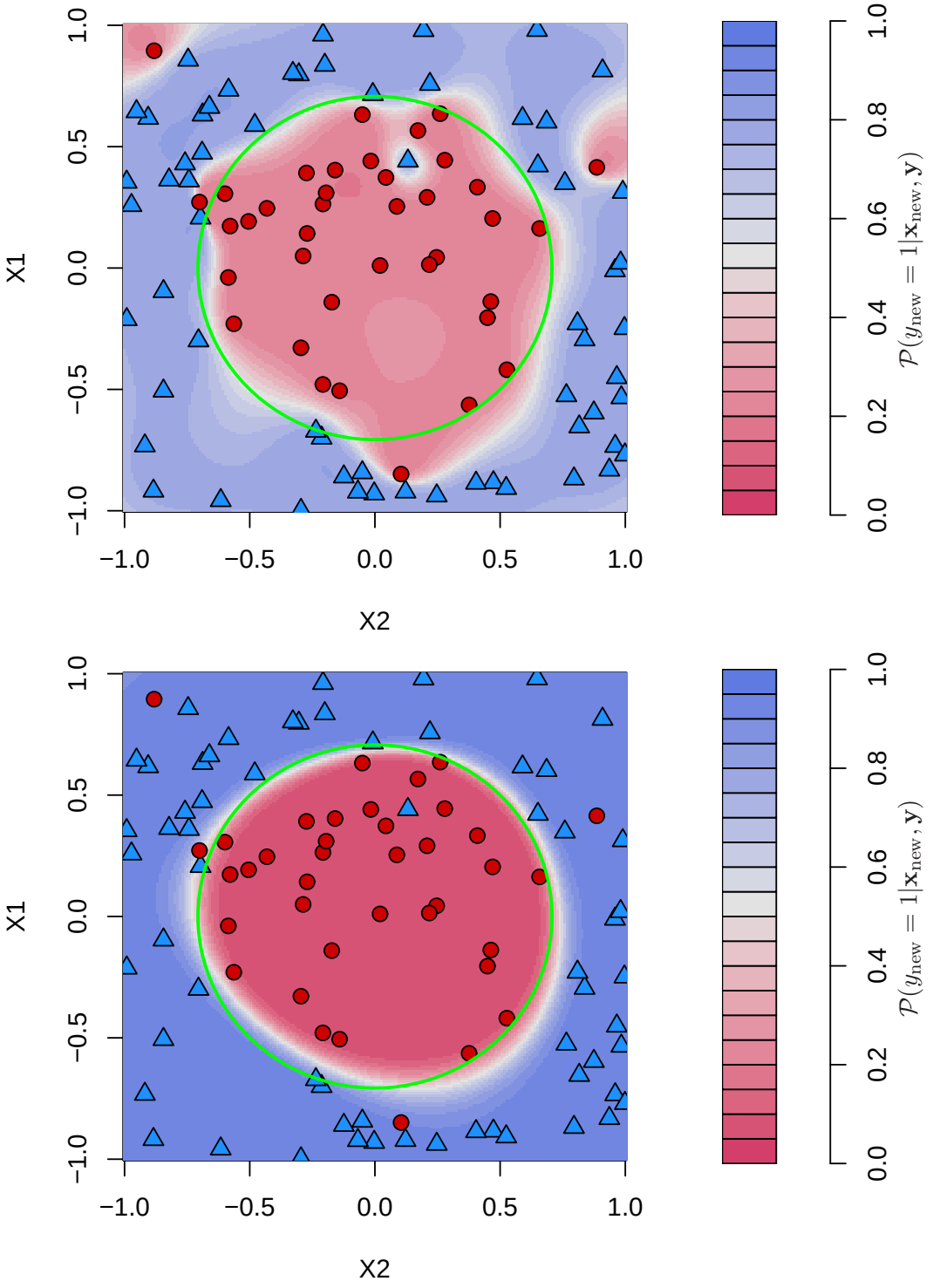

FIGURE 6.1: (top) Training set and decision boundary of a BM that makes the assumption of zero noise ($\epsilon = 0$). (bottom) Training set and decision boundary of a BM that is able to learn the noise intrinsic in the data. Optimal decision boundary is a $\sqrt{0.5}$ radius circumference centered at the point $(0, 0)$. Five outliers have been injected in the training set. Contour curves describe the predictive distribution of each model.

TABLE 6.2: Description of the classification problems used in the experiments.

Problem	**Attributes**	**Train**	**Test**	**Classes**
Australian	14	100	590	2
Boston	12	100	406	2
Breast	9	100	583	2
Crabs	7	66	134	2
Digits	64	70	695	2
German	20	100	900	2
Heart	13	90	180	2
Ionosphere	33	117	234	2
New-Thyroid	6	100	115	2
Magic	10	100	18,920	2
Pima	8	100	668	2
Sonar	60	69	139	2
Spam	58	100	4,501	2
Threenorm	20	100	1,000	2
Twonorm	20	100	1,000	2

where γ is a parameter that has to be tuned in some way. In the SVM this parameter is tuned carrying out a grid search using 10-fold cross validation on the training set to evaluate the performance for each value of γ considered. The natural logarithm of the values of the knots in the grid are ten evenly distributed points in the interval $[-3, 3]$. For the problems *Boston*, *Crabs*, *Digits*, *German*, *Magic* and *Spam* a wider interval $[-6, 6]$ is considered. In the case of the BM the same grid search is carried out, but instead of the average error over the left-out data, the evidence given by (6.82) is used to select the value of γ. The GPC selects automatically this parameter and the noise parameter ϵ by maximizing the log-evidence of the data using an expectation maximization (EM) algorithm (Kim and Ghahramani, 2006). In this method, the initial values of γ and ϵ are fixed to 1 and 0.01 respectively. In the SVM the cost parameter C is automatically determined by an additional nested grid search in which the performance of the different values of C are evaluated by 10-fold cross validation. The grid of values employed are the same as the ones described in first place. Unlike the SVM and the GPC, the BM learns a posterior distribution over the noise parameter ϵ with no additional cost. Because the approach followed by (Kim and Ghahramani, 2006) needs to invert the Gram matrix $\mathbf{C}$, in this method we add a small constant (10^{-3}) to the diagonal terms of $\mathbf{C}$ given by (6.87) when $i = j$. This guarantees that the inverse $\mathbf{C}^{-1}$ actually exists and that the EM-EP algorithm can be carried out. The addition of this constant to the diagonal of $\mathbf{C}$ is equivalent to adding a small amount of Gaussian noise to the latent function of the GPC (MacKay, 2003).

To get reliable results, each dataset is randomly split 50 times into training and test partitions according to the sizes given by Table 6.2. In each partition, the features are normalized so that they have zero mean and unit standard deviation in the training set. Then, each kernel classifier is built on the training set and validated on the test set. The prediction errors reported are averages over the different train and test partitions. These values are displayed in Table 6.3. The lowest error for each problem is highlighted in boldface. We note that the BM obtains the lowest error in eleven of the fifteen

classification problems whereas the GPC obtains the best error in three of the problems and the SVM only in one.

TABLE 6.3: Average error in % and standard deviation for each problem and method. For each classification problem, it is highlighted in bold face the best error value.

Problem	**BM**	**SVM**	**GPC**
Australian	**15.1±1.3**	15.8±2.4	15.5±1.6
Boston	14.8±1.8	16.4±1.9	**14.7±1.8**
Breast	**3.3±0.5**	3.5±0.5	3.5±0.7
Crabs	**0.9±0.9**	1.0±1.1	1.3±1.3
Digits	1.6±0.6	1.9±0.9	**1.6±0.6**
German	**27.7±1.8**	29.0±2.0	28.3±1.9
Heart	**18.1±2.1**	18.9±2.8	19.2±2.2
Ionosphere	11.1±2.0	**6.4±1.7**	10.3±1.7
Magic	**21.1±2.5**	21.1±2.0	21.4±2.4
New-thyroid	4.6±1.9	5.1±1.6	**4.5±1.9**
Pima	**27.0±2.0**	27.2±2.5	27.6±2.3
Sonar	**19.3±3.8**	24.6±4.6	20.0±3.0
Spam	**12.1±1.5**	14.7±2.3	12.7±1.9
Threenorm	**15.6±1.6**	16.2±1.8	16.6±1.9
Twonorm	**3.0±0.5**	3.2±0.7	3.4±0.7

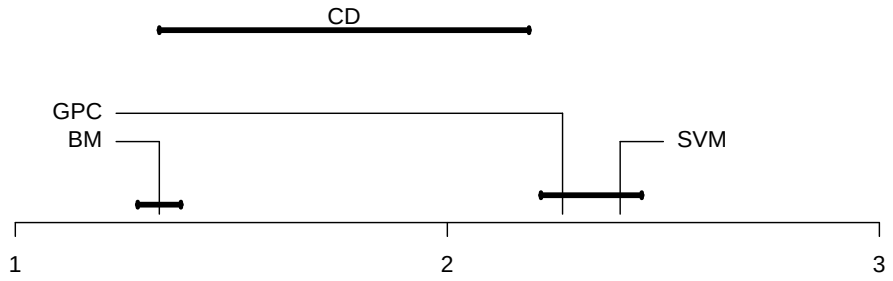

FIGURE 6.2: Graphical representation of a Nemenyi test comparing the overall performance of the BM, the SVM and the GPC. The average rank of each different method is displayed. The critical difference (CD) between these ranks appears at the top as a solid segment. Groups of classifiers that are not significantly different with a confidence level of 5% are conected.

To compare overall results we use the framework proposed by Demšar (2006). We carry out a Friedman average rank test to determine whether the differences in performance of the models investigated are statistically significant in the set of problems considered. The results of the test is that the null-hypothesis, which states that the performance of all models is equivalent, can be discarded with a p-value below 5%. As a result, we apply a Nemenyi test to determine whether the differences in average rank among the different models being compared are statistically significant. Figure 6.2.2 displays the result of this statistical test. The figure shows the average rank of each model on the set of fifteen problems investigated. Classifiers that are not statistically different with a significance level of 5% are connected with a solid black line. The results of this

test show that BMs perform better than SVMs and GPCs on this set of classification problems. The average rank of the GPC is better than the average rank of the SVM. However, this difference is not statistically significant.

The computational cost of BMs and GPCs is $\mathcal{O}(n^3)$, where n is the size of the training set. On the other hand, the cost of SVMs is at worst $\mathcal{O}(n^2)$ because we employed the SMO-type training algorithm provided in the *LIBSVM* (Chang and Lin, 2001). However, the grid search used to determine the parameters of γ and C in this method requires training several SVMs with the corresponding additional cost. Training the GPC is also very expensive because the convergence of EM-EP is rather slow. In the implementation made, the algorithm is halted after 100 iterations if it has not converged.

6.3 Sparse Bayes Machines

As described in the previous section, the Bayes machine (BM) is a kernel classifier for binary classification problems. The BM extends the Bayes point machine (Herbrich et al., 2001) by taking into account all hypotheses that are consistent with the training data in feature space, considering the possibility of mislabeled data. To make inference, Bayes' theorem is employed to compute a posterior distribution for the model parameters. This posterior distribution includes a parameter that quantifies the level of noise in the class labels of the training data. Exact inference in the BM is infeasible and hence, approximate techniques have to be employed. In the previous section, approximate inference is carried out using the expectation propagation (EP) algorithm (Minka, 2001b). EP can be employed to approximate the posterior distribution extended with the noise level parameter without increasing the computational complexity of the algorithm. In consequence, the BM can learn this parameter without any significant increment in the training cost, unlike support vector machines (SVMs) (Vapnik, 1995) or Gaussian process classifiers (GPCs) (Kim and Ghahramani, 2006). These methods also introduce a parameter to model noisy labels. However, this parameter has to be tuned by cross-validation or by type-II maximum likelihood estimation, which involve repeatedly running the training algorithm. BMs trained using the EP algorithm provide competitive results, in terms of prediction accuracy, when compared to these kernel classifiers. The cost of the EP algorithm is $\mathcal{O}(n^3)$, where n is the number of training instances. This cubic dependence in the size of the training set becomes a major drawback when large scale classification problems are considered. In this section we propose to overcome this limitation by obtaining a sparse representation for the BM (Hernández-Lobato, 2008). This sparse representation is based on the approach used in the informative vector machine (IVM) (Lawrence et al., 2003; Seeger, 2003), which can be used to train sparse GPCs.

Besides the IVM, another approach for obtaining sparse GPCs is proposed in (Csató and Opper, 2002). As in the IVM, this method employes assumed density filtering (ADF) (Minka, 2001b; Opper, 1998) to implement approximate Bayesian inference and hence, it can be considered as an online approach. In particular, sparseness is introduced by using a suitable approximation to the posterior distribution of the GPC. This posterior distribution is described in terms of a linear combination of kernel functions. In standard GPCs, every time a new data instance is observed the coefficients of this linear combination are updated and a new kernel function is incorporated. If a certain measure of the approximation error is not exceeded, Csató and Opper (2002) propose to perform this update without increasing the number of kernel functions included in the representation of the posterior distribution of the GPC. For this purpose, the new

kernel function is approximated as a linear combination of the previous kernel functions included in the model. This approach has been extended in (Csató et al., 2001) where an expectation-propagation type algorithm that performs multiple sweeps through the training data is proposed. Even though the robustness of this method has been demonstrated on a range of experiments (Csató and Opper, 2002), we believe that the approach followed by the IVM is more intuitive and hence, we focus on it.

In the IVM a sparse GPC is built using only a reduced set of active instances $\mathcal{I}$, of size $d < n$, which is extracted from the training set. The active set is computed in such a way that the posterior approximation obtained by using these instances only is as close as possible to the one that is obtained when all the available training instances are used. In particular, the IVM determines the active set using a greedy algorithm. Starting from an empty active set, instances are iteratively included in $\mathcal{I}$ by using the Kullback-Liebler (KL) divergence as a measure of dissimilarity between the current posterior distribution and an updated one that includes on more instance. The algorithm stops when $d < n$ instances have been included in the active set. A disadvatage of this greedy approach is that the algorithm can include some instances at the beginning that should not be included and vice-versa. To mitigate this problem, in this section we perform additional refining iterations in the greedy selection procedure. The purpose of these extra iterations is twofold. First, they allow to exlude from the active set instances that had been included in earlier iterations of the algorithm. Thus, we expect them to correct some of the suboptimal selections made by the greedy algorithm. Second, the additional preprocessing of the elements included in $\mathcal{I}$ improves the accuracy of the approximation to the posterior distribution. The proposed algorithm works in a similar way as the EP algorithm, which corrects some of the mistakes of the ADF approach (Minka, 2001b).

If efficient matrix factorizations based on the Cholesky decomposition are employed in the training algorithm, the cost of learning this sparse representation is reduced to $\mathcal{O}(nd^2)$. This is a great improvement if the size of the active set, d, is much smaller than n. The cost of classifying new instances is also reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(d^2)$ when this representation in used. Unlike in the SVM, one of the advantages of the proposed approach is that the size of the active set, d, can be controlled by the user. This can be useful for intance to identify those instances that are more relevant for classification. However, setting the value of sparsity parameter d to get a good generalization performance can be difficult. Finally, even though the final model obtained by using this sparse representation actually depends only on a reduced subset of the total data, experiments carried out on the MNIST handwritten digits dataset (LeCun et al., 1998) confirm that its preformance is comparable to the standard BM. Additionally, these experiments show that this sparse representation can outperform the SVM in terms of prediction accuracy in the classification problems investigated

6.3.1 A Sparse Representation for the Bayes Machine

In this section we describe an algorithm to train a sparse BM (SBM). For this purpose, we follow a procedure that is simular to the one used in the IVM (Lawrence et al., 2003; Seeger, 2003). However, we introduce some modifications to obtain a better approximation to the posterior distribution of the model parameters.

The simplest way to obtain a sparse representation for the BM is to set $v_i = +\infty$, $a_i = 1$ and $b_i = 1$ in some of the approximate terms $\tilde{t}_i$, with $i = 1, \ldots, n$, corresponding to the likelihood. With these values for the parameters, the corresponding terms $\tilde{t}_i$ in the approximation are uniform. Therefore, they do not contribute to the posterior

approximation computed by EP. In particular, assume that we want to build our model using only a subset $\mathcal{S}$ of the total observed data. Furthermore, assume that the active set $\mathcal{I} \subset \{1, \ldots, n\}$ is a set with the indices of the data instances in $\mathcal{S}$ and that $\mathcal{J} = \{1, \ldots, n\} \setminus \mathcal{I}$ is a set containing the indices of the remaining observed instances. If we set $v_j = +\infty$ and $a_j = b_j = 1$, $\forall j \in \mathcal{J}$

$$\mathcal{Q}(\mathbf{w}, \epsilon) = \frac{1}{Z} \prod_{i=1}^{n+1} \tilde{t}_i(\mathbf{w}, \epsilon) = \frac{1}{Z'} \tilde{t}_{n+1}(\mathbf{w}, \epsilon) \prod_{i \in \mathcal{I}} \tilde{t}_i(\mathbf{w}, \epsilon)\,, \tag{6.88}$$

where

$$Z = \int \prod_{i=1}^{n+1} \tilde{t}_i(\mathbf{w}, \epsilon) d\mathbf{w} d\epsilon\,, \qquad Z' = \int \tilde{t}_{n+1}(\mathbf{w}, \epsilon) \prod_{i \in \mathcal{I}} \tilde{t}_i(\mathbf{w}, \epsilon) d\mathbf{w} d\epsilon\,. \tag{6.89}$$

In (6.88) we have used the relation

$$\tilde{t}_i(\mathbf{w}, \epsilon) \tilde{t}_j(\mathbf{w}, \epsilon) \propto \tilde{t}_i(\mathbf{w}, \epsilon)\,, \quad \forall i \in \mathcal{I} \text{ and } \forall j \in \mathcal{J}\,, \tag{6.90}$$

which can be derived using (A.27), (A.42) and the special form of the approximate terms $\tilde{t}_i$, defined in (6.45) as the unnormalized product of a Gaussian distribution and a beta distribution. In consequence, setting $v_j = +\infty$, $a_j = 1$ and $b_j = 1$ $\forall j \in \mathcal{J}$ has the same effect in the EP algorithm as removing from the training set those instances associated to the approximate terms $\tilde{t}_j$, $j \in \mathcal{J}$. Thus, from this point on, we maintain an active set of instances $\mathcal{I} \subset \{1, \ldots, n\}$, $|\mathcal{I}| = d < n$ so that $\forall j \in \mathcal{J} = \{1, \ldots, n\} \setminus \mathcal{I}$, $v_j = +\infty$ and $a_j = b_j = 1$. The instances whose indices are not included in $\mathcal{I}$ are automatically discarded by the model and hence, they do not contribute to the computation of the posterior approximation.

The difficult part of this approach is how to select $\mathcal{I}$ so that the posterior approximation that is obtained is as similar as possible to the one that results from considering all the training instances. For this purpose, we implement the greedy algorithm proposed in (Lawrence et al., 2003), albeit with some refinements. The algorithm is as follows: (i) As in the standard EP algorithm, the posterior approximation $\mathcal{Q}$ and the approximate term $\tilde{t}_{n+1}$ are initialized to the prior (6.6). The approximate terms $\tilde{t}_i$ corresponding to the likelihood are initialized to be uniform by setting $v_i = +\infty$, $a_i = 1$ and $b_i = 1$ $\forall i \in \{1, \ldots, n\}$. (ii) The active set $\mathcal{I}$ is initialized to a random set of d elements extracted from $\{1, \ldots, n\}$. (iii) Until convergence of the active set $\mathcal{I}$ and the approximate terms $\tilde{t}_i$, with $i = 1, \ldots, n$, we iterate over all the elements of $\mathcal{I}$. (iv) For each $i \in \mathcal{I}$, we update $\mathcal{I}$ by setting $\mathcal{I} = \mathcal{I} \setminus \{i\}$. The corresponding approximate term $\tilde{t}_i$ is set uniform by fixing $v_i = +\infty$ and $a_i = b_i = 1$. Finally, the posterior approximation $\mathcal{Q}$ is updated to $\mathcal{Q}^{\setminus i}$ as in the standard EP algorithm. (v) We select a candidate instance $j \in \mathcal{J} = \{1, \ldots, n\} \setminus \mathcal{I}$ to be included in the active set $\mathcal{I}$. Note that after the first removal $i \in \mathcal{J}$, so it is possible that the instance that has been removed could be selected and included again in $\mathcal{I}$. As suggested by Seeger (2003), we select the instance in $\mathcal{J}$ that maximizes the KL-divergence between $\mathcal{Q}_j^{\text{new}}$ and the current posterior approximation $\mathcal{Q}^{\setminus i}$, where $\mathcal{Q}_j^{\text{new}}$ is the distribution that results from processing the corresponding exact term t_j

$$\mathcal{Q}_j^{\text{new}} = \arg\min_{\mathcal{Q}} \quad \text{KL}\left(\hat{\mathcal{P}}_j || \mathcal{Q}\right)\,, \tag{6.91}$$

where

$$\hat{\mathcal{P}}_j(\mathbf{w}, \epsilon) = \frac{1}{Z_j} t_j(\mathbf{w}, \epsilon) \mathcal{Q}^{\setminus i}(\mathbf{w}, \epsilon) \tag{6.92}$$

and

$$Z_j = \int t_j(\mathbf{w}, \epsilon) \mathcal{Q}^{\setminus i}(\mathbf{w}, \epsilon) d\mathbf{w} d\epsilon \,. \tag{6.93}$$

The score Δ_j associated to each candidate instance $j \in \mathcal{J}$ is

$$\Delta_j = \mathrm{KL}\left(\mathcal{Q}_j^{\text{new}} || \mathcal{Q}^{\setminus i}\right) . \tag{6.94}$$

Expression (6.94) measures the change in the posterior approximation when $\mathcal{Q}^{\setminus i}$ is updated to $\mathcal{Q}_j^{\text{new}}$. The instance that maximizes (6.94) is the one that has the largest impact in the update of the approximation to the posterior distribution. In consequence, such an instance brings the approximation closer to the actual posterior distribution. (vi) Once a candidate instance j has been selected from $\mathcal{J}$, we update the active set and the posterior approximation by setting $\mathcal{I} = \mathcal{I} \cup \{j\}$ and $\mathcal{Q} = \mathcal{Q}_j^{\text{new}}$. Then, we set the approximate term $\tilde{t}_j$ to be $Z_j \mathcal{Q}_j^{\text{new}} / \mathcal{Q}^{\setminus i}$, as in the standard EP algorithm. (vii) After convergence, the model evidence is computed using only those instances included in $\mathcal{I}$

$$\mathcal{P}(\mathbf{y}) \approx \int \tilde{t}_{n+1}(\mathbf{w}, \epsilon) \prod_{i \in \mathcal{I}} \tilde{t}_i(\mathbf{w}, \epsilon) d\mathbf{w} d\epsilon \,. \tag{6.95}$$

Figure 6.3 displays the pseudo-code of the proposed algorithm for training SBMs.

Input: instances $(\mathbf{x}_1, y_1), \ldots (\mathbf{x}_n, y_n)$ and d.

Output: posterior approximation $\mathcal{Q}$ and active set $\mathcal{I}$.

1. Initialize $\mathcal{Q}$ to the prior distribution and the terms $\tilde{t}_i$ to be uniform.
2. Initialize the active set $\mathcal{I}$ to d randomly chosen instances.
3. Repeat until $\mathcal{I}$ and all the terms $\tilde{t}_i$ converge:
   (a) For each $i \in \mathcal{I}$:
      i. Update $\mathcal{I}$ to $\mathcal{I} \setminus \{i\}$.
      ii. Set $\tilde{t}_i$ to be uniform and compute $\mathcal{Q}^{\setminus i}$ as in EP.
      iii. For each $j \in \mathcal{J} = \{1, \ldots, n\} \setminus \mathcal{I}$:
         A. Compute an updated posterior distribution $\mathcal{Q}_j^{\text{new}}$.
         B. Compute the KL-divergence between $\mathcal{Q}_j^{\text{new}}$ and $\mathcal{Q}^{\setminus i}$.
      iv. Select the j with the largest score.
      v. Using EP update $\tilde{t}_j$ and set $\mathcal{Q}$ to $\mathcal{Q}_j^{\text{new}}$.
      vi. Update $\mathcal{I}$ to $\mathcal{I} \cup \{j\}$.
4. Return $\mathcal{Q}$, $\mathcal{I}$ and the approximation to $\mathcal{P}(\mathbf{y})$.

FIGURE 6.3: Pseudo-code of the training algorithm for the SBM. $\mathcal{Q}^{\setminus i}$ is the posterior approximation after the term $\tilde{t}_i$ is removed and $\mathcal{Q}_j^{\text{new}}$ is the same approximation after the exact term t_j is processed by the EP algorithm.

Performing a single iteration of this algorithm over the active set $\mathcal{I}$ is equivalent to the approach followed by the IVM. However, we expect that additional refining iterations will significantly improve the posterior approximation and will correct some of the mistakes

that result from this greedy approach. Even though the convergence of the proposed algorithm is not guaranteed, we expect it to work in a similar way as the EP algorithm. In fact, experimental results seem to confirm this expectation. Note that the mechanism for introducing and eliminating instances from the active set is similar to the exchange movements proposed in (Seeger, 2003). However, these had not been implemented in practice as they actually increase the computational cost of the IVM. The algorithm described is also different from computing the active set $\mathcal{I}$ at the first iteration of the algorithm and then running EP to refine the approximate terms $\tilde{t}_i$ corresponding to the instances included in $\mathcal{I}$. The difference lies in the fact that the proposed algorithm allows $\mathcal{I}$ to change during the subsequent refining iterations.

We now describe in terms of inner products the main steps of the algorithm presented in Figure 6.3. In general, the steps of this algorithm are very similar to the steps of the standard EP algorithm written in terms of inner products and hence, they can be implemented using the matrix $\mathbf{A}$ and the vector $\mathbf{h}$ defined in (6.57) and (6.58), respectively. To describe these steps, let

$$\lambda_j^{\setminus i} = \mathbf{x}_j^T \mathbf{V}_{\mathbf{w}}^{\setminus i} \mathbf{x}_j \,, \tag{6.96}$$

$$h_j^{\setminus i} = \mathbf{x}_j^T \mathbf{m}_{\mathbf{w}}^{\setminus i} \,, \tag{6.97}$$

where $\mathbf{m}_{\mathbf{w}}^{\setminus i}$ and $\mathbf{V}_{\mathbf{w}}^{\setminus i}$ are the mean and covariance parameters of the Gaussian part of the posterior approximation $\mathcal{Q}^{\setminus i}$, respectively. The first step of the algorithm removes $\tilde{t}_i$ from $\mathcal{Q}$ to get $\mathcal{Q}^{\setminus i}$ and sets $\tilde{t}_i$ to be uniform. This is performed as in the standard EP algorithm by computing

$$\lambda_j^{\setminus i} = A_{jj} + A_{ji}^2/(v_i - A_{ii}) \,, \quad \forall j \in \mathcal{J} \,, \tag{6.98}$$

$$h_j^{\setminus i} = h_j + \frac{A_{ji}}{v_i - A_{ii}}(h_i - m_i) \,, \quad \forall j \in \mathcal{J} \,, \tag{6.99}$$

$$a_\epsilon^{\setminus i} = a_\epsilon - a_i + 1 \,, \tag{6.100}$$

$$b_\epsilon^{\setminus i} = b_\epsilon - b_i + 1 \,, \tag{6.101}$$

and by setting $a_i = b_i = 1$, $m_i = 0$ and $v_i = +\infty$. A detailed derivation of (6.98) and (6.99) is given in Appendix B.4. Next, we compute an updated posterior distribution $\mathcal{Q}_j^{\text{new}}$ for each candidate instance $j \in \mathcal{J}$. Because of the closure property of the exponential family, the form of $\mathcal{Q}_j^{\text{new}}$ is similar to the form of $\mathcal{Q}$ in the standard EP algorithm

$$\mathcal{Q}_j^{\text{new}}(\mathbf{w}, \epsilon) = \mathcal{N}(\mathbf{w}|\mathbf{m}_{\mathbf{w}}^{\text{new}}(j), \mathbf{V}_{\mathbf{w}}^{\text{new}}(j))\text{Beta}(\epsilon|a_\epsilon^{\text{new}}(j), b_\epsilon^{\text{new}}(j)) \,, \tag{6.102}$$

where $\mathbf{m}_{\mathbf{w}}^{\text{new}}(j)$, $\mathbf{V}_{\mathbf{w}}^{\text{new}}(j)$, $a_\epsilon^{\text{new}}(j)$ and $b_\epsilon^{\text{new}}(j)$ are the parameters of the approximation. Computing each $\mathcal{Q}_j^{\text{new}}$ is almost the same as computing $\mathcal{Q}^{\text{new}}$ in the standard Bayes machine, when EP is written in terms of inner products. However, instead of computing a single updated posterior distribution $\mathcal{Q}^{\text{new}}$, we compute one for each candidate instance $j \in \mathcal{J}$ by setting

$$\overline{\epsilon}^{\setminus i} = \frac{a_\epsilon^{\setminus i}}{a_\epsilon^{\setminus i} + b_\epsilon^{\setminus i}} \,, \tag{6.103}$$

$$z_j = \frac{h_j^{\setminus i}}{\sqrt{\lambda_j^{\setminus i}}}, \quad \forall j \in \mathcal{J}, \tag{6.104}$$

$$Z_j = \overline{\epsilon}^{\setminus i} + (1 - 2\overline{\epsilon}^{\setminus i})\Phi(z_j), \quad \forall j \in \mathcal{J}, \tag{6.105}$$

$$\alpha_j = \frac{1}{\sqrt{\lambda_j^{\setminus i}}} \frac{(1 - 2\overline{\epsilon}^{\setminus i})\mathcal{N}(z_j|0,1)}{\overline{\epsilon}^{\setminus i} + (1 - 2\overline{\epsilon}^{\setminus i})\Phi(z_j)}, \quad \forall j \in \mathcal{J}, \tag{6.106}$$

$$h_j^{\text{new}} = \mathbf{x}_j^T \mathbf{m}_{\mathbf{w}}^{\text{new}}(j) = h_j^{\setminus i} + \lambda_j^{\setminus i}\alpha_j, \quad \forall j \in \mathcal{J}. \tag{6.107}$$

The parameters $a_\epsilon^{\text{new}}(j)$ and $b_\epsilon^{\text{new}}(j)$ of $\mathcal{Q}_j^{\text{new}}$ are also computed as in the standard EP algorithm using (6.31) and (6.32), where

$$\mathbb{E}_{\hat{\mathcal{P}}_j}[\epsilon] = \frac{1}{Z_j}\left[\Phi(z_j)(1 - \overline{\epsilon}^{\setminus i})\frac{a_\epsilon^{\setminus i}}{a_\epsilon^{\setminus i} + b_\epsilon^{\setminus i} + 1} + (1 - \Phi(z_j))\overline{\epsilon}^{\setminus i}\frac{a_\epsilon^{\setminus i} + 1}{a_\epsilon^{\setminus i} + b_\epsilon^{\setminus i} + 1}\right], \tag{6.108}$$

$$\begin{aligned}\mathbb{E}_{\hat{\mathcal{P}}_j}[\epsilon^2] = \frac{1}{Z_j}\Bigg[&\Phi(z_j)(1 - \overline{\epsilon}^{\setminus i})\frac{a_\epsilon^{\setminus i}(a_\epsilon^{\setminus i} + 1)}{(a_\epsilon^{\setminus i} + b_\epsilon^{\setminus i} + 1)(a_\epsilon^{\setminus i} + b_\epsilon^{\setminus i} + 2)} \\ &+ (1 - \Phi(z_j))\overline{\epsilon}^{\setminus i}\frac{(a_\epsilon^{\setminus i} + 1)(a_\epsilon^{\setminus i} + 2)}{(a_\epsilon^{\setminus i} + b_\epsilon^{\setminus i} + 1)(a_\epsilon^{\setminus i} + b_\epsilon^{\setminus i} + 2)}\Bigg].\end{aligned} \tag{6.109}$$

Candidate instances are then ranked according to Δ_j, which is defined as the KL-divergence between $\mathcal{Q}_j^{\text{new}}$ and $\mathcal{Q}^{\setminus i}$

$$\begin{aligned}\Delta_j &= \text{KL}\left(\mathcal{Q}_j^{\text{new}} | \mathcal{Q}^{\setminus i}\right) \\ &= -\frac{1}{2}\log(1 - \alpha_j h_j^{\text{new}}) - \frac{1}{2}\alpha_j h_j^{\text{new}} + \frac{1}{2}\alpha_j^2 \lambda_j^{\setminus i} \\ &\quad + \log\left(\frac{\beta(a_\epsilon^{\setminus i}, b_\epsilon^{\setminus i})}{\beta(a_\epsilon^{\text{new}}(j), b_\epsilon^{\text{new}}(j))}\right) - \left(a_\epsilon^{\setminus i} - a_\epsilon^{\text{new}}(j)\right)\Psi(a_\epsilon^{\text{new}}(j)) \\ &\quad - \left(b_\epsilon^{\setminus i} - b_\epsilon^{\text{new}}(j)\right)\Psi(b_\epsilon^{\text{new}}(j)) \\ &\quad + \left(a_\epsilon^{\setminus i} - a_\epsilon^{\text{new}}(j) + b_\epsilon^{\setminus i} - b_\epsilon^{\text{new}}(j)\right)\Psi(a_\epsilon^{\text{new}}(j) + b_\epsilon^{\text{new}}(j)),\end{aligned} \tag{6.110}$$

where Ψ is the digamma function and β is the beta function (Abramowitz and Stegun, 1964). The derivation of (6.110) is given in Appendix B.8. After selecting the $j \in \mathcal{J}$ with the largest impact in the approximation, as measured by Δ_j, we update the approximate term $\tilde{t}_j$ as in the standard EP algorithm

$$a_j = a_\epsilon^{\text{new}}(j) - a_\epsilon^{\setminus i} + 1, \tag{6.111}$$

$$b_j = b_\epsilon^{\text{new}}(j) - b_\epsilon^{\setminus i} + 1, \tag{6.112}$$

$$v_j = \lambda_j^{\setminus i}\left(\frac{1}{\alpha_j h_j^{\text{new}}} - 1\right), \tag{6.113}$$

$$m_j = h_j^{\setminus i} + (v_j + \lambda_j^{\setminus i})\alpha_j = h_j^{\text{new}} + v_j\alpha_j, \tag{6.114}$$

$$s_j = Z_j \frac{\beta(a_\epsilon^{\setminus i}, b_\epsilon^{\setminus i})}{\beta(a_\epsilon^{\text{new}}(j), b_\epsilon^{\text{new}}(j))}\sqrt{1 + v_j^{-1}\lambda_j^{\setminus i}}\exp\left(\frac{\lambda_j^{\setminus i}\alpha_j}{2h_j^{\text{new}}}\right). \tag{6.115}$$

Once $\tilde{t}_j$ has been updated, the posterior approximation $\mathcal{Q}$ is set to $\mathcal{Q}_j^{\text{new}}$. In particular, we update $a_\epsilon = a_\epsilon^{\text{new}}(j)$, $b_\epsilon = b_\epsilon^{\text{new}}(j)$ and the matrix $\mathbf{A}$ and the vector $\mathbf{h}$ using (6.79) and (6.80), as in the standard EP algorithm. In principle, updating $\mathbf{A}$ and $\mathbf{h}$ requires $\mathcal{O}(n^2)$ operations. To speed up this process, we consider the decomposition of the matrix $\mathbf{A}$ proposed by Lawrence et al. (2003). This decomposition can be obtained using the Woodbury formula (B.1) and (B.12)

$$\mathbf{A} = \mathbf{C} - \mathbf{M}^T\mathbf{M}\,, \tag{6.116}$$

$$\mathbf{M} = \mathbf{L}^{-1}\boldsymbol{\Lambda}_{\mathcal{I}}^{-1/2}\mathbf{C}_{\mathcal{I},\cdot} \in \mathbb{R}^{d,n}\,, \tag{6.117}$$

where $\mathbf{L}$ is the lower-triangular Cholesky factor of

$$\mathbf{B} = \mathbf{I} + \boldsymbol{\Lambda}_{\mathcal{I}}^{-1/2}\mathbf{C}_{\mathcal{I}}\boldsymbol{\Lambda}_{\mathcal{I}}^{-1/2} \in \mathbb{R}^{d,d} \tag{6.118}$$

and $\mathbf{I}$ is the identity matrix. In (6.117) $\mathbf{C}_{\mathcal{I},\cdot}$ denotes the sub-matrix of $\mathbf{C}$ given by considering only the rows whose indices are included in $\mathcal{I}$ and all the columns. Similarly, in (6.118) $\mathbf{C}_{\mathcal{I}}$ denotes the sub-matrix of $\mathbf{C}$ given by considering only the rows and columns whose indices are included in $\mathcal{I}$. The same notation is used for the other matrix $\boldsymbol{\Lambda}_{\mathcal{I}}^{-1/2}$. Note that in (6.116) we have used the fact that the approximate terms corresponding to the instances that are not included in $\mathcal{I}$ are set to be uniform, i.e. $v_j = +\infty\ \forall j \in \mathcal{J}$. Furthermore, it is possible that the matrix $\boldsymbol{\Lambda}_{\mathcal{I}}^{-1/2}$ in (6.118) is undefined because some of the parameters v_i of the approximate terms $\tilde{t}_i$ can be negative in the EP algorithm (Minka, 2001b). This problem can be overcome by using complex numbers in the implementation of the training algorithm of the SBM. The advantage of using (6.116) to represent the matrix $\mathbf{A}$ is that the update of the matrix $\mathbf{M}$ requires only $\mathcal{O}(nd)$ steps. From this matrix any column of $\mathbf{A}$ can be retrieved in $\mathcal{O}(nd)$ steps. These columns are required, for example, to compute (6.98) and (6.99). Finally, the vector $\mathbf{h}$ can also be updated in $\mathcal{O}(nd)$ steps when (6.116) is used.

We now describe how to update the matrix $\mathbf{A}$ using the representation provided in (6.116). Specifically, when an instance i from the active set is replaced by a new candidate instance $j \in \mathcal{J}$, the matrix $\mathbf{A}$ is updated in terms of the matrix $\mathbf{M}$. However, the update of $\mathbf{M}$ requires first the update of the Cholesky factor $\mathbf{L}$, which depends on the changes in the matrix $\mathbf{B}$. Assume that l is the position of the element i in $\mathcal{I}$. As described in (Seeger, 2003), the new matrix $\mathbf{B}$ is

$$\mathbf{B}_{\text{new}} = \mathbf{B} + \left(\tilde{\boldsymbol{\delta}}_l + \tilde{\boldsymbol{\upsilon}}\right)\left(\tilde{\boldsymbol{\delta}}_l + \tilde{\boldsymbol{\upsilon}}\right)^T - \tilde{\boldsymbol{\upsilon}}\tilde{\boldsymbol{\upsilon}}^T\,, \tag{6.119}$$

where $\tilde{\boldsymbol{\delta}}_l = \eta^{1/2}\boldsymbol{\delta}_l$, $\tilde{\boldsymbol{\upsilon}} = \eta^{-1/2}\boldsymbol{\upsilon}$, $\eta = v_j^{-1}C_{jj} + v_i^{-1}C_{ii} - 2v_j^{-1/2}v_i^{-1/2}C_{ij} > 0$, $\boldsymbol{\upsilon} = \boldsymbol{\Lambda}_{\mathcal{I}}^{-1/2}(v_j^{-1/2}\mathbf{C}_{\mathcal{I},j} - v_i^{-1/2}\mathbf{C}_{\mathcal{I},i})$ and $\boldsymbol{\delta}_l$ is a vector whose components are all zeros except the l-th component, which takes value one. The derivation of (6.119) is given in Appendix B.9. This update rule is in fact two rank-one updates. In consequence, the Cholesky factor $\mathbf{L}$ can be updated in $\mathcal{O}(d^2)$ steps because for each rank-one update, $\mathbf{L}_{\text{new}} = \mathbf{L}\tilde{\mathbf{L}}$, where $\tilde{\mathbf{L}}$ has a special form that allows for the use of a fast multiplication algorithm (Gill et al., 1974). The structure of the Cholesky factor $\tilde{\mathbf{L}}$ is discussed in Appendix B.11.

Once the Cholesky factor $\mathbf{L}$ has been updated, we can update the matrix $\mathbf{M}$. The update rule of the matrix $\mathbf{M}$ that appears in the Appendix of (Seeger, 2003) is not complete because it does not take into account the changes in $\mathbf{C}_{\mathcal{I}}$ after the update of $\mathcal{I}$. To derive a more accurate update equation for $\mathbf{M}$ consider the updated Cholesky factor

$\mathbf{L}_{\text{new}} = \mathbf{L}\tilde{\mathbf{L}}_1\tilde{\mathbf{L}}_2$ that results from the two rank-one updates of the matrix $\mathbf{B}$. The update rule for the matrix $\mathbf{M}$ is

$$\mathbf{M}_{\text{new}} = \tilde{\mathbf{L}}_2^{-1}\tilde{\mathbf{L}}_1^{-1}\mathbf{M}' \,, \tag{6.120}$$

where

$$\mathbf{M}' = \mathbf{M} + \left(\mathbf{L}^{-1}\boldsymbol{\delta}_l\right)\left(v_j^{-1/2}\mathbf{C}_{j,\cdot} - v_i^{-1/2}\mathbf{C}_{i,\cdot}\right) \,. \tag{6.121}$$

The derivation of (6.120) is given in Appendix B.10. The evaluation of (6.120) can be made in $\mathcal{O}(nd)$ steps because a multiplication by the factor $\tilde{\mathbf{L}}^{-1}$ can be easily computed in terms of the standard factor $\tilde{\mathbf{L}}$. Appendix B.12 presents the details of this operation. However, to compute (6.120) we need $\mathbf{L}^{-1}$. Thus, we also have to store this matrix in memory and update it as follows $\mathbf{L}_{\text{new}}^{-1} = \tilde{\mathbf{L}}^{-1}\mathbf{L}^{-1}$ after each rank-one update of $\mathbf{B}$.

The vector $\mathbf{h}$ can similarly be updated in $\mathcal{O}(nd)$ steps using the representation of $\mathbf{A}$ given in (6.116) (Seeger, 2003)

$$\mathbf{h}_{\text{new}} = \mathbf{h} + \mathbf{A}_{\cdot,\{ji\}}\boldsymbol{\Delta}^{1/2}\mathbf{P}^{-1}\left(\boldsymbol{\Delta}^{-1/2}\boldsymbol{\gamma} - \boldsymbol{\Delta}^{1/2}\mathbf{h}_{\{ji\}}\right) \,, \tag{6.122}$$

where

$$\mathbf{P} = \mathbf{I} + \boldsymbol{\Delta}^{1/2}\mathbf{A}_{\{ji\}}\boldsymbol{\Delta}^{1/2} \,, \quad \boldsymbol{\Delta} = \text{diag}(v_j^{-1}, v_i^{-1}) \,, \quad \boldsymbol{\gamma} = (v_j^{-1}m_j, v_i^{-1}m_i)^T \,. \tag{6.123}$$

In (6.122) $\boldsymbol{\Delta}$ is a 2×2 diagonal matrix that denotes the changes in $\boldsymbol{\Lambda}^{-1}$ and $\boldsymbol{\gamma}$ is a vector denoting the changes in $\boldsymbol{\Lambda}^{-1}\mathbf{m}$. The two rows of the matrix $\mathbf{A}$ that are required for evaluating (6.122) can be computed in $\mathcal{O}(nd)$ steps using (6.116) and the matrix $\mathbf{M}$. Furthermore, computing the inverse of $\mathbf{P}$ is inexpensive because it is a 2×2 matrix. The derivation of (6.122) is given in Appendix B.13.

In summary, when (6.116) is used to represent the matrix $\mathbf{A}$ the preprocessing of an instance in the active set $\mathcal{I}$ in the training algorithm of the SBM has a total cost of $\mathcal{O}(nd)$ steps. Under the assumption that the elements of the kernel matrix $\mathbf{C}$ can be computed when required, an iteration of the whole algorithm takes $\mathcal{O}(nd^2)$ steps because $|\mathcal{I}| = d$ (otherwise we have to compute first the matrix $\mathbf{C}$, an operation that has a cost $\mathcal{O}(n^2)$). In practice, this algorithm converges after only a few iterations, like the standard EP algorithm. In consequence, the total cost of the algorithm is $\mathcal{O}(nd^2)$. The memory requirements are dominated by the matrix $\mathbf{M}$ which demands $\mathcal{O}(nd)$ storage space.

Once training has been completed, the marginal likelihood of the training data (6.95) can be used for kernel selection. The evaluation of (6.95) can be done as in the standard BM (see (6.82))

$$\mathcal{P}(\mathbf{y}) \approx \sqrt{\frac{|\boldsymbol{\Lambda}_{\mathcal{I}}|}{|\mathbf{C}_{\mathcal{I}} + \boldsymbol{\Lambda}_{\mathcal{I}}|}} \frac{\beta(a_\epsilon, b_\epsilon)}{\beta(a_0, b_0)} \exp(\frac{B}{2}) \prod_{i \in \mathcal{I}} s_i \,, \tag{6.124}$$

where

$$B = \sum_{i,j \in \mathcal{I}} A_{ij} \frac{m_i m_j}{v_i v_j} - \sum_{i \in \mathcal{I}} \frac{m_i^2}{v_i} . \tag{6.125}$$

The predictive distribution for new instances is obtained in a similar way

$$\begin{aligned} \mathcal{P}(y_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{y}) &\approx \int \mathcal{P}(y_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{w}, \epsilon)\mathcal{N}(\mathbf{w}|\mathbf{m}_{\mathbf{w}}, \mathbf{V}_{\mathbf{w}})\text{Beta}(\epsilon|a_\epsilon, b_\epsilon)d\mathbf{w}d\epsilon \\ &= \overline{\epsilon} + (1 - 2\overline{\epsilon})\Phi(z) \,, \end{aligned} \tag{6.126}$$

where $\overline{\epsilon} = a_\epsilon/(a_\epsilon + b_\epsilon)$ and

$$z = \frac{\mathbf{m}_{\mathbf{w}}^T \mathbf{x}_{\text{new}}}{\sqrt{(\mathbf{x}_{\text{new}})^T \mathbf{V}_{\mathbf{w}} \mathbf{x}_{\text{new}}}} = \frac{\sum_{i \in \mathcal{I}} \alpha_i \mathbf{x}_{\text{new}} \mathbf{x}_i^T}{\sqrt{(\mathbf{x}_{\text{new}})^T \mathbf{x}_{\text{new}} - (\mathbf{x}_{\text{new}})^T \mathbf{X}_{\cdot,\mathcal{I}} \mathbf{K} \mathbf{X}_{\cdot,\mathcal{I}}^T \mathbf{x}_{\text{new}}}} , \tag{6.127}$$

with $\mathbf{K} = \mathbf{\Lambda}_{\mathcal{I}}^{-1}(\mathbf{\Lambda}_{\mathcal{I}} - \mathbf{A}_{\mathcal{I}})\mathbf{\Lambda}_{\mathcal{I}}^{-1}$.

6.3.2 Experiments

In this section the performance of the SBM is assessed in experiments on several classification problems. First, a simple problem is analyzed to visualize the selection of instances in the SBM. We generate 100 instances from a simple 2-class 2-dimensional dataset whose input features x_1 and x_2 are independently obtained from two Gaussian distributions with unit covariance matrix and mean vectors $\boldsymbol{\mu}_1 = (1, 1)^T$ and $\boldsymbol{\mu}_2 = (-1, -1)^T$, respectively. The class label of each instance, either 1 or -1, is assigned according to the Gaussian distribution from which the instance has been generated. This means that the optimal decision boundary of the problem is the hyper-plane defined by the vector $\mathbf{w} = (1, 1, 0)^T$. Two SBMs are built using these training data. The first machine uses a linear kernel function, and the second one the Gaussian kernel given by

$$C(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma |\mathbf{x}_i - \mathbf{x}_j|^2\right) , \tag{6.128}$$

with $\gamma = 0.1$. This kernel allows for non-linear decision boundaries. The sparsity parameter d of the SBM is fixed to 20 in both cases. Thus, we retain 20% of the initial training instances. In the case of the linear kernel, after sixteen iterations the algorithm converges to a stable configuration. However, the active set $\mathcal{I}$ converges after the first three iterations. In the case of the Gaussian kernel, the algorithm requires nine iterations to converge. The active set converges again after the first three iterations of the algorithm. Figure 6.4 displays the decision boundary of the resulting SBMs. The instances that are selected after the training algorithm ends are highlighted with green circumferences. As expected, when the linear kernel is used only the instances located near the decision boundary are included in the model. By contrast, when the Gaussian kernel is used, the instances that surround each cluster of points are included in the model. These instances are representative of the limits of each cluster. They carry useful information about the classification task when a non-linear decision boundary is employed. The instances located inside the clusters do not carry this information and are therefore discarded.

The performance of SBMs is assessed by additional experiments in the MNIST handwritten digits dataset (LeCun et al., 1998). In particular, we compare the prediction performance of SBMs with standard BMs, SBMs that follow the approach of the IVM (Lawrence et al., 2003; Seeger, 2003) and SVMs (Vapnik, 1995). Recall that the training algorithm of the IVM is a special case of the training algorithm of the SBM where the algorithm displayed in Figure 6.3 stops after the first iteration. The purpose of comparing with this last model is to determine whether the additional refining iterations proposed in this chapter are beneficial or not.

The experimental procedure is as follows: For each digit i different from 9, we consider binary tasks that consist in discriminating instances of the digit i from instances of the digit 9. The bitmaps of size 28×28 pixels are down-sampled to size 14×14 pixels so that only 196 attributes are considered. For each of these classification problems,

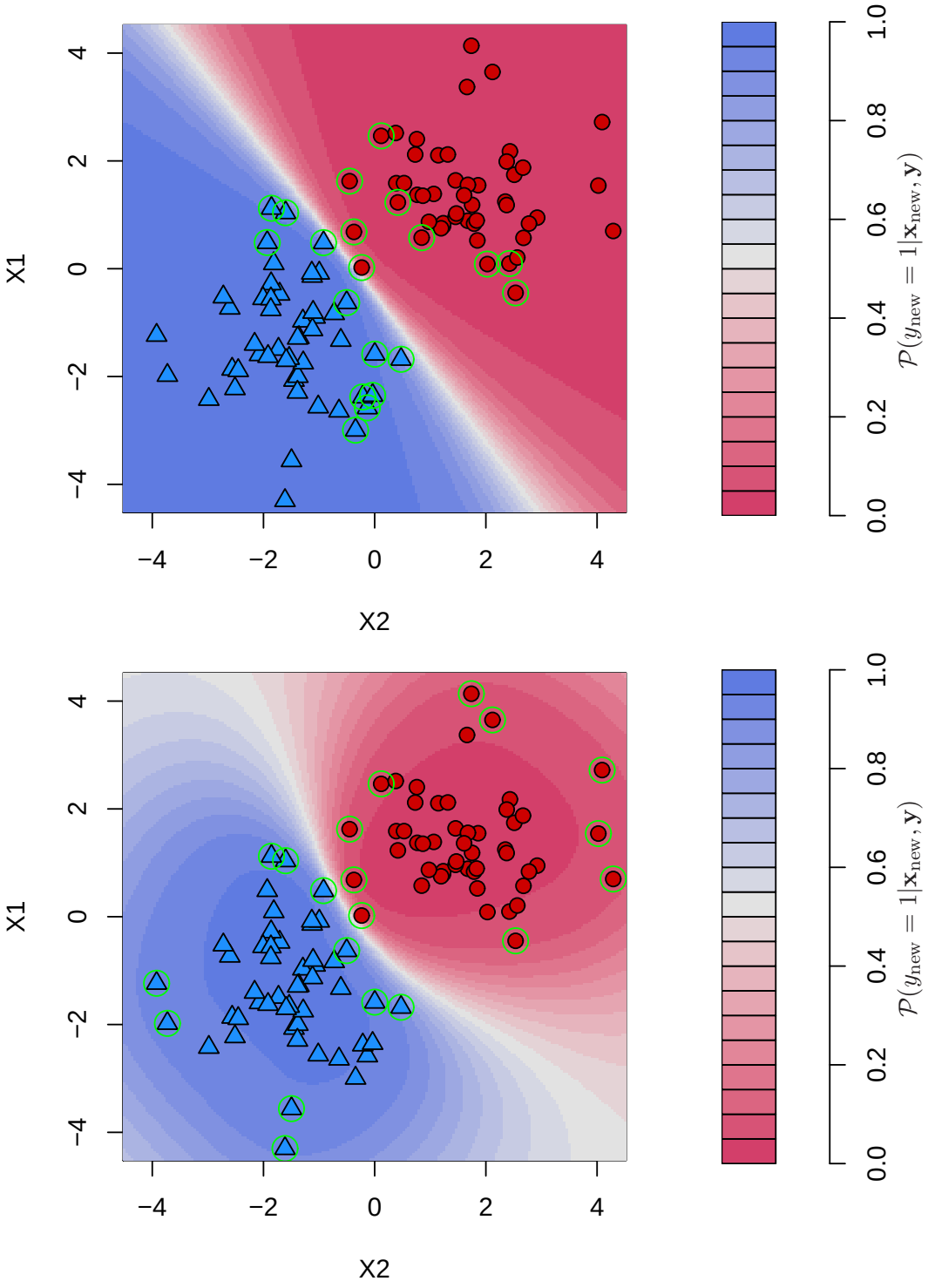

FIGURE 6.4: Training set and decision boundary of a SBM trained with a linear kernel (top) and with a Gaussian kernel (bottom) on the simple problem. Contour curves describe the predictive distribution of each model. Patterns included in the models are those that are highlighted with green circumferences. The parameter d is set to 20% of the number of available instances.

we randomly extract from the MNIST training set of 60,000 instances 500 instances associated to the digit i, and 500 instances associated to the digit 9. Then, each one of these datasets of 1,000 instances is randomly split 50 times into a training set of 100 instances and a test set of 900 instances. The reason for using small training sets is that we also compare results with standard BMs, whose training time scales like $\mathcal{O}(n^3)$, where n is in the number of training instances. The data attributes are then normalized to have zero mean and unit standard deviation on the training set. The different classifiers are built using the training set and then evaluated on the test set. Because the SBM can not determine the value of the sparsity parameter d automatically, for each classification task we initialized this parameter with the average number of support vectors identified by the SVM. In all methods the same kernel is used; namely, the Gaussian kernel described in (6.128). The γ parameter of this kernel is estimated in all methods by a grid search in which the natural logarithm of the values of the knots in the grid are ten equally spaced points in the interval $[-10, -4]$. In the case of the SVM, a 10-fold cross-validation procedure on the training set is used to estimate which parameter value performs best. In the other methods, the marginal likelihood of the training data is used for this purpose. The cost parameter C of the SVM is determined in a nested grid search that uses 10-fold cross validation to discriminate among different values of this parameter. In this case, the natural logarithm of the values of the knots in the grid are ten equally spaced points in the interval $[-3, 3]$. Finally, we note that the prediction rule of the BM and the SBM requires convergence of the training algorithm (see Appendix B.7 for further details). Thus, the prediction rule of the SBM without the additional refining iterations is computed differently. This prediction rule is described in Appendix B.7.

TABLE 6.4: Error values of each method and average number of support vectors.

Problem	**BM**	**SBM**	**SBM$_{\text{IVM}}$**	**SVM**	**# SV**
0 vs 9	**1.5 ±0.6**	1.5±0.6	1.7±0.7	3.2±1.1	54.2±16.4
1 vs 9	**1.9 ±0.9**	2.0±1.0	2.5±1.3	3.5±1.3	56.1±17.9
2 vs 9	2.5±0.6	**2.5 ±0.6**	2.5±0.6	4.5±1.6	59.3±16.2
3 vs 9	**3.5 ±1.0**	3.5±1.0	4.0±1.2	6.1±1.6	54.4±15.8
4 vs 9	8.3±1.6	**8.2 ±1.6**	9.0±1.6	11.6±2.1	61.9±13.9
5 vs 9	4.1±0.9	**4.0 ±0.9**	4.5±1.1	6.8±2.1	59.2±12.8
6 vs 9	**1.3 ±0.8**	1.3±0.8	1.8±0.9	4.2±2.0	72.0±17.3
7 vs 9	9.1±1.5	**9.1 ±1.5**	9.5±1.5	10.6±2.0	56.1±13.1
8 vs 9	**4.1 ±0.8**	4.2±0.9	4.4±0.8	6.4±2.0	59.7±13.2

Table 6.4 displays the average classification error of each method and the average number of support vectors identified by the SVM in the problems investigated. The lowest error value for each problem is highlighted in bold face. These results show that the BM obtains the best performance in five of the nine classification tasks investigated. On the other hand, the SBM obtains the best performance in the remaining four tasks. The error rates of the SBM without the additional refining iterations (fourth column of the table) are larger than the ones of the BM and the complete SBM. These results confirm that the additional refining iterations proposed have a beneficial effect in the quality of the approximation. Finally, the SVM has the highest error rates. The average number of support vectors computed by this method (last column of the table) varies between $\approx 54\%$ and $\approx 72\%$ of the number of initial instances in the training set.

The testing framework described in the previous section is also used here to compare the performance of the classifiers investigated. A Friedman average rank test rejects the hypothesis that all methods have an equivalent performance in this set of problems (a p-value below 5% is obtained). We then apply a Nemenyi test to determine wherther the differences in average rank among the classifiers investigated are significant. A significance level of 10% is employed for this purpose. The differences in performance between the SBM and the SBM without the additional refining iterations are not significant when the level is set to 5%. Figure 6.5 displays the result of this test, as suggested by Demšar (2006). When the difference between the average ranks of two classifiers is not significant they are linked with a solid black segment. The critical difference (CD) between average ranks is displayed at the top. The figure shows that there is statistical evidence supporting that both the BM and the SBM perform better than the SVM and the SBM without the additional refining iterations over the set of classification problems investigated. The differences between the performance of the BM and the SBM, and between the performance of the SVM and the SBM without the additional refining iterations, are not statistically significant.

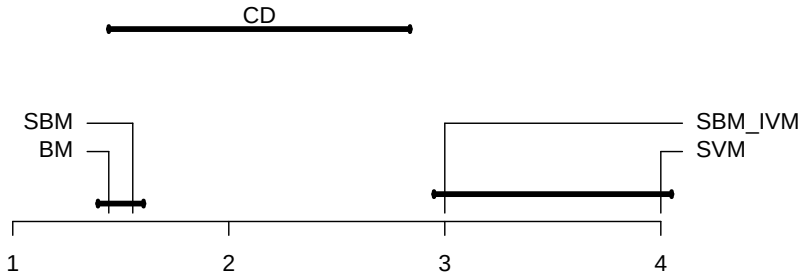

FIGURE 6.5: Graphical result of a Nemenyi test comparing the average rank of each method. The critical difference (CD) between average ranks is displayed at the top.

6.4 Conclusions

In this chapter we have proposed a Bayesian probabilistic model for binary classification that is an extension of the Bayes point machine (BPM) (Herbrich et al., 2001). This extension is performed in two ways. First, we take into account the whole *version space*, not only its center of mass. Second, the model considers the possibility of mislabeled instances in the training set. We call this model the Bayes machine (BM) and follow a Bayesian approach to compute a posterior distribution for all the parameters in the model, including a parameter ϵ that quantifies the noise in the labels of the training data. Exact inference in the BM is not tractable and hence, approximation techniques have to be used. In this work we use the approximate inference algorithm expectation propagation (EP) to train the BM (Minka, 2001b). Unlike Gaussian process classifiers (GPCs) (Barber and Williams, 1997; Gibbs and MacKay, 2000; Kuss and Rasmussen, 2005; Opper and Winther, 2000b) or support vector machines (SVMs) (Vapnik, 1995), the BM learns the noise parameter directly from the traning data without any additional cost. This is the most compelling feature of the proposed model. Additionally, the BM induces an approximate predictive distribution that takes into account the uncertainty in the model parameters, unlike the SVM or the BPM, which yield a single point estimate.

The effectiveness of the proposed model for learning the intrinsic level of noise in the class labels of the training data is illustrated in a simple classification problem. These

experiments show that for increasing values of the noise level injected in the training data the average value of the error rate estimated in the posterior distribution of the BM also increases. The performance of the BM is investigated in fifteen synthetic and real-world classification problems. In these experiments, the prediction accuracy of the BM is compared with the SVM (Vapnik, 1995) and the GPC based on the EM-EP algorithm of Kim and Ghahramani (2006). The BM performs better than these classifiers in the set of problems considered. These improvements in performance are statistically significant according to the Nemenyi test.

The cost of training a BM for binary classification is $\mathcal{O}(n^3)$, where n is the number of training instances. This is the same computational cost of training the GPC using EP (Kim and Ghahramani, 2006; Minka, 2001b; Opper and Winther, 2000b). If other techniques are used to perform approximate inference in GPCs (Gibbs and MacKay, 2000; Kuss and Rasmussen, 2005), the cost only differs in a proportionality constant. In fact, all GPCs require a matrix inversion that demands $\mathcal{O}(n^3)$ steps. On the other hand, the SVM scales better with the number available training instances as its training cost is at worst $\mathcal{O}(n^2)$ when using a SMO-type algorithm. This means that the BM can only be applied to binary classification problems with few training instances.

To reduce the cost of training the BM without significantly deteriorating its performance, we have proposed a *sparse* representation for the BM called the sparse Bayes machine (SBM). The approach followed is based on the IVM (Lawrence et al., 2003; Seeger, 2003). However, we have introduced some modifications that provide a better approximation to the posterior distribution. Specifically, we have included additional refining iterations to the training algorithm of the IVM. Two types of improvements are obtained from the additional iterations. First, the active set of selected instances $\mathcal{I}$ can be updated during these iterations. These updates correct some of the errors made in the greedy selection of active instances. Second, the extra processing of these instances improves the posterior approximation. These additional iterations can be thought as a backfitting algorithm that tries to fix some of the errors that result from the greedy selection of instances. As a matter of fact, they actually work in a similar way as the EP algorithm, which improves the approximation obtained by ADF (Minka, 2001b). The main advantage of using the proposed sparse representation is that the training cost of the BM is reduced to $\mathcal{O}(d^2 n)$, where n is the number of training instances available and d is some sparsity parameter that has to be fixed in some way. If d is much smaller than n this is an important improvement over the training cost of the BM, which is $\mathcal{O}(n^3)$. The time needed for prediction is also improved in a similar way. Specifically, the cost is reduced from $\mathcal{O}(n^2)$, in the standard BM, to $\mathcal{O}(d^2)$, in the SBM. A series of experiments carried out over the MNIST dataset of hand-written digits (LeCun et al., 1998) is performed to compare the performance of the SBM with the standard BM, the SBM without the additional refining iterations and the SVM (Vapnik, 1995). These experiments show that the SBM outperfoms both the SVM and the SBM without the additional refining iterations. Furthermore, they show that the SBM is comparable to the standard BM in terms of prediction error, even though only a subset of the total available instances are used for prediction. In consequence, these experiments confirm that implementing additional refining iterations in the algorithm of the SBM is beneficial. A disadvatage of the proposed approach is that the sparsity parameter d has to be fixed in some way. In general, this is a difficult task.

Chapter 7

A Bayesian Model for Microarray Data Classification

Microarray experiments are a promising tool for disease treatment and early diagnosis. However, the datasets obtained in these experiments typically have a small number of instances and a large number of covariates, most of which are irrelevant for discrimination. These characteristics make them difficult problems for standard classification algorithms. In particular, the predictors induced from these datasets can be rather unstable. Bayesian models can be useful to overcome this problem because they compute a posterior probability distribution for the model coefficients instead of a single point estimate. Using this posterior distribution, these models consider different values for the model coefficients, leading to more robust estimates of the quantities of interest. However, exact Bayesian inference is often infeasible. In practice, some form of approximation has to be made. In this chapter we consider a Bayesian model for microarray data classification based on a prior distribution that favors sparsity in the model coefficients. Expectation propagation (EP) is then used to perform approximate inference. EP is a practicable alternative to more computationally demanding methods such as Markov Chain Monte Carlo (MCMC) sampling. The model considered is evaluated on fifteen microarray datasets and compared with other robust classification algorithms. These experiments show that the model trained with EP performs well on the datasets investigated and can also be used to identify relevant genes for subsequent analysis.

7.1 Introduction

Microarray chips based on c-DNA hybridization technology generate large amounts of data by simultaneously measuring the expression level of several thousands of genes. However, biomedical experiments based on this technology are very expensive and hence, they are available only for a small number of subjects, e.g. (Bourquin et al., 2006; Ramaswamy et al., 2003). The reduced number of samples and the large dimensionality of the attributes measured make the analysis of microarray data a very challenging task that requires specialized statistical algorithms. A common application of microarray experiments is to diagnose some disease on the basis of the gene expression level measured for each individual (Dudoit and Fridlyand, 2003). Previous research in this field

has shown that only a reduced number of genes are actually relevant for classification (Dudoit et al., 2002; Guyon et al., 2002; Lee et al., 2005; Tibshirani et al., 2002). In consequence, most microarray classification algorithms perform a selection process to identify a set of possibly correlated genes which are relevant for the classification task considered. This feature selection mechanism is usually implemented by assuming a linear labeling function and by enforcing sparsity in the estimates of the model coefficients. Specifically, several model coefficients are driven to zero during learning and therefore do not contribute to the classification process. Zeroed coefficients are often determined by either cross-validation error minimization (Díaz-Uriarte and Alvarez de Andrés, 2006; Guyon et al., 2002; Tibshirani et al., 2002) or by maximum posterior estimation (MAP) with a prior distribution that favors sparse models (Cawley and Talbot, 2006; Krishnapuram et al., 2004; Li et al., 2002). Irrespective of the method used, the selection process can be unreliable because of the limited data available in microarray experiments (Dougherty, 2001). In particular, Li et al. (2002) and Cawley and Talbot (2006) find that the selected features can change substantially even when the training dataset is only slightly modified. Bayesian models with prior distributions that encourage sparsity can be useful to overcome this problem because they compute posterior probability distributions for the model coefficients instead of a single point estimate (Bishop, 2006; MacKay, 2003). Unlike approaches in which the posterior probability is maximized, these models do not generate sparse solutions. If only a finite amount of data is available, the values of the model coefficients are uncertain and, in consequence, the posterior probability of a coefficient being exactly zero is zero. Nevertheless, Bayesian models with prior distributions that encourage sparsity can be used to separate the model coefficients into two sets: a large set of coefficients whose value is close to zero with high posterior probability and a small set of coefficients whose posterior probability of being different from zero is large (Seeger, 2008). Additionally, Bayesian models can be used to incorporate prior knowledge into the classification task. This prior knowledge can compensate for the limited amount of data available.

In this chapter we consider a Bayesian model for microarray data classification based on the *spike and slab* sparse prior distribution (George and McCulloch, 1997). This prior introduces a binary latent variable for each gene to indicate whether the expression level of that particular gene will be used for classification or not. Bayes' theorem is then used to compute an estimate of the probability that each one of these variables is active. This estimate can be used to discriminate among different subsets of genes (i.e. those with a high probability of being excluded and those whose associated coefficients in the model are different from zero with a high probability) and hence, it can be useful for identifying relevant genes. Because exact inference in this model is infeasible, approximation techniques need to be used. Sparse prior distributions often lead to complicated posteriors that can have a large number of modes (Seeger, 2008). This, combined with the large dimensionality of microarray data, makes approximate inference a very difficult task. MCMC sampling techniques are typically used to address this problem, e.g. (Bae and Mallick, 2004; Lee et al., 2003; Zhou et al., 2004). These techniques are based on sampling from the posterior distribution by running a Markov chain whose stationary distribution coincides with the posterior distribution of the model. The samples are then used to compute probability estimates or to approximate the predictive distribution of the model for new instances. A difficulty with this approach is that obtaining independent samples demands running very long Markov chains. This significantly increases the training cost of the Bayesian model.

In this chapter we propose to use expectation propagation (EP) (Minka, 2001b), which is an efficient approximate inference algorithm. EP approximates the posterior distribution of the model using a simple distribution that is computed as the product of several terms that belong to the exponential family of probability distributions. These terms are iteratively updated until convergence by a series of rules obtained from matching the expected values of the sufficient statistics. A drawback of EP is that poor approximations can be obtained when the posterior distribution is multi-modal (Bishop, 2006). The typical scenario for such poor behavior is a bimodal posterior distribution that is approximated by a Gaussian. EP places the mean of the Gaussian between the two modes of the true posterior, where the probability can be rather low. In spite of this limitation, experimental evaluation of EP applied to the considered model on a set of fifteen microarray datasets shows that it does not seem to be affected by this problem. The overall performance of EP is comparable with other microarray classification techniques. Furthermore, genes whose coefficients have a high a probability of being different from zero in the posterior approximation are good candidates for subsequent analysis.

The chapter is organized as follows: Section 7.2 introduces the Bayesian model for microarray data classification based on a sparse prior distribution. Section 7.3 describes how the EP algorithm can be used to make approximate inference in this model. Section 7.4 presents experimental results on several microarray datasets. These experiments are carried out to evaluate the performance of the proposed method and compare it with other classifiers. Finally, Section 7.5 summarizes the conclusions of this investigation.

7.2 A *Spike and Slab* Model for Microarray Classification

The starting point of this analysis is the Bayesian model for microarray data classification of Lee et al. (2003). The goal of this model is to learn a decision function that discriminates, based on gene expression measurements, between tissues belonging to different classes (e.g. tumor and normal samples). For this purpose, a set of n d-dimensional input examples $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and the corresponding target class labels $\mathbf{y} = \{y_1, \ldots, y_n\}$, $y_i \in \{-1, 1\}$ is available. Typically, the number of observations n is small and the number of genes d is large. The model proposed in (Lee et al., 2003) assumes that there is a monotonic relation between the probability of the target value y_i and the value of a linear combination of the gene expression measurements in $\mathbf{x}_i$. In particular, the classification rule of this model is

$$y_i = \begin{cases} 1 & \text{if } \mathbf{w}^T \mathbf{x}_i + \epsilon_i \geq 0 \\ -1 & \text{otherwise,} \end{cases} \tag{7.1}$$

where $\mathbf{w}$ are the model coefficients, ϵ_i follows a standard Gaussian distribution and the superscript T means transpose. The classification rule (7.1) is obtained from the assumption that the gene expression measurements $\mathbf{x}_i$ are contaminated by some d-dimensional Gaussian noise vector $\boldsymbol{\xi}_i$ such that $\mathbf{w}^T \boldsymbol{\xi}_i$ has a standard Gaussian distribution. An alternative to (7.1) results from assuming that the noise term ϵ_i follows the logistic distribution (Krishnapuram et al., 2004), which is less sensitive to outliers. This functional form for the noise distribution has not been considered here because it makes Bayesian inference harder (Bishop, 2006). The hyper-plane defined by $\mathbf{w}$ is required to contain the origin. This can be readily achieved by including an additional attribute in $\mathbf{x}_i$ that is constant and equal to one. In this model the likelihood function for a vector of weights

$\mathbf{w}$ given $\mathbf{y}$ and $\mathbf{X}$ is

$$\mathcal{P}(\mathbf{y}|\mathbf{w},\mathbf{X}) = \prod_{i=1}^{n} \mathcal{P}(y_i|\mathbf{x}_i,\mathbf{w}) = \prod_{i=1}^{n} \Phi(y_i \mathbf{w}^T \mathbf{x}_i), \tag{7.2}$$

where $\Phi(\cdot)$ is the cumulative probability function of a standard Gaussian distribution. For notational convenience, $\mathbf{X}$ will not appear in the subsequent expressions, where it is understood that it is always included in the set of conditioning variables. To further simplify the notation, we also assume that each $\mathbf{x}_i$ vector is scaled by the corresponding class label y_i.

To complete the Bayesian model we assume a prior distribution for $\mathbf{w}$. Because in microarray datasets only a small subset of the genes are actually relevant for discrimination, it seems natural to make use of a prior distribution for $\mathbf{w}$ that encourages sparsity. In particular, we employ the *spike and slab* prior introduced by George and McCulloch (1997) and later used by Lee et al. (2003) to make Bayesian inference in microarray classification problems. In this prior all components of the vector $\mathbf{w}$ are assumed to be independent. Binary latent variables γ_i are introduced to reflect whether the expression level of the i-th gene is used for classification ($\gamma_i = 1$) or not ($\gamma_i = 0$). Given $\boldsymbol{\gamma}$, the prior for $\mathbf{w}$ is

$$\mathcal{P}(\mathbf{w}|\boldsymbol{\gamma}) = \prod_{i=1}^{d} \mathcal{N}(w_i|0,\sigma_1^2)^{\gamma_i} \mathcal{N}(w_i|0,\sigma_0^2)^{1-\gamma_i}, \tag{7.3}$$

where $\mathcal{N}(w_i|0,\sigma_1^2)$ denotes a Gaussian density with zero mean and σ_1^2 variance evaluated at w_i. In order to enforce sparsity, the standard deviation of the *slab* σ_1 is set to 1 and the distribution of *spike* is assumed to be a delta function centered at the origin ($\sigma_0 \to 0^+$). Finally, independent *Bernoulli* priors are assumed for the components of $\boldsymbol{\gamma}$

$$\mathcal{P}(\boldsymbol{\gamma}) = \prod_{i=1}^{d} \rho_0^{\gamma_i} (1-\rho_0)^{1-\gamma_i}. \tag{7.4}$$

This prior assumes that each gene contributes independently to the classification process with probability equal to ρ_0.

The joint posterior distribution for $\boldsymbol{\gamma}$ and $\mathbf{w}$ is computed using Bayes' theorem

$$\mathcal{P}(\boldsymbol{\gamma},\mathbf{w}|\mathbf{y}) = \frac{\mathcal{P}(\mathbf{y}|\mathbf{w})\mathcal{P}(\mathbf{w}|\boldsymbol{\gamma})\mathcal{P}(\boldsymbol{\gamma})}{\mathcal{P}(\mathbf{y})}, \tag{7.5}$$

where $\mathcal{P}(\mathbf{y})$ is a normalization constant that can be used to perform model selection (Bishop, 2006; MacKay, 2003).

In this model, a new instance $\mathbf{x}^{\text{new}}$ is classified using the predictive distribution for its target class $y^{\text{new}} \in \{1,-1\}$

$$\mathcal{P}(y^{\text{new}}|\mathbf{x}^{\text{new}},\mathbf{y}) = \int \mathcal{P}(y^{\text{new}}|\mathbf{x}^{\text{new}},\mathbf{w})\mathcal{P}(\boldsymbol{\gamma},\mathbf{w}|\mathbf{y}) d\boldsymbol{\gamma} d\mathbf{w}. \tag{7.6}$$

This probabilistic output is useful to quantify the uncertainty in the prediction. Finally, the genes with the highest contribution to the classification process can be identified using the posterior distribution of $\boldsymbol{\gamma}$

$$\mathcal{P}(\boldsymbol{\gamma}|\mathbf{y}) = \int \mathcal{P}(\boldsymbol{\gamma},\mathbf{w}|\mathbf{y}) d\mathbf{w}. \tag{7.7}$$

Unfortunately, the exact evaluation of (7.5), (7.6) and (7.7) is too costly to be practicable and one has to resort to approximation techniques. Lee et al. (2003) have proposed an approach based on MCMC sampling. However, this algorithm requires rather long simulations of the Markov chain. Inspired by the success of EP (Minka, 2001b) in a related problem involving time-dependent gene expression data (Hernández-Lobato et al., 2008), we propose to apply this efficient algorithm to perform approximate inference in the Bayesian model introduced in this section.

7.3 EP for the *Spike and Slab* Model

In this section we describe the application of the EP algorithm to the Bayesian model for microarray data analysis introduced in the previous section. We begin by factorizing the joint distribution of the observed class labels and the model parameters as the product of $n+d+1$ terms t_i. Namely, n terms for the likelihood, d terms for the prior for $\mathbf{w}$ given $\boldsymbol{\gamma}$ and one term for the prior for $\boldsymbol{\gamma}$. Each of these terms is approximated by a simple term $\tilde{t}_i$ that is restricted to belong to the exponential family of probability distributions

$$\mathcal{P}(\mathbf{y}, \mathbf{w}, \boldsymbol{\gamma}) = \left[\prod_{i=1}^{n} \mathcal{P}(y_i|\mathbf{w})\right] \left[\prod_{i=1}^{d} \mathcal{P}(w_i|\gamma_i)\right] \mathcal{P}(\boldsymbol{\gamma}) = \prod_{i=1}^{n+d+1} t_i(\mathbf{w}, \boldsymbol{\gamma}) \approx \prod_{i=1}^{n+d+1} \tilde{t}_i(\mathbf{w}, \boldsymbol{\gamma})\,. \tag{7.8}$$

Next, we assume that the posterior approximation of (7.5), $\mathcal{Q}$, is the product of d Bernoulli distributions and a factorized Gaussian distribution

$$\mathcal{Q}(\mathbf{w}, \boldsymbol{\gamma}) = \prod_{j=1}^{d} \rho_j^{\gamma_j} (1-\rho_j)^{1-\gamma_j} \mathcal{N}(w_j|\mu_j, \nu_j)\,, \tag{7.9}$$

where the d-dimensional vectors $\boldsymbol{\rho}$, $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ are the parameters of the posterior approximation. Note that (7.9) belongs to the exponential family because it is the product of distributions that also belong to this family. In this approximation, the vectors $\boldsymbol{\gamma}$ and $\mathbf{w}$ and their components are assumed to be independent. The purpose of this independence assumption is to obtain a faster EP algorithm. Approximations that model correlations between $\boldsymbol{\gamma}$ and $\mathbf{w}$ and their components can be more accurate, but also increase the computational cost of EP. Thus, the approximate terms $\tilde{t}_i$ have a similar form as (7.9), although they do not have to be normalized

$$\tilde{t}_i(\boldsymbol{\gamma}, \mathbf{w}) = s_i \prod_{j=1}^{d} p_{ij}^{\gamma_j} (1-p_{ij})^{1-\gamma_j} \exp\left(-\frac{1}{2v_{ij}}(w_j - m_{ij})^2\right)\,, \tag{7.10}$$

where s_i is a constant that ensures that the integral of $\tilde{t}_i \prod_{j\neq i} \tilde{t}_j$ is the same as the integral of $t_i \prod_{j\neq i} \tilde{t}_j$ and the d-dimensional vectors $\mathbf{p}_i$, $\mathbf{m}_i$ and $\mathbf{v}_i$ are free parameters. Note that (7.10) has the same form (except for the Gaussian part) as the prior for $\boldsymbol{\gamma}$ given in (7.4). In consequence, the optimal update of the approximate term $\tilde{t}_{n+d+1}$ corresponding to the prior for $\boldsymbol{\gamma}$ is to set $s_{n+d+1} = 1$, $p_{(n+d+1)j} = \rho_0$, $m_{(n+d+1)j} = 0$ and $v_{(n+d+1)j} = +\infty$ for $j = 1, \ldots, d$. Furthermore, if $\mathcal{Q}$ is initialized to $\tilde{t}_{n+d+1}$ then the preprocessing of the exact term t_{n+d+1} is not needed in the main loop of the EP algorithm (see Section 5.5.1.3). This is obtained by setting $\rho_j = \rho_0$, $\mu_j = 0$ and $\nu_j = +\infty$ for $j = 1, \ldots, d$.

The first step of the EP algorithm consists in initializing all the approximate terms $\tilde{t}_i$ corresponding to the likelihood and the prior for $\mathbf{w}$ to be uniform. In the case of the Bernoulli part of $\tilde{t}_i$ this is obtained by setting all the probability values to 1/2. In the case of the Gaussian part, the means have to be set to zero and the variances to infinity. Then, the approximate term $\tilde{t}_{n+d+1}$ corresponding to the prior for $\boldsymbol{\gamma}$ and the posterior approximation $\mathcal{Q}$ are initialized to (7.4), as described before.

The next step of the EP algorithm is to remove an approximate term $\tilde{t}_i$ from $\mathcal{Q}$ to compute $\mathcal{Q}^{\setminus i}$. $\mathcal{Q}^{\setminus i}$ has the same form as $\mathcal{Q}$ because of the closure property of the exponential family

$$\mathcal{Q}^{\setminus i}(\mathbf{w}, \boldsymbol{\gamma}) = \prod_{j=1}^{d} (\rho_j^{\setminus i})^{\gamma_j} (1 - \rho_j^{\setminus i})^{1-\gamma_j} \mathcal{N}(w_j | \mu_j^{\setminus i}, \nu_j^{\setminus i}) \,. \tag{7.11}$$

The parameters $\boldsymbol{\rho}^{\setminus i}$, $\boldsymbol{\mu}^{\setminus i}$ and $\boldsymbol{\nu}^{\setminus i}$ of $\mathcal{Q}^{\setminus i}$ are found by computing the quotient $\mathcal{Q}/\tilde{t}_i$ and normalizing

$$\boldsymbol{\nu}^{\setminus i} = (\boldsymbol{\nu}^{-1} - \mathbf{v}_i^{-1})^{-1} \,, \tag{7.12}$$

$$\boldsymbol{\mu}^{\setminus i} = \boldsymbol{\nu}^{\setminus i} \circ \left(\boldsymbol{\nu}^{-1} \circ \boldsymbol{\mu} - \mathbf{v}_i^{-1} \circ \mathbf{m}_i \right) \,, \tag{7.13}$$

$$\boldsymbol{\rho}^{\setminus i} = \frac{\boldsymbol{\rho} \circ \mathbf{p}_i^{-1}}{\boldsymbol{\rho} \circ \mathbf{p}_i^{-1} + (1 - \boldsymbol{\rho}) \circ (1 - \mathbf{p}_i)^{-1}} \,, \tag{7.14}$$

where the operator $\circ$ indicates the Hadamard (element-wise) product and the inverse of a vector is defined as a new vector whose components are the inverse of the components of the original vector. For the computation of (7.12) and (7.13) we have used (A.46), and for the computation of (7.14) we have used (A.10).

The next step of the algorithm consists in computing an updated posterior distribution $\hat{\mathcal{P}}$

$$\hat{\mathcal{P}}(\mathbf{w}, \boldsymbol{\gamma}) = \frac{1}{Z_i} t_i(\mathbf{w}, \boldsymbol{\gamma}) \mathcal{Q}^{\setminus i}(\mathbf{w}, \boldsymbol{\gamma}) \,, \tag{7.15}$$

where Z_i is just a normalization constant. In the case of the likelihood terms t_i, with $i = 1, \ldots, n$, this constant is

$$\begin{aligned} Z_i &= \sum_{\boldsymbol{\gamma}} \int t_i(\mathbf{w}, \boldsymbol{\gamma}) \mathcal{Q}^{\setminus i}(\mathbf{w}, \boldsymbol{\gamma}) d\mathbf{w} \\ &= \sum_{\boldsymbol{\gamma}} \int \Phi(\mathbf{w}^T \mathbf{x}_i) \prod_{j=1}^{d} (\rho_j^{\setminus i})^{\gamma_j} (1 - \rho_j^{\setminus i})^{1-\gamma_j} \mathcal{N}(w_j | \mu_j^{\setminus i}, \nu_j^{\setminus i}) d\mathbf{w} \\ &= \Phi(z_i) \,, \end{aligned} \tag{7.16}$$

where

$$z_i = \frac{\mathbf{x}_i^T \boldsymbol{\mu}^{\setminus i}}{\sqrt{\mathbf{x}_i^T (\boldsymbol{\nu}^{\setminus i} \circ \mathbf{x}_i) + 1}} \,. \tag{7.17}$$

In the case of the terms t_{n+i}, with $i = 1, \ldots, d$, corresponding to the prior for $\mathbf{w}$ given $\boldsymbol{\gamma}$, this constant is

$$\begin{aligned} Z_{n+i} &= \sum_{\boldsymbol{\gamma}} \int t_{n+i}(\mathbf{w}, \boldsymbol{\gamma}) \mathcal{Q}^{\backslash n+i}(\mathbf{w}, \boldsymbol{\gamma}) d\mathbf{w} \\ &= \sum_{\boldsymbol{\gamma}} \int \mathcal{N}(w_i|0, \sigma_1^2)^{\gamma_i} \mathcal{N}(w_i|0, \sigma_0^2)^{1-\gamma_i} \\ &\quad \prod_{j=1}^{d} (\rho_j^{\backslash n+i})^{\gamma_j} (1 - \rho_j^{\backslash n+i})^{1-\gamma_j} \mathcal{N}(w_j|\mu_j^{\backslash n+i}, \nu_j^{\backslash n+i}) d\mathbf{w} \\ &= \rho_i^{\backslash n+i} \mathcal{G}_1 + (1 - \rho_i^{\backslash n+i}) \mathcal{G}_0 \,, \end{aligned} \tag{7.18}$$

where

$$\mathcal{G}_0 = \mathcal{N}(0|\mu_i^{\backslash n+i}, \nu_i^{\backslash n+i} + \sigma_0^2) \,, \qquad \mathcal{G}_1 = \mathcal{N}(0|\mu_i^{\backslash n+i}, \nu_i^{\backslash n+i} + \sigma_1^2) \,. \tag{7.19}$$

Once $\hat{\mathcal{P}}$ has been normalized, we have to minimize the Kullback-Liebler (KL) divergence between $\hat{\mathcal{P}}$ and $\mathcal{Q}$ to find the parameters of the updated posterior approximation $\mathcal{Q}^{\text{new}}$. Since $\mathcal{Q}$ belongs to the exponential family of probability distributions and factorizes with respect to $\mathbf{w}$, $\boldsymbol{\gamma}$ and each component of these vectors, the parameters of $\mathcal{Q}^{\text{new}}$ are found by matching the sufficient statistics of each marginal distribution. In particular, from the minimization of the KL-divergence we get the following expectation constraints

$$\mathbb{E}_{\mathcal{Q}^{\text{new}}}[\mathbf{w}] = \mathbb{E}_{\hat{\mathcal{P}}}[\mathbf{w}] \,, \tag{7.20}$$

$$\mathbb{E}_{\mathcal{Q}^{\text{new}}}[\mathbf{w} \circ \mathbf{w}] = \mathbb{E}_{\hat{\mathcal{P}}}[\mathbf{w} \circ \mathbf{w}] \,, \tag{7.21}$$

$$\mathbb{E}_{\mathcal{Q}^{\text{new}}}[\boldsymbol{\gamma}] = \mathbb{E}_{\hat{\mathcal{P}}}[\boldsymbol{\gamma}] \,. \tag{7.22}$$

The expression of the sufficient statistics of the Bernoulli and the Gaussian distribution are given in Appendix A.1 and Appendix A.3, respectively. To match constraints (7.20) and (7.21) we can use (A.40) and (A.41), respectively. Constraint (7.22) can be matched using (A.6). In the case of the likelihood terms t_i, with $i = 1, \ldots, n$, this gives the following parameters for $\mathcal{Q}^{\text{new}}$

$$\boldsymbol{\mu}^{\text{new}} = \boldsymbol{\mu}^{\backslash i} + \alpha_i \boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i \,, \tag{7.23}$$

$$\boldsymbol{\nu}^{\text{new}} = \boldsymbol{\nu}^{\backslash i} - \frac{\alpha_i (\mathbf{x}_i^T \boldsymbol{\mu}^{\text{new}} + \alpha_i)}{\mathbf{x}_i^T (\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i) + 1} \left(\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i \right) \circ \left(\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i \right) \,, \tag{7.24}$$

$$\boldsymbol{\rho}^{\text{new}} = \boldsymbol{\rho}^{\backslash i} \,, \tag{7.25}$$

where

$$\alpha_i = \frac{1}{\sqrt{\mathbf{x}_i^T (\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i) + 1}} \frac{\mathcal{N}(z_i|0,1)}{\Phi(z_i)} \,, \qquad z_i = \frac{\mathbf{x}_i^T \boldsymbol{\mu}^{\backslash i}}{\sqrt{\mathbf{x}_i^T (\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i) + 1}} \,. \tag{7.26}$$

In the case of the terms t_{n+i}, with $i = 1, \ldots, d$, corresponding to the prior for $\mathbf{w}$ given $\boldsymbol{\gamma}$, the parameters of $\mathcal{Q}^{\text{new}}$ are

$$\boldsymbol{\mu}^{\text{new}} = \boldsymbol{\mu}^{\backslash n+i} + \boldsymbol{\delta}_i c_1 \nu_i^{\backslash n+i}, \tag{7.27}$$

$$\boldsymbol{\nu}^{\text{new}} = \boldsymbol{\nu}^{\backslash n+i} - \boldsymbol{\delta}_i c_3 (\nu_i^{\backslash n+i})^2, \tag{7.28}$$

$$\boldsymbol{\rho}^{\text{new}} = \boldsymbol{\rho}^{\backslash n+i} + \boldsymbol{\delta}_i \frac{\mathcal{G}_1 - \mathcal{G}_0}{Z_{n+i}} \rho_i^{\backslash n+i} (1 - \rho_i^{\backslash n+i}), \tag{7.29}$$

where

$$c_1 = \frac{1}{Z_{n+i}} \left(\rho_i^{\backslash n+i} \mathcal{G}_1 \frac{-\mu_i^{\backslash n+i}}{\nu_i^{\backslash n+i} + \sigma_1^2} + (1 - \rho_i^{\backslash n+i}) \mathcal{G}_0 \frac{-\mu_i^{\backslash n+i}}{\nu_i^{\backslash n+i} + \sigma_0^2} \right), \tag{7.30}$$

$$c_2 = \frac{1}{Z_{n+i}} \frac{1}{2} \left(\rho_i^{\backslash n+i} \mathcal{G}_1 \left(\frac{(\mu_i^{\backslash n+i})^2}{(\nu_i^{\backslash n+i} + \sigma_1^2)^2} - \frac{1}{\nu_i^{\backslash n+i} + \sigma_1^2} \right) + (1 - \rho_i^{\backslash n+i}) \mathcal{G}_0 \left(\frac{(\mu_i^{\backslash n+i})^2}{(\nu_i^{\backslash n+i} + \sigma_0^2)^2} - \frac{1}{\nu_i^{\backslash n+i} + \sigma_0^2} \right) \right), \tag{7.31}$$

$$c_3 = c_1^2 - 2c_2 \tag{7.32}$$

$$\tag{7.33}$$

and $\boldsymbol{\delta}_i$ is a d-dimensional vector with all components equal to zero except component i which takes value one

The next step of the EP algorithm consists in updating the corresponding approximate term $\tilde{t}_i$ by setting $\tilde{t}_i = Z_i \mathcal{Q}^{\text{new}} / \mathcal{Q}^{\backslash i}$. This process is slightly different depending on the approximate term being processed. For the approximate terms $\tilde{t}_i$, with $i = 1, \ldots, n$, corresponding to the likelihood, both $\mathcal{Q}^{\text{new}}$ and $\mathcal{Q}^{\backslash i}$ have the same marginal distribution over $\boldsymbol{\gamma}$. This is so because (7.25) does not change the parameter $\boldsymbol{\rho}$ of the posterior approximation, i.e. $\boldsymbol{\rho}^{\text{new}} = \boldsymbol{\rho}^{\backslash i}$. In consequence, when $\tilde{t}_i$ is computed as $Z_i \mathcal{Q}^{\text{new}} / \mathcal{Q}^{\backslash i}$ both distributions over $\boldsymbol{\gamma}$ cancel and we can write

$$\tilde{t}_i(\mathbf{w}, \boldsymbol{\gamma}) = s_i \prod_{j=1}^{d} \exp\left(-\frac{1}{2v_{ij}} (w_j - m_{ij})^2 \right). \tag{7.34}$$

The parameters s_i, $\mathbf{m}_i$ and $\mathbf{v}_i$ of $\tilde{t}_i$ are updated according to

$$\mathbf{v}_i = \left((\boldsymbol{\nu}^{\text{new}})^{-1} - (\boldsymbol{\nu}^{\backslash i})^{-1} \right)^{-1} = \frac{\mathbf{x}_i^T (\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i) + 1}{\alpha_i (\mathbf{x}_i^T \boldsymbol{\mu}^{\text{new}} + \alpha_i)} \mathbf{x}_i^{-1} \circ \mathbf{x}_i^{-1} - \boldsymbol{\nu}^{\backslash i}, \tag{7.35}$$

$$\mathbf{m}_i = \mathbf{v}_i \left((\boldsymbol{\nu}^{\text{new}})^{-1} \boldsymbol{\mu}^{\text{new}} - (\boldsymbol{\nu}^{\backslash i})^{-1} \boldsymbol{\mu}^{\backslash i} \right) = \boldsymbol{\mu}^{\backslash i} + \alpha_i \left(\mathbf{v}_i + \boldsymbol{\nu}^{\backslash i} \right) \circ \mathbf{x}_i, \tag{7.36}$$

$$s_i = Z_i \prod_{j=1}^{d} \sqrt{1 + v_{ij}^{-1} \nu_j^{\backslash i}} \exp\left(\frac{d}{2} \frac{\alpha_i \mathbf{x}_i^T (\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i) + \alpha_i}{\mathbf{x}_i^T \boldsymbol{\mu}^{\text{new}} + \alpha_i} \right), \tag{7.37}$$

where d is the dimensionality of the data vectors. For the computation of (7.35) we have used (A.46), (7.24) and the Woodbury formula (B.1). For the computation of (7.36) we have used (A.46), $(\boldsymbol{\nu}^{\text{new}})^{-1} = \mathbf{v}_i^{-1} + (\boldsymbol{\nu}^{\backslash i})^{-1}$, which can be derived from (7.35), and (7.23). The derivation of (7.37) is given in Appendix C.1. Using (7.34) instead of (7.10) to represent the approximate terms corresponding to the likelihood has the advantage of

reducing the storage space because these terms no longer include a product of Bernoulli distributions. However, when this representation is used (7.14) is no longer valid to compute the value of $\boldsymbol{\rho}^{\backslash i}$, for $i = 1, \ldots, n$. For these terms, this equation has to be replaced by

$$\boldsymbol{\rho}^{\backslash i} = \boldsymbol{\rho}\,. \tag{7.38}$$

For the approximate terms $\tilde{t}_{n+i}$, with $i = 1, \ldots, d$, corresponding to the prior for $\mathbf{w}$ given $\boldsymbol{\gamma}$, the parameters of the posterior approximation $\mathcal{Q}^{\text{new}}$ computed by (7.27), (7.28) and (7.29) only differ from the parameters of $\mathcal{Q}^{\backslash n+i}$ in the i-th component of each parameter vector. In consequence, these approximate terms also have a simpler form when computed as $\tilde{t}_{n+i} = Z_{n+i}\mathcal{Q}^{\text{new}}/\mathcal{Q}^{\backslash n+i}$. In particular,

$$\tilde{t}_{n+i}(\mathbf{w}, \boldsymbol{\gamma}) = s_{n+i} p_{n+i}^{\gamma_i} (1 - p_{n+i})^{1-\gamma_i} \exp\left(-\frac{1}{2v_{n+i}} (w_i - m_{n+i})^2\right), \tag{7.39}$$

where s_{n+i}, p_{n+i}, m_{n+i} and v_{n+i} are free parameters. The update equations for the parameters of $\tilde{t}_{n+i}$, with $i = 1, \ldots, d$, are

$$v_{n+i} = \left((\nu_i^{\text{new}})^{-1} - (\nu_i^{\backslash n+i})^{-1}\right)^{-1} = c_3^{-1} - \nu_i^{\backslash n+i}, \tag{7.40}$$

$$m_{n+i} = v_{n+i}\left((\nu_i^{\text{new}})^{-1}\mu_i^{\text{new}} - (\nu_i^{\backslash n+i})^{-1}\mu_i^{\backslash n+i}\right) = \mu_i^{\backslash n+i} + c_1\left(v_{n+i} + \nu_i^{\backslash n+i}\right), \tag{7.41}$$

$$p_{n+i} = \frac{\rho_i^{\text{new}}/\rho_i^{\backslash n+i}}{\rho_i^{\text{new}}/\rho_i^{\backslash n+i} + (1 - \rho_i^{\text{new}})/(1 - \rho_i^{\backslash n+i})} = \frac{\mathcal{G}_1}{\mathcal{G}_1 + \mathcal{G}_0}, \tag{7.42}$$

$$s_{n+i} = (\mathcal{G}_1 + \mathcal{G}_0)\sqrt{1 + \nu_i^{\backslash n+i} v_{n+i}^{-1}} \exp\left(\frac{1}{2}\frac{c_1^2}{c_3}\right). \tag{7.43}$$

For the computation of (7.40) we have used (A.46) and (7.28). For the computation of (7.41) we have used (A.46), $(\nu_i^{\text{new}})^{-1} = (\nu_i^{\backslash n+i})^{-1} + v_{n+i}^{-1}$, which can be derived from (7.40), and (7.27). Finally, for the computation of (7.42) we have used (A.10), (7.29) and (7.18). The derivation of (7.43) is given in Appendix C.2. Using (7.39) instead of (7.10) to represent the approximate terms $\tilde{t}_{n+i}$, with $i = 1, \ldots, d$, reduces the storage space and also increases the speed of the EP algorithm. Specifically, we only need to store four parameters for each approximate term $\tilde{t}_{n+i}$. These terms can be updated in constant time. However, when (7.39) is used instead of (7.10), equations (7.12), (7.13) and (7.14) can not be used to compute the parameters of $\mathcal{Q}^{\backslash n+i}$, with $i = 1, \ldots, d$. These equations have to be replaced by

$$\nu_i^{\backslash n+i} = \left(\nu_i^{-1} - v_{n+i}^{-1}\right)^{-1}, \tag{7.44}$$

$$\mu_i^{\backslash n+i} = \nu^{\backslash n+i}\left(\nu_i^{-1}\mu_i - v_{n+i}^{-1} m_{n+i}\right), \tag{7.45}$$

$$\rho_i^{\backslash n+i} = \frac{\rho_i/p_{n+i}}{\rho_i/p_{n+i} + (1 - \rho_i)/(1 - p_{n+i})}, \tag{7.46}$$

where have used (A.46) for the computation of (7.44) and (7.45), and (A.10) for the computation of (7.46). Note that we only need $\nu_i^{\backslash n+i}$, $\mu_i^{\backslash n+i}$ and $\rho_i^{\backslash n+i}$ to process the exact terms t_{n+i}, with $i = 1, \ldots, d$, corresponding to the prior for $\mathbf{w}$ given $\boldsymbol{\gamma}$. The remaining parameters of $\mathcal{Q}^{\backslash n+i}$, i.e. $\nu_j^{\backslash n+i}$, $\mu_j^{\backslash n+i}$ and $\rho_j^{\backslash n+i}$, with $j \neq i$, are not required.

Once that EP has converged, we can approximate the model evidence by

$$\mathcal{P}(\mathbf{y}) \approx \sum_{\boldsymbol{\gamma}} \int \prod_{i=1}^{n+d+1} \tilde{t}_i(\mathbf{w}, \boldsymbol{\gamma}) d\mathbf{w} = \sqrt{(2\pi)^d} \exp\left(\frac{B}{2}\right) \left[\prod_{i=1}^{d} \sqrt{\nu_i} C_i\right] \left[\prod_{i=1}^{n+d} s_i\right], \quad (7.47)$$

where we have used (A.7) and (A.42), and where

$$C_i = p_{n+i}\rho_0 + (1 - p_{n+i})(1 - \rho_0), \quad (7.48)$$

$$B = \boldsymbol{\mu}^T \left(\boldsymbol{\nu}^{-1} \circ \boldsymbol{\mu}\right) - \sum_{i=1}^{n} \mathbf{m}_i^T \left(\mathbf{v}_i^{-1} \circ \mathbf{m}_i\right) - \sum_{i=1}^{d} m_{n+i}^2 v_{n+i}^{-1}. \quad (7.49)$$

Furthermore, in the computation of (7.47) we have also used that $\tilde{t}_{n+d+1}$ is initialized to the prior distribution for $\boldsymbol{\gamma}$ and never modified.

The optimal classification rule for an arbitrary instance $\mathbf{x}^{\text{new}}$ can be approximated using (7.9)

$$\begin{aligned} \mathcal{P}(y_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{y}) &\approx \sum_{\boldsymbol{\gamma}} \int \mathcal{P}(y_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{w}) \mathcal{Q}(\mathbf{w}, \boldsymbol{\gamma}) d\mathbf{w} \\ &\approx \int \Phi\left(y_{\text{new}} \mathbf{w}^T \mathbf{x}_{\text{new}}\right) \prod_{j=1}^{d} \mathcal{N}(w_j|\mu_j, \nu_j) d\mathbf{w} \\ &\approx \Phi\left(\frac{y_{\text{new}} (\mathbf{x}_{\text{new}})^T \boldsymbol{\mu}}{\sqrt{(\mathbf{x}_{\text{new}})^T (\boldsymbol{\nu} \circ \mathbf{x}_{\text{new}}) + 1}}\right). \end{aligned} \quad (7.50)$$

Finally, to identify the genes that contribute the most to the classification process, the EP approximation of (7.7) can be used

$$\mathcal{P}(\boldsymbol{\gamma}|\mathbf{y}) \approx \prod_{j=1}^{d} \rho_j^{\gamma_i} (1 - \rho_j)^{1-\gamma_j}. \quad (7.51)$$

In this expression ρ_j is an approximation to the posterior probability of the expression level of gene j being used for classification. These probability values can be used for feature ranking.

Note that the update of the terms $\tilde{t}_i$, with $i = 1, \ldots, n$ corresponding to the likelihood takes $\mathcal{O}(nd)$ steps. However, the terms $\tilde{t}_{n+i}$, with $i = 1, \ldots, d$, corresponding to the prior for $\mathbf{w}$ given $\boldsymbol{\gamma}$, can be updated in only $\mathcal{O}(d)$ steps because of the factorized form of the approximation. In consequence, the total cost of one iteration of EP is $\mathcal{O}(nd)$. By contrast, if efficient matrix factorizations are employed (George and McCulloch, 1997), the cost of the MCMC sampling algorithm proposed by Lee et al. (2003) is on average $\mathcal{O}(\rho_0^2 d^3 k)$, where k is the number of samples generated from the Markov chain. Because the samples generated are not independent, this number is typically set very large, e.g. several thousands, which is the same order as d.

7.4 Experiments

The performance of the Bayesian model trained with EP is investigated in fifteen publicly available microarray classification problems. The characteristics and the sources of

the datasets are shown in Table 7.1. Non-binary classification problems are binarized as follows: in *Lymphoma* we discriminate between 42 samples of diffuse large B-cell lymphoma and 20 samples of follicular lymphoma and chronic lymphocytic leukemia; in *SRBCT* we discriminate between 23 Ewing family of tumors and 20 rhabdomyosarcomas; in *Metastasis* we discriminate between patients that developed metastasis within 5 years and those who did not; finally, in *Brain A* we discriminate between 10 malignant gliomas and 10 atypical teratoid/rhabdoid tumors. The datasets are preprocessed as follows: *Adenocarcinoma* and *Metastasis* are preprocessed as in (Díaz-Uriarte and Alvarez de Andrés, 2006); *Colon* is preprocessed as in (Guyon et al., 2002); *Breast ER*, *Breast LN*, *Brain A*, *Leukemia*, *Lymphoma*, *Prostate* and *SRBCT* are preprocessed as in (Dettling and Bühlmann, 2002); *Mutation* is log-transformed and normalized across genes and samples [1]; *Brain B*, *Brain C* and *Down Syndrome* are preprocessed following a protocol equal to the one described in the supplementary material of (Pomeroy et al., 2002). These datasets are thresholded, filtered by variation, log-transformed and normalized across genes and then across samples. The floor of the threshold step is set to 20 for *Brain B* and *Down Syndrome* and to 100 for *Brain C*. The ceiling is set to $16,000$ in these three datasets. Filtering by variation is performed by removing those genes with $\max/\min \leq 5$ or $(\max - \min) \leq 500$. Finally, the *Ovarian* dataset does not require any further preprocessing (Li et al., 2002).

TABLE 7.1: Main characteristics of the microarray datasets used.

Dataset	**Genes**	**Patients**	**Original Paper**
Adenocarcinoma	9,868	76	(Ramaswamy et al., 2003)
Brain A	5,597	20	(Pomeroy et al., 2002)
Brain B	2,275	34	”
Brain C	4,452	60	”
Breast ER	5,313	49	(West et al., 2001)
Breast LN	5,313	49	”
Colon	2,000	62	(Alon et al., 1999)
Down Syndrome	4,656	63	(Bourquin et al., 2006)
Leukemia	3,571	72	(Golub et al., 1999)
Lymphoma	4,026	62	(Alizadeh et al., 2000)
Metastasis	4,869	77	(van ’t Veer et al., 2002)
Mutation	3,226	22	(Hedenfalk et al., 2001)
Ovarian	1,536	54	(Schummer et al., 1999)
Prostate	6,033	102	(Singh et al., 2002)
SRBCT	2,308	43	(Khan et al., 2001)

The generalization performance of the Bayesian model trained with EP is compared with a range of classification systems: (a) a Bayesian model trained with EP that uses a standard Gaussian prior (i.e. $\rho = 1$). This model is included to determine whether the use of a *spike and slab* prior, which favors sparse models, is beneficial; (b) the Bayesian model with the MCMC sampling approach proposed by Lee et al. (2003). This method has been included to determine whether the EP algorithm, which can fail when the posterior distribution is multi-modal (Bishop, 2006), is sufficiently accurate; (c) three methods that have shown a good performance in several reviews of classification

[1] This normalization process is implemented by subtracting the mean value and dividing by the standard deviation.

methods applied to microarray data (Dettling, 2004; Dudoit et al., 2002; Lee et al., 2005) but which do not include any gene selection procedure: the support vector machine (SVM) with a linear kernel (Vapnik, 1995), the k-nearest neighbor classifier (KNN) (Hastie et al., 2001) and the diagonal linear discriminant analysis (DLDA) (Dudoit et al., 2002); (d) four methods that involve variable selection: the recursive feature elimination (RFE) algorithm introduced in (Guyon et al., 2002), the automatic relevance determination algorithm based on the relevance vector machine (RVM) (Li et al., 2002), the nearest shrunken centroids (NSC) method analyzed in (Tibshirani et al., 2002), and the *random forests* (RF) classifier investigated in (Díaz-Uriarte and Alvarez de Andrés, 2006). We do not compare with the Bayes machine described in Chapter 6 because this model is not expected to perform well in microarray classification problems. The Bayes machine assumes noise in the class labels of the training data. By contrast, in microarray experiments the class labels are free of noise and this appears only in the gene expression measurements.

In each experiment the data are randomly partitioned into two disjoint sets. The test sets need to be sufficiently large, so that significant differences in performance among the classifiers investigated can actually be observed. Following (Dudoit et al., 2002), two thirds of the instances available are used for training and the remaining one third for testing. To reduce the variability of the results, this splitting process is repeated 50 times and the test error estimates are averaged over the different realizations of the problem. The gene expression levels are normalized so that they have zero mean and unit standard deviation on the training set. The hyper-parameters of the different methods are obtained by cross-validation using exclusively the training set, which avoids any selection bias (Ambroise and McLachlan, 2002). In the RFE algorithm, half of the variables are removed at each step until 500 variables remain. Then, variables are removed one at a time. The C parameter of the SVM is fixed to 100 as suggested by Guyon et al. (2002). In the k-NN classifier the Euclidean distance is used and the parameter k is selected from the range of odd values $k = 1, 3, \ldots, 19$. In the RF approach we use the default settings, i.e. s.e. $= 1$, mtryFactor $= 1$ and ntree $= 5,000$ and employ the out-of-bag error for variable elimination, as indicated by Díaz-Uriarte and Alvarez de Andrés (2006). In the RVM algorithm we use the regression likelihood function and fix $\sigma = 1$ as suggested by Li et al. (2002). In the EP algorithm, the parameter ρ_0 of the prior for $\boldsymbol{\gamma}$ is fixed so that, on average, 32 components of the vector are set to one, similarly as in (Lee et al., 2003). The hyper-parameters σ_0^2 and σ_1^2 of the *spike and slab* prior are set to zero and one respectively. The same values of the priors and the parameters are used in the MCMC method. The Markov chain is implemented with Gibbs sampling (Lee et al., 2003). A fast updating algorithm based on efficient matrix factorizations (Gill et al., 1974) is used as suggested by George and McCulloch (1997). The posterior distribution is approximated using $5,000$ samples after a burn-in period of $1,000$ samples. This number of samples is limited due to the large cost of the sampling algorithm. In the NSC method we use the default settings and employ an unbalanced cross validation procedure (default is balanced) to find the optimal threshold values.

Table 7.2 reports the estimated prediction error of the methods investigated on the different datasets. The last two rows of this table display the average test error and average rank of the classifiers. For each problem, the method that has the lowest error is highlighted in boldface. The overall performance of the Bayesian model with *spike and slab* priors trained using EP is fairly good. This method obtains the lowest average prediction error and the lowest average rank in the datasets investigated. Using the *spike and slab* prior instead of a standard Gaussian prior improves the performance of

the method. In most problems EP also outperforms the method based on Gibbs sampling (Lee et al., 2003). This indicates that EP provides a sufficiently accurate approximation to the posterior probability at a reduced computational cost. This is an unexpected result because EP may have difficulties when the posterior distribution is multi-modal, as one would expect to obtain when a *spike and slab* prior is used (Seeger, 2008). The good performance of EP may indicate that there is a dominant mode in the posterior distribution of the problems investigated. Among the other benchmark classifiers, the SVM with a linear kernel has a good overall accuracy and obtains the best results in several problems. The predictors obtained by k-NN and DLDA are clearly inferior to those of the SVM. However these predictors obtain a remarkably high accuracy in a few datasets. Finally, RFE and NSC have the best performance among the remaining feature selection techniques. However, RFE is worse than the SVM in several problems. This is in agreement with the findings of Ambroise and McLachlan (2002).

7.4.1 Identification of Relevant Genes

The EP algorithm is also useful to identify relevant genes for subsequent analysis. To illustrate this point, we use EP to train a Bayesian model assuming *spike and slab* priors on the *Leukemia* dataset using all the instances available. In particular, we analyze the parameter vector $\boldsymbol{\rho}$ of the posterior approximation $\mathcal{Q}$ obtained in the EP algorithm. Recall that the different components of this vector can be interpreted as an estimate of the posterior probability of each individual gene being used for classification. Figure 7.1 displays the different components of $\boldsymbol{\rho}$ and the prior probability level ρ_0 of each component. This prior probability is displayed as a discontinuous horizontal line. The figure shows that most of the components of this vector have low probability values that are close to the prior level ρ_0. Nevertheless, there is also a small subset of the components with significantly larger values. The genes associated to these components are the most relevant ones for classification, as estimated by the Bayesian model. We further investigate these genes and select the 16 genes that are ranked at the top according to the probability vector $\boldsymbol{\rho}$. Figure 7.2 displays a heat map of these genes for each patient in the dataset. The columns of the map correspond to different patients. The rows represent the selected genes. The ID of the genes is indicated on the right-hand-side of the table. The probability of the corresponding gene appears on the left-hand side of the table. Each cell in the map represents the expression of one gene in one patient and its color depicts the intensity (normalized expression level) from blue (large negative) to yellow (large positive). Patients are grouped according to the assigned class. Genes are ordered according to their average level of expression in the majority class. In particular, upper rows correspond to high average normalized expression levels in the majority class. This ordering of patients and genes uncovers a clear clustering pattern. There is a first group of genes (M31523 to D42043) whose normalized expression levels are consistently high for *AML* patients and low for *ALL* patients. The remaining genes exhibit the opposite pattern. The relevance of these genes for discriminating between the two class labels is apparent in this graph. Further evidence of the relevance of these genes comes from the literature: The *M27891* and *M84526* genes are listed among the top ranked features in (Bø and Jonassen, 2002; Lee et al., 2003) and are deemed useful for classification in (Golub et al., 1999; Krishnapuram et al., 2004); *M23197* appears as a relevant gene in (Golub et al., 1999; Lee et al., 2003; Li et al., 2002; Wang et al., 2005); *HG1612* is also

TABLE 7.2: Misclassification error of each method for each microarray dataset and average error and average rank of each method over all the datasets.

Dataset	**EP$_{\rho=1}$**	**EP**	**MCMC**	**SVM**	**KNN**	**DLDA**	**RFE**	**RVM**	**NSC**	**RF**
Adenocarcinoma	38.0±10.1	15.2±5.5	15.0±6.0	**13.8±6.0**	17.0±4.7	30.6±8.0	19.2±6.6	22.5±8.1	16.2±5.5	18.7±7.0
Brain A	5.4±9.5	4.9±9.4	4.9±9.4	2.3±5.3	**0.6±2.8**	6.9±10.1	5.7±10.4	14.6±15.7	18.3±21.4	18.0±14.4
Brain B	28.4±15.7	21.3±15.7	24.7±14.7	**15.1±10.0**	24.4±13.4	17.5±12.2	20.0±11.8	24.2±12.8	18.4±12.9	23.8±12.4
Brain C	39.9±10.0	37.7±10.7	39.2±10.5	**33.8±8.8**	40.0±8.0	38.6±8.9	38.3±9.2	45.1±9.1	38.5±8.3	42.4±11.1
Breast ER	12.2±7.4	**12.1±7.3**	12.4±7.4	12.5±8.1	18.9±9.6	13.6±8.3	17.1±9.8	16.4±10.4	15.6±10.2	20.2±9.2
Breast LN	38.5±12.1	33.4±10.8	35.6±12.6	41.1±11.3	44.0±10.2	39.4±11.9	33.6±12.3	**27.0±9.7**	30.4±16.0	34.9±12.0
Colon	17.3±7.3	16.3±7.2	17.2±7.6	17.6±8.5	28.8±10.4	16.4±7.3	19.0±7.7	18.8±7.5	**13.9±7.3**	23.4±8.7
Down Syndrome	9.3±7.0	8.2±7.2	8.4±6.8	6.1±5.2	15.3±7.4	11.3±6.7	5.6±5.1	7.9±4.7	**5.5±6.1**	6.8±5.9
Leukemia	6.0±4.8	4.2±3.5	5.4±4.2	**2.0±2.4**	5.7±4.0	2.8±3.2	3.2±3.8	6.1±4.7	3.9±4.4	5.8±5.3
Lymphoma	4.2±3.3	4.0±3.4	4.2±3.3	**0.5±1.4**	2.0±2.6	1.6±2.5	2.8±3.2	3.0±2.7	3.3±3.6	5.6±5.5
Metastasis	36.2±7.4	36.0±7.4	36.4±7.6	38.8±9.1	42.2±8.0	**35.6±8.2**	38.3±10.0	38.5±8.0	38.2±8.8	38.5±8.5
Mutation	13.7±12.6	**12.0±12.4**	12.9±12.3	24.0±13.7	24.0±12.1	23.1±13.5	23.1±15.0	32.6±16.3	27.4±18.4	38.3±14.6
Ovarian	9.7±5.8	**9.3±5.7**	9.8±6.2	9.7±4.2	15.8±7.8	9.9±5.6	12.3±7.0	12.4±7.2	11.0±5.8	12.1±7.9
Prostate	9.0±4.4	9.2±6.1	**7.8±4.2**	9.4±4.0	20.0±7.1	37.6±12.4	8.8±4.6	10.4±4.9	10.8±5.2	8.9±4.7
SRBCT	4.7±5.7	4.0±5.0	3.9±5.0	3.9±5.2	21.6±9.4	7.7±7.2	**3.1±4.1**	5.4±6.0	3.7±5.8	8.4±6.6
Avg. Error	18.2±13.8	**15.2±11.7**	15.8±12.3	15.4±13.4	21.3±13.4	19.5±13.6	16.7±12.4	19.0±12.6	17.0±11.8	20.4±12.9
Avg. Rank	6.0±2.7	**3.4±2.0**	4.6±2.4	3.8±2.9	7.8±2.8	5.4±3.0	4.7±2.4	7.1±2.3	4.7±2.7	7.5±2.3

selected for further analysis in (Bø and Jonassen, 2002; Guyon et al., 2002; Krishnapuram et al., 2004; Lee et al., 2003); *U46499* appears as an important gene in (Bø and Jonassen, 2002; Lee et al., 2003; Li et al., 2002; Wang et al., 2005); *X95735* is ranked first in (Guyon et al., 2002; Wang et al., 2005), fourth in (Lee et al., 2003), and is listed as useful for classification in (Golub et al., 1999; Li et al., 2002).

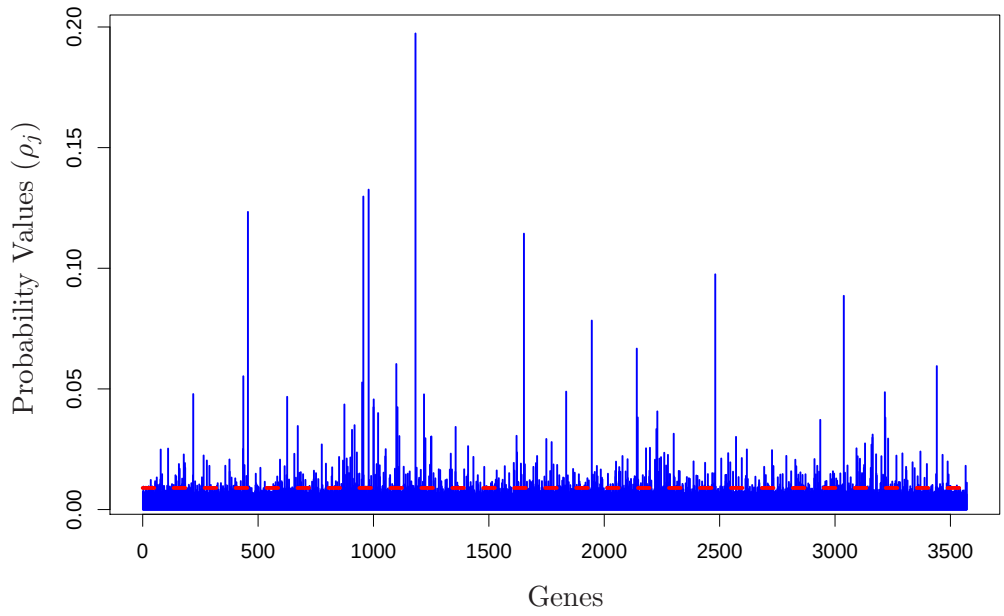

FIGURE 7.1: Values of the different components of the probability vector $\boldsymbol{\rho}$ computed by the EP algorithm in the Leukemia dataset. The prior level ρ_0 for each component of this vector is displayed as a discontinuous horizontal line.

7.4.2 Stability of the Gene Ranking

Finally, we carry out one additional experiment involving a representative subset of four microarray datasets, i.e. *Leukemia*, *SRBCT*, *Mutation* and *Down Syndrome*, to compare the stability of the gene ranking provided by EP with the stability of the rankings provided by other methods. In particular, we compare with the rankings provided by the RFE method (Guyon et al., 2002), the RF classifier of Díaz-Uriarte and Alvarez de Andrés (2006) and the MCMC sampling approach (Lee et al., 2003). This experiment consists in randomly extracting two thirds of the instances available in each microarray dataset to train the different classifiers. This process is repeated 50 times and the rankings provided by each different method for each realization of the training set are computed and stored. Then, a stability measure is evaluated for each method using the different rankings computed. The Kuncheva stability index is employed for this purpose (Kuncheva, 2007). This index measures to which extent T different sets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_T$ of k selected genes share common elements. Assume that $\mathcal{S}_i^k$ is the subset of the first k genes extracted from the ranking provided by one of these classifiers when it is evaluated in the i-th training set. The Kuncheva index for the sets $\mathcal{A}_k = \{\mathcal{S}_i^k : i = 1, \ldots, T\}$ is

$$\mathcal{I}_{\mathcal{S}}(\mathcal{A}_k) = \frac{2}{T(T-1)} \sum_{i=1}^{T-1} \sum_{j=i+1}^{T} \frac{|\mathcal{S}_i^k \cap \mathcal{S}_j^k| - \frac{k^2}{d}}{k - \frac{k^2}{d}}, \tag{7.52}$$

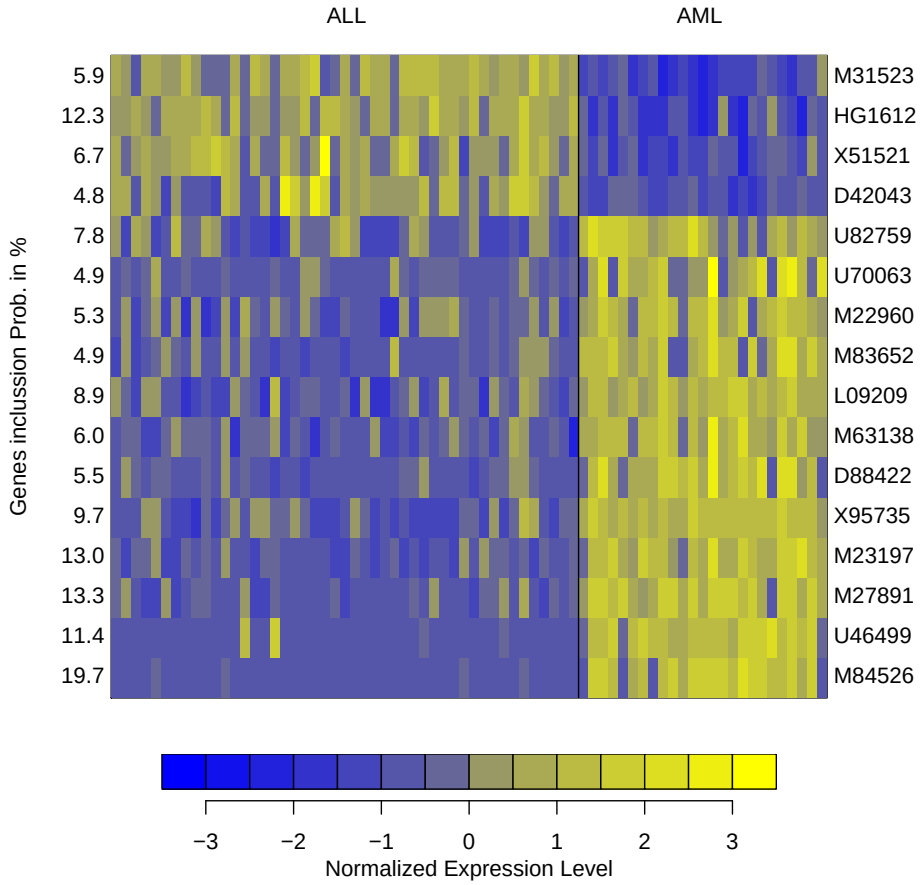

FIGURE 7.2: Normalized expression level of the 16 top ranked genes in the *Leukemia* dataset using the EP algorithm. Each row shows the expression level of a gene for each patient. Patients are grouped according to their class (*AML* and *ALL*) and genes are sorted according to the mean expression level within the majority class.

where d is the total number of different features, K is equal to the different number of training sets (i.e. $K = 50$) and the ratio k^2/d is the expected number of common genes between $\mathcal{S}_i^k$ and $\mathcal{S}_j^k$ just by chance. This index satisfies $-1 < \mathcal{I}_S(\mathcal{A}^k) \leq 1$ and the closer to one its value is, the larger are the number of commonly selected genes in the different sets. A value of the index close to zero indicates that the sets share common genes at the level expected by chance. For each microarray dataset, Figure 7.3 displays the value of the Kuncheva stability index, $\mathcal{I}_S(\mathcal{A}_k)$, for different values of k and for the different classifiers investigated in this experiment. The figure shows that the Bayesian model trained with EP is among the ranking methods that provide the largest values of $\mathcal{I}_S(\mathcal{A}_k)$. The other method that obtains the largest values for the stability index is RF, which is an ensemble method based on averaging the predictions of several classifiers obtained under different training conditions. Nevertheless, because RF employs the out-of-bag error estimate to perform the ranking (Díaz-Uriarte and Alvarez de Andrés, 2006), the stability of this method is severely impaired in datasets with a small number of training instances, such as *Mutation*. In this dataset the out-of-bag error estimate can be very similar when different genes are removed, leading to an unreliable ranking. Finally, the two methods that show the lowest values of $\mathcal{I}_S(\mathcal{A}_k)$ are the RFE method and the method based on MCMC sampling. The lower stability of the MCMC algorithm when compared to the EP approach can be explained because it is a stochastic method where

the probability vector $\boldsymbol{\rho}$ is estimated by computing an average over different samples of $\boldsymbol{\gamma}$ generated from a Markov chain whose stationary distribution coincides with the posterior distribution of the model (Lee et al., 2003). In particular, because the samples generated from this Markov chain are not independent, the variance of this estimate can be large.

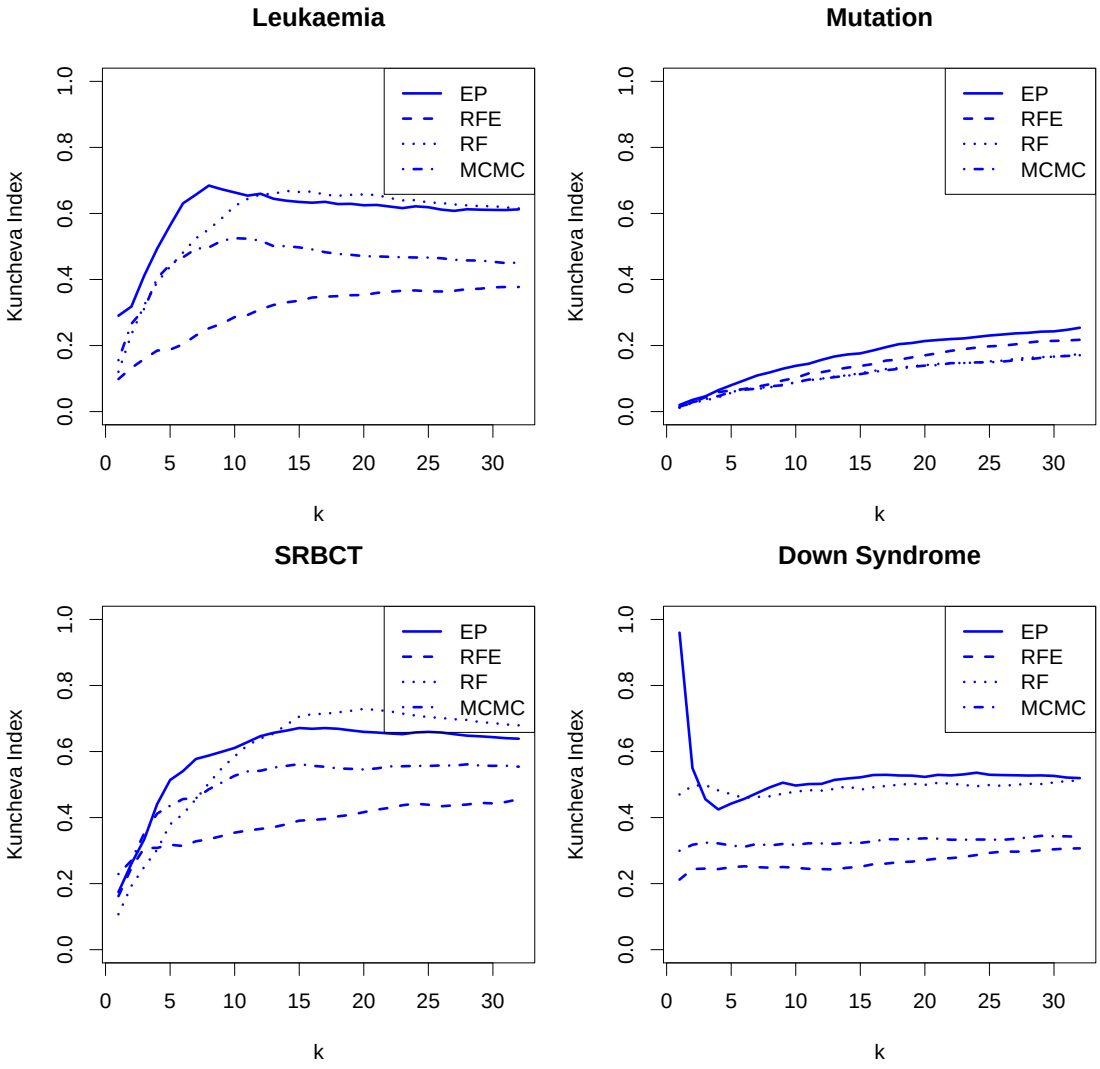

FIGURE 7.3: Kuncheva stability index for each microarray dataset and ranking method as a function of k, i.e. the number of genes extracted from each ranking.

7.5 Conclusions

In this chapter a Bayesian approach based on the use of the *spike and slab* prior distribution (George and McCulloch, 1997) is used to address the problem of microarray data classification. Assuming a linear relation between the gene expression levels and the target value, this prior distribution encourages sparsity in the model coefficients by introducing a binary latent variable $\gamma_i \in \{0, 1\}$ for each different gene. The value of this variable indicates whether the i-th gene contributes to the classification process ($\gamma_i = 1$)

or not ($\gamma_i = 0$). Exact Bayesian inference in such a model is infeasible. In consequence, approximate techniques have to be employed. Because sparse prior distributions often lead to posterior distributions that are multi-modal, spreading the probability mass among the different modes, accurate approximate inference can be a difficult problem (Seeger, 2008). Here, we make use of the efficient approximate inference algorithm EP (Minka, 2001b). EP approximates the posterior distribution by a product of simple terms that belong to the exponential family of probability distributions, although they do not have to be normalized. These terms are iteratively updated until convergence by rules obtained from solving moment matching constraints. Because the posterior approximation factorizes, the algorithm can be run in $\mathcal{O}(nd)$ steps, where n is the number of instances and d is the number of attributes (genes) per instance. This cost is typically much lower than the cost of alternative approximate inference algorithms like MCMC sampling (Bae and Mallick, 2004; Lee et al., 2003; Zhou et al., 2004). Experimental evaluation on a set of fifteen microarray datasets confirms that EP is competitive with other microarray classification techniques. This means that the posterior approximation obtained by EP is sufficiently accurate to provide *state-of-the-art* performance at a reduced computational cost. The EP algorithm is also useful for the identification of genes that are relevant for classification. A detailed investigation on the *Leukemia* dataset (Golub et al., 1999) shows that genes with the highest rank in the posterior approximation have expression levels that are either high for *AML* patients and low for *ALL* patients or the other way round. In consequence, these genes are good candidates for subsequent analysis. Furthermore, most of these genes appear labeled in several works as relevant for the classification process of this microarray dataset (Bø and Jonassen, 2002; Golub et al., 1999; Guyon et al., 2002; Krishnapuram et al., 2004; Lee et al., 2003; Li et al., 2002; Wang et al., 2005). Finally, additional experiments confirm the stability of the ranking computed by the EP algorithm in this Bayesian model when the training set is slightly modified.

Chapter 8

Conclusions and Future Work

Supervised machine learning methods face severe difficulties when the amount of available data is limited and the training instances are contaminated with noise. In particular, the selection of a learning model to describe the observed data is problematic. If the model is too simple, it can fail to describe some of the intrinsic regularities of the data. By contrast, if the model is too complex, it can fit spurious patterns of the training set producing over-fitting. Another problem that arises under these conditions is that the estimators of the model parameters can have large variance. The consequence of this variability is that the resulting estimates may differ significantly from the optimal values. If the amount of data available for induction is large, the severity of these problems is reduced. However, for small datasets it is important to take them into account to obtain reliable predictions.

An effective approach that can be used to alleviate these problems comes from considering simultaneously different models or different values for the model parameters. Given an unlabeled instance, a final decision can then be obtained by averaging the individual responses of the resulting predictors. The combination of the responses of predictors that provide complementary views on the training data has several advantages over machine learning methods based on using a single predictor. In particular, it offers a mechanism to obtain more robust and accurate decisions.

Ensemble methods are a supervised machine learning paradigm that can be used to combine the decisions of different predictors. Therefore, they can be useful to alleviate the problems previously described. Nevertheless, the practical implementation of ensemble methods presents some complications. Specifically, these methods have large storage requirements. The predictors of the ensemble need to be kept in memory so that they are readily accessible. Furthermore, computing the final ensemble decision requires querying every predictor in the ensemble. This increases the final prediction cost. It is also generally difficult to estimate an adequate value for the ensemble size.

The memory requirements and prediction times of ensembles can be improved by using pruning methods. These methods replace the original ensemble by a subensemble with good generalization properties. Because this subensemble is smaller, it requires less storage space than the original ensemble and also has better prediction times. An effective approach to finding subensembles with good generalization properties is to select the subset of predictors with the minimum training error. Nevertheless, the problem of extracting this subensemble is NP-hard. Thus, optimal subensembles defined in these terms can be found in practice only in ensembles of intermediate size. In ensembles of

larger size two approximate methods can be used: ordered aggregation and SDP-pruning. The subensembles identified by these methods need not be optimal. Nevertheless, they share a large fraction of predictors with the optimal subensembles. Furthermore, selected subensembles with only 20% of the initial predictors, besides being smaller than the original ensemble, have better generalization properties than this ensemble. A bias-variance-covariance analysis relate these improvements to the selection of predictors with low bias and small correlations.

Instead of selecting a subensemble with good generalization properties, an alternative pruning method can be used to improve the prediction time of classification ensembles. This method makes inference about the final ensemble prediction after querying only a subset of the classifiers in the ensemble. For this purpose, we introduce a probabilistic framework to describe the majority voting process. This framework is the basis of a novel ensemble pruning method called instance-based (IB) pruning. IB-pruning stops the voting process when there is enough evidence to infer that the majority class label will not be modified by the predictions of the remaining ensemble members. IB-pruning significantly reduces the prediction time of classification ensembles with a negligible deterioration in the generalization performance. Furthermore, IB-pruning tends to give conservative pruning rates: The disagreement rates between the predictions of IB-pruned ensembles and complete ensembles are often below the confidence level specified in the experiments.

The probabilistic framework employed in IB-pruning also provides an effective method to estimate an appropriate size for the ensemble. Specifically, this framework can be employed to infer the size of a classification ensemble so that the resulting ensemble predicts the same class label as an ensemble of infinite size with a specified confidence level. For high confidence levels, the differences between the predictions of this optimal finite ensemble and the predictions of an ensemble of infinite size are very small. Furthermore, the optimal ensemble size is strongly problem dependent: While some classification problems require a few tens of classifiers in the ensemble, others require several thousands. In contrast to other procedures described in the literature, the probabilistic framework considered does not require labeled data to determine the optimal ensemble size.

Besides ensemble methods, another paradigm that can be used to consider different models or different values for the model parameters is Bayesian machine learning. Therefore, it can also be useful to alleviate the problems associated to learning from small training sets which can be furthermore contaminated by random noise. The practical implementation of this learning paradigm also poses some difficult problems. In particular, Bayesian approaches require the evaluation of a posterior distribution that is obtained in terms of complicated integrals in very high dimensions or summations that involve an exponential number of elements. In practice, these calculations are intractable and the posterior distribution has to be approximated using methods such as Markov chain Monte Carlo (MCMC) or type-II maximum likelihood estimation which can be computationally expensive.

The computational efficiency of Bayesian models can be significantly improved using the approximate inference algorithm expectation propagation (EP). This algorithm approximates the posterior distribution of the model using a simple distribution for which the required computations are tractable. In this thesis we have introduced a Bayesian model for binary classification: the Bayes machine. In this model EP is used to approximate the posterior distribution of a parameter that quantifies the level of noise in the labels of the training data. This parameter is difficult to estimate and in previous

studies type-II maximum likelihood is frequently employed for this purpose. Type-II maximum likelihood requires re-training the model multiple times. By contrast, when the EP algorithm is used to compute the approximation, the Bayes machine does not require any re-training for this estimation. In terms of prediction accuracy the Bayes machine is competitive with support vector machines (Vapnik, 1995) and Gaussian process classifiers (Kim and Ghahramani, 2006). Unlike the Bayes machine, these classifiers use type-II maximum likelihood or cross-validation for the estimation of the noise parameter and hence, they have to be re-trained multiple times.

A disadvantage of the Bayes machine is that its training cost is $\mathcal{O}(n^3)$, where n is the number of training instances. Thus, the practical use of this model is limited to small training sets containing only a few hundred instances. The computational cost of the Bayes machine can be reduced by training the model using only a reduced set of d instances called the active set (Lawrence et al., 2003). The active set can be computed using the greedy algorithm proposed in (Lawrence et al., 2003; Seeger, 2003). This algorithm is improved in this thesis by considering additional refining iterations. These extra iterations correct some of the mistakes of the original greedy approach and also improve the posterior approximation. The resulting classifier, called the sparse Bayes machine, has a training cost that scales like $\mathcal{O}(d^2 n)$. When $d \ll n$ this is an important improvement in the time-complexity of the algorithm. Additionally, experiments on several classification problems confirm that the sparse Bayes machine is competitive in terms of prediction accuracy with the standard Bayes machine and the support vector machine.

Another model in which the EP algorithm can be used to carry out approximate inference is the Bayesian model for microarray data classification proposed by Lee et al. (2003). In this model approximate inference has been traditionally implemented using MCMC sampling methods. When EP is used instead of these methods, the cost of training the model is significantly reduced. In particular, the cost of the EP algorithm is $\mathcal{O}(nd)$, where n is the number of training instances and d is the number of attributes. By contrast, the cost of MCMC sampling methods is on average $\mathcal{O}(\rho_0^2 d^3 k)$, where ρ_0 is some small constant and k is of the same order as d. In addition to this speed-up of the training process, the EP algorithm is competitive in terms of prediction accuracy with MCMC and with other state-of-the-art microarray classification techniques. The posterior approximation given by EP is also useful to rank relevant genes for subsequent analysis and this ranking is stable when the training set is slightly modified.

8.1 Future Work

In this section we give some directions for future research on the topics analyzed in this thesis.

- **Ensemble pruning methods:** Future work on pruning regression bagging ensembles includes research on alternative measures to the training error for the identification of subensembles with good generalization properties. For example, the ensemble ambiguity described in (Krogh and Vedelsby, 1995) or the de-correlation function proposed by Rosen (1996). Additionally, it would be interesting to investigate whether ordered aggregation and SDP-pruning are valid for pruning ensembles generated by other methods different from bagging, e.g. negative-correlation learning (Liu and Yao, 1999) or boosting (Drucker, 1997; Friedman, 2001).

A limitation of IB-pruning is that it is restricted to classification ensembles whose elements are generated independently when conditioned to the training data. This method should be extended to address classification ensembles that do not satisfy this restriction. For example, ensembles generated by Adaboost (Freund and Schapire, 1997) or heterogeneous ensembles. Another interesting research work can be the development of IB-pruning methods for regression ensembles.

- **Ensemble size:** One disadvantage of the statistical method proposed in this thesis for the estimation of the optimal ensemble size is that it requires to over-produce and then discard classifiers. Future research on this topic can focus on trying to alleviate this problem. Possible solutions include using out-of-bag samples to perform the estimations (Banfield et al., 2007; Breiman, 1996c, 2001; Hernández-Lobato et al., 2007). Additionally, it can be interesting to combine this method for setting the ensemble size with IB-pruning to get improved classification ensembles.

- **Applications of the EP algorithm:** All the Bayesian models considered in this thesis are restricted to binary classification problems. Thus, the extension of these models to multi-class problems can be considered for future research work. EP can then be used to perform approximate Bayesian inference following the proposal of Kim and Ghahramani (2006). However, possible difficulties include dealing with multi-variate Gaussian integrals and a much larger computational cost. A more challenging problem is to use EP to approximate a posterior distribution for the kernel parameters of the Bayes machine and the sparse Bayes machine.

  The prior distribution for $\boldsymbol{\gamma}$ considered in the Bayesian model for microarray data classification of Lee et al. (2003) is rather simple. It may be possible to extend this model to consider prior distributions with dependences among the different components of $\boldsymbol{\gamma}$ (Li and Li, 2008; Zhu et al., 2009). This prior information can be extracted for example from gene regulatory networks (Gardner and Faith, 2005) or from additional unlabeled data.

  One of the problems of EP is that it is not guaranteed to converge. A possible way of overcoming this limitation is, for example, by making use of double-loop algorithms (Heskes and Zoeter, 2002; Opper and Winther, 2005).

Chapter 9

Conclusiones

Los métodos de aprendizaje automático supervisado tienen que enfrentarse a varios problemas cuando el número de datos de entrenamiento es reducido y éstos se encuentran contaminados por ruido. En particular, la selección de un modelo de aprendizaje no es fácil. Si el modelo es demasiado simple puede que éste no tenga la suficiente capacidad de representación para describir algunas de las regularidades de los datos. Por el contrario, si el modelo es demasiado complejo puede que llegue a aprender patrones espurios del conjunto de entrenamiento produciendo sobreajuste. Otro problema que surge en estas circunstancias es que la varianza de los estimadores de los parámetros del modelo puede ser elevada. La consecuencia de esta variabilidad es que las estimaciones resultantes pueden diferir significativamente de los valores óptimos. Si el número de datos disponibles para realizar las estimaciones es grande, la severidad de estos problemas se reduce. Sin embargo, para conjuntos con pocos datos es importante tenerlos en cuenta para obtener predicciones fiables.

Una estrategia efectiva para hacer frente a estos problemas consiste en considerar simultáneamente diferentes modelos o diferentes valores para los parámetros del modelo. Dado un dato sin etiquetar, la decisión final se puede obtener promediando sobre las respuestas individuales de los predictores resultantes. La combinación de las respuestas de predictores que proporcionan puntos de vista complementarios sobre los datos de entrenamiento presenta ciertas ventajas frente a los métodos de aprendizaje automático que consideran un solo predictor. En particular, ofrece un mecanismo para obtener decisiones más robustas y certeras.

Los métodos basados en conjuntos son un paradigma del aprendizaje automático supervisado que puede ser usado para combinar las decisiones de varios predictores. Por lo tanto, estos métodos pueden ser útiles para aliviar los problemas descritos previamente. Sin embargo, la implementación práctica de los métodos basados en conjuntos presenta ciertas complicaciones. Específicamente, los requisitos de almacenamiento de estos métodos son elevados. Los elementos del conjunto necesitan ser almacenados en memoria para ser accesibles rápidamente. Además, el cálculo de la decisión final del conjunto requiere obtener la predicción de todos y cada uno de los elementos del conjunto. Esto incrementa el coste de predicción. En general, también resulta difícil estimar un valor adecuado para el tamaño del conjunto.

Los requisitos de memoria y el tiempo de predicción de los métodos basados en conjuntos se pueden mejorar utilizando métodos de poda. Estos métodos reemplazan el conjunto original por un subconjunto con buenas propiedades de generalización. Debido

a que el subconjunto resultante tiene un tamaño menor, éste requiere menos espacio de almacenamiento y tiene además mejores tiempos de predicción que el conjunto original. Un enfoque efectivo para encontrar subconjuntos con buenas propiedades de generalización consiste en seleccionar el subconjunto con el mínimo error sobre los datos de entrenamiento. Sin embargo, encontrar este subconjunto es un problema NP-difícil. Por ello, los subconjuntos óptimos definidos en estos términos sólo pueden ser encontrados en conjuntos de tamaño intermedio. En conjuntos de mayor tamaño se pueden usar métodos aproximados: agregación ordenada y poda SDP. Los subconjuntos identificados por estos métodos no son necesariamente óptimos. Sin embargo, comparten una gran fracción de elementos con los subconjuntos óptimos. Además, subconjuntos compuestos únicamente por un 20% de los elementos originales, aparte de ser más pequeños que el conjunto original, tienen mejores propiedades de generalización. Una descomposición del error de generalización en términos de sesgo, varianza y covarianza muestra que estas mejoras son consecuencia de la selección de predictores poco correlacionados (baja covarianza) y con poco sesgo.

En lugar de seleccionar un subconjunto con buenas propiedades de generalización, se puede usar un método de poda alternativo para mejorar el tiempo de predicción en los conjuntos de clasificadores. Este método consiste en hacer inferencia sobre la predicción final del conjunto tras preguntar a un subconjunto del total de clasificadores del conjunto. Para ello, introducimos un marco probabilístico que describe el proceso de voto por mayoría. Este marco probabilístico es la base de un nuevo método de poda basada en instancia (poda BI). La poda BI detiene el proceso de votación cuando hay suficiente evidencia para inferir que la clase mayoritaria no será modificada por las restantes predicciones de los elementos del conjunto. La poda BI reduce ampliamente el tiempo de predicción de los conjuntos de clasificadores a costa de un pequeño deterioro del error de generalización. Además, la poda BI tiende a proporcionar tasas de poda conservadoras: la tasa de desacuerdo entre las predicciones de los conjuntos podados y los conjuntos completos se encuentra generalmente por debajo del nivel de confianza especificado en los experimentos.

El marco probabilístico empleado en la poda BI también proporciona un método eficaz para estimar un tamaño adecuado para el conjunto. En particular, este marco probabilístico se puede emplear para inferir el tamaño de un conjunto de clasificadores tal que el conjunto resultante prediga con un nivel de confianza especificado la misma etiqueta de clase que un conjunto de tamaño infinito. Para niveles de confianza elevados, las diferencias entre las predicciones de este conjunto óptimo de tamaño finito y las predicciones de un conjunto de tamaño infinito son muy pequeñas. Además, el tamaño óptimo del conjunto resulta ser muy distinto dependiendo del problema considerado. Mientras que algunos problemas de clasificación requieren conjuntos con sólo unos cuantos clasificadores, otros requieren conjuntos con varios miles. Al contrario que otros métodos descritos en la literatura, el marco probabilístico considerado en esta tesis no requiere disponer de datos etiquetados para determinar el tamaño óptimo del conjunto.

Otro paradigma que puede ser utilizado para considerar varios modelos o varios valores para los parámetros del modelo es el aprendizaje automático Bayesiano. Por lo tanto, los métodos Bayesianos también pueden ser utilizados para aliviar los problemas que surgen del aprendizaje a partir de conjuntos de entrenamiento reducidos que pueden estar contaminados por ruido aleatorio. La implementación del los métodos Bayesianos también presenta ciertos problemas de tipo práctico. Concretamente, este paradigma requiere la evaluación de una distribución posterior que se obtiene mediante complicadas integrales multidimensionales o mediante sumas que involucran un número exponencial

de términos. En la práctica, estos cálculos son intratables y la distribución posterior ha de ser aproximada utilizando métodos tales como las cadenas de Markov o la máxima verosimilitud de tipo II. Estos métodos pueden resultar costosos computacionalmente.

La eficiencia computacional de los modelos Bayesianos se puede mejorar significativamente utilizando el algoritmo de propagación de esperanzas (PE). Este algoritmo aproxima la distribución posterior del modelo mediante una distribución sencilla para la que los cálculos requeridos son tratables. En esta tesis se ha introducido un modelo Bayesiano para la clasificación binaria: la máquina de Bayes. En este modelo el algoritmo PE se puede utilizar para aproximar la distribución posterior de un parámetro que cuantifica el nivel de ruido en las etiquetas de clase de los datos de entrenamiento. Este parámetro es difícil de estimar y en métodos propuestos con anterioridad la máxima verosimilitud de tipo II se utiliza frecuentemente para este propósito. La máxima verosimilitud de tipo II tiene el inconveniente de que necesita re-entrenar el modelo múltiples veces para estimar dicho parámetro. Por el contrario, cuando el algoritmo PE se utiliza para calcular la aproximación, la máquina de Bayes no necesita volver a ser entrenada. En términos de error de generalización la máquina de Bayes es competitiva con las máquinas de vectores soporte (Vapnik, 1995) y los procesos Gaussianos para clasificación (Kim and Ghahramani, 2006). A diferencia de la máquina de Bayes, estos clasificadores usan máxima verosimilitud de tipo II o validación cruzada para estimar el parámetro del ruido y por lo tanto necesitan ser entrenados de manera repetida.

Una desventaja de la máquina de Bayes es que su coste computacional es $\mathcal{O}(n^3)$, donde n es el número de datos de entrenamiento. Por esta razón, el uso práctico de este modelo está limitado a conjuntos de entrenamiento pequeños que contengan sólo algunos cientos de datos. El coste computacional de la máquina de Bayes se puede reducir entrenando el modelo usando solamente un conjunto reducido de d datos llamado el conjunto activo (Lawrence et al., 2003). El conjunto activo se puede calcular usando el algoritmo codicioso propuesto en (Lawrence et al., 2003; Seeger, 2003). Este algoritmo es mejorado en esta tesis mediante la consideración de iteraciones de refinamiento adicionales. Estas iteraciones adicionales corrigen algunos de los errores del enfoque codicioso original y además mejoran la aproximación a la distribución posterior. El clasificador resultante, llamado máquina de Bayes dispersa, tiene un coste de entrenamiento $\mathcal{O}(d^2 n)$. Cuando $d \ll n$ esta es una mejora importante en la complejidad del algoritmo. Los resultados de experimentos en varios problemas de clasificación confirman que la máquina de Bayes dispersa es competitiva en términos de error de generalización con la máquina de Bayes estándar y con la máquina de vectores soporte.

Otro modelo en el que el algoritmo PE se puede utilizar para llevar a cabo inferencia aproximada es el modelo Bayesiano para la clasificación de datos de microarray propuesto por Lee et al. (2003). En este modelo la inferencia aproximada se ha implementado tradicionalmente utilizando métodos Monte Carlo basados en la simulación de cadenas de Markov. Cuando se utiliza el algoritmo PE en lugar de estos métodos el coste de entrenamiento del modelo se reduce significativamente. En particular, el coste del algoritmo PE es $\mathcal{O}(nd)$, donde n es el número de datos de entrenamiento y d es el número de atributos. Por el contrario, el coste de los métodos Monte Carlo es en promedio $\mathcal{O}(\rho_0^2 d^3 k)$, donde ρ_0 es una constante pequeña y k es del mismo orden que d. Además de esta mejora en el tiempo de entrenamiento, el algoritmo PE es competitivo en términos de error de generalización con los métodos Monte Carlo y con otras técnicas representativas para la clasificación de datos de microarray. La aproximación a la distribución posterior facilitada por el algoritmo PE es también útil para calcular

una ordenación de genes relevantes para su posterior análisis. Esta ordenación es estable cuando los datos de entrenamiento son ligeramente modificados.

Appendix A

Probability Distributions

A.1 Bernoulli

This is the distribution of a single binary variable $x \in \{0, 1\}$. It is governed by a single continuous parameter $p \in [0, 1]$ that represents the probability of $x = 1$. The probability mass function of a Bernoulli random variable is

$$\text{Bern}(x|p) = p^x(1-p)^{1-x}\,. \tag{A.1}$$

The Bernoulli distribution belongs to the exponential family of probability distributions as (A.1) can also be written as

$$\text{Bern}(x|p) = \exp(\eta u(x) - g(\eta))\,, \tag{A.2}$$

with the definitions

$$g(\eta) = \log\left(1 + \exp(\eta)\right)\,, \qquad u(x) = x\,, \qquad \eta = \log\left(\frac{p}{1-p}\right)\,. \tag{A.3}$$

The mean and variance of x are respectively p and $p(1-p)$. The natural moment is given by the expectation of $u(x)$. This expectation is equal to $dg(\eta)/d\eta$. In consequence,

$$\mathbb{E}[x] = \frac{1}{1+\exp(-\eta)} = p\,. \tag{A.4}$$

Let $t(x)$ be an arbitrary function of x and let

$$Z = \sum_x t(x)\text{Bern}(x|p)\,, \qquad \hat{\mathcal{P}}(x) = \frac{1}{Z}t(x)\text{Bern}(x|p)\,. \tag{A.5}$$

Then, we have that

$$\mathbb{E}_{\hat{\mathcal{P}}}[x] = \frac{\partial \log Z}{\partial p}p(1-p) + p\,. \tag{A.6}$$

Because of the closure property of the exponential family of probability distributions, the product of two Bernoulli distributions is another Bernoulli distribution, although no

longer normalized. In particular,

$$\text{Bern}(x|p_1)\text{Bern}(x|p_2) \propto \text{Bern}(x|p)\,, \tag{A.7}$$

where

$$p = \frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)} \tag{A.8}$$

and the normalization constant z of the product is

$$z = p_1 p_2 + (1-p_1)(1-p_2)\,. \tag{A.9}$$

Similarly, the quotient of two Bernoulli distributions is another Bernoulli distribution, although no longer normalized

$$\text{Bern}(x|p_1)/\text{Bern}(x|p_2) \propto \text{Bern}(x|p)\,, \tag{A.10}$$

where

$$p = \frac{p_1/p_2}{p_1/p_2 + (1-p_1)/(1-p_2)} \tag{A.11}$$

and the normalization constant z is in this case

$$z = p_1/p_2 + (1-p_1)/(1-p_2)\,. \tag{A.12}$$

A.2 Beta

This is the distribution of a continuous variable $x \in [0,1]$. It is governed by two parameters a and b that are constrained by $a > 0$ and $b > 0$ to ensure that the distribution can be normalized. The probability density function of a beta random variable is

$$\text{Beta}(x|a,b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1}\,, \tag{A.13}$$

where $\Gamma(x)$ is the gamma function (Abramowitz and Stegun, 1964). The cumulative probability function is

$$\int_0^z \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1} dx = I_z(a,b)\,, \tag{A.14}$$

where $I_z(a,b)$ is the regularized incomplete beta function (Abramowitz and Stegun, 1964). The beta distribution belongs to the exponential family of probability distribution as (A.13) can also be written as

$$\text{Beta}(x|a,b) = \exp(\boldsymbol{\eta}^T \mathbf{u}(x) - g(\boldsymbol{\eta}))\,, \tag{A.15}$$

where

$$\begin{aligned} g(\boldsymbol{\eta}) &= \log\left(\Gamma(\boldsymbol{\eta}^T\boldsymbol{\delta}_a + 1)\right) + \log\left(\Gamma(\boldsymbol{\eta}^T\boldsymbol{\delta}_b + 1)\right) \\ &\quad - \log\left(\Gamma(\boldsymbol{\eta}^T\mathbf{1} + 2)\right)\,, \qquad \text{(A.16)} \\ \mathbf{u}(x) &= (\log(x), \log(1-x))^T\,, \qquad \text{(A.17)} \\ \boldsymbol{\eta} &= (a-1, b-1)^T\,, \qquad \text{(A.18)} \end{aligned}$$

$\boldsymbol{\delta}_a = (1,0)^T$, $\boldsymbol{\delta}_b = (0,1)^T$ and the superscript T means transpose. The first and second moments of x are given by

$$\mathbb{E}[x] = \frac{a}{a+b}\,, \tag{A.19}$$

$$\mathbb{E}[x^2] = \frac{a(a+1)}{(a+b)(a+b+1)}\,. \tag{A.20}$$

The natural moments are given by the expectation of $\mathbf{u}(x)$ under (A.13). This expectation is equal to $\partial g(\boldsymbol{\eta})/\partial \boldsymbol{\eta}$. In consequence,

$$\mathbb{E}[\log(x)] = \Psi(a) - \Psi(a+b)\,, \tag{A.21}$$

$$\mathbb{E}[\log(1-x)] = \Psi(b) - \Psi(a+b)\,, \tag{A.22}$$

where Ψ is the digamma function defined as $\Psi(x) = d\log(\Gamma(x))/dx$ (Abramowitz and Stegun, 1964).

Let x bet distributed according to $\mathrm{Beta}(x|a,b)$ and y according to $\mathrm{Beta}(x|a',b')$. Then, the Kullback-Leibler divergence between these two distributions is

$$\begin{aligned}\mathrm{KL}\,(x,y) = \log\left(\frac{\beta(a',b')}{\beta(a,b)}\right) - (a'-a)\Psi(a) \\ - (b'-b)\Psi(b) + (a'-a+b'-b)\Psi(a+b)\,,\end{aligned} \tag{A.23}$$

where $\beta(a,b)$ is the beta function defined as $\Gamma(a)\Gamma(b)/\Gamma(a+b)$ (Abramowitz and Stegun, 1964).

Let $t(x)$ be an arbitrary function of x and let

$$Z = \int t(x)\mathrm{Beta}(x|a,b)dx\,, \qquad \hat{\mathcal{P}}(x) = \frac{1}{Z}t(x)\mathrm{Beta}(x|a,b)\,. \tag{A.24}$$

Then, we have that

$$\mathbb{E}_{\hat{\mathcal{P}}}[\log(x)] = \Psi(a) - \Psi(a+b) + \frac{\partial \log(Z)}{\partial a}\,, \tag{A.25}$$

$$\mathbb{E}_{\hat{\mathcal{P}}}[\log(1-x)] = \Psi(b) - \Psi(a+b) + \frac{\partial \log(Z)}{\partial b}\,. \tag{A.26}$$

Because of the closure property of the exponential family of probability distributions, the product of two beta distributions is another beta distribution, although no longer normalized. In particular,

$$\mathrm{Beta}(x|a_1,b_1)\mathrm{Beta}(x|a_2,b_2) \propto \mathrm{Beta}(x|a,b)\,, \tag{A.27}$$

where

$$a = a_1 + a_2 - 1\,, \qquad b = b_1 + b_2 - 1\,. \tag{A.28}$$

The normalization constant z of the product is

$$z = \frac{\beta(a_1+a_2-1, b_1+b_2-1)}{\beta(a_1,b_1)\beta(a_2,b_2)}\,. \tag{A.29}$$

where $\beta(a,b) = \Gamma(a)\Gamma(b)/\Gamma(a+b)$. Similarly, the quotient of two beta distributions is another beta distribution. Namely,

$$\text{Beta}(x|a_1,b_1)/\text{Beta}(x|a_2,b_2) \propto \text{Beta}(x|a,b)\,, \tag{A.30}$$

where

$$a = a_1 - a_1 + 1\,, \qquad b = b_1 - b_2 + 1\,, \tag{A.31}$$

and the normalization constant z is in this case

$$z = \frac{\beta(a_1 - a_2 + 1, b_1 - b_2 + 1)\beta(a_2,b_2)}{\beta(a_1,b_1)}\,. \tag{A.32}$$

A.3 Gaussian

This is the distribution of a continuous d-dimensional vector $\mathbf{x} \in \mathbb{R}^d$. It is governed by a d-dimensional mean vector $\boldsymbol{\mu}$ and a $d \times d$ covariance matrix $\boldsymbol{\Sigma}$ that must be symmetric and positive-definite. The probability density function of $\mathbf{x}$ is given by

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}\,, \tag{A.33}$$

where the superscript T means transpose. The Gaussian distribution belongs to the exponential family of probability distributions as (A.33) can also be written as

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \exp(\boldsymbol{\eta}^T\mathbf{u}(\mathbf{x}) - g(\boldsymbol{\eta}))\,, \tag{A.34}$$

where

$$\begin{aligned}
\mathbf{u}(\mathbf{x}) = (&x_1,\ldots,x_d,\\
&x_1^2, x_1x_2,\ldots,x_1x_d,\\
&x_1x_2, x_2^2,\ldots,x_2x_d,\\
&\vdots\\
&x_dx_1, x_dx_2,\ldots,x_d^2)^T\,,
\end{aligned}
\qquad
\begin{aligned}
\boldsymbol{\eta} = -\frac{1}{2}(&-2\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\,,\\
&\Sigma_{11}^{-1},\Sigma_{12}^{-1},\ldots,\Sigma_{1d}^{-1},\\
&\Sigma_{21}^{-1},\Sigma_{22}^{-1},\ldots,\Sigma_{1d}^{-1},\\
&\vdots\\
&\Sigma_{d1}^{-1},\Sigma_{d2}^{-1},\ldots,\Sigma_{dd}^{-1})^T
\end{aligned}
\tag{A.35}$$

and $g(\boldsymbol{\eta}) = \frac{1}{2}\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \frac{d}{2}\log(2\pi) + \frac{1}{2}\log(|\boldsymbol{\Sigma}|)$. The natural moments are given by the expectation of $\mathbf{u}(\mathbf{x})$ under (A.33). In particular,

$$\mathbb{E}[\mathbf{x}] = \boldsymbol{\mu}\,, \tag{A.36}$$

$$\mathbb{E}[\mathbf{x}\mathbf{x}^T] = \boldsymbol{\Sigma} + \boldsymbol{\mu}\boldsymbol{\mu}^T\,. \tag{A.37}$$

Let $\mathbf{x}$ bet distributed according to $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma})$ and $\mathbf{y}$ according to $\mathcal{N}(\mathbf{y}|\boldsymbol{\mu}',\boldsymbol{\Sigma}')$. Then, the Kullback-Leibler divergence between these two distributions is

$$\text{KL}\,(\mathbf{x},\mathbf{y}) = \frac{1}{2}\left(\frac{|\boldsymbol{\Sigma}'|}{|\boldsymbol{\Sigma}|}\right) + \text{trace}\left(\boldsymbol{\Sigma}'^{-1}\boldsymbol{\Sigma} - \mathbf{I}\right) + (\boldsymbol{\mu}'-\boldsymbol{\mu})^T\boldsymbol{\Sigma}'^{-1}(\boldsymbol{\mu}'-\boldsymbol{\mu})\,. \tag{A.38}$$

Let $t(\mathbf{x})$ be an arbitrary function of $\mathbf{x}$ and let

$$Z = \int t(\mathbf{x})\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})d\mathbf{x}\,, \qquad \hat{\mathcal{P}}(\mathbf{x}) = \frac{1}{Z}t(\mathbf{x})\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\,. \tag{A.39}$$

Then, we have that

$$\mathbb{E}_{\hat{\mathcal{P}}}[\mathbf{x}] = \boldsymbol{\mu} + \boldsymbol{\Sigma}\frac{\partial \log(Z)}{\partial \boldsymbol{\mu}}\,, \tag{A.40}$$

$$\mathbb{E}_{\hat{\mathcal{P}}}[\mathbf{x}\mathbf{x}^T] - \mathbb{E}_{\hat{\mathcal{P}}}[\mathbf{x}]\mathbb{E}_{\hat{\mathcal{P}}}[\mathbf{x}]^T = \boldsymbol{\Sigma} - \boldsymbol{\Sigma}\left(\frac{\partial \log(Z)}{\partial \boldsymbol{\mu}}\left(\frac{\partial \log(Z)}{\partial \boldsymbol{\mu}}\right)^T - 2\frac{\partial \log(Z)}{\partial \boldsymbol{\Sigma}}\right)\boldsymbol{\Sigma}\,. \tag{A.41}$$

Because of the closure property of the exponential family of probability distributions, the product of two Gaussians is another Gaussian function, although no longer normalized. In particular,

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) \propto \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\,, \tag{A.42}$$

where

$$\boldsymbol{\Sigma} = \left(\boldsymbol{\Sigma}_1^{-1} + \boldsymbol{\Sigma}_2^{-1}\right)^{-1}\,, \tag{A.43}$$

$$\boldsymbol{\mu} = \boldsymbol{\Sigma}\left(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2\right)\,. \tag{A.44}$$

The normalization constant z of the product is given by

$$z = \sqrt{\frac{|\boldsymbol{\Sigma}|}{(2\pi)^d|\boldsymbol{\Sigma}_1||\boldsymbol{\Sigma}_2|}}\exp\left\{-\frac{1}{2}\left(\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2^T\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2 - \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right)\right\}\,, \tag{A.45}$$

where d is the dimensionality of the data. Similarly, the quotient of two Gaussians is also Gaussian. Namely,

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)/\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) \propto \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\,, \tag{A.46}$$

where

$$\boldsymbol{\Sigma} = \left(\boldsymbol{\Sigma}_1^{-1} - \boldsymbol{\Sigma}_2^{-1}\right)^{-1}\,, \tag{A.47}$$

$$\boldsymbol{\mu} = \boldsymbol{\Sigma}\left(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2\right)\,. \tag{A.48}$$

In this case, the normalization constant z is given by

$$z = \sqrt{\frac{(2\pi)^d|\boldsymbol{\Sigma}||\boldsymbol{\Sigma}_2|}{|\boldsymbol{\Sigma}_1|}}\exp\left\{-\frac{1}{2}\left(\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2^T\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2 - \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right)\right\}\,. \tag{A.49}$$

Appendix B

Appendix for Chapter 6

B.1 Woodbury Formula

Let $\mathbf{A}$ be a square $n \times n$ invertible matrix and $\mathbf{U}$ and $\mathbf{V}$ two $n \times k$ matrices with $k \leq n$ and β an arbitrary scalar. Then,

$$\left(\mathbf{A} + \beta \mathbf{U}\mathbf{V}^T\right)^{-1} = \mathbf{A}^{-1} - \beta \mathbf{A}^{-1}\mathbf{U}\left(\mathbf{I} + \beta \mathbf{V}^T\mathbf{A}^{-1}\mathbf{U}\right)^{-1}\mathbf{V}^T\mathbf{A}^{-1}\,, \tag{B.1}$$

where $\mathbf{I}$ is the identity matrix.

B.2 Special form of the vector $\mathbf{m}_i$

Recall that $\tilde{t}_i = Z_i \mathcal{Q}^{\text{new}}/\mathcal{Q}^{\backslash i}$. In this section we show that there are many candidate vectors $\mathbf{m}_i$ that can be used as the mean parameter of the approximate term $\tilde{t}_i$ defined in (6.13). Because of (A.46), these vectors satisfy

$$\begin{aligned}
\mathbf{V}_i^{-1}\mathbf{m}_i &= (\mathbf{V}_{\mathbf{w}}^{\text{new}})^{-1}\mathbf{m}_{\mathbf{w}}^{\text{new}} - (\mathbf{V}_{\mathbf{w}}^{\backslash i})^{-1}\mathbf{m}_{\mathbf{w}}^{\backslash i} \\
\mathbf{V}_i^{-1}\mathbf{m}_i &= \left((\mathbf{V}_{\mathbf{w}}^{\backslash i})^{-1} + \mathbf{V}_i^{-1}\right)\left(\mathbf{m}_{\mathbf{w}}^{\backslash i} + \alpha_i \mathbf{V}_{\mathbf{w}}^{\backslash i}\mathbf{x}_i\right) - (\mathbf{V}_{\mathbf{w}}^{\backslash i})^{-1}\mathbf{m}_{\mathbf{w}}^{\backslash i} \\
v_i^{-1}\mathbf{x}_i\mathbf{x}_i^T\mathbf{m}_i &= \alpha_i\mathbf{x}_i + v_i^{-1}\mathbf{x}_i\mathbf{x}_i^T\mathbf{m}_{\mathbf{w}}^{\backslash i} + \alpha_i v_i^{-1}\mathbf{x}_i\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\backslash i}\mathbf{x}_i \\
\mathbf{m}_i &= \mathbf{m}_{\mathbf{w}}^{\backslash i} + \alpha_i \mathbf{V}_{\mathbf{w}}^{\backslash i}\mathbf{x}_i + \alpha_i v_i \boldsymbol{\zeta} + K\boldsymbol{\xi}\,,
\end{aligned} \tag{B.2}$$

where we have used (6.26), $(\mathbf{V}_{\mathbf{w}}^{\text{new}})^{-1} = (\mathbf{V}_{\mathbf{w}}^{\backslash i})^{-1} + \mathbf{V}_i^{-1}$, which can be derived from (6.35), and (6.42). In (B.2) $\boldsymbol{\zeta}$ is any vector such that $\mathbf{x}_i^T\boldsymbol{\zeta} = 1$, $\boldsymbol{\xi}$ is any vector such that $\mathbf{x}_i^T\boldsymbol{\xi} = 0$ and K is any arbitrary constant.

B.3 Derivation of s_i

In this section we show how to evaluate s_i, as defined in (6.39), using the representation of $\tilde{t}_i$ given in (6.45). Because $(\mathbf{V}_{\mathbf{w}}^{\text{new}})^{-1} = (\mathbf{V}_{\mathbf{w}}^{\backslash i})^{-1} + \mathbf{V}_i^{-1}$, which can be derived from (6.35), we have that

$$\frac{\left|\mathbf{V}_{\mathbf{w}}^{\backslash i}\right|}{\left|\mathbf{V}_{\mathbf{w}}^{\text{new}}\right|} = \left|\mathbf{I} + \mathbf{V}_i^{-1}\mathbf{V}_{\mathbf{w}}^{\backslash i}\right| = \left|\mathbf{I} + v_i^{-1}\mathbf{x}_i\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\backslash i}\right| = 1 + v_i^{-1}\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\backslash i}\mathbf{x}_i\,, \tag{B.3}$$

where $\mathbf{I}$ is the identity matrix and where we have used that $|\mathbf{I} + \mathbf{A}\mathbf{B}^T| = |\mathbf{I} + \mathbf{A}^T\mathbf{B}|$ (Bishop, 2006). Next, we compute

$$\begin{aligned}(\mathbf{m}_{\mathbf{w}}^{\text{new}})^T(\mathbf{V}_{\mathbf{w}}^{\text{new}})^{-1}\mathbf{m}_{\mathbf{w}}^{\text{new}} &= \left(\mathbf{m}_{\mathbf{w}}^{\setminus i} + \alpha_i \mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i\right)^T \left((\mathbf{V}_{\mathbf{w}}^{\setminus i})^{-1} + \mathbf{V}_i^{-1}\right)\left(\mathbf{m}_{\mathbf{w}}^{\setminus i} + \alpha_i \mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i\right) \\ &= (\mathbf{m}_{\mathbf{w}}^{\setminus i})^T(\mathbf{V}_{\mathbf{w}}^{\setminus i})^{-1}\mathbf{m}_{\mathbf{w}}^{\setminus i} + (\mathbf{m}_{\mathbf{w}}^{\setminus i})^T\mathbf{V}_i^{-1}\mathbf{m}_{\mathbf{w}}^{\setminus i} + 2\alpha_i\mathbf{x}_i^T\mathbf{m}_{\mathbf{w}}^{\setminus i} + \\ &\quad 2\alpha_i\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{V}_i^{-1}\mathbf{m}_{\mathbf{w}}^{\setminus i} + \alpha_i^2\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i + \alpha_i^2\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{V}_i^{-1}\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i \\ &= (\mathbf{m}_{\mathbf{w}}^{\setminus i})^T(\mathbf{V}_{\mathbf{w}}^{\setminus i})^{-1}\mathbf{m}_{\mathbf{w}}^{\setminus i} + v_i^{-1}(\mathbf{m}_{\mathbf{w}}^{\setminus i})^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{m}_{\mathbf{w}}^{\setminus i} + 2\alpha_i\mathbf{x}_i^T\mathbf{m}_{\mathbf{w}}^{\setminus i} + \\ &\quad 2v_i^{-1}\alpha_i\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i\mathbf{x}_i^T\mathbf{m}_{\mathbf{w}}^{\setminus i} + \alpha_i^2\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i + \\ &\quad v_i^{-1}\left(\alpha_i\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i\right)^2, \end{aligned} \tag{B.4}$$

where we have used that $(\mathbf{V}_{\mathbf{w}}^{\text{new}})^{-1} = (\mathbf{V}_{\mathbf{w}}^{\setminus i})^{-1} + \mathbf{V}_i^{-1}$, (6.26) and (6.42). Furthermore,

$$\begin{aligned}\mathbf{m}_i\mathbf{V}_i^{-1}\mathbf{m}_i &= \left(\mathbf{m}_{\mathbf{w}}^{\setminus i} + \alpha_i\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i + \alpha_i v_i\boldsymbol{\zeta} + K\boldsymbol{\xi}\right)^T\mathbf{V}_i^{-1}\left(\mathbf{m}_{\mathbf{w}}^{\setminus i} + \alpha_i\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i + \alpha_i v_i\boldsymbol{\zeta} + K\boldsymbol{\xi}\right) \\ &= (\mathbf{m}_{\mathbf{w}}^{\setminus i})^T\mathbf{V}_i^{-1}\mathbf{m}_{\mathbf{w}}^{\setminus i} + 2\alpha_i(\mathbf{m}_{\mathbf{w}}^{\setminus i})^T\mathbf{V}_i^{-1}\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i + 2\alpha_i v_i(\mathbf{m}_{\mathbf{w}}^{\setminus i})^T\mathbf{V}_i^{-1}\boldsymbol{\zeta} + \\ &\quad 2K(\mathbf{m}_{\mathbf{w}}^{\setminus i})^T\mathbf{V}_i^{-1}\boldsymbol{\xi} + \alpha_i^2\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{V}_i^{-1}\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i + \alpha_i^2 v_i^2\boldsymbol{\zeta}^T\mathbf{V}_i^{-1}\boldsymbol{\zeta} + K^2\boldsymbol{\xi}^T\mathbf{V}_i^{-1}\boldsymbol{\xi} + \\ &\quad 2\alpha_i^2 v_i\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{V}_i^{-1}\boldsymbol{\zeta} + 2\alpha_i v_i K\boldsymbol{\zeta}^T\mathbf{V}_i^{-1}\boldsymbol{\xi} \\ &= v_i^{-1}(\mathbf{m}_{\mathbf{w}}^{\setminus i})^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{m}_{\mathbf{w}}^{\setminus i} + 2v_i^{-1}\alpha_i(\mathbf{m}_{\mathbf{w}}^{\setminus i})^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i + 2\alpha_i(\mathbf{m}_{\mathbf{w}}^{\setminus i})^T\mathbf{x}_i + \\ &\quad v_i^{-1}\left(\alpha_i\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i\right)^2 + \alpha_i^2 v_i + 2\alpha_i^2\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i, \end{aligned} \tag{B.5}$$

where we have used (B.2) and (6.42). With these values, i.e. (B.4) and (B.5), we have that

$$\begin{aligned}(\mathbf{m}_{\mathbf{w}}^{\text{new}})^T(\mathbf{V}_{\mathbf{w}}^{\text{new}})^{-1}\mathbf{m}_{\mathbf{w}}^{\text{new}} - (\mathbf{m}_{\mathbf{w}}^{\setminus i})^T(\mathbf{V}_{\mathbf{w}}^{\setminus i})^{-1}\mathbf{m}_{\mathbf{w}}^{\setminus i} - \mathbf{m}_i\mathbf{V}_i^{-1}\mathbf{m}_i &= \alpha_i^2\mathbf{x}_i\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i - \alpha_i^2 v_i \\ &= -\frac{\alpha_i\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i}{\mathbf{x}_i^T\mathbf{m}_{\mathbf{w}}^{\text{new}}}, \end{aligned} \tag{B.6}$$

where we have used the definition of v_i given in (6.43). In consequence,

$$s_i = Z_i\frac{\beta(a_\epsilon^{\setminus i}, b_\epsilon^{\setminus i})}{\beta(a_\epsilon^{\text{new}}, b_\epsilon^{\text{new}})}\sqrt{1 + v_i^{-1}\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i}\exp\left(\frac{1}{2}\frac{\alpha_i\mathbf{x}_i^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_i}{\mathbf{x}_i^T\mathbf{m}_{\mathbf{w}}^{\text{new}}}\right). \tag{B.7}$$

B.4 Derivation of $\lambda_j^{\setminus i}$ and $h_j^{\setminus i}$

In this section we derive the values of $\lambda_j^{\setminus i}$ and $h_j^{\setminus i}$, defined as

$$\lambda_j^{\setminus i} = \mathbf{x}_j^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_j, \tag{B.8}$$

$$h_j^{\setminus i} = \mathbf{x}_j^T\mathbf{m}_{\mathbf{w}}^{\setminus i}. \tag{B.9}$$

The standard BM only requires the computation of $\lambda_j^{\setminus i}$ and $h_j^{\setminus i}$ with $j = i$. However, in the sparse representation of the BM $\lambda_j^{\setminus i}$ and $h_j^{\setminus i}$ are required for $j \neq i$. Thus, in this

section we derive the general expressions for (B.8) and (B.9). From these, $\lambda_i^{\setminus i}$ and $h_i^{\setminus i}$ can be computed by setting $j = i$

$$\begin{aligned}
\lambda_j^{\setminus i} &= \mathbf{x}_j^T \mathbf{V}_{\mathbf{w}}^{\setminus i} \mathbf{x}_j \\
&= \mathbf{x}_j^T \left(\mathbf{V}_{\mathbf{w}}^{-1} - v_i^{-1} \mathbf{x}_i \mathbf{x}_i^T \right)^{-1} \mathbf{x}_j \\
&= \mathbf{x}_j^T \left(\mathbf{V}_{\mathbf{w}} + v_i^{-1} \mathbf{V}_{\mathbf{w}} \mathbf{x}_i (1 - v_i^{-1} \mathbf{x}_i^T \mathbf{V}_{\mathbf{w}} \mathbf{x}_i)^{-1} \mathbf{x}_i^T \mathbf{V}_{\mathbf{w}} \right) \mathbf{x}_j \\
&= A_{jj} + v_i^{-1} \frac{A_{ji}^2}{1 - v_i^{-1} A_{ii}} \\
&= A_{jj} + \frac{A_{ji}^2}{v_i - A_{ii}} \,, \qquad \text{(B.10)}
\end{aligned}$$

where we have used (6.47), the Woodbury formula (B.1) and (6.57). Furthermore,

$$\begin{aligned}
h_j^{\setminus i} &= \mathbf{x}_j^T \mathbf{m}_{\mathbf{w}}^{\setminus i} \\
&= \mathbf{x}_j^T \left(\mathbf{m}_{\mathbf{w}} + v_i^{-1} \mathbf{V}_{\mathbf{w}}^{\setminus i} \mathbf{x}_i (h_i - m_i) \right) \\
&= \mathbf{x}_j^T \left(\mathbf{m}_{\mathbf{w}} + v_i^{-1} \left(\mathbf{V}_{\mathbf{w}}^{-1} - v_i^{-1} \mathbf{x}_i \mathbf{x}_i^T \right)^{-1} \mathbf{x}_i (h_i - m_i) \right) \\
&= \mathbf{x}_j^T \left(\mathbf{m}_{\mathbf{w}} + v_i^{-1} \left(\mathbf{V}_{\mathbf{w}} + v_i^{-1} \frac{\mathbf{V}_{\mathbf{w}} \mathbf{x}_i \mathbf{x}_i^T \mathbf{V}_{\mathbf{w}}}{1 - v_i^{-1} \mathbf{x}_i^T \mathbf{V}_{\mathbf{w}} \mathbf{x}_i} \right) \mathbf{x}_i (h_i - m_i) \right) \\
&= h_j + \frac{A_{ji}}{v_i - A_{ii}} (h_i - m_i) \,, \qquad \text{(B.11)}
\end{aligned}$$

where we have used (6.47), (6.48), the Woodbury formula (B.1) and (6.58).

B.5 Update of the matrix A and the vector h in the BM

In this section we show how to update the matrix $\mathbf{A}$ and the vector $\mathbf{h}$ after each iteration of the EP algorithm in the BM. From the definition of the matrix $\mathbf{A}$ in (6.57)

$$\begin{aligned}
\mathbf{A} &= \mathbf{X}^T \mathbf{V}_{\mathbf{w}} \mathbf{X} \\
&= \mathbf{X}^T \left(\mathbf{I} + \mathbf{X} \boldsymbol{\Lambda}^{-1} \mathbf{X}^T \right)^{-1} \mathbf{X} \\
&= \mathbf{X}^T \left(\mathbf{I} - \left(\mathbf{I} + \mathbf{X} \boldsymbol{\Lambda}^{-1} \mathbf{X}^T \right)^{-1} \mathbf{X} \boldsymbol{\Lambda}^{-1} \mathbf{X}^T \right) \mathbf{X} \\
&= \left(\mathbf{C} - \mathbf{X}^T \left(\mathbf{I} + \mathbf{X} \boldsymbol{\Lambda}^{-1} \mathbf{X}^T \right)^{-1} \mathbf{X} \boldsymbol{\Lambda}^{-1} \mathbf{C} \right) \\
&= \boldsymbol{\Lambda} \left(\boldsymbol{\Lambda}^{-1} - \boldsymbol{\Lambda}^{-1} \mathbf{X}^T \left(\mathbf{I} + \mathbf{X} \boldsymbol{\Lambda}^{-1} \mathbf{X}^T \right)^{-1} \mathbf{X} \boldsymbol{\Lambda}^{-1} \right) \mathbf{C} \\
&= \boldsymbol{\Lambda} \left(\mathbf{C} + \boldsymbol{\Lambda} \right)^{-1} \mathbf{C} \\
&= \left(\mathbf{C}^{-1} + \boldsymbol{\Lambda}^{-1} \right)^{-1} \,, \qquad \text{(B.12)}
\end{aligned}$$

where we have used (6.53) and the Woodbury formula (B.1). We can compute the new matrix $\mathbf{A}^{\text{new}}$ using (B.12) directly. However, this requires computing the inverse of an $n \times n$ matrix which has cost $\mathcal{O}(n^3)$. In practice, it is better to use an alternative way to compute $\mathbf{A}^{\text{new}}$. Because in EP only one component of the diagonal matrix $\boldsymbol{\Lambda}$ changes

each time after the preprocessing of one approximate term $\tilde{t}_i$, the new matrix is

$$\mathbf{A}^{\text{new}} = \left(\mathbf{C}^{-1} + \boldsymbol{\Lambda}^{-1} + \kappa \boldsymbol{\delta}_i \boldsymbol{\delta}_i^T\right)^{-1} = \mathbf{A} - \frac{\mathbf{A}_{\cdot,i}\mathbf{A}_{i,\cdot}}{\kappa^{-1} + A_{ii}}\,, \tag{B.13}$$

where we have used the Woodbury formula (B.1) and

$$\kappa = \left(\frac{1}{v_i^{\text{new}}} - \frac{1}{v_i^{\text{old}}}\right)\,. \tag{B.14}$$

In (B.13) $\boldsymbol{\delta}_i$ denotes an n-dimensional vector whose components are all zero except component i that takes value 1, $\mathbf{A}_{i,\cdot}$ denotes the i-th row of the matrix $\mathbf{A}$ and $\mathbf{A}_{\cdot,i}$ denotes the i-th column. Finally, in (B.14) v_i^{new} is the new value for the parameter v_i of the approximate term and v_i^{old} is the previous value.

Next, from the definition of the vector $\mathbf{h}$ in (6.58)

$$\mathbf{h} = \mathbf{X}^T\mathbf{m}_{\mathbf{w}} = \mathbf{X}^T\mathbf{V}_{\mathbf{w}}\mathbf{X}\boldsymbol{\Lambda}^{-1}\mathbf{m} = \mathbf{A}\boldsymbol{\Lambda}^{-1}\mathbf{m}\,, \tag{B.15}$$

where we have used (6.54) and (6.57). Finally,

$$\mathbf{h}^{\text{new}} = \mathbf{A}^{\text{new}}(\boldsymbol{\Lambda}^{\text{new}})^{-1}\mathbf{m}^{\text{new}}\,, \tag{B.16}$$

where $\boldsymbol{\Lambda}^{\text{new}}$ and $\mathbf{m}^{\text{new}}$ are the matrix $\boldsymbol{\Lambda}$ and the vector $\mathbf{m}$ after the approximate term $\tilde{t}_i$ has been updated.

B.6 Derivation of $|\mathbf{V}_{\mathbf{w}}|$

In this section we compute the value of $|\mathbf{V}_{\mathbf{w}}|$.

$$\begin{aligned} |\mathbf{V}_{\mathbf{w}}| &= \left|\mathbf{I} + \mathbf{X}^T\boldsymbol{\Lambda}^{-1}\mathbf{X}\right|^{-1} \\ &= \left|\mathbf{I} + \mathbf{X}^T\mathbf{X}\boldsymbol{\Lambda}^{-1}\right|^{-1} \\ &= \left|\mathbf{I} + \mathbf{C}\boldsymbol{\Lambda}^{-1}\right|^{-1} \\ &= \frac{|\boldsymbol{\Lambda}|}{|\mathbf{I} + \mathbf{C}\boldsymbol{\Lambda}^{-1}|}\,, \end{aligned} \tag{B.17}$$

where we have used (6.53) and the fact that $|\mathbf{I} + \mathbf{A}\mathbf{B}^T| = |\mathbf{I} + \mathbf{A}^T\mathbf{B}|$ (Bishop, 2006).

B.7 Predictive Distribution of the BM

In this section we derive the value of z in predictive distribution of the BM defined in (6.84). We note that at convergence of the EP algorithm (6.77) is no longer an update rule and becomes an equality for all $i = 1, \ldots, n$. Furthermore, in this situation $h_i^{\text{new}} = h_i$ because $\mathbf{m}_{\mathbf{w}}^{\text{new}} = \mathbf{m}_{\mathbf{w}}$. In consequence,

$$\sum_{i=1}^{n} \frac{m_i\mathbf{x}_i}{v_i} = \sum_{i=1}^{n} \frac{\mathbf{x}_i h_i}{v_i} + \sum_{i=1}^{n} \alpha_i\mathbf{x}_i$$

$$\mathbf{m}_\mathbf{w} + \sum_{i=1}^{n} \frac{m_i \mathbf{x}_i}{v_i} = \mathbf{m}_\mathbf{w} + \sum_{i=1}^{n} \frac{\mathbf{x}_i \mathbf{x}_i^T \mathbf{m}_\mathbf{w}}{v_i} + \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$$
$$\mathbf{m}_\mathbf{w} + \sum_{i=1}^{n} \frac{m_i \mathbf{x}_i}{v_i} = \left(\mathbf{I} + \sum_{i=1}^{n} \frac{\mathbf{x}_i \mathbf{x}_i^T}{v_i} \right) \mathbf{m}_\mathbf{w} + \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$$
$$\mathbf{m}_\mathbf{w} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i \,, \tag{B.18}$$

where we have used (6.53) and (6.54). To compute the value of the denominator in z we describe the matrix $\mathbf{V}_\mathbf{w}$ as follows using (6.53) and the Woodbury formula (B.1)

$$\begin{aligned}
\mathbf{V}_\mathbf{w} &= \left(\mathbf{I} + \mathbf{X} \mathbf{\Lambda}^{-1} \mathbf{X}^T \right)^{-1} \\
&= \mathbf{I} - \mathbf{X} \mathbf{\Lambda}^{-1/2} \left(\mathbf{I} + \mathbf{\Lambda}^{-1/2} \mathbf{C} \mathbf{\Lambda}^{-1/2} \right)^{-1} \mathbf{\Lambda}^{-1/2} \mathbf{X}^T \\
&= \mathbf{I} - \mathbf{X} \left(\mathbf{\Lambda} + \mathbf{C} \right)^{-1} \mathbf{X}^T \\
&= \mathbf{I} - \mathbf{X} \mathbf{\Lambda}^{-1} \left(\mathbf{\Lambda} - \mathbf{A} \right) \mathbf{\Lambda}^{-1} \mathbf{X}^T \,,
\end{aligned} \tag{B.19}$$

where we have also used that

$$\begin{aligned}
\mathbf{\Lambda}^{-1} \left(\mathbf{\Lambda} - \mathbf{A} \right) \mathbf{\Lambda}^{-1} &= \mathbf{\Lambda}^{-1} \left(\mathbf{\Lambda} - \left(\mathbf{C}^{-1} + \mathbf{\Lambda}^{-1} \right)^{-1} \right) \mathbf{\Lambda}^{-1} \\
&= \mathbf{\Lambda}^{-1} \left(\mathbf{\Lambda} \left(\mathbf{C}^{-1} + \mathbf{\Lambda}^{-1} \right) - \mathbf{I} \right) \left(\mathbf{C}^{-1} + \mathbf{\Lambda}^{-1} \right)^{-1} \mathbf{\Lambda}^{-1} \\
&= \mathbf{C}^{-1} \left(\mathbf{C}^{-1} + \mathbf{\Lambda}^{-1} \right)^{-1} \mathbf{\Lambda}^{-1} \\
&= \left(\mathbf{C} + \mathbf{\Lambda} \right)^{-1} \,.
\end{aligned} \tag{B.20}$$

To derive (B.20) we have used (B.12).

The value of z can also be computed without the convergence assumption in the EP algorithm. For this purpose, consider the following relation derived from (6.54)

$$\begin{aligned}
\mathbf{m}_\mathbf{w} &= \mathbf{V}_\mathbf{w} \sum_{i=1}^{n} \mathbf{x}_i \frac{m_i}{v_i} \\
&= \mathbf{V}_\mathbf{w} \mathbf{X} \mathbf{\Lambda}^{-1} \mathbf{m} \\
&= \left(\mathbf{I} - \mathbf{X} \left(\mathbf{\Lambda} + \mathbf{C} \right)^{-1} \mathbf{X}^T \right) \mathbf{X} \mathbf{\Lambda}^{-1} \mathbf{m} \\
&= \left(\mathbf{X} - \mathbf{X} \left(\mathbf{\Lambda} + \mathbf{C} \right)^{-1} \mathbf{C} \right) \mathbf{\Lambda}^{-1} \mathbf{m} \\
&= \mathbf{X} \left(\mathbf{I} - \left(\mathbf{C}^{-1} \mathbf{\Lambda} + \mathbf{I} \right)^{-1} \right) \mathbf{\Lambda}^{-1} \mathbf{m} \\
&= \mathbf{X} \left(\mathbf{I} - \mathbf{\Lambda}^{-1} \left(\mathbf{C}^{-1} + \mathbf{\Lambda}^{-1} \right)^{-1} \right) \mathbf{\Lambda}^{-1} \mathbf{m} \\
&= \mathbf{X} \left(\mathbf{\Lambda}^{-1} - \mathbf{\Lambda}^{-1} \left(\mathbf{C}^{-1} + \mathbf{\Lambda}^{-1} \right)^{-1} \mathbf{\Lambda}^{-1} \right) \mathbf{m} \\
&= \mathbf{X} \left(\mathbf{\Lambda}^{-1} - \mathbf{\Lambda}^{-1} \mathbf{A} \mathbf{\Lambda}^{-1} \right) \mathbf{m} \,,
\end{aligned} \tag{B.21}$$

where we have used (B.19) and (B.12).

Thus, we can easily compute $\mathbf{x}_\text{new}^T \mathbf{m}_\mathbf{w}$ in terms of inner products for an arbitrary instance $\mathbf{x}_\text{new}$ using either (B.18) or (B.21). However, if the EP algorithm has converged, it is better to use (B.18) because it is faster.

B.8 KL-divergence between $\mathcal{Q}_j^{\text{new}}$ and $\mathcal{Q}^{\setminus i}$

In this section we derive the KL-divergence between $\mathcal{Q}_j^{\text{new}}$ and $\mathcal{Q}^{\setminus i}$, which is used for ranking candidate patterns in the training algorithm of the SBM. Note that because both distributions factorize with respect to to the model parameters ϵ and $\mathbf{w}$, the KL-divergence is the sum of the KL-divergences between the marginal distributions. Using the definition of $\mathcal{Q}_j^{\text{new}}$ given in (6.102), the definition of $\mathcal{Q}^{\setminus i}$ given in (6.14), and (A.38), the KL-divergence with respect to the marginal over $\mathbf{w}$ is

$$\begin{aligned}
\mathrm{KL}_{\mathbf{w}}\left(\mathcal{Q}_j^{\text{new}}, \mathcal{Q}^{\setminus i}\right) &= \frac{1}{2}\log\left(\left|\mathbf{V}_{\mathbf{w}}^{\setminus i}(\mathbf{V}_{\mathbf{w}}^{\text{new}}(j))^{-1}\right|\right) + \frac{1}{2}\mathrm{trace}\left((\mathbf{V}_{\mathbf{w}}^{\setminus i})^{-1}\mathbf{V}_{\mathbf{w}}^{\text{new}}(j) - \mathbf{I}\right) \\
&\quad + \frac{1}{2}\left(\mathbf{m}_{\mathbf{w}}^{\text{new}}(j) - \mathbf{m}_{\mathbf{w}}^{\setminus i}\right)^T(\mathbf{V}_{\mathbf{w}}^{\setminus i})^{-1}\left(\mathbf{m}_{\mathbf{w}}^{\text{new}}(j) - \mathbf{m}_{\mathbf{w}}^{\setminus i}\right) \\
&= \frac{1}{2}\log\left(\left|\mathbf{V}_{\mathbf{w}}^{\setminus i}\left((\mathbf{V}_{\mathbf{w}}^{\setminus i})^{-1} + v_j^{-1}\mathbf{x}_j\mathbf{x}_j^T\right)\right|\right) \\
&\quad + \frac{1}{2}\mathrm{trace}\left((\mathbf{V}_{\mathbf{w}}^{\setminus i})^{-1}\left(\mathbf{V}_{\mathbf{w}}^{\setminus i} - v_j^{-1}\frac{\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_j\mathbf{x}_j^T\mathbf{V}_{\mathbf{w}}^{\setminus i}}{1 + v_j^{-1}\mathbf{x}_j^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_j}\right) - \mathbf{I}\right) \\
&\quad + \frac{1}{2}\left(\alpha_j\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_j\right)^T(\mathbf{V}_{\mathbf{w}}^{\setminus i})^{-1}\left(\alpha_j\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_j\right) \\
&= \frac{1}{2}\log\left(\left|\mathbf{I} + v_j^{-1}\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_j\mathbf{x}_j^T\right|\right) - \frac{v_j^{-1}}{1 + v_j^{-1}\mathbf{x}_j^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_j}\frac{1}{2}\mathrm{trace}\left(\mathbf{x}_j\mathbf{x}_j^T\mathbf{V}_{\mathbf{w}}^{\setminus i}\right) \\
&\quad + \frac{1}{2}\left(\alpha_j\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_j\right)^T\alpha_j\mathbf{x}_j \\
&= \frac{1}{2}\log\left(1 + v_j^{-1}\lambda_j^{\setminus i}\right) - \frac{1}{2}\frac{v_j^{-1}\lambda_j^{\setminus i}}{1 + v_j^{-1}\lambda_j^{\setminus i}} + \frac{1}{2}\alpha_j^2\lambda_j^{\setminus i} \\
&= -\frac{1}{2}\log\left(1 - \alpha_j h_j^{\text{new}}\right) - \frac{1}{2}\alpha_j h_j^{\text{new}} + \frac{1}{2}\alpha_j^2\lambda_j^{\setminus i}, \qquad \text{(B.22)}
\end{aligned}$$

where $\mathbf{I}$ is the identity matrix and we have also used that $(\mathbf{V}_{\mathbf{w}}^{\text{new}}(j))^{-1} = (\mathbf{V}_{\mathbf{w}}^{\setminus i})^{-1} + v_j^{-1}\mathbf{x}_j\mathbf{x}_j^T$, which can be derived from (6.35) and (6.42), and that $\mathbf{m}_{\mathbf{w}}^{\text{new}}(j) = \mathbf{m}_{\mathbf{w}}^{\setminus i} + \alpha_j\mathbf{V}_{\mathbf{w}}^{\setminus i}\mathbf{x}_j$, which can be derived from (6.53). Furthermore, we have used the definition of v_j given in (6.113), the Woodbury formula (B.1) and the fact that $|\mathbf{I}+\mathbf{A}\mathbf{B}^T| = |\mathbf{I}+\mathbf{A}^T\mathbf{B}|$ (Bishop, 2006). Finally, the KL divergence with respect to the marginal over ϵ can be computed using (A.23), the definition of $\mathcal{Q}_j^{\text{new}}$ given in (6.102) and the definition of $\mathcal{Q}^{\setminus i}$ given in (6.14)

$$\begin{aligned}
\mathrm{KL}_{\epsilon}\left(\mathcal{Q}_j^{\text{new}}, \mathcal{Q}^{\setminus i}\right) = \log\left(\frac{\beta(a_\epsilon^{\setminus i}, b_\epsilon^{\setminus i})}{\beta(a_\epsilon^{\text{new}}(j), b_\epsilon^{\text{new}}(j))}\right) &- \left(a_\epsilon^{\setminus i} - a_\epsilon^{\text{new}}(j)\right)\Psi(a_\epsilon^{\text{new}}(j)) \\
&- \left(b_\epsilon^{\setminus i} - b_\epsilon^{\text{new}}(j)\right)\Psi(b_\epsilon^{\text{new}}(j)) \\
&+ \left(a_\epsilon^{\setminus i} - a_\epsilon^{\text{new}}(j) + b_\epsilon^{\setminus i} - b_\epsilon^{\text{new}}(j)\right)\Psi(a_\epsilon^{\text{new}}(j) + b_\epsilon^{\text{new}}(j)), \qquad \text{(B.23)}
\end{aligned}$$

where Ψ is the digamma function defined as $\Psi(x) = d\log(\Gamma(x))/dx$ and $\beta(a,b)$ is the beta function (Abramowitz and Stegun, 1964).

B.9 Update of the matrix B

In this section we show how the matrix $\mathbf{B}$, which is defined as

$$\mathbf{B} = \mathbf{I} + \mathbf{\Lambda}_{\mathcal{I}}^{-1/2} \mathbf{C}_{\mathcal{I}} \mathbf{\Lambda}_{\mathcal{I}}^{-1/2} \in \mathbb{R}^{d,d} , \tag{B.24}$$

is updated in the training algorithm of the SBM. All the derivations of this section have been extracted from the Appendix of (Seeger, 2003).

Assume that the element i is removed from the active set $\mathcal{I}$, and that j is the element that is included in $\mathcal{I}$. Furthermore, assume that l was the position of i in $\mathcal{I}$. Then, we have to replace $\mathbf{\Lambda}_{\mathcal{I}}^{-1/2}$ by $\mathbf{\Lambda}_{\mathcal{I}}^{-1/2} + (v_j^{-1/2} - v_i^{-1/2}) \boldsymbol{\delta}_l \boldsymbol{\delta}_l^T$, where $\boldsymbol{\delta}^l \in \mathbb{R}^d$ is a vector with all components zero except component l-th that takes value one. In addition, since $\mathbf{C}_{\mathcal{I}} = \mathbf{I}_{\mathcal{I},\cdot} \mathbf{C} \mathbf{I}_{\cdot,\mathcal{I}}$, after the change in the active set $\mathcal{I}$ we have to replace $\mathbf{I}_{\cdot,\mathcal{I}}$ by $\mathbf{I}_{\cdot,\mathcal{I}} + (\boldsymbol{\delta}_j - \boldsymbol{\delta}_j) \boldsymbol{\delta}_l^T$, where $\boldsymbol{\delta}_i \in \mathbb{R}^n$ and $\boldsymbol{\delta}_j \in \mathbb{R}^n$. In consequence,

$$\mathbf{B}_{\text{new}} = \mathbf{I} + \mathbf{Q}^T \mathbf{C} \mathbf{Q} , \tag{B.25}$$

where

$$\begin{aligned} \mathbf{Q} &= \left(\mathbf{I}_{\cdot,\mathcal{I}} + (\boldsymbol{\delta}_j - \boldsymbol{\delta}_j) \boldsymbol{\delta}_l^T \right) \left(\mathbf{\Lambda}_{\mathcal{I}}^{-1/2} + (v_j^{-1/2} - v_i^{-1/2}) \boldsymbol{\delta}_l \boldsymbol{\delta}_l^T \right) \\ &= \mathbf{I}_{\cdot,\mathcal{I}} \mathbf{\Lambda}_{\mathcal{I}}^{-1/2} + \left(v_j^{-1/2} \boldsymbol{\delta}_j - v_i^{-1/2} \boldsymbol{\delta}_i \right) \boldsymbol{\delta}_l^T . \end{aligned} \tag{B.26}$$

In (B.26) we have used that $\mathbf{I}_{\cdot,\mathcal{I}} \boldsymbol{\delta}_l = \boldsymbol{\delta}_i$ and that $\boldsymbol{\delta}_l^T \mathbf{\Lambda}_{\mathcal{I}}^{-1/2} = v_i^{-1/2} \boldsymbol{\delta}_l^T$. Now, if we set

$$\boldsymbol{v} = \mathbf{\Lambda}_{\mathcal{I}}^{-1/2} \mathbf{I}_{\mathcal{I},\cdot} \mathbf{C} \left(v_j^{-1/2} \boldsymbol{\delta}_j - v_i^{-1/2} \boldsymbol{\delta}_i \right) = \mathbf{\Lambda}_{\mathcal{I}}^{-1/2} \left(v_j^{-1/2} \mathbf{C}_{\mathcal{I},j} - v_i^{-1/2} \mathbf{C}_{\mathcal{I},i} \right) , \tag{B.27}$$

(B.25) becomes

$$\mathbf{B}_{\text{new}} = \mathbf{B} + \boldsymbol{\delta}_l \boldsymbol{v}^T + \boldsymbol{v} \boldsymbol{\delta}_l^T + \eta \boldsymbol{\delta}_l \boldsymbol{\delta}_l^T , \tag{B.28}$$

where

$$\eta = v_j^{-1} C_{jj} + v_i^{-1} C_{ii} - 2 v_j^{-1/2} v_i^{-1/2} C_{ij} > 0 . \tag{B.29}$$

Therefore,

$$\mathbf{B}_{\text{new}} = \mathbf{B} + \left(\tilde{\boldsymbol{\delta}}_l + \tilde{\boldsymbol{v}} \right) \left(\tilde{\boldsymbol{\delta}}_l + \tilde{\boldsymbol{v}} \right)^T - \tilde{\boldsymbol{v}} \tilde{\boldsymbol{v}}^T , \tag{B.30}$$

where $\tilde{\boldsymbol{\delta}}_l = \eta^{1/2} \boldsymbol{\delta}_l$ and $\tilde{\boldsymbol{v}} = \eta^{-1/2} \boldsymbol{v}$.

B.10 Update of the matrix M

In this section we show how to update the matrix $\mathbf{M}$, which is defined as

$$\mathbf{M} = \mathbf{L}^{-1} \mathbf{\Lambda}_{\mathcal{I}}^{-1/2} \mathbf{C}_{\mathcal{I},\cdot} \in \mathbb{R}^{d,n} , \tag{B.31}$$

after each iteration of the training algorithm of the SBM. The update of the matrix $\mathbf{M}$ is described in the Appendix of (Seeger, 2003). However, the rule suggested does not consider the changes in $\mathbf{C}_{\mathcal{I},\cdot}$. In this section we derive a more accurate update rule.

The update of $\mathbf{M}$ can be performed in two steps. First, assume that the element i is removed from the active set $\mathcal{I}$, and that j is the element that is included in $\mathcal{I}$. Furthermore, assume that l was the position of i in $\mathcal{I}$. Then, we have to replace $\mathbf{\Lambda}_{\mathcal{I}}^{-1/2}$

by $\mathbf{\Lambda}_{\mathcal{I}}^{-1/2} + (v_j^{-1/2} - v_i^{-1/2})\boldsymbol{\delta}_l\boldsymbol{\delta}_l^T$, where $\boldsymbol{\delta}^l \in \mathbb{R}^d$ is a vector with all components zero except component l-th that takes value one. Similarly, $\mathbf{C}_{\mathcal{I},\cdot}$ has to be replaced by $\mathbf{C}_{\mathcal{I},\cdot} + \boldsymbol{\delta}_l\left(\mathbf{C}_{j,\cdot} - \mathbf{C}_{i,\cdot}\right)$. Second, from Appendix B.9 we know that the update of $\mathbf{B}$ involves two rank-one updates. After a single rank-one update of $\mathbf{B}$ the new Cholesky factor of $\mathbf{B}$ is $\mathbf{L}_{\text{new}} = \mathbf{L}\tilde{\mathbf{L}}$, where $\tilde{\mathbf{L}}$ has a special form that allows the use of a fast multiplication algorithm. See Appendix B.11 for further details. Thus, after two rank-one updates the new Cholesky factor is $\mathbf{L}_{\text{new}} = \mathbf{L}\tilde{\mathbf{L}}_1\tilde{\mathbf{L}}_2$, where both $\tilde{\mathbf{L}}_1$ and $\tilde{\mathbf{L}}_2$ have a special form. In consequence, the update of the matrix $\mathbf{M}$ involves computing first

$$\begin{aligned}
\mathbf{M}' &= \mathbf{L}^{-1}\left(\mathbf{\Lambda}_{\mathcal{I}}^{-1/2} + (v_j^{-1/2} - v_i^{-1/2})\boldsymbol{\delta}_l\boldsymbol{\delta}_l^T\right)\left(\mathbf{C}_{\mathcal{I},\cdot} + \boldsymbol{\delta}_l\left(\mathbf{C}_{j,\cdot} - \mathbf{C}_{i,\cdot}\right)\right) \\
&= \mathbf{L}^{-1}\Big(\mathbf{\Lambda}_{\mathcal{I}}^{-1/2}\mathbf{C}_{\mathcal{I},\cdot} + \mathbf{\Lambda}_{\mathcal{I}}^{-1/2}\boldsymbol{\delta}_l\left(\mathbf{C}_{j,\cdot} - \mathbf{C}_{i,\cdot}\right) + (v_j^{-1/2} - v_i^{-1/2})\boldsymbol{\delta}_l\boldsymbol{\delta}_l^T\mathbf{C}_{\mathcal{I},\cdot} \\
&\quad + (v_j^{-1/2} - v_i^{-1/2})\boldsymbol{\delta}_l\boldsymbol{\delta}_l^T\boldsymbol{\delta}_l\left(\mathbf{C}_{j,\cdot} - \mathbf{C}_{i,\cdot}\right)\Big) \\
&= \mathbf{L}^{-1}\Big(\mathbf{\Lambda}_{\mathcal{I}}^{-1/2}\mathbf{C}_{\mathcal{I},\cdot} + v_i^{-1/2}\boldsymbol{\delta}_l\left(\mathbf{C}_{j,\cdot} - \mathbf{C}_{i,\cdot}\right) + (v_j^{-1/2} - v_i^{-1/2})\boldsymbol{\delta}_l\mathbf{C}_{i,\cdot} \\
&\quad + (v_j^{-1/2} - v_i^{-1/2})\boldsymbol{\delta}_l\left(\mathbf{C}_{j,\cdot} - \mathbf{C}_{i,\cdot}\right)\Big) \\
&= \mathbf{M} + \left(\mathbf{L}^{-1}\boldsymbol{\delta}_l\right)\left(v_j^{-1/2}\mathbf{C}_{j,\cdot} - v_i^{-1/2}\mathbf{C}_{i,\cdot}\right), \qquad \text{(B.32)}
\end{aligned}$$

where we have used that $\mathbf{\Lambda}_{\mathcal{I}}^{-1/2}\boldsymbol{\delta}_l = v_i^{-1/2}\boldsymbol{\delta}_l$ and that $\boldsymbol{\delta}_l^T\mathbf{C}_{\mathcal{I},\cdot} = \mathbf{C}_{i,\cdot}$. Note that (B.32) can be computed in $\mathcal{O}(nd)$ steps. Once the matrix $\mathbf{M}'$ has been computed, we can update $\mathbf{M}$

$$\mathbf{M}_{\text{new}} = \tilde{\mathbf{L}}_2^{-1}\tilde{\mathbf{L}}_1^{-1}\mathbf{M}', \qquad \text{(B.33)}$$

where the multiplications by the factors $\tilde{\mathbf{L}}_2^{-1}$ and $\tilde{\mathbf{L}}_1^{-1}$ can be computed in terms of $\tilde{\mathbf{L}}_2$ and $\tilde{\mathbf{L}}_1$ in $\mathcal{O}(nd)$ steps. See Appendix B.12 for further details.

B.11 Structure of the Cholesky factor $\tilde{\mathbf{L}}$

In this section we show that $\tilde{\mathbf{L}}$ has a special structure that allows the use of a fast multiplication algorithm. Let $\mathbf{B} = \mathbf{L}\mathbf{L}^T$ be a simetric positive definite matrix of size $d \times d$, where $\mathbf{L}$ is a lower triangular matrix known as the Cholesky factor of $\mathbf{B}$. After performing a rank-one update of $\mathbf{B}$

$$\begin{aligned}
\mathbf{B}_{\text{new}} &= \mathbf{B} + \alpha\mathbf{v}\mathbf{v}^T \\
&= \mathbf{L}\left(\mathbf{I} + \alpha\mathbf{p}\mathbf{p}^T\right)\mathbf{L}^T \\
&= \mathbf{L}\tilde{\mathbf{L}}\tilde{\mathbf{L}}^T\mathbf{L}^T, \qquad \text{(B.34)}
\end{aligned}$$

where $\mathbf{L}\mathbf{p} = \mathbf{v}$, $\mathbf{p}$ is obtained by forward substitution (Gill et al., 1974) and $\tilde{\mathbf{L}}$ is the Cholesky factor of $\mathbf{I} + \alpha\mathbf{p}\mathbf{p}^T$. Thus, the updated Cholesky factor of $\mathbf{B}_{\text{new}}$ is $\mathbf{L}\tilde{\mathbf{L}}$.

It turns out that $\tilde{\mathbf{L}}$ has a special structure and hence, only depends on $\mathcal{O}(d)$ parameters (Gill et al., 1974)

$$\tilde{\mathbf{L}} = \begin{pmatrix}
\tilde{d}_1 & 0 & 0 & \dots & 0 \\
\tilde{d}_1 p_2 \beta_1 & \tilde{d}_2 & 0 & \dots & 0 \\
\tilde{d}_1 p_3 \beta_1 & \tilde{d}_2 p_3 \beta_2 & \tilde{d}_3 & \dots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\tilde{d}_1 p_d \beta_1 & \tilde{d}_2 p_d \beta_2 & \tilde{d}_3 p_d \beta_3 & \dots & \tilde{d}_d
\end{pmatrix}, \qquad \text{(B.35)}$$

where the values of the vectors $\tilde{\mathbf{d}}$ and $\boldsymbol{\beta}$ are computed using the following recurrence relations for $i = 1, \ldots, d$

$$\tilde{d}_i = \sqrt{1 + \alpha_i p_i^2}\,, \qquad \beta_i = \alpha_i p_i / \tilde{d}_i^2\,, \qquad \alpha_{i+1} = \alpha_i / \tilde{d}_i^2\,, \tag{B.36}$$

where $\alpha_1 = \alpha$. See (Gill et al., 1974) for further details. Thus, we can make use of this special structure to derive a fast multiplication algorithm to compute $\mathbf{L}\tilde{\mathbf{L}}$. This algorithm is displayed in Figure B.1.

Input: matrix $\mathbf{L}$ of size $d \times d$, $\mathbf{p}$, $\boldsymbol{\beta}$ and $\tilde{\mathbf{d}}$.

Output: $\mathbf{L}^{\text{new}} = \mathbf{L}\tilde{\mathbf{L}}$.

1. $\mathbf{L}^{\text{new}} \leftarrow \mathbf{L}$
2. For $i = 1, \ldots, d$
   (a) $\sigma \leftarrow \rho \leftarrow 0$
   (b) For $j = i, \ldots, 1$
      i. $\sigma \leftarrow \sigma + \rho$
      ii. $\rho \leftarrow p_j L_{ij}^{\text{new}}$
      iii. $L_{ij}^{\text{new}} \leftarrow \left(L_{ij}^{\text{new}} + \sigma\beta_j\right)\tilde{d}_j$
3. Return $\mathbf{L}^{\text{new}}$.

FIGURE B.1: Algorithm that computes the updated Cholesky factor $\mathbf{L}\tilde{\mathbf{L}}$.

B.12 Multiplication by $\tilde{\mathbf{L}}^{-1}$

In this section we show that a multiplication by $\tilde{\mathbf{L}}^{-1}$ can be expressed in terms of a multiplication by the standard factor $\tilde{\mathbf{L}}$. From Appendix B.11 we know that $\tilde{\mathbf{L}}\tilde{\mathbf{L}}^T = \mathbf{I} + \alpha\mathbf{p}\mathbf{p}^T$. Thus,

$$\begin{aligned}\tilde{\mathbf{L}}^{-1} &= \tilde{\mathbf{L}}^T \left(\mathbf{I} + \alpha\mathbf{p}\mathbf{p}^T\right)^{-1} \\ &= \tilde{\mathbf{L}}^T \left(\mathbf{I} - \frac{\alpha}{1 + \alpha\mathbf{p}^T\mathbf{p}}\mathbf{p}\mathbf{p}^T\right)\,,\end{aligned} \tag{B.37}$$

where we have used the Woodbury formula (B.1). In consequence, computing $\tilde{\mathbf{L}}^{-1}\mathbf{L}^{-1}$ takes $\mathcal{O}(d^2)$ steps, where d is the number of rows and columns of $\mathbf{L}$. Similarly, if $\mathbf{M}$ is an arbitrary matrix of size $d \times n$, the product $\tilde{\mathbf{L}}^{-1}\mathbf{M}$ can be computed in $\mathcal{O}(nd)$ steps. Figure B.2 displays the algorithm that computes the product $\tilde{\mathbf{L}}^T\mathbf{M}$.

B.13 Update of the vector h in the SBM

In this section we show how to update in the training algorithm of the SBM the vector $\mathbf{h}$, which is is defined in (B.15) as

$$\mathbf{h} = \mathbf{A}\boldsymbol{\Lambda}^{-1}\mathbf{m}\,. \tag{B.38}$$

Input: matrix $\mathbf{M}$ of size $d \times n$, $\mathbf{p}$, $\boldsymbol{\beta}$ and $\tilde{\mathbf{d}}$.

Output: $\mathbf{M}' = \tilde{\mathbf{L}}^T\mathbf{M}$.

1. $\mathbf{M}' \leftarrow \mathbf{M}$
2. For $j = 1, \ldots, n$
   (a) $\sigma \leftarrow \rho \leftarrow 0$
   (b) For $i = d, \ldots, 1$
      i. $\sigma \leftarrow \sigma + \rho$
      ii. $\rho \leftarrow p_i M'_{ij}$
      iii. $M'_{ij} \leftarrow \left(M'_{ij} + \sigma\beta_i\right)\tilde{d}_i$
3. Return $\mathbf{M}'$.

FIGURE B.2: Algorithm that computes the product $\tilde{\mathbf{L}}^T\mathbf{M}$.

All the derivations of this section have been extracted from the Appendix of (Seeger, 2003).

Assume that the element i is removed from the active set $\mathcal{I}$, and that j is the element that is included in $\mathcal{I}$. Furthermore, assume that l was the position of i in $\mathcal{I}$. Consider the 2×2 diagonal matrix $\boldsymbol{\Delta} = \text{diag}(v_j^{-1}, -v_i^{-1})$ that denotes the changes in $\boldsymbol{\Lambda}^{-1}$. From the definition of the matrix $\mathbf{A}$ given in (B.12)

$$\begin{aligned}\mathbf{A}^{\text{new}} &= \left(\mathbf{C}^{-1} + \boldsymbol{\Lambda}^{-1} + \mathbf{I}_{\cdot,\{ji\}}\boldsymbol{\Delta}\mathbf{I}_{\{ji\},\cdot}\right)^{-1} \\ &= \mathbf{A} - \mathbf{A}\mathbf{I}_{\cdot,\{ji\}}\boldsymbol{\Delta}^{1/2}\mathbf{P}^{-1}\boldsymbol{\Delta}^{1/2}\mathbf{I}_{\{ji\},\cdot}\mathbf{A} \\ &= \mathbf{A} - \mathbf{A}_{\cdot,\{ji\}}\boldsymbol{\Delta}^{1/2}\mathbf{P}^{-1}\boldsymbol{\Delta}^{1/2}\mathbf{A}_{\{ji\},\cdot}\,, \end{aligned} \tag{B.39}$$

where

$$\begin{aligned}\mathbf{P} &= \mathbf{I} + \boldsymbol{\Delta}^{1/2}\mathbf{I}_{\{ji\},\cdot}\mathbf{A}\mathbf{I}_{\cdot,\{ji\}}\boldsymbol{\Delta}^{1/2} \\ &= \mathbf{I} + \boldsymbol{\Delta}^{1/2}\mathbf{A}_{\{ji\}}\boldsymbol{\Delta}^{1/2}\,. \end{aligned} \tag{B.40}$$

In the computation of (B.39) we have used the Woodbury formula (B.1). Now, consider the vector $\boldsymbol{\gamma} = (v_j^{-1}m_j, -v_i^{-1}m_i)^T$ that denotes the changes in $\boldsymbol{\Lambda}^{-1}\mathbf{m}$. Using (B.38) and (B.39)

$$\begin{aligned}\mathbf{h}_{\text{new}} &= \mathbf{A}^{\text{new}}\left(\boldsymbol{\Lambda}^{-1}\mathbf{m} + \mathbf{I}_{\cdot,\{ji\}}\boldsymbol{\gamma}\right) \\ &= \mathbf{h} + \mathbf{A}\mathbf{I}_{\{ji\},\cdot}\boldsymbol{\gamma} - \mathbf{A}_{\cdot,\{ji\}}\boldsymbol{\Delta}^{1/2}\mathbf{P}^{-1}\boldsymbol{\Delta}^{1/2}\mathbf{h}_{\{ji\}} \\ &\quad - \mathbf{A}_{\cdot,\{ji\}}\boldsymbol{\Delta}^{1/2}\mathbf{P}^{-1}\boldsymbol{\Delta}^{1/2}\mathbf{A}_{\{ji\},\cdot}\mathbf{I}_{\cdot,\{ji\}}\boldsymbol{\gamma} \\ &= \mathbf{h} - \mathbf{A}_{\cdot,\{ji\}}\boldsymbol{\Delta}^{1/2}\mathbf{P}^{-1}\boldsymbol{\Delta}^{1/2}\mathbf{h}_{\{ji\}} + \mathbf{A}^{\text{new}}\mathbf{I}_{\cdot,\{ji\}}\boldsymbol{\gamma} \\ &= \mathbf{h} - \mathbf{A}_{\cdot,\{ji\}}\boldsymbol{\Delta}^{1/2}\mathbf{P}^{-1}\boldsymbol{\Delta}^{1/2}\mathbf{h}_{\{ji\}} + \mathbf{A}_{\cdot,\{ji\}}\boldsymbol{\Delta}^{1/2}\mathbf{P}^{-1}\boldsymbol{\Delta}^{-1/2}\boldsymbol{\gamma} \\ &= \mathbf{h} + \mathbf{A}_{\cdot,\{ji\}}\boldsymbol{\Delta}^{1/2}\mathbf{P}^{-1}\left(\boldsymbol{\Delta}^{-1/2}\boldsymbol{\gamma} - \boldsymbol{\Delta}^{1/2}\mathbf{h}_{\{ji\}}\right)\,, \end{aligned} \tag{B.41}$$

where we have used that

$$\begin{aligned}\mathbf{A}^{\text{new}}\mathbf{I}_{\cdot,\{ji\}} &= \mathbf{A}_{\cdot,\{ji\}} - \mathbf{A}_{\cdot,\{ji\}}\boldsymbol{\Delta}^{1/2}\mathbf{P}^{-1}\boldsymbol{\Delta}^{1/2}\mathbf{A}_{\{ji\}} \\ &= \mathbf{A}_{\cdot,\{ji\}}\boldsymbol{\Delta}^{1/2}\mathbf{P}^{-1}\left(\mathbf{P}\boldsymbol{\Delta}^{-1/2} - \boldsymbol{\Delta}^{1/2}\mathbf{A}_{\{ji\}}\right) \\ &= \mathbf{A}_{\cdot,\{ji\}}\boldsymbol{\Delta}^{1/2}\mathbf{P}^{-1}\left(\boldsymbol{\Delta}^{-1/2} + \boldsymbol{\Delta}^{1/2}\mathbf{A}_{\{ji\}} - \boldsymbol{\Delta}^{1/2}\mathbf{A}_{\{ji\}}\right) \\ &= \mathbf{A}_{\cdot,\{ji\}}\boldsymbol{\Delta}^{1/2}\mathbf{P}^{-1}\boldsymbol{\Delta}^{-1/2}\,. \end{aligned} \tag{B.42}$$

For the derivation of (B.42) we have used (B.40).

In consequence, the new vector $\mathbf{h}_{\text{new}}$ can be computed in $\mathcal{O}(nd)$ steps because any column of the matrix $\mathbf{A}$ can be computed in $\mathcal{O}(nd)$ steps using the decomposition (6.116). Furthermore, the inversion process of $\mathbf{P}$ is inexpensive because it is a 2×2 matrix.

Appendix C

Appendix for Chapter 7

C.1 Derivation of s_i

In this section we show how to compute the value of s_i, as defined in (7.37). From the definition of $\tilde{t}_i = Z_i \mathcal{Q}^{\text{new}}/\mathcal{Q}^{\backslash i}$ and using (A.46)

$$s_i = Z_i \sqrt{\prod_{j=1}^{d} \frac{\nu_j^{\backslash i}}{\nu_j^{\text{new}}}} \exp\left(-\frac{1}{2}\left((\boldsymbol{\mu}^{\text{new}})^T((\boldsymbol{\nu}^{\text{new}})^{-1} \circ \boldsymbol{\mu}^{\text{new}}) - (\boldsymbol{\mu}^{\backslash i})^T(\boldsymbol{\nu}^{\backslash i} \circ \boldsymbol{\mu}^{\backslash i}) - \mathbf{m}_i^T(\mathbf{v}_i^{-1} \circ \mathbf{m}_i)\right)\right), \tag{C.1}$$

where the operator $\circ$ indicates the Hadamard (element-wise) product and the inverse of a vector is defined as a new vector whose components are the inverse of the components of the original vector. Because $(\boldsymbol{\nu}^{\text{new}})^{-1} = (\boldsymbol{\nu}^{\backslash i})^{-1} + \mathbf{v}_i^{-1}$, which can be derived from (7.35), is easy to show that

$$\sqrt{\prod_{j=1}^{d} \frac{\nu_j^{\backslash i}}{\nu_j^{\text{new}}}} = \prod_{j=1}^{d} \sqrt{1 + v_{ij}^{-1} \nu_j^{\backslash i}}. \tag{C.2}$$

Next, we compute

$$\begin{aligned}
(\boldsymbol{\mu}^{\text{new}})^T((\boldsymbol{\nu}^{\text{new}})^{-1} \circ \boldsymbol{\mu}^{\text{new}}) &= \left(\boldsymbol{\mu}^{\backslash i} + \alpha_i \boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i\right)^T \left(\left((\boldsymbol{\nu}^{\backslash i})^{-1} + \mathbf{v}_i^{-1}\right) \circ \left(\boldsymbol{\mu}^{\backslash i} + \alpha_i \boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i\right)\right) \\
&= (\boldsymbol{\mu}^{\backslash i})^T \left((\boldsymbol{\nu}^{\backslash i})^{-1} \circ \boldsymbol{\mu}^{\backslash i}\right) + 2\alpha_i (\boldsymbol{\mu}^{\backslash i})^T \mathbf{x}_i + \alpha_i^2 \mathbf{x}_i^T \left(\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i\right) + \\
&\quad \alpha_i^2 (\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i)^T \left(\mathbf{v}_i^{-1} \circ \left(\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i\right)\right) + (\boldsymbol{\mu}^{\backslash i})^T \left(\mathbf{v}_i^{-1} \circ \boldsymbol{\mu}^{\backslash i}\right) + \\
&\quad 2\alpha_i (\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i)^T \left(\mathbf{v}_i^{-1} \circ \boldsymbol{\mu}^{\backslash i}\right),
\end{aligned} \tag{C.3}$$

where we have used (7.23) and again that $(\boldsymbol{\nu}^{\text{new}})^{-1} = (\boldsymbol{\nu}^{\backslash i})^{-1} + \mathbf{v}_i^{-1}$. Furthermore,

$$\begin{aligned}\mathbf{m}_i^T(\mathbf{v}_i^{-1} \circ \mathbf{m}_i) &= \left(\boldsymbol{\mu}^{\backslash i} + \alpha_i(\mathbf{v}_i + \boldsymbol{\nu}^{\backslash i}) \circ \mathbf{x}_i\right)^T \left(\mathbf{v}_i^{-1} \circ \left(\boldsymbol{\mu}^{\backslash i} + \alpha_i(\mathbf{v}_i + \boldsymbol{\nu}^{\backslash i}) \circ \mathbf{x}_i\right)\right) \\ &= (\boldsymbol{\mu}^{\backslash i})^T \left(\mathbf{v}_i^{-1} \circ \boldsymbol{\mu}^{\backslash i}\right) + \alpha_i^2 \mathbf{x}_i^T(\mathbf{v}_i \circ \mathbf{x}_i) + \alpha_i^2(\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i)^T \left(\mathbf{v}_i^{-1} \circ \left(\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i\right)\right) \\ &\quad + 2\alpha_i(\boldsymbol{\mu}^{\backslash i})^T \mathbf{x}_i + 2\alpha_i(\boldsymbol{\mu}^{\backslash i})^T \left(\mathbf{v}_i^{-1} \circ \left(\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i\right)\right) + \\ &\quad 2\alpha_i^2 \mathbf{x}_i^T \left(\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i\right) . \end{aligned} \tag{C.4}$$

where we have used (7.36). With these values, i.e. (C.3) and (C.4) we have that

$$\begin{aligned}(\boldsymbol{\mu}^{\text{new}})^T((\boldsymbol{\nu}^{\text{new}})^{-1} \circ \boldsymbol{\mu}^{\text{new}}) - (\boldsymbol{\mu}^{\backslash i})^T(\boldsymbol{\nu}^{\backslash i} \circ \boldsymbol{\mu}^{\backslash i}) & \\ -\mathbf{m}_i^T(\mathbf{v}_i^{-1} \circ \mathbf{m}_i) &= -\alpha_i^2 \mathbf{x}_i^T (\mathbf{v}_i \circ \mathbf{x}_i) \\ &\quad - \alpha_i^2 \mathbf{x}_i^T \left(\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i\right) \\ &= -\alpha_i^2 \mathbf{x}_i^T \left((\mathbf{v}_i + \boldsymbol{\nu}^{\backslash i}) \circ \mathbf{x}_i\right) \\ &= -d\frac{\alpha_i \mathbf{x}_i^T(\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i) + \alpha_i}{\mathbf{x}_i^T \boldsymbol{\mu}^{\text{new}} + \alpha_i} , \end{aligned} \tag{C.5}$$

where we have used (7.35) and d is the data dimensionality. Finally,

$$s_i = Z_i \prod_{j=1}^{d} \sqrt{1 + v_{ij}^{-1} \nu_j^{\backslash i}} \exp\left(\frac{d}{2} \frac{\alpha_i \mathbf{x}_i^T(\boldsymbol{\nu}^{\backslash i} \circ \mathbf{x}_i) + \alpha_i}{\mathbf{x}_i^T \boldsymbol{\mu}^{\text{new}} + \alpha_i}\right) , \tag{C.6}$$

C.2 Derivation of s_{n+i}

In this section we show how to compute the value of s_{n+i}, as defined in (7.43). From the definition of $\tilde{t}_i = Z_i \mathcal{Q}^{\text{new}} / \mathcal{Q}^{\backslash i}$ and using (A.10) and (A.46)

$$\begin{aligned} s_{n+i} = Z_{n+i} &\left(\rho_i^{\text{new}} / \rho_i^{\backslash n+i} + (1 - \rho_i^{\text{new}})/(1 - \rho_i^{\backslash n+i})\right) \sqrt{\frac{\nu_i^{\backslash n+i}}{\nu_i^{\text{new}}}} \\ &\exp\left(-\frac{1}{2}\left((\mu_i^{\text{new}})^2(\nu_i^{\text{new}})^{-1} - (\mu_i^{\backslash n+i})^2(\nu_i^{\backslash n+i})^{-1} - m_{n+i}^2 v_{n+i}^{-1}\right)\right) . \end{aligned} \tag{C.7}$$

Because $(\nu_i^{\text{new}})^{-1} = (\nu_i^{\backslash n+i})^{-1} + v_{n+i}^{-1}$, which can be derived from (7.40), is easy to show that

$$\sqrt{\frac{\nu_i^{\backslash n+i}}{\nu_i^{\text{new}}}} = \sqrt{1 + \nu_i^{\backslash n+i} v_{n+i}^{-1}} . \tag{C.8}$$

In addition,

$$\begin{aligned}\rho_i^{\text{new}} / \rho_i^{\backslash n+i} + (1 - \rho_i^{\text{new}})/(1 - \rho_i^{\backslash n+i}) &= 1 + \frac{\mathcal{G}_1 - \mathcal{G}_0}{Z_{n+i}}(1 - \rho_i^{\backslash n+i}) + 1 - \frac{\mathcal{G}_1 - \mathcal{G}_0}{Z_{n+i}} \rho_i^{\backslash n+i} \\ &= \frac{\mathcal{G}_1 + \mathcal{G}_0}{Z_{n+i}} , \end{aligned} \tag{C.9}$$

where we have used (7.29) and (7.18). Next, we compute

$$\begin{aligned}(\mu_i^{\text{new}})^2(\nu_i^{\text{new}})^{-1} &= \left(\mu_i^{\setminus n+i} + c_1\nu_i^{\setminus n+i}\right)^2 \left((\nu_i^{\setminus n+i})^{-1} + v_{n+i}^{-1}\right) \\ &= (\mu_i^{\setminus n+i})^2(\nu_i^{\setminus n+i})^{-1} + 2\mu_i^{\setminus n+i}c_1 + c_1^2\nu_i^{\setminus n+i} + \\ &\quad (\mu_i^{\setminus n+i})^2 v_{n+i}^{-1} + 2c_1\mu_i^{\setminus n+i}\nu_i^{\setminus n+i}v_{n+i}^{-1} + c_1^2(\nu_i^{\setminus n+i})^2 v_{n+i}^{-1}\,, \end{aligned} \tag{C.10}$$

where we have used (7.27) and that $(\nu_i{}^{\text{new}})^{-1} = (\nu_i^{\setminus n+i})^{-1} + v_{n+i}^{-1}$. Furthermore,

$$\begin{aligned} m_{n+i}^2 v_{n+i}^{-1} &= \left(\mu_i^{\setminus n+i} + c_1(v_{n+i} + \nu_i^{\setminus n+i})\right)^2 v_{n+i}^{-1} \\ &= (\mu_i^{\setminus n+i})^2 v_{n+i}^{-1} + 2c_1\mu_i^{\setminus n+i} + 2c_1\mu_i^{\setminus n+i}\nu_i^{\setminus n+i}v_{n+i}^{-1} + \\ &\quad c_1^2 v_{n+i} + 2c_1^2\nu_i^{\setminus n+i} + c_1^2(\nu_i^{\setminus n+i})^2 v_{n+i}^{-1}\,, \end{aligned} \tag{C.11}$$

where we have used (7.41). With these values, i.e. (C.10) and (C.11) we have that

$$\begin{aligned}(\mu_i^{\text{new}})^2(\nu_i^{\text{new}})^{-1} - (\mu_i^{\setminus n+i})^2(\nu_i^{\setminus n+i})^{-1} - m_{n+i}^2 v_{n+i}^{-1} &= -c_1^2\nu_i^{\setminus n+i} - c_1^2 v_{n+i} \\ &= -\frac{c_1^2}{c_3}\,, \end{aligned} \tag{C.12}$$

where we have used (7.40). Finally,

$$s_{n+i} = (\mathcal{G}_1 + \mathcal{G}_0)\sqrt{1 + \nu_i^{\setminus n+i}v_{n+i}^{-1}}\exp\left(\frac{1}{2}\frac{c_1^2}{c_3}\right)\,. \tag{C.13}$$

Bibliography

Abramowitz, M. and Stegun, I. A. (1964). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, ninth dover printing, tenth GPO printing edition.

Adler, S. L. (1981). Over-relaxation method for the Monte Carlo evaluation of the partition function for multiquadratic actions. *Physical Review D*, 23(12):2901–2904.

Aizerman, A., Braverman, E. M., and Rozoner, L. I. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837.

Alizadeh, A. A., Eisen, M. B., Davis, R. E., Ma, C., Lossos, I. S., Rosenwald, A., Boldrick, J. C., Sabet, H., Tran, T., Yu, X., Powell, J. I., Yang, L., Marti, G. E., Moore, T., Hudson, J., Lu, L., Lewis, D. B., Tibshirani, R., Sherlock, G., Chan, W. C., Greiner, T. C., Weisenburger, D. D., Armitage, J. O., Warnke, R., Levy, R., Wilson, W., Grever, M. R., Byrd, J. C., Botstein, D., Brown, P. O., and Staudt, L. M. (2000). Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling. *Nature*, 403(6769):503–511.

Alon, U., Barkai, N., Notterman, D. A., Gish, K., Ybarra, S., Mack, D., and Levine, A. J. (1999). Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Science of the United States of America*, 96(12):6745–6750.

Alpaydin, E. and Kaynak, C. (1998). Cascading classifiers. *Kybernetika*, 34:369–374.

Ambroise, C. and McLachlan, G. J. (2002). Selection bias in gene extraction on the basis of microarray gene-expression data. *Proceedings of the National Academy of Sciences of the United States of America*, 99(10):6562–6566.

Amit, Y. and Geman, D. (1997). Shape quantization and recognition with randomized trees. *Neural Computation*, 9(7):1545–1588.

Asuncion, A. and Newman, D. (2007). UCI machine learning repository. Available at http://www.ics.uci.edu/~mlearn/MLRepository.html.

Attias, H. (2000). A variational Bayesian framework for graphical models. In *In Advances in Neural Information Processing Systems*, volume 12, pages 209–215. MIT Press.

Auda, G., Kamel, M., and Raafat, H. (1995). Voting schemes for cooperative neural network classifiers. In *IEEE International Conference on Neural Networks*, volume 3, pages 1240–1243.

Avnimelech, R. and Intrator, N. (1999). Boosting regression estimators. *Neural Computation*, 11(2):499–520.

Bacauskiene, M., Verikas, A., Gelzinis, A., and Valincius, D. (2009). A feature selection technique for generation of classification committees and its application to categorization of laryngeal images. *Pattern Recognition*, 42(5):645–654.

Bae, K. and Mallick, B. K. (2004). Gene selection using a two-level hierarchical Bayesian model. *Bioinformatics*, 20(18):3423–3430.

Bahler, D. and Navarro, L. (2000). Methods for combining heterogeneous sets of classifiers. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI 2000), Workshop on New Research Problems for Machine Learning*, Austin, TX, USA.

Bakker, B. and Heskes, T. (2003). Clustering ensembles of neural network models. *Neural Networks*, 16(2):261–269.

Banfield, R. E., Hall, L. O., Bowyer, K. W., and Kegelmeyer, W. P. (2005). Ensemble diversity measures and their application to thinning. *Information Fusion*, 6(1):49–62.

Banfield, R. E., Hall, L. O., Bowyer, K. W., and Kegelmeyer, W. P. (2007). A comparison of decision tree ensemble creation techniques. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(1):173–180.

Barber, D. and Williams, C. K. I. (1997). Gaussian processes for Bayesian classification via hybrid Monte Carlo. In Mozer, M. C., Jordan, M. I., and Petsche, T., editors, *Advances in Neural Information Processing Systems*, volume 9, pages 340–346. The MIT Press.

Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139.

Bernard, S., Heutte, L., and Adam, S. (2009). Influence of hyperparameters on random forest accuracy. In Benediktsson, J. A., Kittler, J., and Roli, F., editors, *Proceedings of the 8th International Workshop on Multiple Classifier Systems*, volume 5519 of *Lecture Notes in Computer Science*, pages 171–180. Springer.

Bishop, C. and Svensen, M. (2003). Bayesian hierarchical mixtures of experts. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 57–64, San Francisco, CA. Morgan Kaufmann.

Bishop, C. M. (1996). *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, UK.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer.

Bishop, C. M., Spiegelhalter, D. J., and Winn, J. M. (2002). VIBES: A variational inference engine for Bayesian networks. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems*, volume 15, pages 777–784. The MIT Press.

Blei, D. M. and Jordan, M. I. (2006). Variational inference for Dirichlet process mixtures. *Bayesian Analysis*, 1(1):121–144.

Bø, T. and Jonassen, I. (2002). New feature subset selection procedures for classification of expression profiles. *Genome Biology*, 3(4):1–11.

Borchers, B. (1999). CSDP : A C library for semidefinite programming. *Optimization Methods and Software*, 11:613–623.

Bourquin, J.-P., Subramanian, A., Langebrake, C., Reinhardt, D., Bernard, O., Ballerini, P., and Baruchel, A. (2006). Identification of distinct molecular phenotypes in acute megakaryoblastic leukemia by gene expression profiling. *Proceedings of the National Academy of Sciences of the United States of America*, 103:3339–3344.

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

Breiman, L. (1996b). Bias, variance, and arcing classifiers. Technical Report 460, Statistics Department, University of California.

Breiman, L. (1996c). Out-of-bag estimation. Technical report, Statistics Department, University of California.

Breiman, L. (1996d). Stacked regressions. *Machine Learning*, 24(1):49–64.

Breiman, L. (1998). Arcing classifiers. *The Annals of Statistics*, 26(3):801–824.

Breiman, L. (1999a). Prediction games and arcing algorithms. *Neural Computation*, 11(7):1493–1517.

Breiman, L. (1999b). Using adaptive bagging to debias regressions. Technical report, Statistics Department, University of California.

Breiman, L. (2000). Randomizing outputs to increase prediction accuracy. *Machine Learning*, 40(3):229–242.

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

Breiman, L. (2002). *Manual On Setting Up, Using, And Understanding Random Forests V3.1*. Available online at
http://oz.berkeley.edu/users/breiman/Using_random_forests_V3.1.pdf.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Chapman & Hall, New York.

Brown, G., Wyatt, J. L., Harris, R., and Yao, X. (2005a). Diversity creation methods: A survey and categorisation. *Information Fusion*, 6(1):5–20.

Brown, G., Wyatt, J. L., and Tino, P. (2005b). Managing diversity in regression ensembles. *Journal of Machine Learning Research*, 6:1621–1650.

Bryll, R., Gutierrez-Osuna, R., and Quek, F. (2003). Attribute bagging: Improving accuracy of classifier ensembles by using random feature subsets. *Pattern Recognition*, 36(6):1291–1302.

Bühlmann, P. (2003). Bagging, subagging and bragging for improving some prediction algorithms. In Akritas, M. and Politis, D., editors, *Recent Advances and Trends in Nonparametric Statistics*, pages 19–34.

Bylander, T. (2002). Estimating generalization error on two-class datasets using out-of-bag estimates. *Machine Learning*, 48(1-3):287–297.

Canuto, A. M. P., Abreu, M. C. C., de Melo Oliveira, L., Jo ao C. Xavier, J., and de M. Santos, A. (2007). Investigating the influence of the choice of the ensemble members in accuracy and diversity of selection-based and fusion-based methods for ensembles. *Pattern Recognition Letters*, 28(4):472–486.

Caruana, R. and Niculescu-Mizil, A. (2004). Ensemble selection from libraries of models. In *Proceedings of the 21st International Conference on Machine Learning*, New York, NY, USA. ACM Press.

Cawley, G. C. and Talbot, N. L. C. (2006). Gene selection in cancer classification using sparse logistic regression with Bayesian regularization. *Bioinformatics*, 22(19):2348–2355.

Chambers, J. M. and Hastie, T. J. (1992). *Statistical Models in S*. Chapman & Hall, London.

Chan, P., Zeng, X., Tsang, E., Yeung, D., and Lee, J. (2007). Neural network ensemble pruning using sensitivity measure in web applications. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 3051–3056.

Chang, C.-C. and Lin, C.-J. (2001). *LIBSVM: A library for support vector machines*. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Chen, H., Tino, P., and Yao, X. (2006). A probabilistic ensemble pruning algorithm. In *Proceedings of the sixth IEEE International conference on Data Mining*, pages 878–882, Washington, DC, USA. IEEE Computer Society.

Cherkauer, K. (1996). Human expert-level performance on a scientific image analysis task by a system using combined artificial neural networks. In Chan, P., Stolfo., S., and Wolpert, D., editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 15–21, Portland, OR. AAAI Press. Workshop on Integrated Multiple Learned Models for Improving and Scaling Machine Learning Algorithms.

Cormen, T. H., Leiserson, C. H., and Rivest, R. L. (1990). *Introduction to Algorithms*. The MIT Press.

Cowell, R. G., Dawid, A. P., and Sebastiani, P. (1996). A comparison of sequential learning methods for incomplete data. In Bernardo, J. M., Berger, J. O., Dawid, A. P., and Smith, A. F. M., editors, *Bayesian Statistics 5*, pages 553–541. Oxford University Press.

Cox, R. T. (1946). Probability, frequency and reasonable expectation. *American Journal of Physics*, 14:1–13.

Csató, L. and Opper, M. (2002). Sparse online Gaussian processes. *Neural Computation*, 14(6):641–692.

Csató, L., Opper, M., and Winther, O. (2001). TAP Gibbs free energy, belief propagation and sparsity. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems*, volume 14, pages 657–663. The MIT Press.

Demir, C. and Alpaydin, E. (2005). Cost-conscious classifier ensembles. *Pattern Recognition Letters*, 26(14):2206–2214.

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

Dettling, M. (2004). Bagboosting for tumor classification with gene expression data. *Bioinformatics*, 20(18):3583–3593.

Dettling, M. and Bühlmann, P. (2002). Supervised clustering of genes. *Genome Biology*, 3:1–15.

Díaz-Uriarte, R. and Alvarez de Andrés, S. (2006). Gene selection and classification of microarray data using random forest. *BMC Bioinformatics*, 7(1):3.

Dietterich, T. G. (1998). Machine-learning research: Four current directions. *The AI Magazine*, 18(4):97–136.

Dietterich, T. G. (2000a). Ensemble methods in machine learning. In Kittler, J. and Roli, F., editors, *Proceedings of the First International Workshop on Multiple Classifier Systems*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15. Springer.

Dietterich, T. G. (2000b). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157.

Dietterich, T. G. and Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286.

Dietterich, T. G. and Kong, E. (1995). Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Oregon State University, Covallis, OR.

Domingos, P. (1997). Knowledge acquisition from examples via multiple models. In Fisher, D. H., editor, *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 98–106, Nashville, TN. Morgan Kaufmann.

Domingos, P. (2000). A unified bias-variance decomposition and its applications. In Langley, P., editor, *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 231–238. Morgan Kaufmann, San Francisco, CA.

Dougherty, E. R. (2001). Small sample issues for microarray-based classification. *Comparative and Functional Genomics*, 2(1):28–34.

Drucker, H. (1997). Improving regressors using boosting techniques. In Fisher, D. H., editor, *Proceedings of the Fourteenth International Conference on Machine Learning*, Nashville, Tennessee. Morgan Kaufmann.

Dudoit, S. and Fridlyand, J. (2003). *Statistical Analysis of Gene Expression Microarray Data*, chapter 3. CRC Press.

Dudoit, S., Fridlyand, J., and Speed, T. P. (2002). Comparison of discrimination methods for the classification of tumors using gene expression data. *Journal of the American Statistical Association*, 97:77–87.

Duin, R. P. W. and Tax, D. M. J. (2000). Experiments with classifier combining rules. In Kittler, J. and Roli, F., editors, *Proceedings of the First International Workshop on Multiple Classifiers Systems*, volume 1857 of *Lecture Notes in Computer Science*, pages 16–29, London, UK. Springer.

Efron, B. and Tibshirani, R. J. (1994). *An Introduction to the Bootstrap*. Chapman & Hall/CRC.

Esposito, R. and Saitta, L. (2003). Monte Carlo theory as an explanation of bagging and boosting. In *Proceeding of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 499–504. Morgan Kaufmann.

Fan, W., Chu, F., Wang, H., and Yu, P. S. (2002). Pruning and dynamic scheduling of cost-sensitive ensembles. In *Eighteenth National Conference on Artificial Intelligence*, pages 146–151, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Fern, X. Z. and Brodley, C. E. (2003). Random projection for high dimensional data clustering: A cluster ensemble approach. In Fawcett, T. and Mishra, N., editors, *Proceedings of the Twentieth International Conference on Machine Learning*, pages 186–193, Washington, DC, USA. AAAI Press.

Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285.

Freund, Y. (2009). A more robust boosting algorithm. Unpublished. Available online at: http://www.citebase.org/abstract?id=oai:arXiv.org:0905.2138.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–374.

Friedman, J. H. (1991). Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–67.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232.

Fumera, G., Fabio, R., and Alessandra, S. (2008). A theoretical analysis of bagging as a linear combination of classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(7):1293–1299.

Fumera, G. and Roli, F. (2005). A theoretical and experimental analysis of linear combiners for multiple classifier systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(6):942–956.

Fürnkranz, J. (2002). Round robin classification. *Journal of Machine Learning Research*, 2:721–747.

Gama, J. and Brazdil, P. (2000). Cascade generalization. *Machine Learning*, 41(3):315–343.

Gardner, T. S. and Faith, J. J. (2005). Reverse-engineering transcription control networks. *Physics of Life Reviews*, 2(1):65–88.

Garey, M. R. and Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

George, E. I. and McCulloch, R. E. (1997). Approaches for Bayesian variable selection. *Statistica Sinica*, 7(2):339–373.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 36(1):3–42.

Giacinto, G. and Roli, F. (2001a). An approach to the automatic design of multiple classifier systems. *Pattern Recognition Letters*, 22(1):25–33.

Giacinto, G. and Roli, F. (2001b). Dynamic classifier selection based on multiple classifier behaviour. *Pattern Recognition*, 34(9):1879–1881.

Giacinto, G., Roli, F., and Fumera, G. (2000). Design of effective multiple classifier systems by clustering of classifiers. In *Proceedings of the 15th International Conference on Pattern Recognition*, volume 4, pages 2160–2163, Barcelona, Spain. IEEE Computer Society.

Gibbs, M. N. and MacKay, D. J. C. (2000). Variational Gaussian process classifiers. *IEEE Transactions on Neural Networks*, 11(6):1458–1464.

Gill, P. E., Golub, G. H., Murray, W., and Saunders, M. A. (1974). Methods for modifying matrix factorizations. *Mathematics of Computation*, 28(126):505–535.

Goemans, M. X. and Williamson, D. P. (1995). Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145.

Golub, T. R., Slonim, D. K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J. P., Coller, H., Loh, M. L., Downing, J. R., Caligiuri, M. A., and Bloomfield, C. D. (1999). Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537.

Gómez-Verdejo, V., Arenas-García, J., and Figueiras-Vidal, A. (2008). A dynamically adjusted mixed emphasis method for building boosting ensembles. *IEEE Transactions on Neural Networks*, 19(1):3–17.

Gómez-Verdejo, V., Ortega-Moral, M., Arenas-García, J., and Figueiras-Vidal, A. R. (2006). Boosting by weighting critical and erroneous samples. *Neurocomputing*, 69(7-9):679 – 685. New Issues in Neurocomputing: 13th European Symposium on Artificial Neural Networks.

Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422.

Hall, P. and Samworth, R. J. (2005). Properties of bagged nearest neighbour classifiers. *Journal of the Royal Statistical Society Series B*, 67(3):363–379.

Han, Q., Ye, Y., and Zhang, J. (2002). An improved rounding method and semidefinite programming relaxation for graph partition. *Mathematical Programming*, 92:509–535.

Hansen, L. and Salamon, P. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001.

Hashem, S. (1993). *Optimal combinations of neural networks*. PhD thesis, Purdue University, School of Engineering, Lafayette, IN.

Hastie, T., Tibshirani, R., and Friedman, J. H. (2001). *The Elements of Statistical Learning*. Springer.

Hedenfalk, I., Duggan, D., Chen, Y., Radmacher, M., Bittner, M., Simon, R., Meltzer, P., Gusterson, B., Esteller, M., Raffeld, M., Yakhini, Z., Ben-Dor, A., Dougherty, E., Kononen, J., Bubendorf, L., Fehrle, W., Pittaluga, S., Gruvberger, S., Loman, N., Johannsson, O., Olsson, H., Wilfond, B., Sauter, G., Kallioniemi, O.-P., Borg, A., and Trent, J. (2001). Gene-Expression Profiles in Hereditary Breast Cancer. *The New England Journal of Medicine*, 344(8):539–548.

Herbrich, R., Graepel, T., and Campbell, C. (2001). Bayes point machines. *Journal of Machine Learning Research*, 1:245–279.

Hernández-Lobato, D. (2008). Sparse Bayes machines for binary classification. In Kurková, V., Neruda, R., and Koutník, J., editors, *Proceedings of the 18th International Conference on Artificial Neural Networks*, volume 5163 of *Lecture Notes in Computer Science*, pages 205–214. Springer.

Hernández-Lobato, D., Hernández-Lobato, J. M., , and Suárez, A. (2009). Expectation propagation for microarray data classification. Submitted for publication.

Hernández-Lobato, D. and Hernández-Lobato, J. M. (2008). Bayes machines for binary classification. *Pattern Recognition Letters*, 29(10):1466–1473.

Hernández-Lobato, D., Hernández-Lobato, J. M., Ruiz-Torrubiano, R., and Valle, Á. (2006a). Pruning adaptive boosting ensembles by means of a genetic algorithm. In Corchado, E., Yin, H., Botti, V. J., and Fyfe, C., editors, *Proceedings of the 7th International Conference on Intelligent Data Engineering and Automated Learning*, volume 4224 of *Lecture Notes in Computer Science*, pages 322–329. Springer.

Hernández-Lobato, D., Martínez-Muñoz, G., and Suárez, A. (2009). Statistical instance-based pruning in ensembles of independent classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):364–369.

Hernández-Lobato, D., Martínez-Muñoz, G., and Suárez, A. (2006b). Pruning in ordered regression bagging ensembles. In *International Joint Conference on Neural Networks*, pages 1266–1273. IEEE.

Hernández-Lobato, D., Martínez-Muñoz, G., and Suárez, A. (2007). Out of bootstrap estimation of generalization error curves in bagging ensembles. In Yin, H., Tiño, P., Corchado, E., Byrne, W., and Yao, X., editors, *Proceedings of the 8th International Conference on Intelligent Data Engineering and Automated Learning*, volume 4881 of *Lecture Notes in Computer Science*, pages 47–56. Springer.

Hernández-Lobato, D., Martínez-Muñoz, G., and Suárez, A. (2009a). How large should ensembles of classifiers be? Submitted for publication.

Hernández-Lobato, D., Martínez-Muñoz, G., and Suárez, A. (2009b). Pruning regression bagging ensembles. Submitted for publication.

Hernández-Lobato, J. M., Dijkstra, T., and Heskes, T. (2008). Regulator discovery from gene expression time series of malaria parasites: A hierarchical approach. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems 20*, pages 649–656. The MIT Press.

Heskes, T. and Zoeter, O. (2002). Expectation propagation for approximate inference in dynamic Bayesian networks. In Darwiche, A. and Friedman, N., editors, *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence*, pages 216–223, Edmonton, Alberta, Canada. Morgan Kaufmann.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844.

Ho, T. K., Hull, J., and Srihari, S. (1994). Decision combination in multiple classifier systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(1):66–75.

Husmeier, D. and Althoefer, K. (1998). Modelling conditional probabilities with network committees: How overfitting can be useful. *Neural Network World*, 8:417–439.

Jaakkola, T. S. (2001). Tutorial on variational approximation methods. In Opper, M. and Saad, D., editors, *Advances in Mean Field Methods: Theory and Practice*, pages 129–159. MIT Press.

Jaakkola, T. S. and Jordan, M. I. (2000). Bayesian parameter estimation via variational methods. *Statistics and Computing*, 10(1):25–37.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87.

Jain, A. K., Duin, R. P. W., and Jianchang, M. (2000). Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37.

Johnson, N. L., Kotz, S., and Balakrishnan, N. (1997). *Discrete Multivariate Distributions*. John Wiley & Sons.

Jordan, M. I. and Jacobs, R. A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214.

Khan, J., Wei, J. S., Ringner, M., Saal, L. H., Ladanyi, M., Westermann, F., Berthold, F., Schwab, M., Antonescu, C. R., Peterson, C., and Meltzer, P. S. (2001). Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. *Nature Medicine*, 7(6):673–679.

Kim, H.-C. and Ghahramani, Z. (2006). Bayesian Gaussian process classification with the EM-EP algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(12):1948–1959.

Kim, H.-C., Pang, S., Je, H.-M., Kim, D., and Bang, S. Y. (2003). Constructing support vector machine ensemble. *Pattern Recognition*, 36(12):2757–2767.

Kittler, J. (1998). Combining classifiers: A theoretical framework. *Pattern Analysis & Applications*, 1(1):18–27.

Kittler, J., Hatef, M., Duin, R., and Matas, J. (1998). On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):226–239.

Kolen, J. F. and Pollack, J. B. (1991). Back propagation is sensitive to initial conditions. In Lippmann, R., Moody, J. E., and Touretzky, D. S., editors, *Advances in neural information processing systems*, volume 3, pages 860–867. Morgan Kaufmann.

Koren, Y. (2009). The Belkor solution to the Netflix grand prize. Online available at: http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf.

Kotsiantis, S., Zaharakis, I., and Pintelas, P. (2006). Machine learning: A review of classification and combining techniques. *Artificial Intelligence Review*, 26(3):159–190.

Krishnapuram, B., Carin, L., and Hartemink, A. J. (2004). Joint classifier and feature optimization for comprehensive cancer diagnosis using gene expression data. *Journal of Computational Biology*, 11(2-3):227–242.

Krogh, A. and Hertz, J. A. (1992). A simple weight decay can improve generalization. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems*, volume 4, pages 950–957. Morgan Kaufmann.

Krogh, A. and Vedelsby, J. (1995). Neural network ensembles, cross validation, and active learning. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. The MIT Press.

Kuncheva, L. (1993). Genetic algorithm for feature selection for parallel classifiers. *Information Processing Letters*, 46(4):163–168.

Kuncheva, L., Whitaker, C., Shipp, C., and Duin, R. (2003). Limits on the majority vote accuracy in classifier fusion. *Pattern Analysis and Applications*, 6(1):22–31.

Kuncheva, L. I. (2004). *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience.

Kuncheva, L. I. (2007). A stability index for feature selection. In *Proceedings of the 25th IASTED International Multi-Conference on Artificial Intelligence and Applications*, pages 390–395, Anaheim, CA, USA. ACTA Press.

Kuncheva, L. I., Bezdek, J. C., and Duin, R. P. W. (2001). Decision templates for multiple classifier fusion: An experimental comparison. *Pattern Recognition*, 34(2):299–314.

Kung, F. H. (1986). Fitting logistic growth curve with predetermined carrying capacity. In *Proceedings of the Statistical Computing Section*, pages 340–343. American Statistical Association.

Kuss, M. and Rasmussen, C. E. (2005). Assessing approximate inference for binary Gaussian process classification. *Journal of Machine Learning Research*, 6:1679–1704.

Lam, L. and Suen, C. (1997). Application of majority voting to pattern recognition: An analysis of its behavior and performance. *IEEE Transactions on System, Man and Cybernetics*, 27(5):553–568.

Latinne, P., Debeir, O., and Decaestecker, C. (2001). Limiting the number of trees in random forests. In Kittler, J. and Roli, F., editors, *Proceedings of the Second International Workshop on Multiple Classifier Systems*, volume 2096 of *Lecture Notes in Computer Science*, pages 178–187. Springer-Verlag.

Lawrence, N., Seeger, M., and Herbrich, R. (2003). Fast sparse Gaussian process methods: The informative vector machine. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*, pages 609–616. MIT Press, Cambridge, MA.

Lawrence, N. D. (2000). *Variational Inference in Probabilistic Models*. PhD thesis, University of Cambridge, UK.

Lazarevic, A. and Obradovic, Z. (2001). Effective pruning of neural network classifier ensembles. In *International Joint Conference on Neural Networks*, volume 2, pages 796–801. IEEE.

LeBlanc, M. and Tibshirani, R. (1993). Combining estimates in regression and classification. Technical Report 9318, Dept. of Statistics, University of Toronto.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lee, J. W., Lee, J. B., Park, M., and Song, S. H. (2005). An extensive comparison of recent classification tools applied to microarray data. *Computational Statistics and Data Analysis*, 48(4):869–885.

Lee, K. E., Sha, N., Dougherty, E. R., Vannucci, M., and Mallick, B. K. (2003). Gene selection: A Bayesian variable selection approach. *Bioinformatics*, 19(1):90–97.

Leisch, F. and Dimitriadou, E. (2007). Mlbench: Machine Learning Benchmark Problems. R package version 1.1-3.

Li, C. and Li, H. (2008). Network-constrained regularization and variable selection for analysis of genomic data. *Bioinformatics*, 24(9):1175–1182.

Li, Y., Campbell, C., and Tipping, M. (2002). Bayesian automatic relevance determination algorithms for classifying gene expression data. *Bioinformatics*, 18(10):1332–1339.

Liu, Y. and Yao, X. (1999). Ensemble learning via negative correlation. *Neural Networks*, 12(10):1399–1404.

MacKay, D. J. C. (2003). *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, Cambridge, UK.

Margineantu, D. D. and Dietterich, T. G. (1997). Pruning adaptive boosting. In Fisher, D. H., editor, *Proceedings of the 14th International Conference on Machine Learning*, pages 211–218. Morgan Kaufmann.

Martínez-Muñoz, G., Hernández-Lobato, D., and Suárez, A. (2007). Selection of decision stumps in bagging ensembles. In Marques de Sá, J., Alexandre, L. A., Duch, W., and Mandic, D. P., editors, *Proceedings of the 17th International Conference on Artificial Neural Networks*, volume 4668 of *Lecture Notes in Computer Science*, pages 319–328. Springer.

Martínez-Muñoz, G., Hernández-Lobato, D., and Suárez, A. (2009). An analysis of ensemble pruning techniques based on ordered aggregation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31:245–259.

Martínez-Muñoz, G., Sánchez-Martínez, A., Hernández-Lobato, D., and Suárez, A. (2008). Class-switching neural network ensembles. *Neurocomputing*, 71(13-15):2521–2528.

Martínez-Muñoz, G. and Suárez, A. (2004). Aggregation ordering in bagging. In *Proceedings of the IASTED International Conference on Artificial Intelligence and Applications*, pages 258–263. Acta Press.

Martínez-Muñoz, G. and Suárez, A. (2005). Switching class labels to generate classification ensembles. *Pattern Recognition*, 38(10):1483–1494.

Martínez-Muñoz, G. and Suárez, A. (2006). Pruning in ordered bagging ensembles. In Cohen, W. W. and Moore, A., editors, *Proceedings of the 23rd international conference on Machine learning*, volume 148, pages 609–616, New York, USA. ACM Press.

Martínez-Muñoz, G. and Suárez, A. (2007). Using boosting to prune bagging ensembles. *Pattern Recognition Letters*, 28(1):156–165.

Meir, R. and Rätsch, G. (2003). An introduction to boosting and leveraging. In Mendelson, S. and Smola, A. J., editors, *Machine Learning Summer School 2002, Advanced Lectures on Machine Learning*, volume 2600 of *Lecture Notes in Computer Science*, pages 118–183. Springer.

Meynet, J. and Thiran, J.-P. (2007). Information theoretic combination of classifiers with application to Adaboost. In Haindl, M., Kittler, J., and Roli, F., editors, *Proceedings of the 7th International Workshop on Multiple Classifier Systems*, volume 4472 of *Lecture Notes in Computer Science*, pages 171–179. Springer.

Minka, T. (2001a). Expectation propagation for approximate Bayesian inference. In Breese, J. S. and Koller, D., editors, *Proceedings of the 17th Annual Conference on Uncertainty in Artificial Intelligence*, pages 362–36, San Francisco, CA. Morgan Kaufmann.

Minka, T. (2001b). *A Family of Algorithms for approximate Bayesian Inference*. PhD thesis, Massachusetts Institute of Technology.

Minka, T., Xiang, R., and Qi, A. (2009). Virtual vector machine for Bayesian online classification. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*.

Narasimhamurthy, A. (2005). Theoretical bounds of majority voting performance for a binary classification problem. *IEEE Transactions Pattern Analysis Machine Intelligence*, 27(12):1988–1995.

Neal, R. M. (1993). Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Dept. of Computer Science, University of Toronto.

Neal, R. M. (1997). Monte Carlo implementation of Gaussian process models for Bayesian regression and classification. Technical Report CRG-TR-97-2, Dept. of Computer Science, University of Toronto.

Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization*. Springer.

Opitz, D. and Maclin, R. (1999). Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198.

Opper, M. (1998). *On-Line Learning in Neural Networks*, chapter A Bayesian Approach to Online Learning, pages 363–378. Cambridge University Press, New York, NY, USA.

Opper, M. and Winther, O. (2000a). Gaussian process classification and SVM: Mean field results. In Bartlett, P., B.Schoelkopf, Schuurmans, D., and Smola, A., editors, *Advances in large margin classifiers*, pages 43–65. MIT Press.

Opper, M. and Winther, O. (2000b). Gaussian processes for classification: Mean-field algorithms. *Neural Computation*, 12(11):2655–2684.

Opper, M. and Winther, O. (2005). Expectation consistent approximate inference. *Journal of Machine Learning Research*, 6:2177–2204.

Ortega, J., Koppel, M., and Argamon, S. (2001). Arbitrating among competing classifiers using learned referees. *Knowledge and Information Systems*, 3(4):470–490.

Partalas, I., Tsoumakas, G., Hatzikos, E. V., and Vlahavas, I. (2008). Greedy regression ensemble selection: Theory and an application to water quality prediction. *Information Sciences*, 178(20):3867–3879. Special Issue on Industrial Applications of Neural Networks, 10th Engineering Applications of Neural Networks 2007.

Partalas, I., Tsoumakas, G., Katakis, I., and Vlahavas, I. P. (2006). Ensemble pruning using reinforcement learning. In Antoniou, G., Potamias, G., Spyropoulos, C., and Plexousakis, D., editors, *Proceedings of the 4th Helenic Conference in Advances in Artificial Intelligence*, volume 3955 of *Lecture Notes in Computer Science*, pages 301–310. Springer.

Perrone, M. P. and Cooper, L. N. (1993). When networks disagree: Ensemble methods for hybrid neural networks. In Mammone, R. J., editor, *Neural Networks for Speech and Image Processing*, pages 126–142. Chapman-Hall.

Piotte, M. and Chabbert, M. (2009). The Pragmatic Theory solution to the Netflix grand prize. Online available at:
http://www.netflixprize.com/assets/GrandPrize2009_BPC_PragmaticTheory.pdf.

Politis, D., Romano, J., and Wolf, M. (1999). *Subsampling*. Springer-Verlang, New York.

Pomeroy, S. L., Tamayo, P., Gaasenbeek, M., Sturla, L. M., Angelo, M., McLaughlin, M. E., Kim, J. Y. H., Goumnerova, L. C., Black, P. M., Lau, C., Allen, J. C., Zagzag, D., Olson, J. M., Curran, T., Wetmore, C., Biegel, J. A., Poggio, T., Mukherjee, S., Rifkin, R., Califano, A., Stolovitzky, G., Louis, D. N., Mesirov, J. P., Lander, E. S., and Golub, T. R. (2002). Prediction of central nervous system embryonal tumour outcome based on gene expression. *Nature*, 415(6870):436–442.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA.

Prodromidis, A. L. and Stolfo, S. J. (2001). Cost complexity-based pruning of ensemble classifiers. *Knowledge and Information Systems*, 3(4):449–469.

Pudil, P., Novovicova, J., Blaha, S., and Kittler, J. (1992). Multistage pattern recognition with reject option. In *Proceedings of the 11th IAPR International Conference on Pattern Recognition*, volume 2, pages 92–95. IEEE Computer Society Press.

Puuronen, S., Terziyan, V. Y., and Tsymbal, A. (1999). A dynamic integration algorithm for an ensemble of classifiers. In Ras, Z. W. and Skowron, A., editors, *Proceedings of the 11th International Symposium on Foundations of Intelligent Systems*, volume 1609 of *Lecture Notes in Computer Science*, pages 592–600, London, UK. Springer.

Quinlan, J. R. (1996). Bagging, boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Conference on Innovative Applications of Artificial Intelligence*, volume 1, pages 725–730. AAAI Press / The MIT Press.

R Development Core Team (2005). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Ramaswamy, S., Ross, K. N., Lander, E. S., and Golub, T. R. (2003). A molecular signature of metastasis in primary solid tumors. *Nature Genetics*, 33:49 – 54.

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

Rätsch, G., Warmuth, M. K., and Shawe-Taylor, J. (2005). Efficient margin maximizing with boosting. *Journal of Machine Learning Research*, 6(12):2131 – 2152.

Ridgeway, G., Madigan, D., and Richardson, T. (1999). Boosting methodology for regression problems. *Artificial Intelligence and Statistics*, pages 152–161.

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

Rodríguez, J. J., Kuncheva, L. I., and Alonso, C. J. (2006). Rotation forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1619–1630.

Rosen, B. (1996). Ensemble learning using decorrelated neural networks. *Connection Science*, 8(3):373–384.

Ruta, D. and Gabrys, B. (2002). A theoretical analysis of the limits of majority voting errors for multiple classifier systems. *Pattern Analysis and Applications*, 5(4):333–350.

Ruta, D. and Gabrys, B. (2005). Classifier selection for majority voting. *Information Fusion*, 6(1):63–81.

Schaffer, C. (1994). A conservation law for generalization performance. In Cohen and Hirsh, editors, *Proceedings of the 11th International Conference on Machine Learning*, pages 259–265. Morgan Kaufmann.

Schapire, R., Freund, Y., Bartlett, P., and Lee, W. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 12(5):1651–1686.

Schapire, R. E. (1997). Using output codes to boost multiclass learning problems. In Fisher, D. H., editor, *Proceedings of the 14th International Conference on Machine Learning*, pages 313–321. Morgan Kaufmann.

Schapire, R. E. and Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336.

Schummer, M., Ng, W. V., Bumgarner, R. E., Nelson, P. S., Schummer, B., Bednarski, D. W., Hassell, L., Baldwin, R. L., Karlan, B. Y., and Hood, L. (1999). Comparative hybridization of an array of 21 500 ovarian cDNAs for the discovery of genes overexpressed in ovarian carcinomas. *Gene*, 238(2):375–385.

Seeger, M. (2003). *Bayesian Gaussian Process Models: PAC-Bayesian Generalisation Error Bounds and Sparse Approximations*. PhD thesis, University of Edinburgh.

Seeger, M. W. (2008). Bayesian inference and optimal design for the sparse linear model. *Journal of Machine Learning Research*, 9:759–813.

Sharkey, A. J. C. (1996). On combining artificial neural nets. *Connection Science*, 8:299–314.

Sharkey, A. J. C. (1999). *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*. Springer-Verlag, London.

Sharkey, A. J. C. and Sharkey, N. E. (1997). Combining diverse neural nets. *The Knowledge Engineering Review*, 12(03):231–247.

Singh, D., Febbo, P. G., Ross, K., Jackson, D. G., Manola, J., Ladd, C., Tamayo, P., Renshaw, A. A., D'Amico, A. V., Richie, J. P., Lander, E. S., Loda, M., Kantoff, P. W., Golub, T. R., and Sellers, W. R. (2002). Gene expression correlates of clinical prostate cancer behavior. *Cancer Cell*, 1:203–209.

Skurichina, M. and Duin, R. P. W. (2002). Bagging, boosting and the random subspace method for linear classifiers. *Pattern Analysis & Applications*, 5(2):121–135.

Skurichina, M. and Duin, R. P. W. (2005). Combining feature subsets in feature selection. In Oza, N. C., Polikar, R., Kittler, J., and Roli, F., editors, *Proceedings of the 6th International Workshop on Multiple Classifier Systems*, volume 3541 of *Lecture Notes in Computer Science*, pages 165–175. Springer.

Sollich, P. and Krogh, A. (1996). Learning with ensembles: How overfitting can be useful. In Touretzky, D. S., Mozer, M., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems*, volume 8, pages 190–196. MIT Press.

Tamon, C. and Xiang, J. (2000). On the boosting pruning problem. In de Mántaras, R. L. and Plaza, E., editors, *Proceedings of the 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 404–412. Springer.

Tibshirani, R., Hastie, T., Narasimhan, B., and Chu, G. (2002). Diagnosis of multiple cancer types by shrunken centroids of gene expression. *Proceedings of the National Academy of Sciences of the United States of America*, 99(10):6567–6572.

Todorovski, L. and Džeroski, S. (2003). Combining classifiers with meta decision trees. *Machine Learning*, 50(3):223–249.

Töscher, A., Jahrero, M., and Bell, R. M. (2009). The Big Chaos solution to the Netflix grand prize. Online available at: http://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf.

Tsoumakas, G., Angelis, L., and Vlahavas, I. (2005). Selective fusion of heterogeneous classifiers. *Intelligent Data Analysis*, 9(6):511–525.

Tsoumakas, G., Katakis, I., and Vlahavas, I. . (2004). Effective voting of heterogeneous classifiers. In Boulicaut, J.-F., Esposito, F., Giannotti, F., and Pedreschi, D., editors, *Proceedings of the 15th European Conference on Machine Learning*, volume 3201 of *Lecture Notes in Computer Science*, pages 465–476. Springer.

Tsymbal, A., Pechenizkiy, M., Puuronen, S., and Patterson, D. W. (2003). Dynamic integration of classifiers in the space of principal components. In Kalinichenko, L. A., Manthey, R., Thalheim, B., and Wloka, U., editors, *Proceedings of the 7th East European Conference on Advances in Databases and Information Systems*, volume 2798 of *Lecture Notes in Computer Science*, pages 278–292. Springer.

Tsymbal, A. and Puuronen, S. (2000). Bagging and boosting with dynamic integration of classifiers. In *Principles of Data Mining and Knowledge Discovery*, pages 116–125.

Tumer, K. and Ghosh, J. (1996). Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8(3/4):385–404.

Tumer, K. and Oza, N. C. (2003). Input decimated ensembles. *Pattern Analysis & Applications*, 6(1):65–77.

Turner, K. and Oza, N. (1999). Decimated input ensembles for improved generalization. In *International Joint Confeerence on Neural Networks*, volume 5, pages 3069–3074.

Ueda, N. and Nakano, R. (1996). Generalization error of ensemble estimators. In *Proceedings of the International Conference on Neural Networks*, pages 90–95.

Valentini, G. and Masulli, F. (2002). Ensembles of learning machines. In Marinaro, M. and Tagliaferri, R., editors, *Proceedings of the 13th Italian Workshop on Neural Nets*, volume 2486 of *Lecture Notes in Computer Science*, pages 3–22. Springer.

van 't Veer, L. J., Dai, H., van de Vijver, M. J., He, Y. D., Hart, A. A., Mao, M., Peterse, H. L., van der Kooy, K., Marton, M. J., Witteveen, A. T., Schreiber, G. J., Kerkhoven, R. M., Roberts, C., Linsley, P. S., Bernards, R., and Friend, S. H. (2002). Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, 415(6871):530–536.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc.

Venables, W. N. and Ripley, B. D. (2002). *Modern Applied Statistics with S*. Springer, New York, fourth edition. ISBN 0-387-95457-0.

Wang, H., Fan, W., Yu, P. S., and Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. In *KDD '03: Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–235, New York, NY, USA. ACM.

Wang, Y., Tetko, I. V., Hall, M. A., Frank, E., Facius, A., Mayer, K. F., and Mewes, H. W. (2005). Gene selection from microarray data for cancer classification–a machine learning approach. *Computational Biology and Chemistry*, 29(1):37–46.

Webb, G. I. (2000). Multiboosting: A technique for combining boosting and wagging. *Machine Learning*, 40(2):159–196.

West, M., Blanchette, C., and Dressman, H. (2001). Predicting the clinical status of human breast cancer by using gene expression profiles. *Proceedings of National Academy of Science*, 98(20):11462–11467.

Wilcoxon, F. (1968). Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83.

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

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2nd edition edition.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, 5(2):241–259.

Wolpert, D. H. (1996). The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390.

Woods, K., Kegelmeyer, W.P., J., and Bowyer, K. (1997). Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):405–410.

Xu, L., Krzyzak, A., and Suen, C. (1992). Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Transactions on Systems, Man and Cybernetics*, 22(3):418–435.

Zhang, Y., Burer, S., and Street, W. N. (2006). Ensemble pruning via semi-definite programming. *Journal of Machine Learning Research*, 7:1315–1338.

Zhou, X., Liu, K.-Y., and Wong, S. T. (2004). Cancer classification and prediction using logistic regression with Bayesian gene selection. *Journal of Biomedical Informatics*, 37(4):249–259.

Zhou, Z.-H. and Tang, W. (2003). Selective ensemble of decision trees. In Wang, G., Liu, Q., Yao, Y., and Skowron, A., editors, *Proceedings of the 9th International Conference on Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing*, volume 2639 of *Lecture Notes in Computer Science*, pages 476–483. Springer.

Zhou, Z.-H., Wu, J., and Tang, W. (2002). Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, 137(1-2):239–263.

Zhu, Y., Shen, X., and Pan, W. (2009). Network-based support vector machine for classification of microarray samples. *BMC Bioinformatics*, 10(Suppl 1):S21.