UNIVERSITÉ DE LIÈGE
Faculté des Sciences Appliquées
Institut d'électricité Montefiore

Contributions to Decision Tree Induction:

Bias/Variance Tradeoff
and
Time Series Classification

Thèse présentée par
Pierre Geurts
en vue de l'obtention du titre
de Docteur en Sciences Appliquées

Année Académique 2001–2002

Summary

Because of the rapid progress of computer and information technology, large amounts of data are nowadays available in almost any field of application. Automatic learning aims at producing synthetic high-level information, or models, from data. Learning algorithms are generally evaluated according to three different criteria: interpretability (how well the model helps to understand the data), predictive accuracy (how well the model can predict unseen situations), and computational efficiency (how fast the algorithm is and how well it scales to very large databases). Data mining refers to the application of automatic learning and visualization techniques in order to help a human expert to extract knowledge from large volumes of raw data. Unlike in other applications of automatic learning, in the context of data mining usually only a small fraction of the available data is actually relevant, and also, there is seldom reliable domain knowledge to help extracting relevant features or models. Actually, the precise objective of data mining is to help creating such knowledge from the available data.

Decision tree induction is a supervised learning algorithm, which focuses on the modeling of input/output relationships if the form of if-then rules. It has been claimed that among the various classes of learning methods, decision trees come closest to meeting the requirements for serving as an off-the-shelf procedure for data mining. Indeed, they are quite fast, produce relatively interpretable models, are immune to outliers, resistant to irrelevant variables, insensitive to variable rescaling, and can be easily extended to cope with a large variety of data types (numerical, symbolic, time series...). However, in terms of accuracy they seldom are competitive with other automatic learning algorithms, such as neural networks and nearest neighbors. It is commonly admitted that this suboptimality is mostly due to the excessive variance of this method. The variance of an automatic learning method measures the dependence of the models it produces on the random nature of the data set used (sampling and noise). In the context of decision trees, this variance affects all the parameters of the extracted rules and is detrimental in terms of both accuracy and interpretability. Furthermore, this variance tends to increase very strongly in applications such as time series classification where the input variables correspond to a large number of low level and, sometimes, very noisy features.

Within this overall context, this thesis explores two complementary issues: the reduction of variance of decision tree based models and the problem of learning interpretable and accurate classification rules from time series data. All our developments and analyses are extensively validated in an empirical fashion through the estimation of variance, bias and average accuracy, on the basis of a diversified set of databases.

We start our investigations by an empirical study which shows quantitatively how important decision tree variance is, i.e. how strongly they depend on the random nature of the database used to infer them. These experiments confirm that the rules induced for a given problem are indeed highly variable from one learning sample to another, and thus that variance is detrimental not only for accuracy but also from the point of view of interpretability. With the goal of improving both interpretability and accuracy, we then investigate three complementary variance reduction techniques. First, we propose several methods to stabilize the parameters chosen during tree induction. While these methods succeed in reducing the variability of the parameters, they produce only a slight improvement of accuracy. Next, we consider perturb and combine algorithms, which aggregate predictions of several models obtained by randomizing in some way the learning process. Among the existing such methods, the most popular ones, namely Bagging and Boosting, strongly improve tree accuracy but jeopardize interpretability and computational efficiency. Thus, inspired by the high variance of tree parameters, we propose a new algorithm based on extremely randomized trees (EXTRA-trees). This method aggregates the predictions of many large trees, each one obtained by choosing most of its parameters fully at random. These EXTRA-trees compare very favorably with both bagging and boosting, in terms of variance reduction, accuracy and computational efficiency. They are thus largely more accurate than standard trees, while remaining competitive with them in terms of computational efficiency. In order to also recover interpretability, we then propose a "dual" perturb and combine algorithm which randomizes input attributes at the prediction stage while using one single model. In combination with decision trees, this method actually yields soft decisions and eventually bridges the gap between single trees and the ensembles of trees produced by the perturb and combine methods. Of the first, it saves most of the interpretability, and with perturb and combine algorithms, it shares part of the accuracy improvement.

The last part of our work concerns a first attempt to design general algorithms solving the specific, but very multifaceted problem of time series classification. The most direct way to solve this problem is to apply existing learning algorithms on low-level variables which correspond to the values of a time series at several time points. Experiments with the tree-based algorithms studied in the first part of the thesis show that this approach has its limitations. We thus design a variance reduction technique specifically for this kind of data, which consists in aggregating the predictions given by a classification model based on small subsequences of the original time series. This method is very successful in terms of accuracy but does not provide fully interpretable models. We therefore propose a second method which extends decision tree tests by allowing them to detect local shift invariant properties, or patterns, in time series. This method, although limited in scope, offers the possibility to extract interpretable and accurate rules from time series.

Aknowledgement

Contents

Chapter 1

Introduction

The research presented in this thesis concerns two different domains related to supervised automatic learning, namely decision tree induction and the treatment of temporal data. Both problems are approached using the bias/variance tradeoff paradigm. Both topics are defined and motivated in Section 1.1. Then, we define in Section 1.2 the general objectives of our research in this context and the way we have tackled these objectives. The organization of the thesis is given in Section 1.3. Eventually, we close this introduction with an enumeration of the main contributions of this thesis.

1.1 Context and motivations

1.1.1 Automatic learning

Automatic learning denotes methods which aim at extracting a model of a system from the sole observation (or the simulation) of this system in some situations. By model, we mean some relationships between the variables used to describe the system. The goal of this model may be to predict the behavior of this system in some unencountered situations or to help understanding its behavior.

The interest of automatic learning is obvious when it is impossible to analytically model a system as it is often easy to gather data from its direct observation. For example, a handwriting recognition system is very difficult to put into equation, notably because of the high variability of handwriting. However, it is easy to ask several people to write some letters or words and hence constitute a database for applying automatic learning techniques. Even in situations where a physical (analytical or numerical) model based on first principles is available, the application of automatic learning can be useful, for various reasons. First, from a practical point of view, automatic learning may provide simpler models which may be exploited more efficiently in practice than the existing ones. In addition, these models constructed by automatic learning may be customized so as to express the relationship among any desired *subset* of system variables. Second, if automatic learning produces models which are comprehensible (interpretable), it can help to understand the physical behavior of the system. Third, automatic learning may take into account uncertainties about some variables which can not be modeled in a deterministic or accurate way (for example, the external environment with which the system interacts, and in particular human operators or users of the system). In all these cases, observations may be gathered from numerical simulations based on the physical model, completed, if necessary, by randomized inputs from the environment in the form of randomized system parameters or disturbances. For example, electrical power systems are very complex systems which can nevertheless be modeled by (approximate) analytical models. However, the practical use of these models is very computationally demanding and they can not be used in real-time to predict the future behavior of the system. In this case, the power system can be simulated offline under different conditions and the particular relationship we want to observe is extracted from data by automatic learning (see [Weh98]).

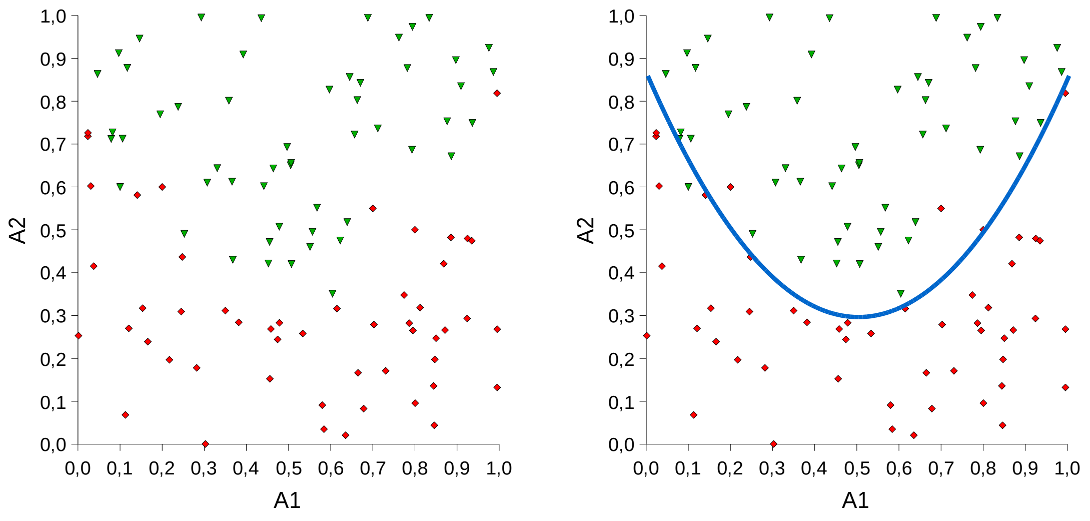


Figure 1.1: Left, a sample corresponding to a simple classification problem in a two-dimensional space. Right, the optimal classification boundary for this problem

Since its beginning, automatic learning has drawn interest from several communities: from the artificial intelligence community where it can solve several shortcomings of expert systems (by providing rules in an automatic way from observations), from the statistical community but also from the system theory community (where it is related to system identification). Each of these communities has brought different ways to state the problem and a large variety of techniques. Recent interest in this field has also grown because of the rapid progress of computer technology and also because of the rapid growth and high accessibility of the available data. The field has thus evolved into a broader field which is called KDD (knowledge discovery in databases). KDD is defined in [FPSS96] as "The non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data". The emergence of this field has also emphasized several needs in automatic learning techniques such as the scalability of the algorithms, the possibility of using them in an adaptative way (in order to not repeat the knowledge extraction from scratch when data grows), and interpretability or understandability of the knowledge extracted.

1.1.2 Supervised learning

Supervised learning is the part of automatic learning which focuses on modeling input/output relationships. More precisely, the goal of supervised learning is to identify a mapping from some input variables to some output variables on the sole basis of a sample of observations of the values of these variables. The variables are often called (input or output) attributes, observations are called objects and the sample of object is the learning sample. For example, consider the bi-dimensional problem of Figure 1.1 and assume that the goal of learning is to find a function which reproduces at best the classification between red and green points. For example, each point of this space may represent a patient which is described by two variables which are the result of two medical tests. Patients marked by a red point are suffering from one disease while patients marked by a green point are healthy. The goal of supervised learning is to provide a rule that may help a doctor to predict the state of a new patient on which the test results are known. Usually, of course, each point of the learning sample is described by more than one or two attributes and these attributes may have other values than a number (discrete values for example).

This kind of problem is solved by a (supervised) learning algorithm. Loosely speaking, a learning algorithm receives a learning sample (like the one represented in Figure 1.1) and returns a function h (an hypothesis) which is chosen in a set of candidate models (the hypothesis space).

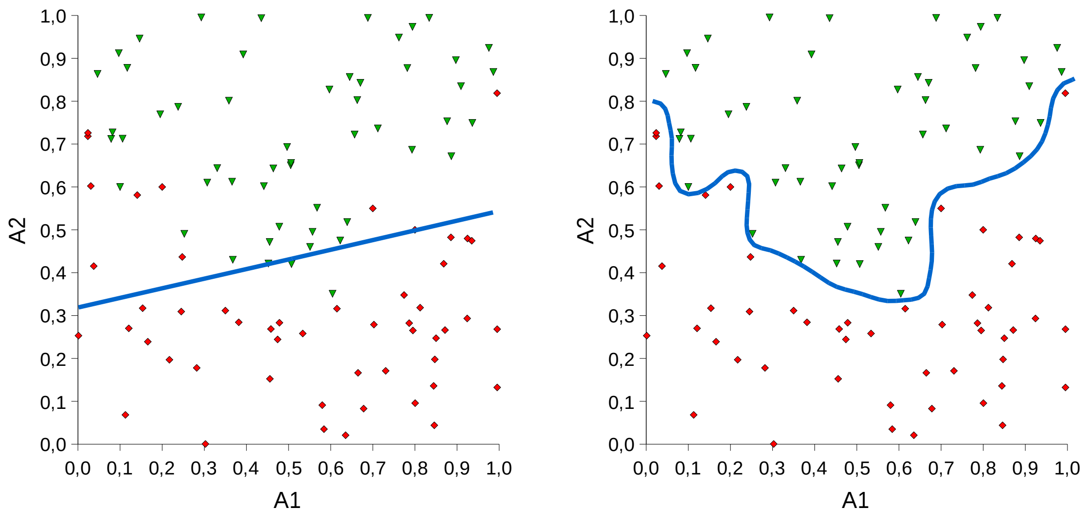


Figure 1.2: Left, a too simple model, right, a too complex one

For example, a potential candidate function returned by a learning algorithm is the one given in Figure 1.1. Even though this function make several mistakes on the learning sample, it seems that these mistakes are due to noise in the data (which may be due either to error in measurements or variables which have not been observed) and the function is a plausible explanation for the function which has served to generate the data.

The main criterion used to assess learning algorithms is their prediction accuracy, i.e. the way the model they produce generalizes to unseen data. Another important criterion, especially in the context of KDD, is the interpretability of this model. For example, in our medical application, it would be interesting to know if one of the results of the two tests is not correlated to the disease. Computational efficiency and scalability are also of great concern, especially when it comes to apply learning algorithms on very large dataset.

1.1.3 The Bias/variance tradeoff in supervised learning

To explain why the supervised learning problem is not trivial, it is useful to give some feeling about the bias/variance tradeoff in this context. Let us assume that we want to separate the two classes in Figure 1.1 with a linear decision boundary. The left part of Figure 1.2 shows one such model. With a linear model, it is impossible to separate perfectly the two classes. Patently, this model is not satisfactory to solve this task (it is very far from the optimal model of Figure 1.1). The chosen family of functions is not flexible enough to model our classification problem. This phenomenon responsible for the error in this case is called the **bias**.

Let us assume now that we use a more complex family of functions among which there exists one or several hypotheses which realize a perfect separation between classes in our sample. Then, from this family, the learning algorithm may output the decision boundary given in the right part of Figure 1.2. This time, the classification boundary perfectly separates elements of the different classes. Like the linear model however, this model is still not appropriate. This time, actually, the model is too complex and hence it learns “too much” information from our data. In this case, we say that the learning algorithm **overfits** the data. If we draw another learning sample for the same problem (by repeating the experiment on another group of patients), then, it is very likely that the model found by the same learning algorithm will be very different from this one. In this case, we say that the learning algorithm suffers from **variance**. The hypothesis which it returns is very variable from one learning sample to another.

When it comes to apply the model on new objects, both bias and variance are sources of error and hence they should be minimized. As they are functions reacting in an opposite direction to a variation of the flexibility of the hypothesis space, there is a tradeoff between these two effects which must be taken into account when designing learning algorithms.

Because of its central role in automatic learning, this tradeoff has received a lot of attention in the supervised learning literature. There are mainly two ways to handle this tradeoff. The first one is to introduce in the learning algorithm some means to control the complexity of the hypothesis space and then adapt this complexity to the problem under consideration. This approach is necessary but it is not plainly satisfying since it supposes to sacrifice some bias to reduce the variance. Recently, a series of methods have been proposed which reduce the variance without increasing the bias. The idea behind these methods is to aggregate the predictions given by several models to reduce the dependence on the random nature of the learning sample and hence the variance. In classification, this approach consists in building several models for the same sample which all overfit the data (to some extent, like the right one in Figure 1.2). To classify a new object, we make a count of the number of times each class is given by one model and then finally output the class which receives the majority of vote among all models. Intuitively, it is clear that such vote should stabilize the classification with respect to the individual classification given by every models (just like the majority vote among several people in a population is generally more stable than the vote of a random person taken from this population). These methods are called averaging techniques or ensemble methods. Among the most popular ensemble methods, there are bagging [Bre96b] and boosting [FS96]. Because of the very impressive results obtained by these methods, a lot of research has been carried out recently in this domain ([Die00b]).

1.1.4 Decision tree induction

Among learning algorithms, one of the most popular one is decision tree induction ([BFOS84]). In its simple form, a decision tree combines several binary tests in a tree structure. For example, Figure 1.3 plots a decision tree and the resulting decision boundary for our bi-dimensional example problem. Each interior node of the tree is labeled with a test (here which compares the attribute value to a threshold) and each terminal node is labeled with a class. To produce a classification for a new object which attribute values are known, we simply propagate it into the tree from the top node according to the test answers. When a terminal node is reached, its classification is attributed to the object. By using such tests, the tree progressively partitions the input space into hyper-rectangular regions so as to yield regions where the classification is constant. In the representation of the decision tree in Figure 1.3, the areas of different colors at a node represent the proportions of patients of each class which go through this node, highlighting how the tests progressively gather information about the output class.

The popularity of this learning algorithm is related to the fact that it combines several nice characteristics:

- **Interpretability.** Because of its structure, the way attributes interact to give a prediction is very readable. For example, the tree of Figure 1.3 tells to the doctor that if the result of the $A2$ test is lower than 0.35 then the patient suffers from the disease and there is no need to carry out the $A1$ test.

- **Efficiency.** The induction of such trees is done by a top-down algorithm which (in its most simple version) recursively splits terminal nodes of the current tree until they all contain elements of only one class. The resulting algorithm is very fast and can be used on very large datasets (e.g. of millions of objects and thousands of variables).

- **Flexibility.** This method is very open-minded. It does not make any hypothesis about the problem under consideration. It can handle both continuous and discrete attributes (by adapting the test choices). Predictions at leaf nodes may be symbolic (like in our example) or numerical (in which case, trees are called regression trees). Furthermore, the tree induction method can be easily extended by improving tests at tree nodes (introducing for example linear combinations of attributes) or providing a prediction at terminal node by means of another model.

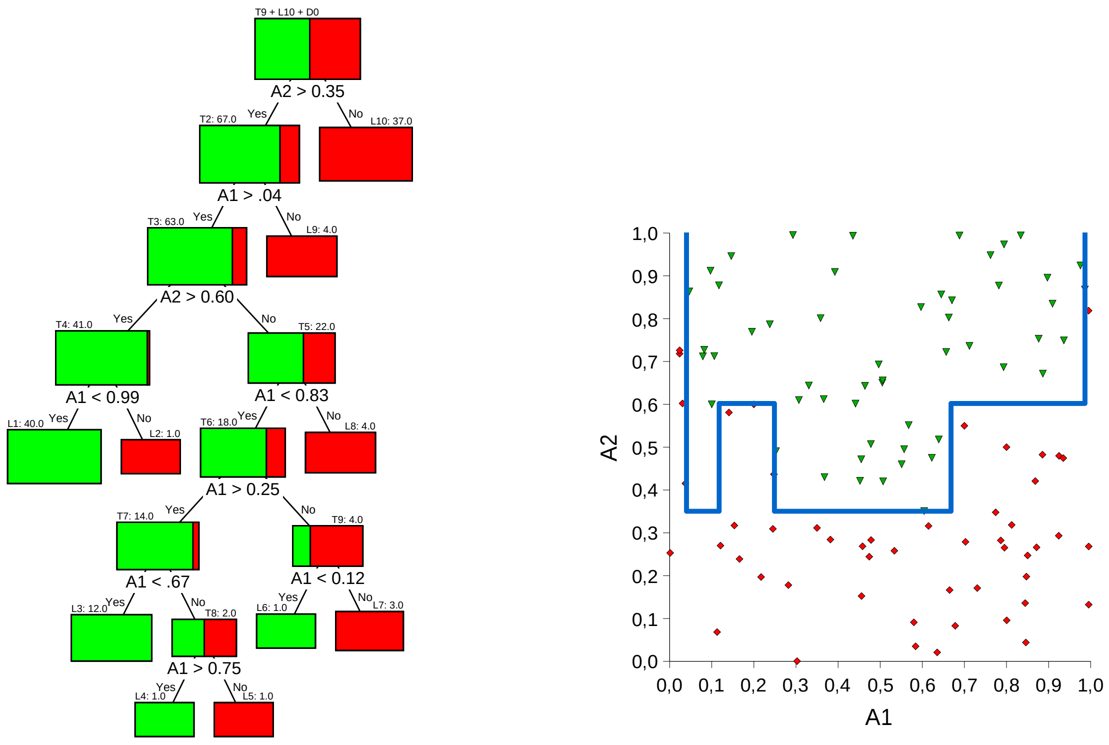


Figure 1.3: Left, a decision tree, right, the corresponding decision boundary

It is one of the only methods in machine learning which present so many advantages at the same time (see [HTF01] for a comparison with other learning algorithms).

However, decision trees still suffer one drawback: their accuracy is often not competitive with other learning algorithms (like e.g. neural networks). This lack of accuracy is mainly due to their variance. Indeed, during induction, the decision tree can be extended as far as needed to perfectly separate objects from the learning sample. So, their bias is low. However, because of this high adaptivity to the learning sample, a small perturbation of the learning sample may result in very different decision trees. So, their variance is high. In addition to the negative effect on accuracy, this instability reduces also their interpretability: we can not really trust the choices of tests since they are changing from one learning sample to another.

There are several existing approaches to improve accuracy of decision trees by reducing their variance. First, pruning consists in simplifying the tree by closing some of its test nodes so as to find the best compromise between bias and variance. Although pruning may reduce substantially the complexity of the tree by removing useless parts of it, its effect on accuracy is often limited ([Min89a]). A more efficient way to improve the accuracy of decision trees is to aggregate the predictions given by several of them for the same problem. Several techniques have been proposed in the literature to generate different decision trees from the same learning sample for aggregation and they often give very impressive improvement of accuracy. Nevertheless, they also destroy some of the main advantages of decision trees, namely computational efficiency and interpretability. Indeed, the computation time needed to build a model is multiplied by the number of trees which have to be constructed. Also, since the prediction is given by majority vote among several trees, the way attributes intervene to make up the final prediction is somewhat blurred by the process.

1.1.5 Temporal data

Classical machine learning algorithms like decision tree induction assume that objects of the learning sample are described by attributes which take simple values, essentially numerical or symbolic. However, because of the success of automatic learning in a lot of domains and because of the few hypotheses underlying such methods (they only need to gather data), automatic learning is applied to a lot of domains which need to handle more complex types of data, like image, text, or biological sequences. Among all these types of data, we focus here on temporal data. Restricting ourself to the supervised classification problem, objects of the learning sample correspond in this case to trajectories of a dynamic system which are described by several temporal attributes and the goal of learning is to discriminate among the trajectories according to a given classification. Many practical problems satisfy this definition. For example, in speech recognition, each object corresponds to the utterance of a word and is described by the evolution of the acoustic speech signal over time. The goal of learning is to recognize the word which has been pronounced. Other application domains concern for example, computer system intrusion detection, gestures recognition, and ECG signal classification.

With the introduction of the temporal dimension, each object is in fact described by a large number of low-level values, each one corresponding to the value of an attribute at a given time point. Often, every such low-level value alone is useless for classification. Only the way they are related in time is useful for classification. The difficulties of this domain are manifold and related to this high dimensionality. First, automatic learning algorithms will suffer from a high variance. Indeed, because of the high number of low-level attributes, it is easy to perfectly separate a learning sample and the "chances" for overfitting are increased. So, variance deserves special attention in such domains. Also, temporal problems often correspond to very large datasets and application of automatic learning is often very demanding in terms of computation times and computer resources. Another difficulty is to produce interpretable rules. Indeed, interpretable methods applied on low-level attributes without taking into account the temporal dimension will likely produce very complex models which are very difficult to understand.

Several algorithms have been developed for handling temporal data in specific domains. Practically, the difficulties mentioned earlier are overcome in these systems by introducing a lot of a priori information about the problem. For example, many systems exist to solve the speech recognition problem. These systems usually use specific information about the language structure to simplify the learning phase. If these problem specific approaches often give very good results in terms of accuracy, they are not widely applicable. Furthermore, they fail in providing interpretable models since on the contrary they assume a prior understanding of the problem to solve. In this context, what is missing is thus general approaches which produce accurate and/or interpretable models.

1.2 Objective of the thesis and our way to solutions

The research presented in this thesis concerns the two domains motivated in the previous section, namely improvement of decision tree induction and treatment of temporal data. In our search for a solution to both these problems, the bias/variance tradeoff will be used to guide most of our choices. One of the side objectives of this thesis is to show how such a theoretical tool may help either to improve existing methods or to design algorithms to handle new types of problems.

1.2.1 Improvements of decision tree induction

In the context of decision tree induction, the main objective of this thesis is to analyze the sources and consequences of the high variance of decision trees and to investigate means to reduce this variance without jeopardizing efficiency and interpretability of this method.

The resolution of this question in this thesis is a four step process. First, this variance is studied in detail by several experiments. The main goals of these experiments are to highlight quantitatively the high variance of decision trees and to find out the main sources of this variance. One of the main conclusions of these experiments is that decision tree variance is (indeed) high and it results in a high variance of the parameters that are used to define the tree, mainly the discretization thresholds for numerical attributes.

The second step consists in trying to find methods which reduce the variance of the decision tree induction algorithm. In this goal, we evaluate classical pruning algorithms that adapt the complexity of decision trees and propose several means to reduce the discretization threshold variance. Although both techniques certainly improve the interpretability of decision trees (by reducing the complexity and stabilizing the parameters), we show that they have however little effect on decision tree accuracy. This experiment shows the limitation of what can be done in terms of accuracy by using a single decision tree. One conclusion of this second step is thus that it is necessary to extend the hypothesis space of decision trees if we want to obtain further improvement in terms of accuracy.

The next step is thus to consider ensemble techniques in the context of decision trees which use more complex hypothesis spaces. In the context of decision trees, several ensemble methods have been proposed which somewhat randomize the induction algorithm to produce different models from the same learning sample. Since our experiments highlight the high variance of test parameters, we introduce extremely randomized trees (Extra-trees) where tests are chosen almost randomly (i.e. without taking into account the learning sample). When several extra-trees are aggregated, the improvement in terms of accuracy by variance reduction is very impressive. Because of the extreme randomization, the result is also a computationally very efficient method which is competitive with single decision tree induction in spite of the fact that it needs to build several trees. However, although very accurate and efficient, the resulting method is intrinsically not interpretable since it uses several models to produce a prediction and furthermore these models are random.

In the last step, we propose a new algorithm which tries to reproduce the prediction of an ensemble of models with only one model. This general algorithm generates several predictions by perturbing the attribute vector corresponding to an object and using a single model to produce a prediction for each of these perturbed vectors. We call this algorithm "dual perturb and combine". In the context of decision trees, a closed form approximation of this algorithm is developed which essentially replaces crisp thresholds at the test nodes of a tree by soft ones. Actually, this method bridges the gap between decision trees and ensembles of decision trees. Of the first, it saves the interpretability (by using only one model), and with ensemble methods, it shares some of the accuracy (by reducing the variance).

1.2.2 Treatment of temporal data

The main objective of this part of the thesis is to design general approaches to solve the problem of supervised classification with temporal attributes. We make the additional assumption that temporal signals in the dataset are represented in discrete time. Discrete time temporal signals are often called time-series in the literature. In this domain, our ambition is not to give a final solution to the problem but rather to give some preliminary steps in the goal of giving accurate and interpretable methods. This objective is realized in three steps.

First, we explore the use of low-level features with classical learning algorithms, with the aim of highlighting the limitations of such approaches. On several problems, provided that we use strong variance reduction techniques (like extra-trees), this approach gives not so bad results. However, on problems which present typical behavior of the temporal domain, variance reduction techniques do not succeed in improving accuracy in an acceptable way.

Second, we propose a new approach which also use classical methods on low-level features but which nevertheless exploits the temporal ordering of values. A time series is divided into all its subsequences of a given length and a model is constructed to classify these subsequences on the basis of low-level features. This algorithm is called "segment and combine" and its validation shows that it gives significant improvement especially on typical temporal problems.

Even if algorithms working with low-level features give acceptable results in terms of accuracy, they do not provide interpretable models even if the classical algorithm behind it was interpretable. So, the third step of this work is to propose a new algorithm which provides interpretable models. To this end, decision tree tests are extended to take into account local shift invariant properties in time-series. Such patterns are extracted during tree induction from the time-series appearing in the learning sample. To reduce the search space for candidate patterns and improve interpretability, time-series are compressed by using piecewise constant models. Although they were introduced mainly to improve the interpretability of decision trees, such tests actually allow to improve also accuracy in problems which are characterized by local shift invariant properties (mainly because of a low bias). On other problems however, it deteriorates accuracy with respect to the best use of low-level features, mainly because of an increase of variance.

1.3 Organization of the thesis

The thesis is organized into three parts. The first part aims at introducing the supervised learning problem and the bias/variance tradeoff. The second part is devoted to the improvements of decision tree induction while the treatment of temporal data is studied in the third part. In order to improve readability, we have collected some technical discussions as well as the detailed presentation of empirical results in appendices. Finally, the conclusions chapter gives a general synthesis of the research developed in this thesis and discusses some ideas for future work. The three parts of the body of the thesis are structured as follows.

Part I: Supervised learning and the bias/variance tradeoff

In chapter 2, we define the supervised learning problem using a probabilistic framework. Several classical learning algorithms are described and discussed in terms of the practical criteria used to compare learning algorithms, namely accuracy, computational efficiency, and interpretability. Eventually, we conclude this chapter with a description of usual methods used to estimate the accuracy of models.

Chapter 3 is devoted to the bias/variance tradeoff in automatic learning. After an informal illustration of this tradeoff on a simple synthetic problem, we give the well-known decomposition of the mean square error into the two positive terms that reflect bias and variance. The problem of the decomposition of mean error rate in classification is then tackled. While the decomposition of square error is well agreed upon, several (actually, quite many) decompositions have been proposed in classification. One of them is selected which will be used throughout our experiments. Other decompositions are nevertheless given and related in Appendix A. In this chapter as well, several experiments are carried out which aim at providing some insight about the way bias and variance evolve according to the parameters of learning algorithms and the studied problem. Eventually, the chapter is concluded with some analytical results found in the literature related to the bias/variance tradeoff.

To conclude this part, we review in Chapter 4 classical variance reduction techniques. Two types of techniques are considered: techniques that control the bias/variance tradeoff in the context of a given learning algorithm and ensemble methods which consists in aggregating several models. In this latter case, we propose a unifying framework based on random algorithms to describe ensemble methods which, like in [Bre96a], are termed “perturb and combine” algorithms. An analysis of the bias/variance decomposition of these algorithms identifies the conditions under which such algorithms should improve accuracy. In this chapter, we give also an introduction of the general dual perturb and combine algorithm which is studied in chapter 8 in combination with decision trees. Related techniques like Bayesian approaches and boosting type of algorithms are also discussed here.

The reader already familiar with supervised learning and the bias/variance tradeoff may skip most of the first part or refer to it only when necessary.

Part II: Bias and variance in decision tree induction

The second part of the thesis is devoted to the analysis and the improvement of the decision tree induction algorithm. This part is decomposed into five chapters.

Chapter 5 contains several experiments that aim at showing and analyzing the high variance of decision trees. This variance is analyzed from both the point of view of accuracy (measuring its effect on error rates) and interpretability (measuring the variance of the parameters that define the decision tree, i.e. the discretization thresholds and the attribute which is chosen at a test node). Among other things, we study the effect on variance of the score measure which is used to assess candidate tests during tree induction.

Chapter 6 is devoted to the reduction of decision tree variance. First, some experiments are carried with a pruning algorithm. Then, we propose several techniques to reduce the variance of the discretization thresholds of numerical attributes. The effect of these techniques on prediction accuracy is also assessed.

In chapter 7, we approach the problem of decision tree variance reduction by perturb and combine algorithms. After some experiments with the popular bagging algorithm, we give a short review of the perturb and combine algorithms specifically designed for decision trees. Then, we introduce our perturb and combine algorithm which builds extremely randomized trees. This algorithm is evaluated and compared with bagging and boosting algorithms. Eventually, we discuss the promising capabilities of this algorithm in terms of scalability and for bias reduction.

Chapter 8 is devoted to the application of the dual perturb and combine approach to decision trees. A closed-form implementation of this algorithm is developed which shows that it actually transforms a classical decision tree into a soft tree model. The method is validated and compared with bagging. Its behavior is related to the one of other perturb and combine algorithms and its combination with bagging is assessed. We conclude this chapter by discussing related work on other approaches to build soft tree models.

Eventually, chapter 9 concludes this part of the thesis by a synthetic comparison of all decision tree variants which have been studied and analysed throughout this part of the thesis.

All experiments of this part of the thesis are carried out on 7 classification problems which are described in Appendix B. While only synthetic results appear in the core of the text, Appendix C collects detailed results on all problems for experiments of all chapters of this part of the thesis.

Part III: Knowledge extraction from time series

The third part of the thesis is devoted to the temporal domain. It is divided into three chapters.

In Chapter 10, we give the motivations and the challenge in considering temporal data (and more generally complex data) with learning algorithms. We then extend the supervised learning framework of Chapter 2 to handle temporal attributes. Several classifications of the temporal problems are given and peculiarities of the time-series classification problem are discussed. Eventually, literature on time-series classification is briefly reviewed and the 6 test problems we used throughout our experiments are introduced. The details concerning the generation of these problems are reported to Appendix B.

Chapter 11 investigates the two approaches based on low-level features. First, classical algorithms are used with non temporal attributes extracted from time-series by two simple sampling approaches. Then, the segment and combine algorithm is described and validated. The limitation of this approach is discussed in the conclusion of this chapter.

Chapter 12 studies our extension of decision trees to handle time-series data. Decision tree tests are extended to detect local shift invariant properties or patterns in time series. A large part of this chapter is devoted to the description of the proposed algorithm. Some algorithmic details are reported in Appendix D. Several experiments are carried out to validate the method in terms of accuracy and interpretability. This chapter is concluded by a section on related work and a discussion of possible improvements of the method.

1.4 Contributions and originality

The aim of this section is to highlight the main contributions of this thesis. Each item is related to a small number of selected references in order to allow one to appraise our own contributions. A more extended discussion of related works in all domains is of course given in the body of the thesis. Contributions which have already been published in the machine learning literature are also mentioned.

So, we believe that the principal contributions of this thesis are (in order of appearance in the main text):

- The quantitative analysis of decision tree variance in Chapter 5. Although we can find several references in the literature to the fact that decision trees are highly variable and also that the variance of the discretization thresholds is large (see for example [DK95] or [HTF01]), we have no knowledge of previous publications that analyse and estimate quantitatively how important this variance is. The experiments of this thesis have been motivated by [Weh97]. A summary has been published in [GW00].

- The stabilization techniques for discretization thresholds introduced in Chapter 6. This work is orthogonal to previous works on discretization which focus exclusively on improvements of global characteristics of trees (complexity, predictive accuracy, or computational efficiency). These methods have been motivated by the work in [Weh97] and some of the results have been published in [GW00].

- Extra-tree algorithm of Chapter 7. Random decision trees are not new (see [Bre01] or [Ho95]). With respect to existing algorithms, our method goes one step further by randomizing both attributes and discretization thresholds. This algorithm originates from our analysis of decision tree variance. The idea of randomizing the attribute at tree nodes already appears in [Geu00].

- The general dual perturb and combine algorithm and its closed-form implementation in the context of decision tree induction presented in Chapter 6. This method has already been published in [Geu01a] and is described in a broader extent in [Geu01c]. The idea of generating several predictions at the prediction stage has also appeared independently of our work in [DKN01] under the name "virtual test sample" approach. In the context of decision trees, this method is also related to soft tree models which have been extensively studied in automatic learning under different names.

- The derivation, from the perspective of the bias/variance tradeoff, of the segment and combine algorithm for time-series classification in Chapter 11. A similar scheme for image classification has been proposed in [FH00a].

- Pattern extraction techniques for decision tree induction in Chapter 12. The approach is substantially different from existing algorithms for time-series classification. A slightly less advanced version of this method has been published in [Geu01b].

All methods, except for the classical decision tree induction algorithm, have been implemented by the author of the thesis in Common Lisp as add-ons to the ATDIDT software which is an extensible data mining package developed at the University of Liège. Validation of improvements or new algorithms rely heavily on empirical studies. For information, the results of the second part of the thesis (gathered in Appendix C) have required the construction of about 100,000 decision trees. Notice that, while computational efficiency of automatic learning algorithms is very important, even in research, our focus has been more on the intrinsic features of the methods (such as algorithm complexity and scalability, and more importantly bias, variance, and accuracy) than on the efficient implementation of the algorithms. Therefore, we report only on few occasions CPU times, and we do this with the sole objective of providing some feeling about *relative* performances.

Part I

Bias and variance in automatic learning

Frequently used notation

We collect below the most important and most frequently used notation. All symbols and notations are defined precisely in the first place where they are introduced in the main text.

o: an object (a variable denoting the outcome of a random experiment)

$\mathcal{U}$: the universe (the set of all possible objects within a given automatic learning problem)

$\mathcal{X}$: a subset of objects from the universe (or an event)

$P(\mathcal{X})$: the probability that an object belongs to $\mathcal{X}$ (or of the event to happen)

$P(o)$: the probability of a singleton (the same as $P(\{o\})$)

$A(.)$, $Y(.)$, $C(.)$: attributes (i.e. functions defined on $\mathcal{U}$). $Y(.)$ denotes the output variable. $C(.)$ denotes a classification output variable

$\mathcal{A}$, $\mathcal{C}$, $\mathcal{Y}$,...: the codomain of the corresponding attributes

A,Y,C,...: general (not necessarily real-valued) random variables induced by the attribute functions on their respective codomain

a,y,c,...: attribute values or equivalently values of random variables

LS (and LS^N): random learning sample (of size N), i.e. an random variable induced by a sampling scheme over $(\mathcal{A} \times \mathcal{Y})^N$

ls (and ls^N): a particular learning sample, i.e. a realization of LS (of size N)

$\underline{A}$, $\underline{\mathcal{A}}$, $\underline{a}$: the vector of input attributes (random variables), the codomain of this vector, and a value of this vector

$P(A)$: probability distribution induced by $A(.)$ on $\mathcal{A}$

$P(a)$, $P(A(o) = a)$, $P(A = a)$: probability that the random variable A takes the value a

$E_X\{f(X)\}$: expectation of a (real-valued) function of a random variable

$var_X\{f(X)\}$: variance of a function (i.e. $E_X\{(f(X) - E_X\{f(x)\})^2\}$)

$\mathcal{H}$: the hypothesis space of a learning algorithm (a set of functions mapping $\underline{\mathcal{A}}$ into $\mathcal{Y}$)

$h(.), f(.)$ (or h, f): a particular hypothesis

$f_B(.)$: Bayes model, i.e. a function produced by an ideal learning algorithm

$f_{ls}(.)$: a particular hypothesis returned by a learning algorithm in response to a particular learning sample ls

$f_{ls}(\underline{a})$: the prediction of this hypothesis at point $\underline{a}$

$f_{LS}(.)$: the random variable induced on $\mathcal{H}$ by a learning algorithm and sampling scheme

$f_{LS}(\underline{a})$: random variable of predictions at point $\underline{a}$, induced on $\mathcal{Y}$ by a learning algorithm and sampling scheme (the source of randomness is the sampling scheme)

$f_{ls}(\underline{A})$: random variable of predictions, induced on $\mathcal{Y}$ by a learning algorithm fed with a particular sample ls (the source of randomness is the input variable $\underline{A}$)

$f_{LS}(\underline{A})$: random variable of predictions, induced on $\mathcal{Y}$ by a learning algorithm fed with a random learning sample (the two sources of randomness are combined)

$P(y|\underline{a})$: the conditional probability of $\{o \in \mathcal{U} | Y(o) = y\}$ given that $\underline{A}(o) = \underline{a}$

$P(c|\underline{a})$: similar, stressing the fact that the output is a classification

$P_{LS}(y|\underline{a})$: the probability distribution of $f_{LS}(\underline{a})$

$\hat{P}_{LS}(.)$: a random variable used to estimate a probability, provided by some estimation scheme using the learning sample

$\hat{P}_{LS}(c|\underline{a})$: a random variable used to estimate the conditional probability $P(c|\underline{a})$, determined by a classification algorithm in response to a random learning sample LS

$1(.)$: indicator function which equals 1 when its argument is true, 0 otherwise.

$\overline{f}_{LS}(.)$: the average model defined at point $\underline{a}$ by $E_{LS}\{f_{LS}(\underline{a})\}$, when the output is real-valued

$f_{LS}^{MAJ}(.)$: the majority vote classifier defined at point $\underline{a}$ by $\arg\max_c E_{LS}\{1(f_{LS}(\underline{a}) = c)\}$, when the output is a classification

Chapter 2

Supervised automatic learning

In this chapter, the supervised learning problem is formally defined. In the first section, the different notions and notations are introduced. Section 2.2 explains the way learning algorithms proceed. Some classical methods are described in section 2.3 and the way algorithms and models are evaluated are discussed in Section 2.4 and 2.5.

2.1 Definitions and notation

Loosely speaking, automatic learning aims at modeling and/or understanding relationships among a set of variables used to describe objects (or situations) encountered in some particular context, on the sole basis of some available observations of the values of these variables on a sample of already encountered objects. By modeling, we mean being able to predict the values of some of the variables (called output variables) in terms of the values of some other variables (the input variables). By understanding, we mean for example sorting out those input variables which influence (or are correlated with) some of the outputs, determining the type of influences, or constructing logical rules which allow to infer information about outputs given information about inputs.

In this thesis we will consider a restrictive version of this type of problem, using the so-called "object-attribute-value" representation and where objects are viewed as outcomes of a random experiment modeled using probability theory.

2.1.1 Universe, objects, attributes

Under these restrictions, the formal definition of a supervised learning problem starts by the definition of the universe $\mathcal{U}$ of all possible objects o. In a given context, each object o has a, generally unknown, probability $P(o)$ of being observed [1]. In practice, we do not have full information about the objects but only have access to a description of them by their attributes (which are measurable properties of objects, or which can be computed as a function of such properties). An attribute A is a function defined on the universe:

$$A(\cdot) : \mathcal{U} \mapsto \mathcal{A},$$

where $\mathcal{A}$ is the codomain of the function $A(.)$. Equivalently, an attribute A is a random variable whose values are supposed to belong to the set $\mathcal{A}$ [2]. The probability distribution on the universe

[1]Although most of the theory that we present can be extended to infinite and non-countable universes, for the sake of mathematical and notational convenience, we assume in this chapter that the universe is a finite (but generally very large) set of objects, and use the notation $P(o)$ instead of $P(\{o\})$ when considering the probability of an atomic event; under these conditions we can further assume that $P(o) > 0, \forall o \in \mathcal{U}$. In the context of our research, we found that the distinction between continuous and discrete probability spaces was not really fundamental. Although we will in some occasions use continuous random variables and densities, mainly in the context of illustrative examples, we will use the discrete notations in most cases.

[2]We use upper case letters to denote random variables and lower case letters to denote values of random variables. Occasionally, we use the notation $A(.)$ to denote the corresponding function defined on $\mathcal{U}$. Finally, we

induces a probability distribution on $\mathcal{A}$ in the following way:

$$P(A = a) = \sum_{o \in \mathcal{U} | A(o) = a} P(o). \tag{2.1}$$

In the literature, attributes are also called features or variables. Usually two types of attributes are considered according to the structure of their codomains:

- **Symbolic attributes:** the codomain is a finite[3] and unstructured set of values, i.e. $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$.
- **Numerical attributes:** the codomain is a subset of the real axis $\mathbb{R}$.

These two general types are sufficient to handle most problems. Symbolic attributes in fact denote all discrete non ordered types of attributes, while the numerical attributes regroup ordered ones (discrete or not, like integer for example). Of course, many other types of attributes are possible. For example, in the third part of this dissertation, we will consider attribute values in the form of time-series or temporal signals.

2.1.2 Learning sample

Supervised learning aims at modeling the relationship between some a priori selected input and output attributes. In this dissertation, we reduce this problem to the modeling of a *single* output variable which is denoted by Y. Besides, the vector of input attributes is denoted by $\underline{A} = (A_1, \ldots, A_n)$. This random vector takes its values in

$$\underline{\mathcal{A}} = \{(A_1(o), \ldots, A_n(o)) | o \in \mathcal{U}\} \subseteq \mathcal{A}_1 \times \ldots \times \mathcal{A}_n,$$

where $\mathcal{A}_i$ is the codomain of the attribute A_i. The joint probability distribution of input and output attributes is denoted by $P(\underline{A}, Y)$ and is determined like (2.1). The relationship between input and output attributes is in principle fully characterized by the conditional probability distribution $P(Y|\underline{A}) = P(\underline{A}, Y)/P(\underline{A})$ (if $P(\underline{A}) \neq 0$).

A learning sample[4] of size N, LS^N, is an N-tuple of objects randomly and independently drawn from $\mathcal{U}$, each one described by its input and output attribute values[5]:

$$LS^N = ((\underline{A}(o_1), Y(o_1)), \ldots, (\underline{A}(o_N), Y(o_N))) = ((\underline{a}_1, y_1), \ldots, (\underline{a}_N, y_N)).$$

LS^N is thus equivalently a random variable distributed according to $P^N(\underline{A}, Y)$. In the sequel we will most often drop the superscript ^{N} from our notation, unless it is necessary to explicitly stress the dependencies on the sample size.

2.1.3 Supervised learning problem

The general problem of supervised learning is formally stated as follows:

For any value of N and a learning set ls^N (a realization of LS^N) and without any a priori knowledge of the functions $P(.)$, $Y(.)$, or, $\underline{A}(.)$, find a function $f(.)$

use calligraphic upper case letter to denote subsets of objects or of values of random variables.

[3]Obviously, when $\mathcal{U}$ is finite all attributes have also a finite codomain.

[4]In the literature, the term "learning set" is often improperly used to denote the learning sample. Historically, this terminology originates from the AI community where learning algorithms were first introduced in a deterministic setting and thus duplicates in the learning sample were meaningless. Propagating this bad habit, we will sometimes improperly use the term "set" to denote a sample.

[5]In practice, some of the attribute values may be unknown for some objects. We exclude this possibility from our discussion and suppose that missing values have been replaced in some ad hoc way before the learning problem is formulated.

defined on $\underline{A}$ which minimizes the expected prediction error defined by[6]:

$$Err(f) = E_{\underline{A},Y}\{L(Y, f(\underline{A}))\}, \tag{2.2}$$

where $L(.,.)$ is a loss function which measures the discrepancy between its two arguments.

In other words, the goal of learning is to find a function of the input attributes which predicts at best the outcome of the output attribute. The goodness of the fit is evaluated according to a predefined loss function. In practice, this loss function depends on the type of values of the output attribute:

- **Classification problem:** Y is a symbolic variable taking its values in $\mathcal{Y} = \{c_1, \ldots, c_m\}$. The most common loss function is the 0-1 loss function $L(Y, f(\underline{A})) = 1(Y \neq f(\underline{A}))$. The error becomes the probability of mis-classification:

$$Err(f) = E_{\underline{A},Y}\{1(Y \neq f(\underline{A}))\} = P(Y \neq f(\underline{A})) \tag{2.3}$$

- **Regression problem:** Y is a real-valued variable, $\mathcal{Y} \subset I\!\!R$. The most commonly used loss function is the square loss $L(Y, f(\underline{A})) = (Y - f(\underline{A}))^2$ and the corresponding error is the mean square error:

$$Err(f) = E_{\underline{A},Y}\{(Y - f(\underline{A}))^2\}, \tag{2.4}$$

To distinguish a classification problem from a regression one and following the machine learning literature, we will often denote a discrete output attribute by C while Y will be reserved to a regression output or a general output attribute whatever its type.

2.1.4 Bayes model and residual error

For a given problem and loss function the best possible model, i.e. the target of a learning algorithm, can be derived in the following way (assuming complete knowledge of the conditional probability distribution $P(Y|\underline{A})$).

Since :

$$Err(f) = E_{\underline{A},Y}\{L(Y, f(\underline{A}))\} = E_{\underline{A}}\{E_{Y|\underline{A}}\{L(Y, f(\underline{A})\}\}, \tag{2.5}$$

the function f_B which minimizes $Err(f_B)$ is obtained by minimizing the inner expectation at each point $\underline{A}$ of the input space, i.e. taking:

$$f_B(\underline{a}) = \arg\min_{y'} E_{Y|\underline{a}}\{L(Y, y')\}. \tag{2.6}$$

This function is called the *Bayes model* in the statistical pattern recognition literature. For the 0-1 loss function, it becomes the *Bayes classifier* :

$$\begin{aligned} f_B(\underline{a}) &= \arg\min_{c} E_{C|\underline{a}}\{1(C \neq c)\} && (2.7)\\ &= \arg\min_{c} P(C \neq c|\underline{a}) && (2.8)\\ &= \arg\max_{c} P(C = c|\underline{a}). && (2.9)\end{aligned}$$

The Bayes classifier thus predicts the most probable class at each point of the input space. For the square loss, a derivation of the inner expectation of 2.5 gives the following minimizer:

$$f_B(\underline{a}) = E_{Y|\underline{a}}\{Y\}. \tag{2.10}$$

[6] $E_X\{f(X)\}$ denotes the expectation of a function $f(.)$ with respect to the distribution of a random variable X; it is defined by

$$E_X\{f(X)\} = \sum_{x\in\mathcal{X}} P(X = x)f(x) = \sum_{o\in\mathcal{U}} P(o)f(X(o)),$$

where X is some random variable.

The best prediction at the point $\underline{a}$ of the input space is thus the conditional expectation of Y given $\underline{a}$.

In both cases, the minimal error vanishes if and only if there exists a functional relationship g between Y and $\underline{A}$, i.e. $P(Y = y|\underline{A} = \underline{a}) = 1(y = g(\underline{a}))$. In this case, g and f_B coincide. In general, however, there exists some random deviation (or noise) around the Bayes model which translates into a non deterministic probability distribution $P(Y|\underline{A})$ and thus there exists a non null minimal attainable error, $Err(f_B)$, which is independent of the learning algorithm. This minimal achievable error is also called the *residual error* in supervised learning.

2.2 Learning algorithms

A learning algorithm is a (computable) function which takes as sole input a learning sample and outputs a (computable) classification or regression model. From the previous consideration, it appears that to solve the supervised learning problem, it would be sufficient to estimate, from the learning sample, the distribution $P(C|\underline{A})$ in classification or $P(Y|\underline{A})$ in regression and then define a model based on this estimate (e.g. by formula (2.9) or (2.10)). In the context of finite universes, a trivial solution would be to use frequency counts to estimate these probabilities. In practice, however, an accurate estimation of this distribution would require a very large learning sample which indeed needs to contain a lot of samples at each point of the input space. Actually, as the number of input variables grows the number of observations required by this trivial approach would grow in an exponential way.

Thus, to work with reasonable sample sizes and in large dimensional input spaces, learning algorithms must make some "smoothness" hypothesis of the function $f(.)$ over the input space and use neighbor points to make a prediction at a point $\underline{a}$. One classical way to do this is to restrict candidate models to a specified form and do optimization in this family of models in order to minimize the error.

2.2.1 Hypothesis space

The design of a learning algorithm thus begins by the definition of a hypothesis space $\mathcal{H}$ which contains potential candidate models h among which this algorithm will select one on the basis of the learning sample ls. In the sequel, we will provide several examples of hypothesis spaces, e.g. linear models, neural networks, or decision trees.

Note that for a particular learning problem, the best possible model belonging to a particular hypothesis space may or may not be close to the Bayes model of this problem. The *approximation error* of a hypothesis space for a given learning problem can be defined by:

$$\min_{h \in \mathcal{H}} E_{\underline{A}}\{L(h(\underline{A}), f_B(\underline{A}))\}. \tag{2.11}$$

Thus, in practice it is often necessary to iterate in order to adjust the hypothesis space a posteriori to the problem specifics, so as to reduce this approximation error.

Of course, the larger the hypothesis space, the smaller the approximation error, and, in particular, if the hypothesis space contains models which are arbitrarily "close" to any function within a given class, then the hypothesis space is said to have the *universal approximation capability* in this class. Unfortunately, as was already suggested above in the context of the trivial frequency counting method, the larger the hypothesis space the more difficult it is to identify the optimal hypothesis from a finite sample, and the larger the so-called *estimation error*. This latter phenomenon is also called *overfitting*, or *bias/variance tradeoff* in the automatic learning literature. The design of good hypothesis spaces through the study of the tradeoff between approximation error and estimation error in the context of finite sample size is one of the main themes of modern automatic learning theory. It is also one of the main themes of this thesis, and is further elaborated in chapter 3 where the notions of *bias* and *variance* will be defined more precisely. For sake of the present discussion, however, we assume that the hypothesis space is already fixed.

2.2.2 Empirical risk minimization

As the only available data is a finite sample of the universe $\mathcal{U}$, the value of the error $Err(h)$ for a hypothesis h can not be computed exactly. This error is thus first approximated by a summation over the objects of the learning set:

$$\hat{Err}(h, ls) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, h(\underline{a}_i)), \tag{2.12}$$

which is also called the *empirical risk* in the automatic learning literature.

Then, learning could consist in selecting a function in $\mathcal{H}$ minimizing this estimation of error, which is the so-called *empirical risk minimization principle.* A learning algorithm is thus fully characterized by its hypothesis space and by the way it optimizes the learning set error. The optimization can be deterministic (i.e. output always the same function of the hypothesis space in response to a fixed learning set) or random (i.e. may output different hypotheses for the same learning set).

In regression, the mean square error (2.4) is smooth with respect to variations of the output of the model being evaluated. Hence, if the hypothesis space is a parametric family and if the output of a hypothesis is smooth with respect to variations of the parameters, traditional optimization techniques (for example, gradient descent) can be used to find the model which minimizes the learning set error. In classification, the output of the model is discrete and direct optimization of the error is often difficult. So, many algorithms first compute an estimate $\hat{P}(C|\underline{a})$ of the conditional class probability distribution $P(C|\underline{a})$ and then use this estimate to make a prediction at each point according to:

$$f(\underline{a}) = \arg\max_{c} \hat{P}(C|\underline{a}). \tag{2.13}$$

There are several ways to find an estimate of $P(C|\underline{a})$. Some methods find a region $R(\underline{a})$ around a point $\underline{a}$ and then estimate probabilities by relative frequency counts of objects from the learning set appearing in these regions. As we will see below, decision trees and the K-nearest neighbors work that way. Another solution is to use a regression algorithm and the square error loss to find a model for a new series of numerical outputs derived from the classification output by:

$$Y_C^i = 1(C = c_i), i = 1, \ldots, m. \tag{2.14}$$

Indeed, We have seen that, under square loss, a regression method tries to find an estimation of $E_{Y|\underline{a}}\{Y\}$. With Y_i^C as output, this amounts to estimating:

$$E_{Y_C^i|\underline{a}}\{Y_C^i\} = P(Y_C^i = 1|\underline{a}) = P(C = c_i|\underline{a}), \tag{2.15}$$

which yields an estimation of conditional class probabilities. This is a possible way to learn classification models with linear regression and neural networks.

2.2.3 Problem specific background knowledge

In our formalization of the supervised learning problem, the sole problem specific information which is used by the algorithm is the learning sample. There is presently research in automatic learning about methods able to exploit other problem specific information in the form of some constraints about the relation among attributes and/or output variable. For example, inductive logic programming (ILP) does this in a logical setting in the form of first order logic statements. Bayesian networks incorporate this information in a probabilistic setting, in the form of conditional independence assumptions. We will not consider these aspects in this thesis as we are focusing on methods which are not exploiting such side information explicitly.

2.3 Main classes of supervised learning algorithms

In this section, we briefly describe five well-known learning algorithms which will be used later in this dissertation. Although there exist of course much more than five learning algorithms, they cover a large spectrum of basic automatic learning algorithms. Linear models and neural networks are representative of parametric techniques. Parametric techniques make assumptions on the hypothesis spaces by means of a fixed number of parameters and optimize these parameters from the learning set. On the other hand, decision trees and k-NN are termed non parametric techniques since they do not make any assumption on the form of the Bayes model and can adapt the shape of the model they produce on the basis of the learning set. Decision trees are also the main representative of the family of symbolic machine learning techniques. The initial motivation behind this method is the induction of logical rules and while all other methods are statistical in nature, decision tree induction has been largely studied within the symbolic artificial intelligence community. Finally, Naive Bayes is a very simple instance of probabilistic models used for classification of which Bayesian networks are a generalization. The main characteristic of these latter probabilistic models with respect to other algorithms is that they provide full descriptive models (as opposed to predictive models) which induction is not directed by constraints of good performance in prediction.

Besides these basic automatic learning algorithms, most of the recent advances in machine learning are devoted to meta-machine learning or wrapper techniques. These are general techniques which are applied on the top of existing learning algorithms so as to improve their accuracy. For example, the variance reduction techniques discussed in subsequent chapters of this dissertation belong to this family of wrapper techniques. As they will be tackled latter, they will not be discussed here.

2.3.1 Linear models

$\mathcal{H}$ contains all linear combinations of input attributes:

$$h(\underline{a}) = w_0 + \sum_{i=1}^{n} w_i \cdot a_i. \tag{2.16}$$

This model can be extended to give a linear discriminant model for two-class problems in the following way:

$$h(\underline{a}) = \begin{cases} c_1 & \text{if } w_0 + \sum_{i=1}^{n} w_i \cdot a_i > 0 \\ c_2 & \text{otherwise} \end{cases}. \tag{2.17}$$

Several alternatives are possible to combine such binary linear discriminant functions to handle the multi-category case.

Learning consists in searching for the value of the vector of parameters $\underline{w} = (w_0, w_1, \ldots, w_n)$ which minimizes $\hat{Err}(h, LS)$. Although the optimal solution can be computed analytically in the regression case, there exist numerous gradient descent procedures which iteratively modify the vector of parameters in a direction which optimizes the error criterion. These algorithms differ in the loss function and in the way they correct $\underline{w}$ at each iteration steps (see [DHS00] for a good review of linear models).

The expressiveness of linear models can be simply improved by the introduction of new attributes which are non-linear functions of existing attributes:

$$h(\underline{a}) = \sum_{i=1}^{d} w_i \cdot f_i(\underline{a}), \tag{2.18}$$

where d is the number of such attributes and f_i can be arbitrary functions. These models are called generalized linear models. For example, quadratic models can be represented by using functions f_i which are products of pairs of attributes. Since model (2.18) is still linear in the transformed input space, optimization of parameters w_i, $i = 1, \ldots, d$, is done similarly as the optimization of non generalized linear models.

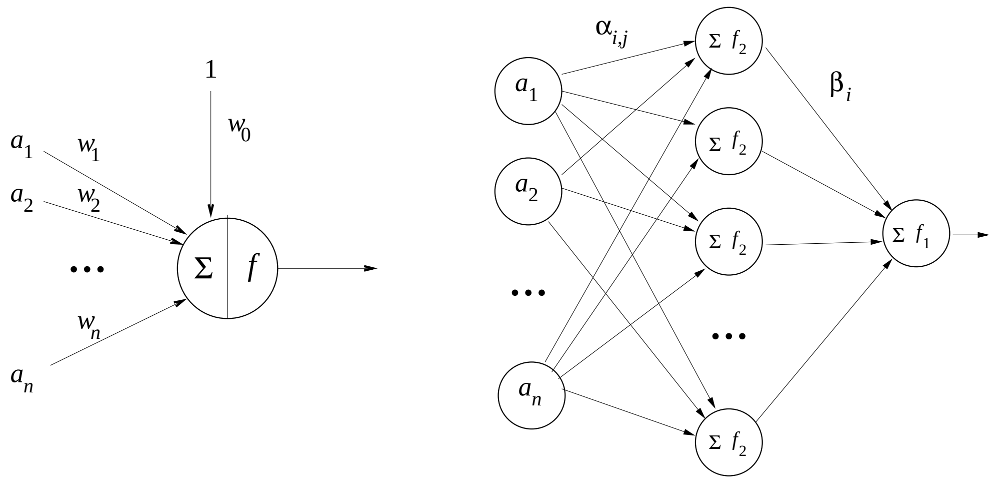


Figure 2.1: Left, a perceptron, right, a three layer neural network with one output

2.3.2 Artificial neural networks

Artificial neural networks are a generalization of linear models inspired by analogies with the biological brain. A perceptron is a simple unit which computes the following function:

$$h(\underline{a}) = f(w_0 + \sum_{i=1}^{n} w_i \cdot a_i),$$

where the activation function f is a non linear function of its argument. Examples are the sign function:

$$f(x) = \operatorname{sgn}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases},$$

or the sigmoid function:

$$f(x) = \frac{1}{1 + e^{-x}}.$$

Note that the first one gives a classification rule similar to (2.17). A single perceptron is thus essentially equivalent to a linear model.

The idea of neural networks is to combine such simple units in order to improve expressiveness. In feed-forward multilayer architectures, the neural networks are organized into successive layers of perceptrons where the outputs of a layer are only connected to the inputs of the next layers. A perceptron and an example of a three-layer neural networks is depicted in Figure 2.1. The first layer is the input layer which transmits the input attribute values to the second layer. The last layer is the output layer which computes the output value. Layers between the input and output layers are called hidden layers (since their outputs do not correspond to observed variables). A neural network with a single hidden layer like the one of Figure 2.1 represents the following parametric family of functions:

$$h(\underline{a}) = f_1(\beta_0 + \sum_{i=1}^{L} \beta_i f_2(\alpha_{i,0} + \sum_{j=1}^{n} \alpha_{i,j} \cdot a_i), \tag{2.19}$$

where L is the number of units in the hidden layer, f_1 and f_2 are respectively the activation functions of the output and hidden layers (which do not need to be identical), and $\alpha_{i,j}, \beta_i$ are the parameters. It has been shown that a neural network with sufficient number of hidden layers and units is able to represent any function with arbitrary accuracy (this is called the universal approximation property, see e.g. [Bar94]).

Learning in neural networks is also an optimization in a parameters space. When the activation and the error functions are smooth, there exists an efficient algorithm to compute the gradient of the error with respect to the vector of parameters (the back-propagation algorithm). This gradient is then used to iteratively modify the parameters in the direction of a minimization of error:

$$\underline{w}^{new} = \underline{w}^{old} - \eta \cdot \nabla_{\underline{w}} \hat{Err}(h, LS), \tag{2.20}$$

where η is called the learning rate, and $\underline{w}$ denotes the vector of parameters. The learning algorithm starts with random initial weights and then iteratively improves them using (2.20) until a local minimum of the error is reached. Several variants of this algorithm have been introduced to accelerate convergence (by introduction of second derivatives in (2.20) for example). In general, (2.20) may have several local minima and thus the hypothesis returned by the learning algorithm may depend to some extent on the (randomized) initial conditions.

As mentioned previously, classification problems are handled with neural networks by the introduction of the numerical class indicator variables Y_C^i defined by (2.14).

2.3.3 K-Nearest Neighbors

For fixed k and fixed learning sample, the hypothesis produced by this method predicts at each point an aggregation of the outputs of its k nearest neighbors in the learning set according to some a priori defined distance measure. When attributes are numerical, the distance is often a weighted Euclidian distance where weights are usually chosen as the inverse variance of the corresponding attribute so as to give every attribute equal importance:

$$d(\underline{a}, \underline{a}') = \sum_{i=1}^{n} \frac{1}{\sigma_i^2}(a_i - a_i')^2,$$

where σ_i is the standard deviation of attribute A_i (estimated from the learning set). The usual aggregated predictions are the average output for regression:

$$h(\underline{a}) = \frac{1}{k} \sum_{(\underline{a},y) \in NN(\underline{a},LS,k)} y,$$

and the majority class for classification:

$$h(\underline{a}) = \arg\max_{c_i \in \mathcal{Y}} \sum_{(\underline{a},y) \in NN(\underline{a},LS,k)} 1(y = c_i),$$

where the set $NN(\underline{a}, LS, k)$ contains the k nearest neighbors of $\underline{a}$ in LS.

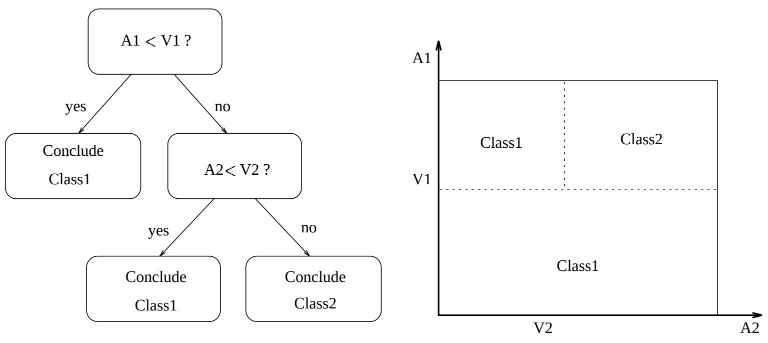
In practice the optimal value of k will depend on the learning sample size N and on the problem specifics (dimensionality of the input space and smoothness of the Bayes model). Note that when N tends towards infinity and k is appropriately adjusted to N, the hypothesis produced by this method converges to the Bayes model under fairly unrestrictive assumptions [DHS00].

2.3.4 Decision and regression trees

As this method will be studied in detail in the sequel of the thesis, the tree induction algorithm that we have been using in our experiments will be explained precisely in later chapters. We therefore restrict the following presentation to a very brief introduction of the general principles of the so-called TDIDT family of algorithms (for Top Down Induction of Decision Trees).

The hypothesis space of decision/regression trees is the set of piecewise constant functions $h(.)$ defined on the input space by:

$$h(\underline{a}) = \begin{cases} h_1 & \text{if } \underline{a} \in R_1 \\ h_2 & \text{if } \underline{a} \in R_2 \\ \dots & \\ h_L & \text{if } \underline{a} \in R_L \end{cases}, \tag{2.21}$$



$$
\begin{array}{c}
\text{if } a_1 < v_1 \text{ then } C = c_1 \\
\text{if } a_1 \geq v_1 \text{ and } a_2 < v_2 \text{ then } C = c_1 \\
\text{if } a_1 \geq v_1 \text{ and } a_2 \geq v_2 \text{ then } C = c_2
\end{array}
$$

Figure 2.2: A decision tree in a two-dimensional input space, the corresponding rule and the resulting partition of the input space

where $R_1, \ldots, R_L$ is a recursive partition of the input space into L regions and h_i is the constant prediction associated to each region R_i (a class or a number).

In order to represent the structure of the recursive partition and how it is defined in terms of the input attributes, a decision tree structure is used. Each interior node of such a tree is labeled with a test defined as a function of the attribute values and each terminal node defines a region R_i. To find the region associated to a point $\underline{a}$, it is sufficient to propagate it into the tree according to test issues until it reaches a terminal node. Each region R_i is thus described by a sequence of tests and the whole tree simply translates into a set of "If ... then ..." rules. In the classical method, tests are binary and involve only one attribute at the time: for numerical attributes they compare the attribute value to a threshold, resulting in a partition of the input space into hyper-rectangular regions. Figure 2.2 shows an example tree and the resulting partition of the input space.

As it is often practically infeasible to compute the best tree of a given complexity for a given problem (finding a tree of fixed complexity minimizing (2.12) is NP-complete), decision tree induction uses a hill-climbing algorithm composed of two stages, namely tree growing and tree pruning. The growing algorithm starts with a one node tree and tries to split this node by selecting a test among a set of candidate tests. The algorithm then proceeds recursively to split the left and right successors of this node. Predictions h_i at leaves are estimated from the subset of learning samples which belong to the corresponding region R_i, by conditional class frequency counts and majority vote in classification and by averaging the output values of these objects in regression. Typically, tree growing proceeds until the terminal nodes correspond to sub-samples in which the output variable is (almost) constant. In order to reduce overfitting it is followed by a pruning stage which aims at determining the correct complexity of the tree by replacing subtrees of the fully grown tree by terminal nodes. The pruning procedure operates in a bottom up fashion.

The decision tree induction method used in our experiments is described in full detail in Section 5.2 (growing) and Section 6.2 (pruning).

2.3.5 Naive Bayes and Bayesian networks

The previously described family of algorithms seeks for a direct approximation of $P(C|\underline{A})$ or $E_{Y|\underline{A}}\{Y\}$ and so are termed discriminative or predictive approaches. Another class of automatic learning methods proceed by first modeling the joint distribution $P(\underline{A}, Y)$ and then inferring

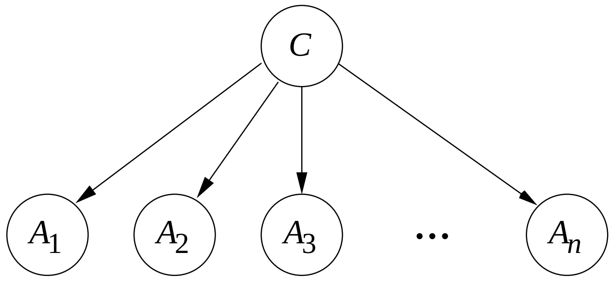


Figure 2.3: The Bayesian network for the Naive Bayes method. Each node represents a random variable and is independent of its non descendants given its parents in the graph.

the conditional distribution $P(Y|\underline{A})$ from the former according to the Bayes rule, i.e.

$$P(Y|\underline{A}) = \frac{P(\underline{A}, Y)}{P(\underline{A})},$$

in order to eventually make predictions according to (2.9) or (2.10). In order to make the learning of $P(\underline{A}, Y)$ easier, some assumptions about its structure must be made. Since the early eighties a large body of work has been carried out on the appropriate modeling of modularity in the context of probabilistic reasoning, which has led to various classes of modular probabilistic models. In this context, modularity essentially corresponds to conditional independence assumptions among sets of random variables, and this modularity is expressed by the factorization of the joint distribution of all variables into a product of functions involving each one only a small number of variables [CDLS99]. The most well-known type of graphical models are the so-called Bayesian networks, where the factorization is graphically represented by a directed acyclic graph (DAG) each node of which corresponds to one of the variables involved in the modeled joint distribution, and where the component functions correspond to conditional probability distributions of each variable given the parents of this variable in the DAG [Pea88].

The naive Bayes method in classification corresponds to a very simple type of Bayesian network which assumes that:

$$P(\underline{A}, C) = P(C) \cdot P(A_1|C) \cdots P(A_n|C), \tag{2.22}$$

which translates the conditional independence of the attributes given the output variable. In terms of graphical structure, this factorization is represented in Figure 2.3 by a shallow directed graph the top-node of which corresponds to variable C and where the nodes corresponding to the input attributes A_i have this node as single parent.

Once this independence hypothesis is assumed, the automatic learning problem reduces to the estimation of $P(C)$ and of the $P(A_i|C)$ for each input attribute on the basis of the learning sample. Thus, in the standard Naive Bayes method the prior class probabilities $P(C = c_i)$ are estimated by counting the frequency of occurrence of class c_i in the learning set (i.e. the maximum likelihood estimates). For a symbolic attribute,, $P(A_i|C = c_i)$ usually is also estimated by direct counting. For numerical attributes, several parametric and non parametric techniques are possible. For example, the distribution $p(A_i|c_i)$ can be assumed to be Gaussian and the parameters of this distribution, the mean and standard deviation, can be estimated from the learning sample. Among non parametric techniques, the most simple one is the histogram. Histogram consists in dividing the range of the attribute A_i into subregions (or bins) and counting the relative number of examples (of a fixed class) falling into each subregion.

Of course, the hypothesis of conditional independence of the attributes given the class is too strong in many problems. Actually, in the special case where distributions $P(A_i|C_i)$ are assumed to be Gaussian, the naive Bayes model is nothing but a quadratic discriminant. More general Bayesian network structure have been also used for classification (see [FGG97] for example).

2.4 Evaluation of learning algorithms

Three main criteria are used to compare learning algorithms:

- **Accuracy**: the way it succeeds in reducing the error.

- **Computational efficiency**: the time needed to learn a model but also the time to apply this model to new cases.

- **Interpretability**: if it gives comprehensible models or not.

Large experiments comparing the accuracy of several learning algorithms (see for example [MS94] or more recently [LLS00]) show that the exact ranking among algorithms depends on the problem and how well the basic hypotheses of the learning algorithm (for example, linearity) are satisfied by this problem. However, neural networks and nearest neighbors are often among the most accurate methods. Although competitive in some problems (for example, when there are redundant or useless attributes), tree based models are often not as accurate as these two methods. Of course, the strong assumptions behind linear models and the Naive Bayes prevent them to reach high accuracy in many problems. Nevertheless, in some cases, these methods are competitive with other more complex algorithms such as decision trees (see for example [DP97] for the Naive Bayes). In the next chapter, we will show that this apparent inconsistency can be explained by analyzing the variance of these methods.

The precise ranking of learning algorithms in terms of computational efficiency would be possible only in the context of specific implementations (the general methods that we have described can be implemented in different ways, with quite significant impacts on their efficiency). Also, the problem size in the context of automatic learning is determined by two parameters characterizing the database size, namely the number of objects N and number of attributes n, which may have a different impact on computational efficiency. Nevertheless, for medium problem sizes (say a few thousand objects and a few hundred attributes) a crude ranking of these methods can be done. For example, because of their simplicity, the learning step of linear models and Naive Bayes are very fast. Although slower, decision tree induction remains very efficient even for large sample sizes and large numbers of variables. The basic K-NN method does not require any learning phase (except for the optimization of K). At the very end of this ranking are neural networks. Indeed, especially complex neural networks can take substantial time to be optimized. On the other hand, new predictions are computed very quickly with all models except for K-NN which requires to compute the distances to each element of the learning set.

Interpretability is highly subjective and also problem dependent. However, it is generally accepted that neural networks are black-box type of models and that on the other side, decision trees are very comprehensible. Linear model and Naive bayes because of their simplicity are also termed interpretable. Nearest neighbors techniques, although they do not provide any synthetic model, can still give useful information by retrieving similar cases in a data set.

In general, there is a tradeoff between these three criteria. An accurate method is likely to be resource demanding or induces non comprehensible models, while interpretable models are often not very accurate. Also, the relative importance assigned to each criterion is highly application-dependent. Sometimes, it will be acceptable to sacrifice some accuracy in order to gain interpretability or computational efficiency. In other cases, accuracy has to be optimized as far as possible, whatever the price to pay in terms of interpretability or computation times. This makes existing algorithms complementary and none of them can be claimed to be globally superior to all other ones.

In this thesis, new improvements or algorithms will be discussed having in mind these three criteria.

2.5 Evaluation of accuracy of models

The evaluation of a model is important first in order to predict its future performance on new data but also in order to provide the learning algorithm with a criterion to choose a model in its hypothesis space. Suppose we want to evaluate the true error (see eqn. (2.2)) of a model f_{ls} induced from a learning set ls by a given learning algorithm and let us denote an error estimate for this model by $\hat{Err}(f_{ls})$.

The *resubstitution estimate* (which is equivalent to the empirical risk) directly evaluates the accuracy on the learning set used to induce the model:

$$\hat{Err}(f_{ls}) = \hat{Err}(f_{ls}, ls), \tag{2.23}$$

where $\hat{Err}(f_{ls}, ls)$ is given by (2.12). While the resubstitution estimate is useful to guide learning algorithms, the error estimation it gives for the induced model is likely to be very optimistic since most of the learning algorithms explicitly try to reduce this estimate.

A more reliable estimate would be obtained from a data set independent of the sample of data used during induction. For example, a *hold-out estimate* consists in partitioning ls into two disjoint subsets vs and $ls \setminus vs$. A new model is induced from $ls \setminus vs$ and the error is estimated by:

$$\hat{Err}(f_{ls}) = \hat{Err}(f_{ls\setminus vs}, vs), \tag{2.24}$$

If vs is too small, this estimate is not stable and if vs is too large, the estimate will be pessimistic as the truly assessed model, f_{ls}, is induced from a larger learning set. A common practice is to save one third of the data for validation.

In cases where the learning sample is very small, even one third of the data will yield an unstable estimate of the error. *Cross-validation* is a way to stabilize error estimates with small validation set vs. In k-fold cross-validation, we divide the learning set into k disjoint subsets of the same size, $ls = ls_1 \cup ls_2 \cup \ldots \cup ls_k$. A model is built from every set $ls \setminus ls_i, i = 1, \ldots, k$ and the error is estimated either by:

$$\hat{Err}(f_{ls}) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, f_{ls\setminus ls_{k(i)}}(\underline{a}_i)), \tag{2.25}$$

where $k(i)$ is the index of the subset ls_i of ls that includes the instance $(\underline{a}_i, y_i)$, or by:

$$\hat{Err}(f_{ls}) = \frac{1}{k}\sum_{i=1}^{k} \hat{Err}(f_{ls\setminus ls_i}, ls_i). \tag{2.26}$$

When the number of objects is an exact multiple of the number of folds, equations (2.25) and (2.26) are in fact equivalent. When this is not the case, the first one should be preferred, especially in the case of small samples, as it gives identical weights to each instance. When k is equal to the number of samples in the learning set, this method is called leave-one-out. Except for some learning algorithms like K-NN, small values of k (between 10 and 20) are preferred for computational efficiency reasons. Stratified cross-validation is a variant of cross-validation where the class proportions in the learning set are kept into each fold.

When comparing different learning algorithms or variants of the same algorithm, it is not crucial to use the whole data set to build models and so the fact that hold-out estimates give pessimistic estimates of the error of a model induced from the whole dataset is not important as long as it gives a reliable estimate of the error of the model induced from the reduced learning set. So in what follows, when comparing algorithms, we will first divide the available data into two disjoint subsets: one subset devoted to the induction of models and the other set devoted to their validation. Occasionally, when the available data set will be too small to allow reliable estimates from hold-out estimate, 10-fold cross-validation will be preferred.

At this point, it is important to note that the methods explained here do not give an idea of the general accuracy of a learning algorithm, but only of a model induced from a particular learning set in the context of one particular problem. If learning algorithms have to be compared, other criteria have to be taken into account. For example, the next chapter will give some guide to study the general behavior of a learning algorithm when it is faced with different learning sets. Another criterion is its behavior on a large spectrum of problems. To this end, there exist several repositories of data sets which are used as benchmarks by many researchers to compare algorithms (for example [BM98]).

Chapter 3

Bias and variance in supervised learning

In this chapter, the important concepts of bias and variance are introduced. After an intuitive introduction to the bias/variance compromise, we discuss in Section 3.2 the bias/variance decompositions of the mean square error (in the context of regression problems) and of the classification error (in the context of classification problems). Section 3.3 provides an empirical study through various experiments which aim at providing some insight about how the parameters of the learning problem and of the learning algorithm influence bias and variance. Section 3.4 discusses the differences between two notions of variance: error variance and parameter variance. The last section of this chapter provides some analytical characterizations of bias and variance obtained in the context of statistical learning theory, in particular quantitative error bounds obtained in the context of neural networks.

3.1 Why more general is not necessarily better

As introduced in the previous chapter, the problem of supervised learning appears to be very straightforward and could be reduced to a simple optimization problem. Indeed, at first sight, it is "sufficient" to find a model in the family $\mathcal{H}$ which realizes the minimal error on the learning set. Further, if no good enough model is found in this family, it should be sufficient to extend the family or to exchange it for a more powerful one in terms of model flexibility. Thus, to avoid too many trials and errors before getting acceptable accuracy, it would be quite natural to directly use the largest available hypothesis space (e.g. one having the so-called universal approximation property). Unfortunately, in practice, good results on the learning set do not necessarily imply good generalization performance on the universe $\mathcal{U}$ (which is the actual goal of supervised learning), especially if the "size" of the hypothesis space is large in comparison to the sample size.

To explain intuitively why larger hypothesis spaces do not necessarily lead to better models, we use the simple unidimensional regression problem described in the top of Figure 3.1. In this synthetic problem, learning outputs are generated according to

$$Y = f_B(A) + \varepsilon,$$

where ε is distributed according to $N(0, \sigma)$. With squared error loss, the best possible model is thus f_B and its average (and also local) squared error is σ^2. Let us consider two extreme situations of a bad model structure choice.[1]

- A too simple model structure: using a linear model $Y = w \cdot A + b$ and minimizing squared error on the learning set, we obtain the estimations given in the left part of Figure 3.1

[1]Notice that in this example both A and Y are *continuous* real valued variables. For the sake of uniformity of notation, we nevertheless proceed by using upper case letters to denote the probability distribution of the variables.

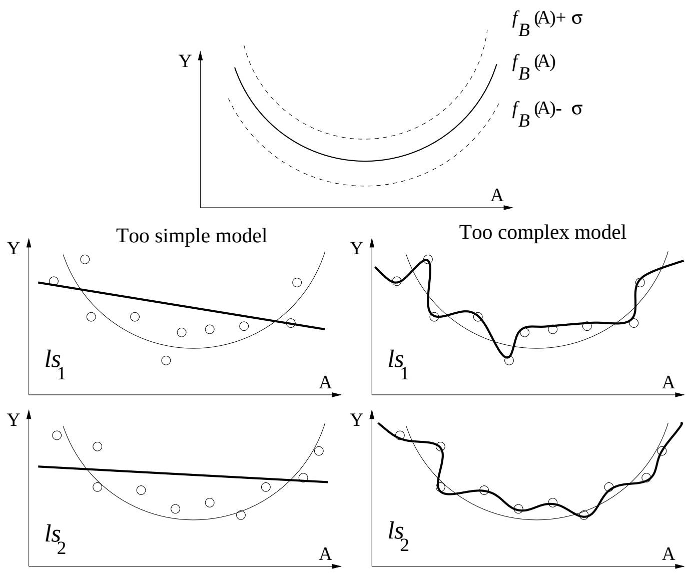


Figure 3.1: Top, our illustrative problem. Left, a linear model fitted to two learning samples. Right, a neural network with two hidden layers of 10 neurons fitted to the same samples

for two different learning set choices. These models are not very good, neither on their learning sets, nor in generalization. Whatever the learning set, there will always remain an error due to the fact that the model is too simple with respect to the complexity of f_B.

- A too complex model: by using a very complex model like a neural network with two hidden layers of ten neurons each, we get the functions at the right part of Figure 3.1 for the same learning sets. This time, models receive an almost perfect score on the learning set. However, their generalization errors are still not very good because of two phenomena. First, the learning algorithm is able to match perfectly the learning set and hence also the noise term. We say in this case that the learning algorithm "overfits" the data. Second, even if there is no noise, there will still remain some errors due to the high complexity of the model. Indeed, the learning algorithm has many different models at its disposal and if the learning set size is relatively small, several of them will realize a perfect match of the learning set. As at most one of them is a perfect image of the Bayes model, any other choice by the learning algorithm will result in suboptimality.

The main source of error is very different in both cases. In the first case, the error is essentially independent of the particular learning set and must be attributed to the lack of complexity of the model. This source of error is called **bias**. In the second case, on the other hand, the error may be attributed to the variability of the model from one learning set to another (which is due on one hand to overfitting and on the other hand to the sparse nature of the learning set with respect to the complexity of the model). This source of error is called **variance**. Note that in the first case there is also a dependence of the model on the learning set and thus some variability of the predictions. However the resulting variance is negligible with respect to bias. In general, bias and variance both depend on the complexity of the model but in opposite direction and thus there must exist an optimal tradeoff between these two sources of error. As a matter of fact (see subsequent discussions), this optimal tradeoff depends also on the smoothness of the Bayes model and on the sample size. An important consequence of this is that automatic learning as *empirical risk minimization in a fixed hypothesis space*, as formulated in the previous chapter, is not suitable for most practical problems. Indeed, because of variance, we should always take care of not increasing too much the complexity of the model structure with respect to the complexity of the problem and the size of the learning sample.

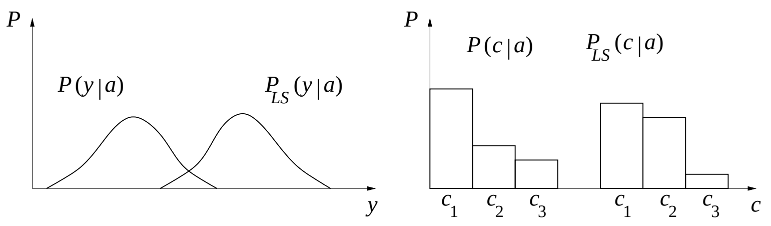


Figure 3.2: distributions of outputs and model predictions at a point $\underline{a}$, left in regression, right in classification

In the next section, we give a formal additive decomposition of the mean (over all learning set choices) squared error into two terms which represent the bias and the variance effect. Some propositions of similar decompositions in the context of 0-1 loss-functions are also discussed. They show some fundamental differences between the two types of problems although bias and variance concepts are always useful.

3.2 Bias and variance decompositions

A learning algorithm produces a model f_{LS} from a learning sample LS of size N[2]. For a fixed sample size N, LS is a random variable. Hence, the model f_{LS} and its prediction $f_{LS}(\underline{a})$ at $\underline{a}$ are also random. In what follows, we will denote by $P_{LS}(y|\underline{a})$ the probability distribution of this output, i.e. the probability that a model (produced by a given learning algorithm, from samples of size N corresponding to a given learning problem) outputs y at the point $\underline{a}$:

$$P_{LS}(y|\underline{a}) \overset{\triangle}{=} P(f_{LS}(\underline{a}) = y) = \sum_{ls \in (\underline{\mathcal{A}}, \mathcal{Y})^N | f_{ls}(\underline{a}) = y} P(LS = ls). \tag{3.1}$$

The error of these models $Err(f_{LS})$ is again a random variable and we are interested in studying the expected value of this error (over the set of all learning sets of a given size N) $E_{LS}\{Err(f_{LS})\}$. The expectations may be inverted to give:

$$\begin{aligned} E_{LS}\{Err(f_{LS})\} &= E_{LS}\{E_{\underline{A},Y}\{L(Y, f_{LS}(\underline{A}))\}\} & (3.2) \\ &= E_{\underline{A}}\{E_{LS}\{E_{Y|\underline{A}}\{L(Y, f_{LS}(\underline{A}))\}\}\} & (3.3) \\ &= E_{\underline{A}}\{E_{LS}\{Err(f_{LS}(\underline{A})\}\}, & (3.4) \end{aligned}$$

where $Err(f_{LS}(\underline{A}))$ is the local error at point $\underline{A}$. Note here the symmetry (interchangeability) between the two random variables Y and $f_{LS}(\underline{A})$ in (3.3). The first one is distributed at $\underline{a}$ according to $P(y|\underline{a})$ ($= P(Y = y|\underline{A} = \underline{a})$) while the second one is distributed according to $P_{LS}(y|\underline{a})$. These distributions are illustrated in Figure 3.2, left in the case of a regression problem (density), right in the case of a three-class classification problem (discrete probability). The bias/variance decompositions will be expressed in terms of the parameters of these two distributions.

3.2.1 Regression error

In the case of a regression problem and if we use the square error criterion as a loss measure, we know that the best possible model is the function $f_B(\underline{a})$ which associates to each input $\underline{a}$ the expectation of Y according to the conditional probability distribution $p(Y|\underline{a})$. Introducing this model in $E_{LS}\{Err(f_{LS}(\underline{a}))\}$, we get:

[2]For simplicity reasons, the dependence on the learning set size N and on the learning algorithm will be forgotten in the notations throughout this chapter.

$$\begin{aligned}
E_{LS}\{Err(f_{LS}(\underline{a}))\} &= E_{LS}\{E_{Y|\underline{a}}\{(Y - f_{LS}(\underline{a}))^2\} && (3.5)\\
&= E_{LS}\{E_{Y|\underline{a}}\{(Y - f_B(\underline{a}) + f_B(\underline{a}) - f_{LS}(\underline{a}))^2\} && (3.6)\\
&= E_{LS}\{E_{Y|\underline{a}}\{(Y - f_B(\underline{a}))^2\} + E_{LS}\{E_{Y|\underline{a}}\{(f_B(\underline{a}) - f_{LS}(\underline{a}))^2\} +\\
&\quad E_{LS}\{E_{Y|\underline{a}}\{2 \cdot (Y - f_B(\underline{a})) \cdot (f_B(\underline{a}) - f_{LS}(\underline{a}))\} && (3.7)\\
&= E_{Y|\underline{a}}\{(Y - f_B(\underline{a}))^2\} + E_{LS}\{(f_B(\underline{a}) - f_{LS}(\underline{a}))^2\}, && (3.8)
\end{aligned}$$

since:

$$E_{Y|\underline{a}}\{(Y - f_B(\underline{a}))\} = E_{Y|\underline{a}}\{Y\} - f_B(\underline{a}) = 0 \qquad (3.9)$$

by definition of the Bayes model. Symmetrically to the Bayes model, we can define the average model, $\overline{f}_{LS}$, which outputs the average prediction among all learning sets:

$$\overline{f}_{LS}(\underline{a}) = E_{LS}\{f_{LS}(\underline{a})\}. \qquad (3.10)$$

Introducing this model in the second term of (3.8), we obtain:

$$\begin{aligned}
E_{LS}\{(f_B(\underline{a}) - f_{LS}(\underline{a}))^2\} &= E_{LS}\{(f_B(\underline{a}) - \overline{f}_{LS}(\underline{a}) + \overline{f}_{LS}(\underline{a}) - f_{LS}(\underline{a}))^2\} && (3.11)\\
&= E_{LS}\{(f_B(\underline{a}) - \overline{f}_{LS}(\underline{a}))^2\} + E_{LS}\{(f_{LS}(\underline{a}) - \overline{f}_{LS}(\underline{a}))^2\} +\\
&\quad E_{LS}\{2 \cdot (f_B(\underline{a}) - \overline{f}_{LS}(\underline{a})) \cdot (\overline{f}_{LS}(\underline{a}) - f_{LS}(\underline{a}))\} && (3.12)\\
&= (f_B(\underline{a}) - \overline{f}_{LS}(\underline{a}))^2 + E_{LS}\{(f_{LS}(\underline{a}) - \overline{f}_{LS}(\underline{a}))^2\} && (3.13)
\end{aligned}$$

since:

$$E_{LS}\{\overline{f}_{LS}(\underline{a}) - f_{LS}(\underline{a})\} = \overline{f}_{LS}(\underline{a}) - E_{LS}\{f_{LS}(\underline{a})\} = 0 \qquad (3.14)$$

by definition of $\overline{f}_{LS}(\underline{a})$. In summary, we have the following well-known decomposition of the square error at point $\underline{a}$:

$$E_{LS}\{Err(f_{LS}(\underline{a}))\} = \sigma^2_R(\underline{a}) + \text{bias}^2_R(\underline{a}) + \text{var}_R(\underline{a}), \qquad (3.15)$$

by defining:

$$\begin{aligned}
\sigma^2_R(\underline{a}) &= E_{Y|\underline{a}}\{(Y - f_B(\underline{a}))^2\}, && (3.16)\\
\text{bias}^2_R(\underline{a}) &= (f_B(\underline{a}) - \overline{f}_{LS}(\underline{a}))^2, && (3.17)\\
\text{var}_R(\underline{a}) &= E_{LS}\{(f_{LS}(\underline{a}) - \overline{f}_{LS}(\underline{a}))^2\}. && (3.18)
\end{aligned}$$

This error decomposition is well-known in estimation theory and has been introduced in the automatic learning community by Geman et al. [GBD92].

The residual square error, $\sigma^2_R(\underline{a})$, is the error obtained by the Bayes model and it provides a theoretical lower bound which is independent of the learning algorithm. Thus the sub-optimality of a particular learning algorithm (its error with respect to the Bayes model, i.e. the second term of (3.8)) is composed of two terms:

- The (squared) bias, $\text{bias}^2_R(\underline{a})$, measuring the discrepancy between the Bayes model and the average model.
- The variance, $\text{var}_R(\underline{a})$, measuring the variability of the predictions with respect to the learning set randomness.

These three terms are illustrated in Figure 3.3. Residual error and variance are measures of the spread of the two densities and the bias is simply the distance between their means.

To explain why these two terms are effectively the consequence of the two phenomena discussed in the introduction of this chapter, we will come back to our simple regression problem. The average model is depicted in the top of Figure 3.4 for the two cases of bad model choice. Residual error, bias and variance for each position a are also drawn in the center part of the

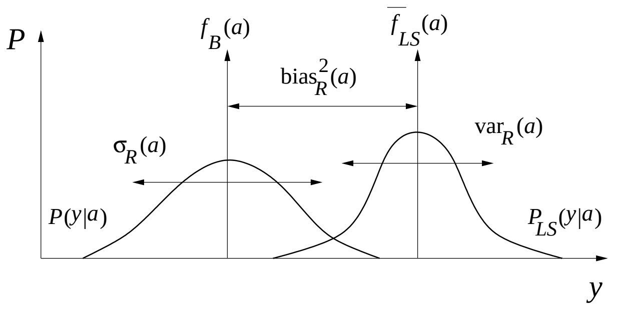


Figure 3.3: Residual error, regression bias and variance

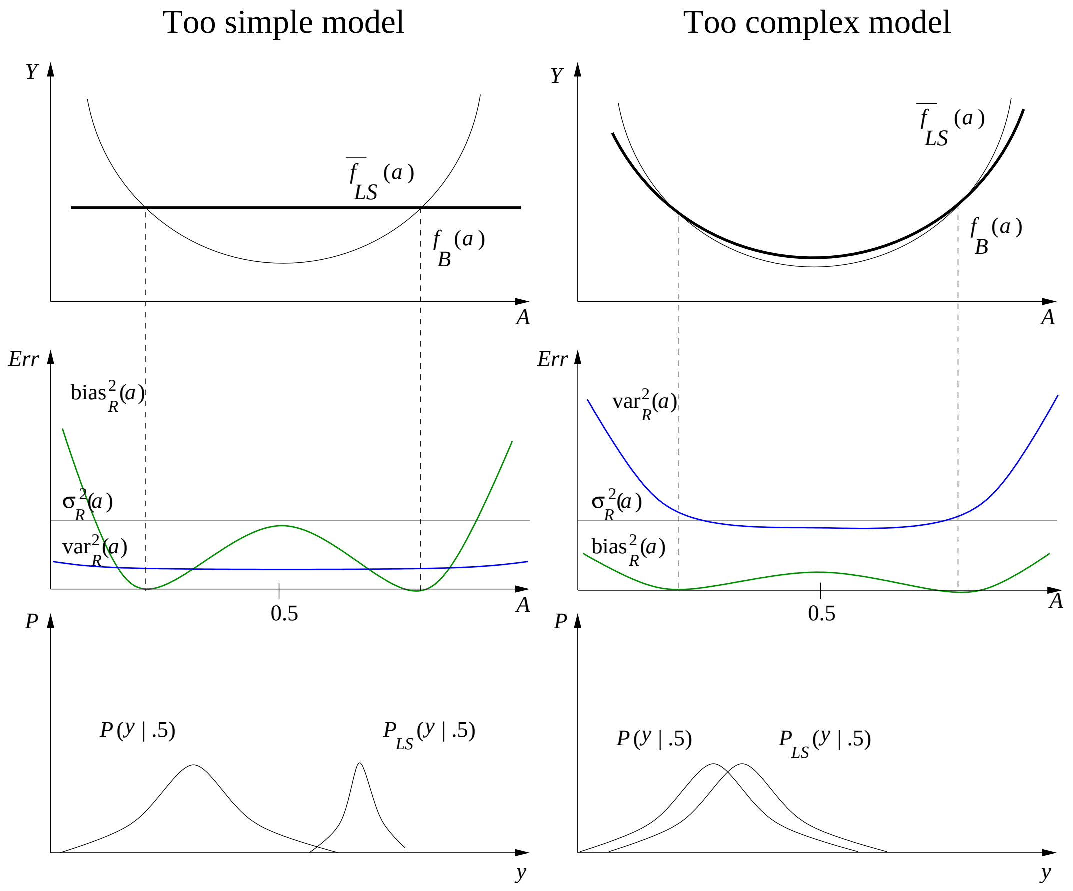


Figure 3.4: top: average model; center: residual error, bias, and, variance; bottom: distributions of outputs and predictions at the point $A = 0.5$

same figure. An illustration of the distributions of true and learned outputs are reproduced at the bottom of the figure. The residual error is entirely specified by the problem and loss criterion and hence independent of the algorithm and learning set used. When the model is too simple, we see on Figure 3.4 that the average model is far from the Bayes model almost everywhere and thus bias is high. The learning algorithm is not powerful enough to represent the Bayes model. On the other hand, the variance is low as the model does not match very strongly the learning set and thus the prediction at each point does not vary too much from one learning set to another. Bias is thus the dominant term of error. When the model is too complex, the distribution of predictions matches very strongly the distribution of outputs at each point. The average prediction is thus close to the Bayes model and the bias is low. However, because of the noise and the small learning set size, predictions are highly variable at each point. In this case, variance is the dominant term of error.

3.2.2 Classification error

The local mean error in classification is the probability of misclassification at point $\underline{a}$:

$$\begin{aligned} E_{LS}\{Err(f_{LS}(\underline{a}))\} &= E_{LS}\{E_{C|\underline{a}}\{1(C \neq f_{LS}(\underline{a}))\}\} && (3.19)\\ &= P(C \neq f_{LS}(\underline{a})|\underline{a}) && (3.20)\\ &= 1 - \sum_c P(c|\underline{a}) \cdot P_{LS}(c|\underline{a}). && (3.21)\end{aligned}$$

The Bayes model in classification associates to each input $\underline{a}$ the most likely class according to the conditional class probability distribution, i.e. $f_B(\underline{a}) = \arg\max_c P(c|\underline{a})$. The residual or irreducible error at point $\underline{a}$, noted[3] $\sigma_C(\underline{a})$, becomes:

$$\sigma_C(\underline{a}) = 1 - P(f_B(\underline{a})|\underline{a}). \tag{3.22}$$

To keep symmetry with the Bayes model and by analogy with the regression problem, the equivalent in classification of the average model of regression is the *majority vote classifier* defined by:

$$f_{LS}^{MAJ}(\underline{a}) = \arg\max_c P_{LS}(c|\underline{a}), \tag{3.23}$$

which outputs at each point the class receiving the majority of votes among the distribution of classifiers induced from the distribution of learning sets (of size N). The bias in regression is the error of the average model with respect to the Bayes model. This definition yields here:

$$\text{bias}_C(\underline{a}) = 1(f_B(\underline{a}) \neq f_{LS}^{MAJ}(\underline{a})). \tag{3.24}$$

So, biased points are those for which the majority vote classifier disagrees with the Bayes classifier. On the other hand, variance can be defined as:

$$\text{var}_C(\underline{a}) = 1 - P_{LS}(f_{LS}^{MAJ}(\underline{a})|\underline{a}). \tag{3.25}$$

Indeed, this definition is certainly a measure of the variability of the predictions at a point $\underline{a}$:

- when $\text{var}_C(\underline{a}) = 0$, every model outputs the same class whatever the learning set from which it is induced;

- $\text{var}_C(\underline{a})$ is maximal when the probability of the class given by the majority vote classifier is equal to $\frac{1}{m}$ (with m the number of classes), which indeed corresponds to the most uncertain distribution of classifiers.

[3] In our error decompositions for classification, we made the arbitrary choice of not using the "square" notations. Hence, we will use σ_C instead of σ_C^2 and bias_C instead of bias_C^2.

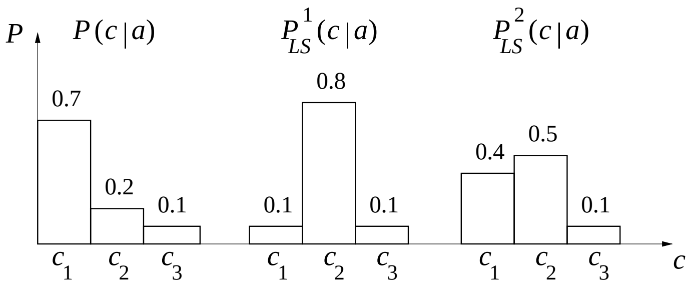


Figure 3.5: left, a classification problem; center and right, two different learning algorithms

Unfortunately, defined in this way by analogy with the regression case, residual error, bias and variance do not sum up to give the local classification error. In other words:

$$E_{LS}\{Err(f_{LS}(\underline{a}))\} \neq \sigma_C(\underline{a}) + \text{bias}_C(\underline{a}) + \text{var}_C(\underline{a}). \tag{3.26}$$

Indeed, let us illustrate on a simple example how increased variance may decrease the average classification error in some situations. Figure 3.5 represents (at a certain point $\underline{a}$ of the input space) three probability distributions: the leftmost histogram represents the true conditional class probability distribution $P(c|\underline{a})$, showing also that the Bayes classifier would decide class c_1 at this point; the two other histograms represent the sampling distributions of the class decided by models produced by two different learning algorithms.

From Figure 3.5 we observe that both algorithms produce models which most probably will decide class c_2 (respectively with probability 0.8 and 0.5). Thus the two methods are biased ($\text{bias}^1_C(\underline{a}) = \text{bias}^2_C(\underline{a}) = 1$) since their majority vote classifiers would choose class c_2 instead of the most likely class (c_1). On the other hand, the variances of the two methods are obtained in the following way:

$$\begin{aligned} \text{var}^1_C(\underline{a}) &= 1 - 0.8 = 0.2 \\ \text{var}^2_C(\underline{a}) &= 1 - 0.5 = 0.5, \end{aligned}$$

and their average classification error rates are found to be

$$\begin{aligned} E_{LS}\{Err(f^1_{LS}(\underline{a}))\} &= 1 - 0.7 \cdot 0.1 - 0.2 \cdot 0.8 - 0.1 \cdot 0.1 = 0.76 \\ E_{LS}\{Err(f^2_{LS}(\underline{a}))\} &= 1 - 0.7 \cdot 0.4 - 0.2 \cdot 0.5 - 0.1 \cdot 0.1 = 0.61. \end{aligned}$$

Thus among these two methods with identical bias, it is the one having the largest variance which has the smallest average error rate.

It is easy to see that this happens here because of the existence of a bias. Indeed, in classification an algorithm which has small variance and high bias is an algorithm[4] which systematically (i.e. whatever the learning sample) produces a wrong[5] answer, whereas an algorithm which has a high bias but also a high variance is only wrong for a majority of learning samples, but not necessarily systematically. In other words, in classification, much variance can be beneficial because it can lead the system closer to the Bayes classification.

As a result of this counter-intuitive interaction between bias and variance terms in classification, a number of alternative measures of classification bias and variance have been proposed in the machine learning literature none of which is really satisfactory (a detailed discussion of the most representative decompositions found in the literature is given in Appendix A). In order to study experimentally classification algorithms, we have selected among these latter a small subset of measures which seemed the most sensible to us. These measures are discussed below. One of the goals of our experiments in subsequent chapters is to show that such measures are still useful to analyze classification learning algorithms, although not plainly satisfactory from a theoretical point of view.

[4]in our example, this corresponds to algorithm 1

[5]a class different from the one chosen by the Bayes classifier

Decomposition proposed by Tibshirani

This decomposition was proposed first by Tibshirani [Tib96] and later on by James and Hastie [JH96]. They define the bias at point $\underline{a}$ as the difference between the (local) error probability of the majority vote classifier and that of the Bayes classifier:

$$\text{bias}_T(\underline{a}) = (1 - P(f_{LS}^{MAJ}(\underline{a})|\underline{a})) - (1 - P(f_B(\underline{a})|\underline{a})). \tag{3.27}$$

This quantity is necessarily non-negative and combined with the residual error gives the classification error of the majority vote classifier:

$$\begin{aligned} \sigma_C(\underline{a}) + \text{bias}_T(\underline{a}) &= 1 - P(f_B(\underline{a})|\underline{a}) + P(f_B(\underline{a})|\underline{a}) - P(f_{LS}^{MAJ}(\underline{a})|\underline{a}) & (3.28) \\ &= 1 - P(f_{LS}^{MAJ}(\underline{a})|\underline{a}) & (3.29) \\ &= Err(f_{LS}^{MAJ}(\underline{a})). & (3.30) \end{aligned}$$

In order to obtain an additive decomposition, Tibshirani then logically defines "variance" as the difference between the expected classification error of the models produced by the algorithm and the error of the majority vote classifier:

$$\text{var}_T(\underline{a}) = E_{LS}\{Err(f_{LS}(\underline{a}))\} - Err(f_{LS}^{MAJ}(\underline{a})). \tag{3.31}$$

Because this quantity can be negative, Tibshirani names it *aggregation effect* instead of variance.

Thus the bias term in this decomposition corresponds to the additional error with respect to the residual error which would remain if we could drive variance to zero without affecting the majority vote classifier. Variance can thus be interpreted as the modification of error resulting from a complete reduction of the prediction variability. As it has been illustrated above that an increase of prediction variability may increase the error in classification, it is not surprising to note that var_T is not strictly positive. For example, for the second classifier of Figure 3.5, $E_{LS}\{Err(f_{LS}(\underline{a}))\}$ is 0.61 and $Err(f_{LS}^{MAJ}(\underline{a}))$ is 0.8, yielding an aggregation effect of -0.19. If the variance term is negative, a reduction of the prediction variability which does not affect the bias is likely to increase the error rate. On the contrary, if the variance term is positive, a reduction of the prediction variability results in a decrease of error. Notice that in many situations, the majority of points in the input space will be unbiased because most of them are located far from the optimal decision boundary, and for such points it is generally quite easy to identify the most probable class. Thus, only the points lying in regions where classes overlap may possibly be biased and hence present a negative variance. Thus, if we focus on the global expected error rate of these methods (averaged over the complete input space distribution) it is normally the case that the variance term is positive and that its reduction leads to accuracy improvement[6].

Variance measure proposed by Kohavi and Wolpert

While var_T is a measure of the effect on error of a modification of the prediction variability, it does not really measure the stability of the prediction according to the learning set randomness. Therefore we will use in our analysis of the bias/variance profile of classification algorithms another, more adequate measure of this variability, namely the variance term introduced by Kohavi and Wolpert (see [KW96] and the appendix A):

$$\text{var}_{KW}(\underline{a}) = \frac{1}{2}(1 - \sum_c P_{LS}(c|\underline{a})^2). \tag{3.32}$$

This measure is actually equivalent to an entropy measure[7] for the categorical variable $f_{LS}(\underline{a})$. It is non-negative and equal to zero if and only if $P_{LS}(f_{LS}^{MAJ}(\underline{a})|\underline{a}) = 1$, and maximal when $P_{LS}(c|\underline{a}) = \frac{1}{m}$, where m is the number of classes. However, contrary to var_T, a decrease of the value of this measure, for fixed majority vote classifier, should not be interpreted as a decrease of the average error rate.

[6]Actually, in none of our experiments, we have found a negative global variance term.

[7]Actually, it is proportional to the so-called Gini index (or quadratic entropy) of the discrete random variable $f_{LS}(\underline{a})$ (see for example [Weh96] for a discussion of entropy measures)

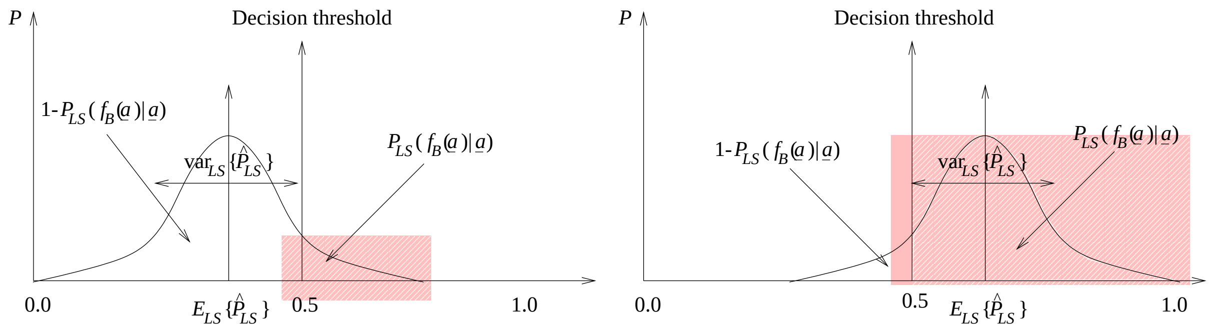


Figure 3.6: distribution of $\hat{P}_{LS}(f_B(\underline{a})|\underline{a})$, left when $E_{LS}\{\hat{P}_{LS}(f_B(\underline{a})|\underline{a})\}$ is lower than 0.5, right when $E_{LS}\{\hat{P}_{LS}(f_B(\underline{a})|\underline{a})\}$ is greater than 0.5

3.2.3 Variance of probability estimates

Many classification algorithms work by first computing an estimate $\hat{P}_{LS}(c|\underline{a})$ of the conditional class probability distribution at $\underline{a}$ and then deriving a classification model by:

$$f_{LS}(\underline{a}) = \arg\max_c \hat{P}_{LS}(c|\underline{a}). \tag{3.33}$$

Obviously, if these probability estimates have a small variance and bias the corresponding classifier is stable with respect to random variations of the learning set and close to the Bayes classifier. Thus, a complementary approach to study bias and variance of a classification algorithm is to connect, in a quantitative way, this probability estimate variance, i.e.

$$\mathrm{var}_{LS}\{\hat{P}_{LS}(c|\underline{a})\} = E_{LS}\{(\hat{P}_{LS}(c|\underline{a}) - E_{LS}\{\hat{P}_{LS}(c|\underline{a})\})^2\}, \tag{3.34}$$

to the classification error of the resulting classification rule (3.33).

We reproduce here the derivations carried out by Friedman [Fri97], in the particular case of a two-class problem. Let us denote by $f_B(\underline{a})$ the most probable class at $\underline{a}$ (i.e. the class chosen by the Bayes classifier) and by $f_{\neg B}(\underline{a})$ the other class. The mean classification error at $\underline{a}$ may then be written (with some elementary manipulations) as:

$$\begin{aligned} E_{LS}\{Err(f_{LS}(\underline{a}))\} &= 1 - P_{LS}(f_B(\underline{a})|\underline{a}) \cdot P(f_B(\underline{a})|\underline{a}) - P_{LS}(f_{\neg B}(\underline{a})|\underline{a}) \cdot P(f_{\neg B}(\underline{a})|\underline{a}) \\ &= 1 - P(f_B(\underline{a})|\underline{a}) + (1 - P_{LS}(f_B(\underline{a})|\underline{a})) \cdot (2P(f_B(\underline{a})|\underline{a}) - 1). \end{aligned} \tag{3.35}$$

Notice that in (3.35) $P_{LS}(f_B(\underline{a})|\underline{a})$ denotes the probability that a model induced from a random learning set outputs the class $f_B(\underline{a})$, and, according to (3.33), this is the case when $\hat{P}_{LS}(f_B(\underline{a})|\underline{a})$ is greater than 0.5.

In order to relate this quantity to the probability estimate variance let us have a look at the graphs of Figure 3.6 which represent two hypothetical distributions of $\hat{P}_{LS}(f_B(\underline{a})|\underline{a})$. Note that on these figures the probability $P_{LS}(f_B(\underline{a})|\underline{a})$ is represented by the area at the right of the decision threshold 0.5. Hence, $1 - P_{LS}(f_B(\underline{a})|\underline{a})$ is the area to the left of 0.5. Assuming a Gaussian distribution[8] for $\hat{P}_{LS}(f_B(\underline{a})|\underline{a})$, this area is computed by:

$$1 - P_{LS}(f_B(\underline{a})|\underline{a}) = \Phi(\frac{E_{LS}\{\hat{P}_{LS}(f_B(\underline{a})|\underline{a})\} - 0.5}{\mathrm{var}_{LS}\{\hat{P}_{LS}(f_B(\underline{a})|\underline{a})\}}) \tag{3.36}$$

where $\Phi(z)$ is the upper tail of the standard normal distribution (see Figure 3.7).

So the local error $Err(f_{LS}(\underline{a}))$ is linked to the regression variance of $\hat{P}_{LS}(f_B(\underline{a})|\underline{a})$ by the following expression:

$$E_{LS}\{Err(f_{LS}(\underline{a}))\} = 1 - P(f_B(\underline{a})|\underline{a}) + \Phi(\frac{E_{LS}\{\hat{P}_{LS}(f_B(\underline{a})|\underline{a})\} - 0.5}{\mathrm{var}_{LS}\{\hat{P}_{LS}(f_B(\underline{a})|\underline{a})\}}) \cdot (2.P(f_B(\underline{a})|\underline{a}) - 1). \tag{3.38}$$

[8]This Gaussian assumption is certainly not satisfied in all cases (e.g. $\hat{P}_{LS}(c|\underline{a})$ estimated by a tree with only pure terminal nodes will take only two values, 0 or 1), but the main conclusions drawn upon this assumption still remain valid.

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{z}^{+\infty} e^{-\frac{u^2}{2}} du. \tag{3.37}$$

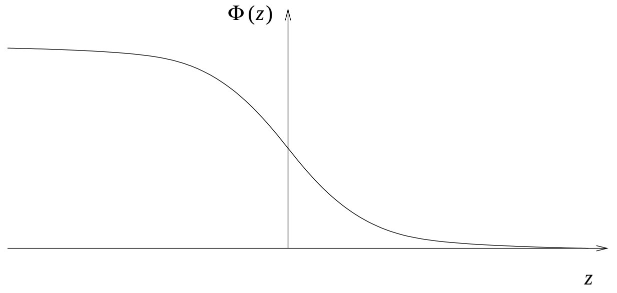


Figure 3.7: Upper tail area of the standard normal distribution

Their interaction is easy to understand from Figure 3.6 and the form of $\Phi(z)$:

- when the average probability estimate of the majority class (i.e. $E_{LS}\{\hat{P}_{LS}(f_B(\underline{a})|\underline{a})\}$) is lower than 0.5, an increase of variance will increase $P_{LS}(f_B(\underline{a})|\underline{a})$ (at the left part of Figure 3.6) and hence yield a decrease of the classification error;

- on the other hand, when $E_{LS}\{\hat{P}_{LS}(f_B(\underline{a})|\underline{a})\}$ is greater than 0.5, an increase of variance will increase $1 - P_{LS}(f_B(\underline{a})|\underline{a})$ (at the right part Figure 3.6) and hence yield an increase of the classification error.

Actually, $E_{LS}\{\hat{P}_{LS}(f_B(\underline{a})|\underline{a})\}$ greater than 0.5 corresponds to the case $f_{LS}^{MAJ}(\underline{a}) = f_B(\underline{a})$, i.e. $\text{bias}_C(\underline{a}) = 0$ and $E_{LS}\{\hat{P}_{LS}(f_B(\underline{a})|\underline{a})\}$ lower than 0.5 to the case $f_{LS}^{MAJ}(\underline{a}) \neq f_B(\underline{a})$, i.e. $\text{bias}_C(\underline{a}) = 1$. The conclusions are thus similar to what we found in our illustrative problem above: *in classification, variance is beneficial for biased points and detrimental for unbiased ones.*

Another conclusion can be drawn from (3.38): whatever the regression bias on the approximation of $\hat{P}_{LS}(f_B(\underline{a})|\underline{a})$, the classification error can be driven to its minimum value by reducing solely the variance, under the assumption that $E_{LS}\{\hat{P}_{LS}(f_B(\underline{a})|\underline{a})\}$ is greater than 0.5. This means that perfect classification rules can be induced from very bad (rather biased, but of small variance) probability estimators.

In experiments of subsequent chapters, we will often complete our analysis of bias/variance profile of classification algorithms by the regression decomposition of these probability estimates.

3.2.4 Bias and variance estimation (bootstrap method)

Bias and variance definitions make intensive use of the knowledge of the distribution of learning samples, $P(\underline{A}, Y)$. In the machine learning problem definition however, the only knowledge we have about this distribution is a data set of randomly drawn samples. So, in practice, true values of bias/variance terms have to be exchanged for some estimates. To this end, and assuming we have enough data, the available dataset is divided into two disjoint parts, PS (P for "pool") and TS (T for "test"), used in the following way:

- As we do not have access to an infinite source of learning sets, PS is used to approximate the learning set generation mechanism. A good candidate for this is to replace sampling from $P(\underline{A}, Y)$ by sampling with replacement from PS. This is called "bootstrap" sampling in the statistical literature [ET93] and the idea behind it is to use the empirical distribution of the finite sample as an approximation of the true distribution. The bigger is PS with respect to the learning set size, the better will be the approximation. Thus, starting from a set $PS = \{(\underline{a}_1, y_1), \ldots, (\underline{a}_{M_p}, y_{M_p})\}$, the estimation procedure for the expectation of a function $g(LS)$ is the following: Draw T learning sets of size N (with $N \leq M_p$)with replacement from PS, $\{ls_1, \ldots, ls_T\}$, compute $g(ls_i)$ for $i = 1, \ldots, T$ and average:

$$E_{LS}\{g(LS)\} \simeq \frac{1}{T}\sum_{i=1}^{T} g(ls_i). \tag{3.39}$$

For example, to estimate the average regression model, build a model f_{ls_i} from each learning set ls_i and compute:

$$\overline{f}_{LS}(\underline{a}) \simeq \frac{1}{T}\sum_{i=1}^{T} f_{ls_i}(\underline{a}). \tag{3.40}$$

Similarly, the probability $P_{LS}(c|\underline{a})$ that a model induced from a random learning set gives the class c at point $\underline{a}$ is estimated by:

$$P_{LS}(c|\underline{a}) \simeq \frac{1}{T}\sum_{i=1}^{T} 1(f_{ls_i}(\underline{a}) = c), \tag{3.41}$$

which yields the majority vote classifier:

$$f_{LS}^{MAJ}(\underline{a}) \simeq \arg\max_{c}\sum_{i=1}^{T} 1(f_{ls_i}(\underline{a}) = c). \tag{3.42}$$

- The set TS, which is independent of PS, is used as a test sample to estimate errors and bias and variance terms. Denoting by $\{(\underline{a}_1, y_1), \ldots, (\underline{a}_{M_v}, y_{M_v})\}$ this set, the expectation of a function $g(\underline{a}, y)$ is approximated by:

$$E_{\underline{A},Y}\{g(\underline{A}, Y)\} \simeq \frac{1}{M_v}\sum_{i=1}^{M_v} g(\underline{a}_i, y_i). \tag{3.43}$$

  For example, the mean error of a model f is estimated by:

$$Err(f) = E_{\underline{A},Y}\{L(f(\underline{A}), Y)\} \simeq \frac{1}{M_v}\sum_{i=1}^{M_v} L(f(\underline{a}_i), y_i). \tag{3.44}$$

However, some of the previously defined terms are difficult to estimate without any knowledge of the problem other than a dataset TS. Indeed, the estimation of the Bayes model $f_B(\underline{a})$ (or more generally of the distribution $P(Y|\underline{A})$) from data is nothing but the goal of supervised learning and if we had a way to estimate it from data, we would not need to bother about estimating bias and variance to study machine learning algorithms. So, the bias and variance terms which make explicit use of these latter will be mostly impossible to estimate for real datasets (from which we do not have any knowledge of the underlying distribution). For example, the regression noise and bias terms (3.16 - 3.17) depend on the Bayes model and thus are impossible to estimate separately only from data. The solution adopted by most authors to circumvent this problem is to assume that there is no noise, i.e. $f_B(a_i) = y_i, i = 1, \ldots, M_v$, and to estimate the bias term from TS by:[9]

$$E_{\underline{A}}\{\text{bias}_R^2(\underline{A})\} = E_{\underline{A}}\{(f_B(\underline{A}) - \overline{f}_{LS}(\underline{A}))^2\} \simeq \frac{1}{M_v}\sum_{i=1}^{M_v}(y_i - \overline{f}_{LS}(\underline{a}_i))^2, \tag{3.45}$$

using in this last expression the estimation of the average model given by (3.40). If it happens that actually there is noise, the last expression is an estimation of the error of the average model

[9]Note that obviously only average values (over the input space) and not local values of bias or variance terms may be estimated as a finite sample can not cover the whole input space.

and hence, this amounts at estimating the sum of the residual error and bias terms. The fact that we can not distinguish errors which are due to noise or bias is not very dramatic, since in our experiments we will be mainly interested in studying relative variations of bias and variance more than their absolute values and the part of 3.45 which is due to residual error is constant. The regression variance may then be estimated by the following expression:

$$E_{\underline{A}}\{\mathrm{var}_R(\underline{A})\} \simeq \frac{1}{M_v}\sum_{i=1}^{M_v}\frac{1}{T}\sum_{i=1}^{T}(f_{ls_i}(\underline{a}_i) - \overline{f}_{LS}(\underline{a}_i))^2, \tag{3.46}$$

or equivalently by the difference between the mean error and the sum of noise and bias as estimated by (3.45).

In classification, the estimate of the classification bias given by Tibshirani can also be obtained by assuming no noise (i.e. if $P(c|\underline{a}) = 1(c = f_B(\underline{a}))$) by:

$$E_{\underline{A}}\{\mathrm{bias}_T(\underline{A})\} \simeq \frac{1}{M_v}\sum_{i=1}^{M_v} 1(y_i \neq f_{LS}^{MAJ}(\underline{a}_i)), \tag{3.47}$$

where the majority vote classifier is estimated according to (3.42). Again, if there is noise, expression (3.47) is actually an estimation of the mean error of the majority vote classifier which is also the additive combination of residual error σ_C and Tibshirani's bias term bias_T. According to its definition, Tibshirani's variance is estimated by the difference between the mean error and expression (3.47).

The preceding procedure may yield very unstable estimators (suffering of high variance) especially if the available amount of data is not sufficiently large. Several techniques are possible to stabilize the estimations. For example, a simple method is to use several random divisions of the datasets into PS and TS and to average the estimates found for each separation. More complex estimates may be constructed to further reduce the variance of the estimation (see for example [Tib96, Wol97, Web00]). However in our study, we will try to choose sufficiently large data set so that avoidance of such complex estimation techniques would not be too damaging.

3.3 Empirical study of bias and variance

Bias and variance both contribute to the error and we have explained in the beginning of this chapter that they are affected in an opposite direction by variations in the complexity of the model. In addition to the model complexity, it is interesting to find out and demonstrate how the parameters of the learning problem and algorithm influence bias and variance. To evaluate these influences, some experiments are carried out in this section on a regression and a classification problem. Both are derived from an artificial problem introduced in [Fri91] which has 10 input attributes $(A_1, \ldots, A_{10})$ all independently and uniformly distributed in $[0, 1]$. The regression output variable Y is obtained as follows:

$$Y = f_B(\underline{A}) + \varepsilon, \tag{3.48}$$

with

$$f_B(\underline{A}) = 10sin(\pi A_1 A_2) + 20(A_3 - 0.5)^2 + 10A_4 + 5A_5, \tag{3.49}$$

and ε is a noise term independent of $\underline{A}$ and distributed according to a Gaussian distribution of zero mean and unit variance. Thus, Y is actually independent of the last five input variables which are included to make the problem more representative of real situations where in general some variables are irrelevant. A classification problem is defined from this regression problem by defining the following binary classification output:

$$C = \begin{cases} c_1 & \text{if } Y < 14.0 \\ c_2 & \text{if } Y \geq 14.0 \end{cases} \tag{3.50}$$

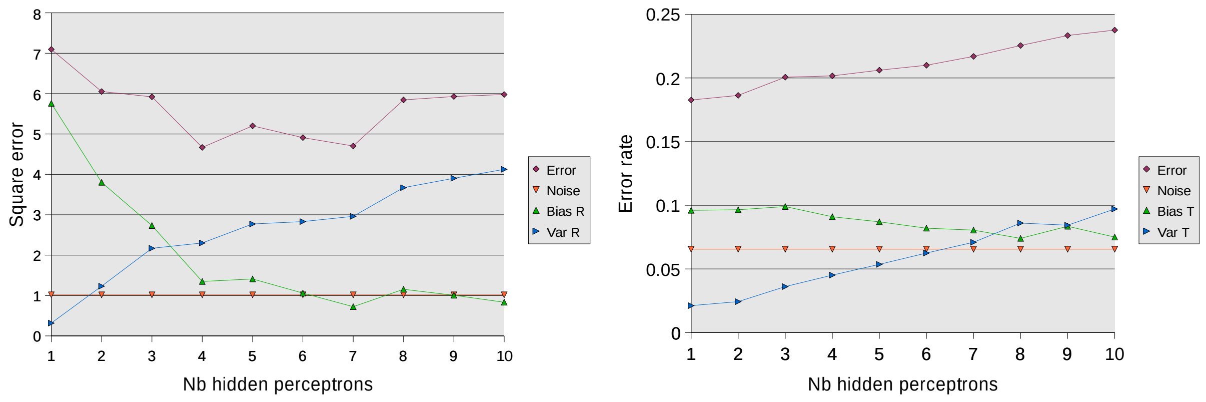


Figure 3.8: Evolution of regression (left) and classification (right) bias and variance with respect to the number of perceptrons in the hidden layer of a neural network

The threshold on Y has been chosen in order to balance the two classes. Bias and variance decompositions are estimated using the protocol given in the previous section. 8000 cases are forming the pool sample and the test sample contains 2000 cases. When not stated otherwise, the learning sample size N is equal to 300 and 50 of them are drawn from PS to estimate expectations according to LS.

3.3.1 Hypothesis space size

Bias mainly depends on the model complexity (the size of the hypothesis space) but also on the amount of optimization carried out by the machine learning method (search procedure): the larger the number of candidate models implicitly visited during the search, the smaller bias. On the other hand, the more search the learning algorithm carries out, the more dependent is the model on the learning sample and hence the higher will be variance.

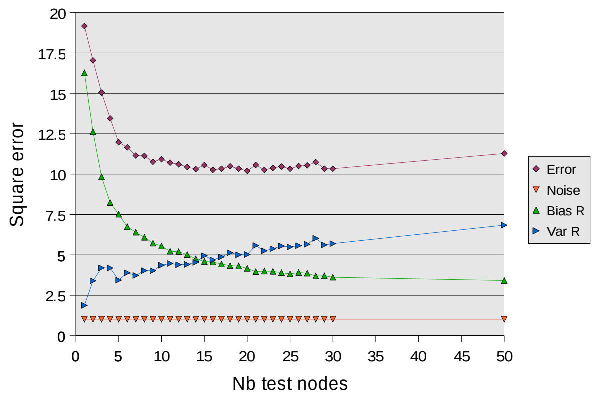
Single hidden layer perceptrons

Figure 3.8 corresponds to a sequence of simulations carried out on the above described regression and classification problems with a neural network package using the Levenberg-Marquardt optimization method [Muñ96]. Notice that this algorithm starts from initial weights which values are chosen according to a random number generator, thus introducing some randomness in the algorithm in addition to the variance related to the learning sample randomness itself. The maximum number of iterations and stopping criteria where chosen in these simulations so as to ensure good convergence to the local minimum of the empirical risk.

The horizontal axis in these figures corresponds to the number of neurons in the hidden layer, i.e. the parameter which controls here the size of the hypothesis space. Notice that the number of adjustable parameters of these neural networks varies between about 10 (a single hidden neuron) and 100 (ten hidden neurons).

The left graph shows the results corresponding to the regression problem and the right one corresponds to classification (in the latter case, the neural network has two neurons in the output layer, trained on the indicator variables of the two classes; the corresponding classification rule consist of choosing the class corresponding to the largest output layer activation). The yellow curves correspond to the residual error which, obviously, does not depend on the number of neurons in the hidden layer. The three other curves show the variation of bias, variance and average error (the latter curve is obtained as the sum of the three other ones).

In the regression case, the curves clearly show the expected evolution of bias, which decreases with increasing complexity, of variance, which increases, and, of the mean square error which first decreases and then increases. The curves thus are consistent with the theory of the bias/variance compromise, which in this particular problem corresponds to 4 hidden neurons.



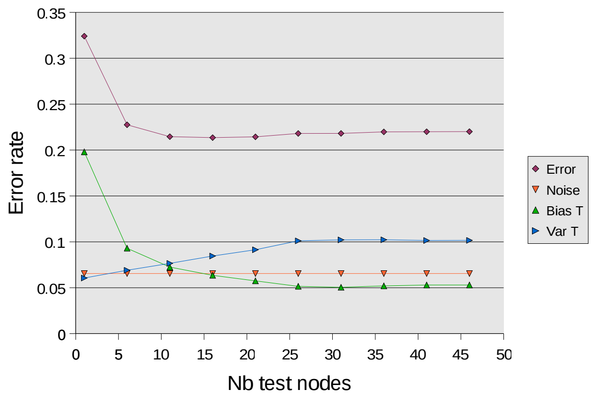


Figure 3.9: Evolution of regression (left) and classification (right) bias and variance with respect to the number of tests in a decision/regression tree

Considering the curves corresponding to the classification case, where Tibshirani's bias and variance terms are used, similar behavior can be observed, with the only difference that in this case the optimal complexity corresponds to one single hidden neuron (which actually corresponds to a linear decision boundary).

Decision/regression trees

Figure 3.9 shows similar curves, where the complexity parameter corresponds to the number of terminal nodes of the tree[10]. The same protocol and learning samples than in the preceding simulation have been used to yield these graphics. Again, as predicted by the theory, variance increases monotonically with complexity while bias decreases, yield in terms of average error and optimal complexity, which is slightly large for regression than for classification.

Comparing the average error rates of Figs. 3.9 and 3.8 we notice, in passing, that the neural networks significantly outperform decision and regression trees on this particular problem.

3.3.2 The learning problem: Bayes model and noise

Bias and variance are also related to the complexity of the Bayes model $f_B(\underline{a})$. At fixed complexity of the model, the bias increases with the complexity of f_B, because the absolute value of the bias depends on the way the complexity of the model matches the complexity of the function. The variance on the other hand remains mainly unaffected by the complexity of the Bayes model[11]. The best tradeoff between bias and variance with respect to the model complexity is thus - unfortunately, but not surprisingly - problem dependent. Note that the selection of the optimal value of complexity, in a problem dependent way, is discussed in the next chapter in the context of several learning algorithms.

Another problem dependent parameter that influences the tradeoff is the noise. In regression, bias which depends essentially only on the Bayes model and the average model should be unaffected by random additive noise. Variance however increases with the noise. This is confirmed by the results shown in Figure 3.10, corresponding to simulations with our synthetic problem where the noise level has been artificially increased progressively. The curves show the bias/variance profile for increasing value of the standard deviation of the noise ε in equation (3.48). The residual error is of course increasing. And indeed, while bias remains almost constant, variance is monotonically increasing. Note that, the fact that variance decreases with

[10]The trees have been built according to the best first procedure which is described in chapter 5.

[11]This statement may appear here as unjustified, but it is essentially supported by various theoretical considerations, in particular those provided in section 3.5.

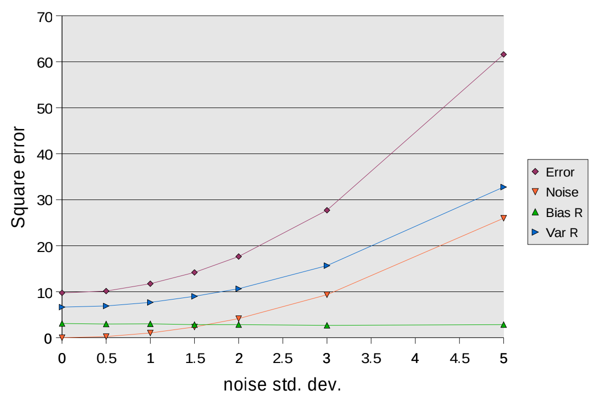


Figure 3.10: Evolution of bias and variance of regression trees with respect to the standard deviation of the noise

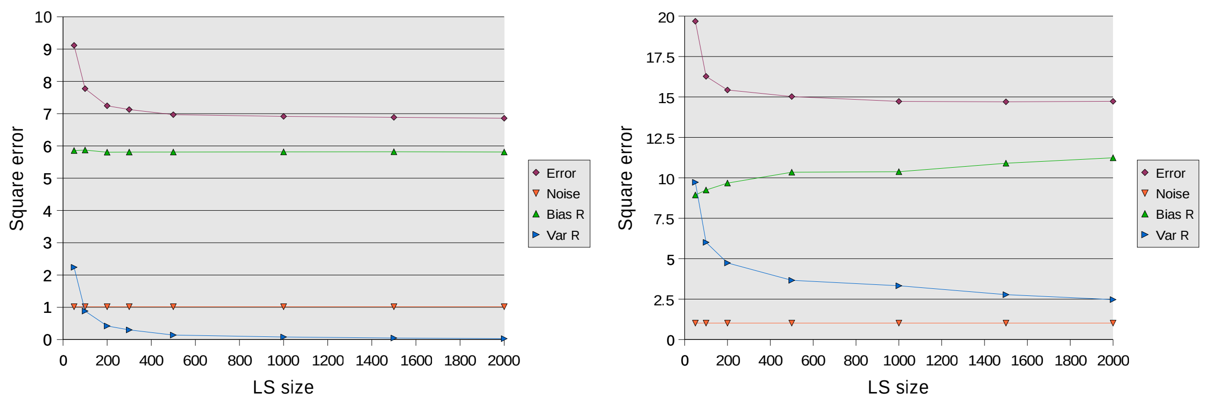


Figure 3.11: Influence of the learning set size at fixed complexity. Left, with linear regression, right, with regression trees of 7 terminal nodes

decreasing noise level must not be interpreted as "no noise = no variance"[12]. Besides, the variance of regression trees with zero noise remains quite important in Figure 3.10.

3.3.3 Learning sample size

Whatever the learning algorithm, we expect the variance to decrease when the learning sample size increases, since the more samples are gathered, the more stable should be the induced model.[13]

On the other hand, the evolution of bias with respect to the sample size is algorithm dependent. When the complexity of the model is fixed, bias is typically independent of the learning sample size. The left part of Figure 3.11 shows that this is indeed the case with linear regression for example. However, the right part of the same figure obtained with regression trees of fixed number of terminal nodes (7 here) shows that in this case, bias is a slightly increasing function of the learning sample size although the complexity is fixed. This counterintuitive result can nevertheless be explained. In the case of regression trees, the average model $\overline{f}_{LS}$ is impossible to represent with a tree of fixed complexity (the hypothesis space of fixed complexity trees is not closed with respect to the aggregation operator used). Indeed, the average model corresponding to a finite number of trees of small complexity can be represented only by a tree

[12]Which is a statement found in various forms in the literature on automatic learning, e.g. in the form "Overfitting is due to noise...".

[13]Note that the "whatever" in this sentence, should be understood as "whatever reasonable", because it is of course possible to imagine learning algorithms which variance would increase with sample size.

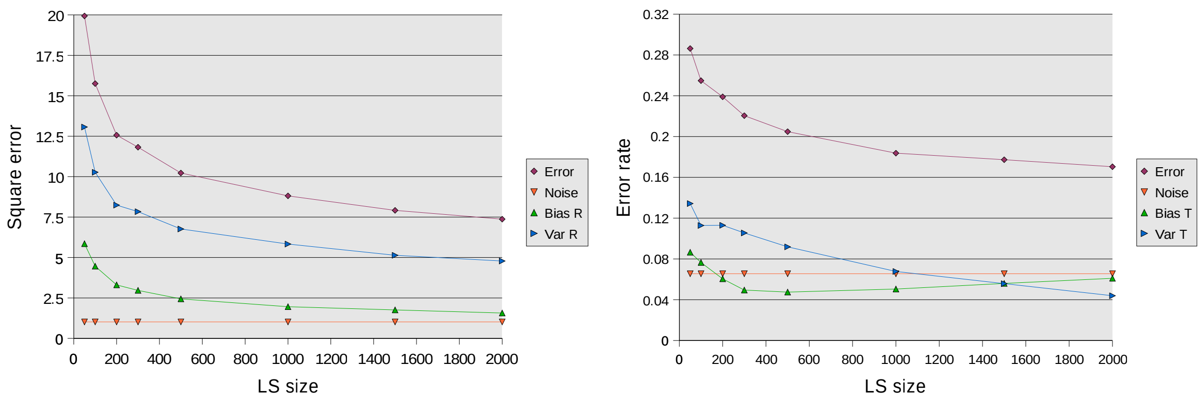


Figure 3.12: Influence of the learning set size at adjustable complexity. Left, with regression trees, right, with decision trees

of much larger complexity. And, in the limit, the average of an infinite number of trees can actually represent a smooth function, which is not at all representable by a finite tree, which can only represent piecewise constant functions. Thus, when the learning sample size is small, the variance is high and trees induced from different learning sets are very diverse. On the other hand, when the learning sample size increases, trees becomes closer and closer to each other and the average model is less and less flexible, hence bias increases (slightly in Figure 3.11). Notice however that, because variance decreases much more significantly than bias increases this figure suggests that for increasing learning sample size the optimal complexity will also increase, which is indeed the case (see chapter 5).

In several algorithms however, the model complexity or the search algorithm depends indirectly[14] on the size of the learning set and hence the bias should decrease in these cases with the learning sample size. For example, the size of fully grown decision/regression trees (i.e. which are build so as to minimize the empirical risk) is automatically adapted to the learning sample size and hence their bias is actually a decreasing function of the learning set size. Figure 3.12 shows us that under these conditions bias and variance indeed decrease as the sample size goes up, left with regression trees, right with decision trees.

3.3.4 Learning algorithm

The bias/variance tradeoff is also different from one learning algorithm to another. Some algorithms intrinsically present high variance but low bias and other algorithms present high bias but low variance. For example, linear models and Naive Bayes method because of their strong hypothesis often suffer from a high bias. On the other hand, because of their small number of parameters, their variance is small and, on some problems, they may therefore be competitive with more complex algorithms, even if their underlying hypothesis is clearly violated. While the bias of the nearest neighbor method (1-NN), neural networks and regression/decision trees generally are smaller, the increase in flexibility of their model is paid by an increase of the variance.

The variance of decision trees will be studied in detail later but it is partly due to the discretization of numerical attributes and to the hierarchical nature of their induction.

The variance of 1-NN rule (which bias is negligible, for large sample sizes) is easier to understand. Since, the decision at a point of the input space depends on the closest sample in the learning set, a small perturbation of this sample may lead to a large change in this prediction. In the case of complex neural networks, there are generally several (possibly an infinite number

[14]Actually, those algorithms which aim at optimizing the bias/variance tradeoff by adjusting the model complexity to the problem specifics.

Table 3.1: Bias/variance decompositions of several algorithms in regression

Method	Square error	σ_R^2	bias_R^2	var_R
Linear regression	7.0	1.0	5.8	0.2
k-NN (k=1)	15.4	1.0	4.0	10.4
k-NN (k=10)	8.5	1.0	6.2	1.3
MLP (10)	2.0	1.0	0.2	0.8
MLP (10-10)	4.6	1.0	0.4	3.2
Regression tree	10.2	1.0	2.5	6.7

of) possible parameterizations of the networks which allow to realize very small values of the empirical risk on the learning sample. So, according to the initial choice of parameters and the particular learning sample, the optimization procedure will be trapped into different local optima and hence the induced model will also be highly variable.

To illustrate this discussion, we have carried out a number of simulations on the regression problem in order to compare different variants of four basic learning algorithms. Table 3.1 provides results obtained with learning samples of size 500, the numbers being obtained by the same protocol as before (50 random learning samples of size 500 used to estimate expected values with respect to the LS variable, and a validation sample composed of 2000 objects, in order to compute averages over the input space). The table confirms the following general trends

- **Linear regression:** variance is negligible, which is mainly due here to the fact the number of parameters of this method (11) is very small with respect to the number of objects in the learning sample. The error of the linear regression is mainly due to its bias, which is a consequence of the non-linearity of the Bayes model corresponding to this regression problem.

- **k-Nearest-Neighbor:** in our experiments we have used the Euclidean distance in the space of 10 attributes scaled to make their variance equal to 1. Note that among the 10 input attributes their are actually 5 which are irrelevant, and this is known to be detrimental to the performance of k-NN. The table shows a very high variance when $k = 1$ and also a rather high bias. Increasing the value of k to 10 allows to reduce variance significantly, but at the price of increasing the bias by 50%. All in all the k-NN method is less accurate than linear regression in spite of the fact that it may in principle handle the non-linearity. Note that the rather high bias displayed here by the k-NN method is due to the fact that the average distance to the nearest neighbors is too high compared to the speed of variation of the Bayes model. This distance is a result of the combined effect of high dimensionality in the input space and small sample size.

- **Multilayer perceptrons:** this method provides overall the best results in this problem, having both negligible bias and small variance. The two simulations correspond respectively to one single hidden layer with ten neurons (denoted (10) in the table) and two hidden layers with ten neurons each (noted (10-10) in the table). The variance of the more complex structure is significantly more important.

- **Regression trees:** in this table the trees are fully grown. Their bias is small but not negligible, and variance is so high that in the end the method is less accurate than linear regression and the 10-NN method. Note that the bias is due here to the fact that the smooth function f_B is difficult to approximate with a simple staircase model, whose complexity is moreover limited by the sample size (to about 20 terminal nodes, on average).

3.4 Two other kinds of variances

The variance expressed in our decompositions and studied in the preceding experiments:

$$E_{\underline{A}}\{\text{var}_{LS}\{f_{LS}(\underline{A})\}\} \tag{3.51}$$

is the variance of the predictions given by the model, averaged over the whole input space. It translates the effect of the variability of the model on the mean square error or on the classification error. Nevertheless, the variability of the prediction comes with (and is correlated to) two other types of variances: the variance of the error realized by a model and the variance of the parameters of this model.

3.4.1 Error variance

The variance of the error of a model is expressed by:

$$\text{var}_{LS}\{Err(f_{LS})\}. \tag{3.52}$$

If this variance is high, the accuracy of a model produced from a learning set is very variable from one learning set to another. At the time of writing this thesis, and to our best knowledge, no detailed study of the link between this variance and the variance of the predictions has been carried out. In what follows we will merely provide a very short discussion of it.

Intuitively, if error variance is high at a point $\underline{a}$, then the variance of the predictions at this point must also be high. In other words, small prediction variance implies small error variance. On the other hand, the opposite is not necessarily true: a high prediction variance does not necessarily translate into a high error variance. This is due to the fact that errors at different points in the input space can be correlated in different ways, and it is possible that the increase of error in one region is cancelled by a decrease in another region. Thus errors may be rather stable in spite of a significant prediction variance.

In this dissertation, we will focus on the reduction of prediction variance, which will as a by-product also reduce error variance, should it be high.

3.4.2 Parameter variance

If a model is parameterized by a parameter α (e.g. a weight in a neural network or a discretization threshold in a decision tree), the learned value of this parameter α_{LS} depends on the learning set and hence is also a random variable. We can thus compute its variance with respect to the learning set randomness:

$$\text{var}_{LS}\{\alpha_{LS}\}. \tag{3.53}$$

If this variance is high, the value of the parameter is highly unstable from one learning set to another. When the model is intrinsically interpretable, its practical interpretability is limited if the parameter variance is high and so this variance should be reduced as well. Again, there is not a bi-directional quantitative relationship between parameter variance and prediction variance. Indeed, high prediction variance should come with a high parameter variance but the opposite is not true: even if the prediction variance is not high, the parameter variance may be important. It depends on the way these parameters intervene to make up the final prediction.

We will see in subsequent chapters that the variance of the parameters of decision trees (discretization thresholds and attribute choice at interior nodes of the tree) turns out to be very high, which makes their interpretation questionable. Thus, we will study in detail this variance in Chapter 5 and investigate several techniques liable to reduce it in order to improve interpretability.

3.5 Analytical results related to bias and variance

Bias/variance decompositions split the error into two terms whose interpretation gives some feeling about the way error evolves with respect to various parameters. The empirical study of the previous section gives some clues about the qualitative behavior of bias and variance with respect to these parameters but what we can quantitatively expect is still an open question. We should add that, although the main trends that have been illustrated hold in a large number of situations, there are also a number of exceptions[15].

In some simple cases, it is possible to express analytically bias or variance.

3.5.1 Linear models

Let us assume a regression problem where $Y = f_B(\underline{A}) + \varepsilon$ with $E_\varepsilon\{\varepsilon\} = 0$ and $\text{var}_\varepsilon\{\varepsilon\} = \sigma^2$.

Let us further assume that $f_B(\underline{A})$ is a linear function of the attributes and that we use a least squares multiple linear regression method to approximate this function from a learning sample. Then one can show (see [HTF01]) that the following two conditions are satisfied:

$$E_{\underline{A}}\{\text{bias}^2_R(\underline{A})\} = 0, \tag{3.54}$$

and

$$E_{\underline{A}}\{\text{var}_R(\underline{A})\} = \frac{n}{N}\sigma^2, \tag{3.55}$$

where n is the number of attributes and N the sample size.

Thus variance is proportional to the dimensionality of the input space and inversely proportional to the sample size. More generally, even if the Bayes model is not linear, the variance of linear models is increasing with the number of attributes (and hence of weights) and a decreasing function of the learning sample size (as confirmed by Figure 3.11 p. 43).

Notice also the fact that under "no noise" conditions variance vanishes, which led some people to extrapolate that variance is *due* to noise in general (which is not true). Indeed, even for linear regression, variance may be observed under no-noise conditions if the Bayes model is not linear.

3.5.2 K-NN

If we further assume input attribute values of the learning cases are fixed and only their output values are changing from one learning set to another, the variance of the k-NN may be approximated by (see also [HTF01]):

$$E_{\underline{A}}\{\text{var}_R(\underline{A})\} = \frac{\sigma^2}{k}, \tag{3.56}$$

showing that variance is a decreasing function of the number of neighbors. Again, variance is increasing with the noise level.

3.5.3 Single hidden layer perceptrons

Derivation of analytical formulas for bias and variance is possible only in very simple cases, such as those just discussed. For example, in more general situations, like in the case of neural networks or decision trees, no such formulas have been (and possibly could be) derived.

Nevertheless, sometimes it is possible to give at least analytical bounds on error. Barron [Bar94], for example, gives bounds on the mean square error for single hidden layer neural networks. Assuming there is no noise ($\sigma^2_R = 0$), these bounds are:

$$E_{LS}\{Err(f_{LS})\} \leq O(\frac{C^2_{f_B}}{L}) + O(\frac{nL}{N}\log N), \tag{3.57}$$

[15] e.g. it is possible to find situations where the increase of the number of parameters of a model leads actually to a decrease of variance, rather than an increase.

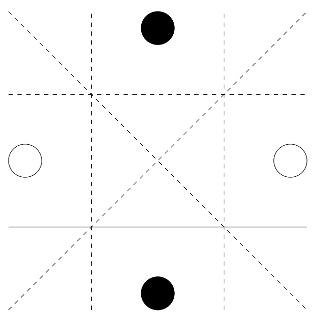


Figure 3.13: a configuration of four points that can not be shattered by a single line

where L is the number of units in the hidden layer and C_{f_B} is a measure of the complexity of the Bayes model based on its Fourier representation. This decomposition is similar to a bias/variance decomposition. Actually, the first contribution is a bound on the error of the best neural network of this complexity, denoted by f_L^*, with respect to the Bayes model:

$$E_{\underline{A}}\{(f_B(\underline{A}) - f_L^*(\underline{A}))^2\}. \tag{3.58}$$

This latter error is called *approximation error* by Barron.

The second term in (3.57) is actually a bound on the average error of a learned neural network with respect to f_L^*:

$$E_{LS,\underline{A}}\{(f_{LS}(\underline{A}) - f_L^*(\underline{A}))^2\}. \tag{3.59}$$

This term is called *estimation error* by Barron and the bound is obtained under the assumption that the neural network optimization method is indeed able to find the set of parameters minimizing the empirical risk.

Although this decomposition does not correspond exactly to the usual decomposition, these two terms may be assimilated respectively to a bias and a variance term. An interpretation of these bounds confirms our intuitive and empirical analyses. Indeed, the bound on bias is an increasing function of the Bayes model complexity and decreases with the complexity of the model, L. On the other hand, the bound on variance increases with the model complexity and the number of input attributes, while it decreases with the learning sample size. So, these bounds give an analytical justification of the form of the curves obtained in the previous section.

3.5.4 Statistical learning theory

When the complexity of the model increases, the learning algorithm is able to realize lower and lower errors on the learning sample (empirical risk) even if the Bayes model is very complex, and simultaneously approximation error and bias are decreasing. On the other hand, the estimation error of the learned model and the variance of the method will increase. Thus, the empirical risk is correlated with bias and approximation error, and the difference between the generalization error $Err(f_{ls})$ and the empirical risk $\hat{Err}(f_{ls}, ls)$ is a quantity related to variance.

While several authors have given bounds on this difference, the statistical learning theory of Vapnik [Vap95] certainly gives rise to the most general bounds. In classification, these bounds are based on a general measure of the representation capacity of a set of functions which is called the VC-dimension. The VC-dimension of a hypothesis space $\mathcal{H}$ of binary-valued functions is the largest number of points such that there exists a configuration of this number of points that can be *shattered* by members of $\mathcal{H}$. A set of points is said to be shattered by $\mathcal{H}$ if whatever the binary class assigned to each point, there exists a function in $\mathcal{H}$ that perfectly reproduces this classification. For example, in a bi-dimensional space, the VC-dimension of linear discriminant functions is three since it is impossible to find four points in the two-dimensional space which can be shattered by the set of linear discriminants (see Figure 3.13, for an example of 4 points which classification can not be obtained by a linear discriminant), while it is possible to shatter three points in general position. Notice that if the VC-dimension is infinite, then there exist samples of any size for which the empirical risk can be driven to zero within the hypothesis space whatever the chosen labels. The main contribution of Vapnik in automatic learning theory was to show that the empirical risk minimization principle in automatic learning is justifiable only if the VC-dimension of the hypothesis space is finite.[16]

The other main contribution of Vapnik was to provide errors bounds in the case of finite samples, based on the VC-dimension. For example, he proves the following result for binary classification problems with misclassification error: for any function $h \in \mathcal{H}$, we have with probability at least $1-\alpha$:

$$Err(h) \leq \hat{Err}(h, ls) + \varepsilon, \tag{3.60}$$

with

$$\varepsilon = 4\frac{d}{N}[\ln(\frac{2N}{d}) + 1] - \frac{1}{N}\ln(\frac{\alpha}{4}), \tag{3.61}$$

where d is the VC-dimension of the set of functions $\mathcal{H}$. This bound shows again that the difference between generalization error and empirical risk increases with the model complexity (by means of the VC-dimension) and decreases with the learning set size. Vapnik provides similar bounds for the regression case (see [Vap95]).

Because of their generality (the results are distribution independent), these bounds are often too loose to be of practical use from the quantitative point of view. Another drawback of this approach is the difficulty to compute the exact VC-dimension of many hypothesis spaces used in automatic learning. However, qualitatively, these theoretical results are very useful to understand and design learning algorithms. A practical application of these bounds will be described in the next chapter.

3.6 Discussion

In automatic learning, bias and variance both contribute to prediction error, whatever the learning problem, algorithm, and sample size. The error decomposition allows us to better understand the way an automatic learning algorithm will respond to changing conditions. It allows us also to compare different methods in terms of their weaknesses. This understanding can then be exploited in order to select methods in practice, to study their performances in research, and to guide us in order to find appropriate ways to improve automatic learning methods.

The empirical simulations provided in Section 3.3 aimed at providing a quantitative comparison of methods and of various effects. As far as this thesis is concerned, one of the main outcomes is the very high variance of decision and regression tree induction, which significantly decreases the attractiveness of this method in spite of its other intrinsic qualities (interpretability, flexibility, computational efficiency). It suggests that, in order to improve these methods it is necessary to find a way to reduce its variance, ideally without jeopardizing its other nice features. We believe that these observations justify our focus on variance in decision tree induction in this thesis, and our research for general methods able to reduce this variance in an effective way. This is the topic of the second part of this thesis, while in the third part we will concentrate on practical applications in the context of time-series classification, where variance is even more important than in non-temporal problems. Before we switch to these considerations, the next chapter is dedicated to classical variance reduction techniques in supervised learning.

[16]Infinite VC-dimension means that the hypothesis space is so large that it becomes too "easy" to find apparently excellent hypotheses, for most learning problems and any sample size, so that the empirical risk minimization principle looses its discrimination power between hypotheses.

Chapter 4

Variance reduction techniques

Bias and variance are two sources of error reacting in opposite way to the model complexity. There thus exists a tradeoff between these two sources of error. After a short introduction, we review in Section 4.2 the "classical" way this tradeoff is handled in the context of different learning algorithms. Section 4.3 then considers a class of variance reduction techniques based on the aggregation of several models (also called "ensemble techniques") which have been proposed in the recent years, and which are potentially able to reduce variance without increasing bias. The last section provides examples of such techniques and, in particular, introduces our "dual perturb and combine" algorithm which we further develop in the context of decision tree induction in chapter 8.

Note that in this chapter we provide no empirical comparisons of the variance reduction techniques, our aim being essentially to provide a broad and synthetic view of the methods. Many of these methods will be studied more in detail later on in the context of decision tree induction.

4.1 Introduction

Automatic learning methods like decision/regression trees or neural networks have the nice property of providing a very flexible hypothesis space. They have the universal approximation property which means that they are able to model with arbitrary accuracy any practically relevant input/output function, however complex, provided that sufficiently complex tree or neural network structures are used. The main problem of these automatic learning methods is thus not bias (which, in fact, can be made arbitrarily small) but variance which arises when such complex models are needed and must be chosen on the basis of finite samples. The increase in interest in variance reduction has been recently strengthened by two things. First, the success of automatic learning techniques makes people want to apply them in more and more complex domains, e.g. in bioinformatics, cryptography, or, for temporal data modeling. Even if traditional algorithms are in principle applicable in these domains, they are not really adapted to these new kinds of data (see the third part of this thesis for time series data) and even more complex model spaces have to be designed. Second, at the same time, the very fast increase in computer power makes it possible to search larger and larger hypothesis spaces. These new improvements in representational power of the learning algorithms will be useless if they are not combined with some variance reduction techniques.

Variance reduction techniques can be classified into two main families:

- techniques that control the bias/variance tradeoff in the context of one particular learning algorithm. These methods do not change the bias/variance configuration of the learning algorithm but rather adapt the optimization part of the algorithm in order to find the best tradeoff between bias and variance.
- techniques that change the bias/variance configuration of a learning algorithm. These are general techniques which can be applied to any learning algorithm; all of them work by

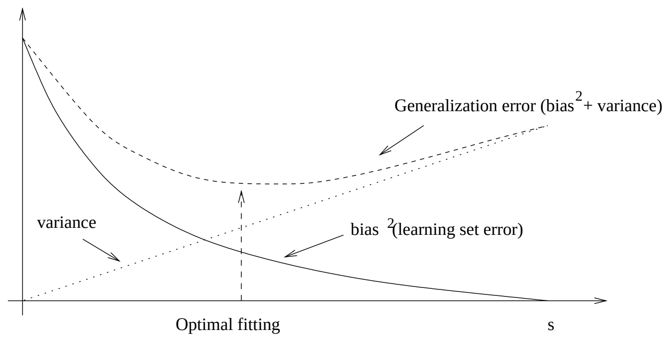


Figure 4.1: Bias and variance with respect to the learning set fitting (neglecting residual error).

averaging several (unstable) predictions.

The next two sections will describe both classes of techniques. The fourth section will discuss among other things Bayesian learning paradigms and relate them to variance reduction techniques.

4.2 Controlling the bias-variance tradeoff of a method

4.2.1 General framework

In every method which uses a learning set to select one model, there is a need to control the bias/variance tradeoff to avoid overfitting. The more you give means to an algorithm to fit the learning set, the more it will suffer from variance. Figure 4.1 shows the typical evolution of the bias and variance of a machine learning algorithm with respect to a parameter s measuring the means given to the learning algorithm to fit the learning set (e.g. number of decision tree nodes, number of weights for neural networks). Bias generally is a monotonic decreasing function of s and variance on the other hand is a monotonic increasing function of the same parameter. The error thus reaches a minimum for some given value s^* of s. Since, as we commented above, variance decreases with sample size and bias is often not at all dependent of sample size, the best value of s should be an increasing function of the sample size. Also since bias is always dependent on the problem (by means of the Bayes model), the best value s^* is also problem dependent.

In practice, there are three ways of controlling a parameter like s to avoid overfitting:

1. First, a reasonable value of s can be fixed according to some prior knowledge (a thumb rule) and the best model selected on the learning set taking into account this value. For example, in the neural network literature one can read rules like "the number of weights of a neural network should not exceed 10% of the learning sample size N".

   Note that, even if this procedure is fast as it needs only one optimization, the best value of the parameter is most of the time unknown a priori and intrinsically dependent on the problem. Thus, in practice, this approach turns out to be suboptimal.

2. The second approach aims at identifying the optimal value of s on the basis of the available sample of objects. It works as follows :

   for increasing values of s:

   - use the learning set to find a model taking into account the current value of s;
   - evaluate the generalization error of this model and stop the iteration when this error is not decreasing anymore.

   As bias and variance are difficult to estimate from one learning set, a generalization error estimate is used to select the best model. To evaluate the generalization error, we can use either cross-validation or an independent validation set (see section 2.5). Apart from the increase in computational burden, the main drawback of this technique is that it can be trapped in a local minimum of the generalization error. This leads to the third algorithm.

3. For increasing values of $s < s_{max}$:

   - use the learning set to find a model taking into account the current value of s;
   - evaluate the generalization error of this model;

   Among the resulting sequence of models, select the one which yields the best generalization error. When there is no natural maximum value s_{max}, iterations can be stopped either when the learning set error vanishes or at least when this error stops decreasing. The problem of local minimum is thus avoided (or at least mitigated) but at the price of higher computation times.

Note that the computation of the optimal model on the learning set for increasing values of s can be very complex in some cases. However, sometimes, it is possible to compute efficiently the optimal model for one value of s from the previous model in the sequence. If this is impossible, one solution is to design heuristics that compute an approximation of this model from the previous one.

4.2.2 Examples in the context of different learning algorithms

In the context of a particular machine learning algorithm there are often many possible parameters s, each one regulating a different bias/variance tradeoff. One general way to control the goodness of fit to the learning set is to control the size of the hypothesis spaces. The parameter s thus defines a sequence of nested hypothesis spaces $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots \subset \mathcal{H}_s \subset \ldots$ which are all included in the original space $\mathcal{H}$. For a fixed value of s, learning consists in searching $\mathcal{H}_s$ for the best model on the learning set. Almost all parameter choices discussed below comply with this framework of nested hypothesis spaces. Usually, to limit the size of the hypothesis space, a complexity measure $C(h)$ of models is defined and the subspace $\mathcal{H}_s$ is restricted to models whose complexity is less than a threshold s, i.e. $\mathcal{H}_s = \{h \in \mathcal{H} | C(h) \leq s\}$. From the theory of Lagrange multipliers, the sequence of models obtained by minimizing the error $\hat{Err}(h, ls)$ in $\mathcal{H}_s$ for increasing values of s is equivalent to the sequence of models obtained by minimizing $\hat{Err}(h, ls) + \lambda \cdot C(h)$ for decreasing values of λ. This second formulation often makes the problem easier to solve.

Variance controlling techniques are reviewed here in the context of the five learning algorithms described in the first chapter.

Linear regression

Even if this algorithm generally does not suffer from very high variance, the first variance reduction techniques were devoted to it. Ridge regression, for example, consists in bounding the norm of the weight vector or equivalently in adding a penalty term to the learning set error which is proportional to the norm of this vector. More precisely, the full hypothesis space $\mathcal{H}$ contains all linear models and the hypothesis subspace $\mathcal{H}_s$ for some s contains only those functions which satisfy

$$h(\underline{a}) = w_0 + \sum_{i=1}^{n} w_i \cdot a_i \text{ with } \sum_{i=0}^{n} w_i^2 < s. \tag{4.1}$$

Note that, as we have argued above, minimizing the empirical risk

$$\hat{Err}(h, ls), \tag{4.2}$$

in $\mathcal{H}_s$, is equivalent to minimizing in $\mathcal{H}$ the criterion

$$\hat{Err}(h, ls) + \lambda(s) \cdot \sum_{i=0}^{n} w_i^2, \tag{4.3}$$

for some suitably chosen decreasing function $\lambda(s)$. Once λ is fixed, the minimization of (4.3) can be carried out solving a set of linear equations derived by slightly modifying the normal equations of linear least squares.

Irrespectively of variance reduction, another advantage of ridge regression is that even small values of λ stabilize the analytical solution of equation (4.3) when the solution for $\lambda = 0$ may be ill-conditioned (e.g. in the case of singularities resulting from quasi-linear relationships between attributes).

Neural networks

There exist many parameters controlling the bias/variance tradeoff in neural networks. First, the complexity of the neural networks (in terms of the number of neurons and hidden layers) is certainly a valid choice for s. This is a typical case where heuristics have to be designed to select the right complexity. Indeed, as optimization of neural networks can be very time consuming, exhaustive search on structure is essentially intractable. Growing (starting with a simple network and iteratively adding new units to it) and pruning strategies (starting from complex networks and gradually removing connections or units) both have been explored.

Another obvious parameter is the number of iterations of the optimization algorithm. If we let this algorithm drive the resubstitution error to zero, the model will certainly overfit the data and hence suffer high variance. So, stopping the optimization earlier could reduce the generalization error (the combination of this parameter and one of the two ways of finding the optimal value of s is called early stopping in the neural network community).

The equivalent of ridge regression for neural networks is called "weight decay" in the neural network literature. Like (4.3), weight decay consists in adding to the error function a term which penalizes large weights. The parameter s is the inverse of the constant multiplying the norm of the weights and is called the decay constant. Weight decay is based on the idea that very large weights in a neural network can cause the output function to be very irregular and thus to fit the noise in the data. Techniques penalizing irregular output functions in favor of smoother ones are called regularization techniques and weight decay is only one of them. Another alternative is to use a penalty term involving various derivatives of the output function.

Decision/regression trees

The obvious parameter regulating the bias/variance tradeoff is the complexity of the tree measured for example by its number of terminal nodes. In practice, to control the complexity, a quality measure $Q(\mathcal{T}, LS)$ for a tree $\mathcal{T}$ is defined by:

$$Q(\mathcal{T}, LS) = R(\mathcal{T}, LS) - \frac{1}{s}.C(\mathcal{T}), \tag{4.4}$$

where $R(\mathcal{T}, LS)$ is some measure of the reliability of the tree on the learning set (resubstitution error and information quantity have been proposed in the literature). Small values of s favor simple trees while large values of s favor more complex ones. Fixing a priori the value of s and optimizing the quality is called stop splitting (or pre-pruning). Optimization of s following the third procedure of subsection 4.2.1 is called pruning (or post-pruning). Note that these ideas will be explained in detail in Chapter 6 of the second part of this thesis, focusing on variance reduction in decision tree induction.

K-Nearest Neighbors

The only parameter in K-nearest neighbors is precisely K the number of neighbors which are taken into account when doing a prediction. A parameter s satisfying the conditions is $N - K$ where N is the size of the learning set. When $s = 0$, the bias is maximum and the variance minimum. Indeed, the prediction is constant allover the input space but may still change from one learning set to another. On the other hand, variance is maximum and bias is minimum when

K is equal to 1. Note that in the context of K-NN, leave-one-out is an obvious candidate for estimation of the generalization error during optimization of K. Indeed, the learning phase is very efficient and one leave-one-out test for some value of K can take into account the distances between objects which have already been computed for testing previous values of K.

Naive Bayes

In the Naive Bayes method, a bias/variance tradeoff appears when it comes to learn a model for probability distributions $P(A_i|C)$. If the model for this distribution is too complex, it will not generalize. On the other hand, if the model is too simple, bias will be high. For example, if we use histograms to model such a distribution, the number of cells of the histogram (bins) is a parameter which controls a bias/variance tradeoff. If this number is too large, some subregions will be empty of any cases from the learning set and thus the histogram will not generalize. On the other hand, too few subregions will hide some information contained in the learning set.

General techniques

There exist also some general model independent techniques to regulate some bias/variance tradeoff. For example, attribute selection is a way to reduce the variance. Indeed, if there are some useless or redundant attributes, they will give some further means to the learning algorithm to overfit the data. The number of attributes is thus a valid choice of parameter s and a lot of work has been done in this field (see for example [KJ97] for a model-independent wrapper technique). Note that tree based methods intrinsically are doing some attribute selection and actually can be used as an attribute selection method for other learning algorithms.

Another general variance reduction technique is jitter ([HK92, Bis95]). Jitter consists in generating new pseudo-samples cases obtained by adding a small amount of random variations (noise) to copies of the input attribute vectors of existing learning cases while leaving the output attribute unchanged. Assuming the goal functions are mostly smooth and injecting a small amount of noise, expanding the learning set in such a way will not change very strongly the goal function (and hence bias) but will certainly force the learning algorithm to build smoother models (and hence decrease variance). In the context of linear regression, it can be shown that jitter with white Gaussian noise is strictly equivalent to ridge regression. In the context of neural networks, although not strictly equivalent, jitter and weight decay have been shown to produce very similar results.

4.2.3 Related approaches for model selection

Several general frameworks have been proposed in the machine learning literature in order to formalize the compromise between complexity of model structure and empirical risk, which are more or less strongly related to the approaches discussed above. Below, we briefly discuss three of them, namely the Bayesian approach to model selection, the Minimum Description Length (MDL) principle, and the structural risk minimization approach. Rather than providing a detailed theoretical development of these approaches, our aim is to give an intuitive description of how they work and how they are related to the other techniques presented in this chapter.

Bayesian model selection

We have seen that the classical approach to machine learning (inherited from classical statistics) can be summarized as follows: define an error measure and a set of candidate models, then apply the empirical risk minimization principle in order to select one model on the basis of the learning sample. Note that under some additional hypotheses on the underlying data generation process, most of the approaches following this principle can be viewed as a *maximum likelihood* approach. In other words, the model they select is the model (within the set of candidate models) which maximizes the likelihood of the learning sample. Under some restrictive conditions on the hypothesis space (essentially, finite VC-dimesion) one can show that this approach is optimal asymptotically, i.e. when the sample size tends towards infinity. On the other hand, as we have argued in the preceding sections, when sample size is small or moderate compared to the size of the hypothesis space, this approach is essentially suboptimal.

In the Bayesian approach, rather than considering the model as a deterministic object which has to be identified from the data, we consider the model as a random variable and the observable samples as random variables which have a conditional distribution given the model. Then, the prior distribution over the hypothesis space represents our prior beliefs about the possible explanations of a sample, and these beliefs can be updated once a learning sample has been observed, yielding a posterior distribution. In practice, to apply the approach we thus need to specify two distributions:

- a prior distribution, $P(H)$, defined on the hypothesis space $\mathcal{H}$ which measures how plausible are a priori the models of the hypothesis space (without any knowledge of the data);
- a conditional distribution $P(LS|H)$ which measures the likelihood (also called the evidence) of a learning sample under the assumption that it is generated according to the model H.

From these two distributions, Bayes theorem allows us to compute the posterior distribution of the model given the data:

$$\begin{aligned} P(H|LS) &= \frac{P(LS|H)P(H)}{P(LS)} && (4.5)\\ &\propto P(LS|H)P(H), && (4.6) \end{aligned}$$

where $P(LS) = \sum_H P(LS|H)P(H)$ is the probability of observing a particular sample, and can be viewed here as a normalization constant which ensures that the distribution $P(H|LS)$ over $\mathcal{H}$ sums to 1.

From a technical point of view, the Bayesian approach to machine learning essentially consists in computing the posterior distribution $P(H|ls)$ from the priors and from the observed learning sample ls. The output of the approach is thus a *distribution* over $\mathcal{H}$, rather than a particular model $H(ls)$. Thus, unlike classical techniques, the Bayesian approach gives information about the uncertainty which remains about the model when the data is known. The posterior probability distribution gives us a ranking of the plausible models of the hypothesis space for our data and may be used for several inference tasks: parameter estimation, prediction, and model selection.

As concerns the use of $P(H|LS)$ for model selection, this approach thus constitutes an alternative to the other automatic learning methods described earlier. The most direct way to select a model from data is to select the most probable one according to the posterior probability distribution:

$$f_{LS} = \arg\max_{h\in\mathcal{H}} P(H = h|LS) = \arg\max_{h\in\mathcal{H}} P(LS|H = h)P(H = h). \quad (4.7)$$

This formulation of model selection allows to take into account the bias/variance compromise in two ways.

First, it is possible to choose the prior distribution of models in order to penalize too complex models. For example, under some hypotheses, it is possible to give a Bayesian interpretation of the weigth decay approach for learning neural networks (see [CM98]). Indeed, let us assume that the hypothesis space is composed of all possible parameterizations of a neural network of a given complexity and the goal of learning is to find the most probable choice of parameters given the learning sample. A model is then defined by the vector of parameters $\underline{W}$ and the Bayesian approach aims at computing $P(\underline{W}|LS)$. In the previous section, we have seen that large weights imply irregular functions and hence should be avoided. A way to penalize large weights is to assume that small values of the weights are a priori more probable, e.g. taking:

$$P(\underline{W}) = \frac{1}{Z_\beta}\exp(-\frac{\beta}{2}\sum_{i=1}^{d} W_i^2), \quad (4.8)$$

where d is the number of parameters of the neural network, β is some parameter of the distribution, and Z_β is a normalization constant. On the other hand, the likelihood of the learning sample ls (of size N) for some value of the parameter w is given by:

$$\begin{aligned} P(LS = ls|\underline{W} = w) &= \prod_{i=1}^{N} P(\underline{A} = \underline{a_i}, Y = y_i|\underline{W} = \underline{w}) && (4.9) \\ &= \prod_{i=1}^{N} P(Y = y_i|\underline{A} = \underline{a_i}, \underline{W} = \underline{w}) \prod_{i=1}^{N} P(\underline{A} = \underline{a_i}), && (4.10) \end{aligned}$$

since parameters $\underline{W}$ does not predict input values. Assuming that the output values are generated according to

$$Y = f(\underline{A}; \underline{w_o}) + \varepsilon,$$

where $f(\underline{A}; \underline{w_0})$ is the prediction of our neural network with some parameters $\underline{w_0}$ and ε is a Gaussian random variable, we have:

$$P(Y = y_i|\underline{A} = \underline{a_i}, \underline{W} = \underline{w}) = \frac{1}{Z_\alpha} \exp(-\frac{\alpha}{2}(y_i - f(\underline{a_i}; \underline{w}))^2), \qquad (4.11)$$

where α depends on the variance of ε and Z_α is some normalization constant. The most probable choice of parameters is the one which maximizes $P(\underline{W}|LS) \propto P(LS|\underline{W})P(\underline{W})$ and hence which minimizes $-log(P(\underline{W}|LS))$. Using equations (4.8), (4.10), and (4.11), one can show that:

$$-log(P(\underline{W} = \underline{w}|LS = ls)) = \frac{\alpha}{2} \sum_{i=1}^{N} (y_i - f(\underline{a_i}; \underline{W}))^2 + \frac{\beta}{2} \sum_{i=1}^{d} W_i^2 + C, \qquad (4.12)$$

where C is some constant which does not depend on the parameters $\underline{w}$. Hence, minimizing $P(\underline{W} = \underline{w}|LS = ls)$ according to $\underline{w}$ is equivalent to minimizing an expression like (4.3) and so, the weight decay approach of the previous section has been given a Bayesian interpretation. In the context of decision tree induction, the "growing+pruning" strategy can be viewed as the selection of the a posteriori most probable model in a Bayesian framework. Indeed, the quality measure (4.4) using information quantity may be derived from a posterior distribution of decision trees where prior distribution penalizes large trees, see [Weh93].

Even without adopting a specific choice of prior distribution to penalize complex models, the Bayesian approach also encompasses some built-in prior preference for simpler models and hence implicitly penalizes complex models (see [Mac92]). More precisely, let us consider a hypothesis space of models which are defined by two sets of variables: a structure S (for example, for the hypothesis space of single hidden layer neural networks, the structure would denote the number of nodes in the hidden layer) and the parameters characterizing a model of this structure $\underline{W}$ (the exact value of the weights of the neural network). Note that the prior distribution over the hypothesis space can thus be written in the form

$$P(H) = P(S, \underline{W}) = P(S)P(\underline{W}|S),$$

and the Bayesian approach may be used in order to determine the most probable structure given the observed data by computing:

$$P(S|LS) \propto P(LS|S)P(S). \qquad (4.13)$$

Depending on the assumptions made on the prior distribution $P(H)$ this criterion will favor more or less complex models. Note that one can argue that a natural assumption is to consider that $P(S)$ is a uniform distribution (so called uninformative prior distribution) over the range of structures covered by $\mathcal{H}$. As concerns $P(\underline{W}|S)$ a similar argument would then also lead to a uniform distribution. However, since the dimensionality of $\underline{W}$ is an increasing function of the complexity of the structure, $P(\underline{W}|S)$, as well as $P(S, \underline{W})$, will necessarily be a decreasing function of the model complexity. On the other hand, the assumption that all the structures are a priori equiprobable, implies that the a posteriori most probable structure is the one which maximizes the evidence $P(LS|S)$ which may be written:

$$P(LS|S) = \int P(LS|S, \underline{W}) dP(\underline{W}|S), \tag{4.14}$$

where the integration is over all possible parameterizations of the model. The question is whether this criterion will favor simple models or complex ones. Actually, while the maximum likelihood approach (or the empirical risk minimization approach) which selects structures maximizing

$$\arg\max_{\underline{W}} P(LS|S, \underline{W}),$$

will systematically select the most complex models of the hypothesis space, the criterion (4.14) is not systematically biased towards the more complex structures. This is essentially due to the assumption of uniform conditional distribution $P(\underline{W}|S)$ which implies an exponentially fast decrease of $P(\underline{W}|S)$ with increasing complexity. This effect is generally strong enough to overcompensate (for complex enough models) the increase of $\arg\max_{\underline{W}} P(LS|S, \underline{W})$. We refer the interested reader to [Sch78] and [Mac92] for a more in depth discussion of this natural complexity penalization property.

Minimum description length (MDL) principle

The MDL principle, and its cousin, the *minimum message length* principle, are based on a data compression argument. Let us assume that we want to send output values corresponding to the learning cases to a receiver which already knows the input values. Then we have several alternatives, e.g. we could send an explicit description of the output values of each object, or we could determine and send to the receiver a model able to recompute the output values from the inputs, or we could send a model computing approximations of the output values together with error corrections. The MDL principle ([Ris78]) states that the best model for the learning sample is the one which minimizes the length of the message which has to be sent to the receiver:

$$L(ls, h) = L(ls|h) + L(h) \tag{4.15}$$

where $L(h)$ is the minimal number of bits necessary to describe the model to the receiver and $L(ls|h)$ is the minimal number of bits necessary to describe the errors made by this model on the learning sample. Intuitively, the lower is the empirical risk of the model, the lower will be the length $L(ls|h)$; in the limit if the model perfectly fits the ls this term vanishes. On the other hand, the number of bits used to code a model is certainly an increasing function of the number of parameters used to represent this model and hence of its complexity. Thus the two terms making up the description length are essentially representing a tradeoff between the empirical risk and the model complexity. The MDL principle thus embodies a preference for simpler models and is of the form $Err(f) + \lambda.C(h)$ which was proposed in Section 4.2.2.

Notice that in eqn. (4.15) the two terms defining $L(ls, h)$ are in theory defined as the Kolmogorov complexities of the corresponding objects. These quantities are essentially non computable [CT91] and hence various approximations have been proposed in the literature, leading to various practical versions of the MDL and MML principles.

On the other hand, the link which exists between the Bayesian approach and these principles can be made on the basis of Shannon's information theory [CT91]. Indeed, Shannon's first theorem says that the optimal encoding of long sequences of a random variable X amounts to taking codeword lengths $l_i = -\log_2(P(X = x_i))$ for the i^{th} value of this variable. So, by taking the length $L(ls|h) = -\log_2 P(LS = ls|H = h)$ and $L(h) = -\log_2 P(H = h)$ in (4.15), we get minus the logarithm of (4.6) and hence minimizing (4.15) is equivalent to maximizing (4.6). We refer the interested reader to [ZRF99] for a more detailed discussion of the MDL and MML principles and their relationship with Bayesian model selection, in the specific context of decision tree induction.

Structural risk minimization

The main benefits of the Bayesian and MDL principle is that they provide a language (using either a probabilistic interpretation or a encoding scheme) which facilitates the design of optimization criteria which take into account the tradeoff between error and complexity. However, we do not have assurance that minimizing the posterior probability or the number of bits to encode data and model will give low generalization error (which is the final goal of learning). Actually, it is easy to find a very bad choice of prior probability distribution or encoding scheme which will lead to inappropriate models for many problems. Furthermore, both approaches will often rely on some parameters (like α and β in (4.12)) which would require to use one of the techniques presented in Section 4.2.1 to be fixed automatically.

Structural risk minimization on the other hand is based on the bounds on generalization error given by Vapnik (see Section 3.5.4 or [Vap95]). This bound is the sum of the empirical risk and an increasing function of the VC-dimension, denoted h (which is some measure of the expressiveness of the hypothesis space). Hence, when the VC-dimension increases, empirical risk is decreasing like the bias in Figure 4.1 and the bound on the difference is increasing like the variance. All in all, the generalization error is bounded by a function which follows a trajectory similar to the one of the mean error in the same figure. Vapnik proposes structural risk minimization to minimize the error bounds. It works as follows:

- Structure the hypothesis space into nested subsets $\mathcal{H}_i$ such that $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots \subset \mathcal{H}_i \subset \ldots$ and the VC-dimension h_i of each $\mathcal{H}_i$ is finite. Hence, we have $h_1 \leq h_2 \leq \ldots \leq h_i \leq \ldots$.
- Find the function f_i^* which minimizes the empirical risk in each subspace $\mathcal{H}_i$.
- Among these functions, choose the one which together with its subspace minimize the error bound.

Structural risk minimization is thus strictly equivalent to our approach where the parameter s is the VC-dimension. However, this method uses the derived bound on the generalization error to select the optimal value of complexity and hence does not require an estimate of the generalization error. The practical application of structural risk minimization is however limited by the fact that often the VC-dimension is very difficult to compute.

4.3 Aggregation techniques

The techniques discussed in the previous section are dependent on the intrinsic bias/variance curves of the learning algorithms they are applied to. For a given value of the parameter s, bias and variance are determined by the learning algorithm and the optimization of s is tied to respect these values. The techniques described in the present section, on the contrary, try to attack the intrinsic bias/variance tradeoff of a learning algorithm essentially by shifting its variance curve towards zero without changing too much its bias curve. To this end, the common idea of all these techniques is to generate by some means a set of predictions using the original (unstable) learning algorithm, and then to aggregate these predictions using a voting or an averaging scheme, in order to yield a more stable final prediction. These aggregation techniques essentially differ in the way they operate on the original algorithm in order to build a set of predictions, but the usual way is to build an ensemble[1] of *models* whose predictions are aggregated.

This section is organized as follows : we start by presenting “Bagging”, one of the most popular aggregation technique; we then proceed by analyzing the effect of averaging[2] techniques applied to randomized algorithms; we conclude by proposing a general framework encompassing

[1]The term “ensemble methods” is used in the literature to denote techniques which combine different models produced by one or several different automatic learning algorithms ([Die00b]).

[2]Although in the strict sense “averaging” is a particular case of “aggregation” we will use the two terms interchangeably.

all these averaging techniques and show how bagging can be viewed as a particular case of this general framework.

4.3.1 Bootstrap aggregating (Bagging)

Underlying idea

By definition, the average model $\overline{f}_{LS}(\underline{a})$ corresponding to a learning algorithm has zero variance and it has the same bias as the original algorithm. So if we could find an algorithm producing on the basis of the learning sample an good approximation of this average model, this would reduce variance without too much altering bias. Let us study this idea in the case of regression and in an ideal situation, i.e. assuming that we can draw as many learning sets as we want from the distribution $P(LS) = P^N(\underline{A}, Y)$. An approximation of $\overline{f}_{LS}$ is then simply built by drawing several learning sets ls_i, $i = 1, \dots, T$, from $P(LS)$, inducing a model f_{ls_i} from each of the ls_i and then computing the average predictor by :

$$f_{ls^T}(\underline{a}) = \frac{1}{T}\sum_{i=1}^{T} f_{ls_i}(\underline{a}), \tag{4.16}$$

for some value of T. This expression defines a new model whose predictions are also random. Since the ls_i are independently drawn from the same distribution $P(LS)$, the average model corresponding to the new learning algorithm, which generates $f_{LS^T}(\underline{a})$, is nothing but the average model of the original algorithm :

$$E_{LS^T}\{f_{LS^T}(\underline{a})\} = \frac{1}{T}\sum_{i=1}^{T} E_{LS_i}\{f_{LS_i}(\underline{a})\} = \overline{f}_{LS}(\underline{a}). \tag{4.17}$$

On the other hand, its variance is given by:

$$\mathrm{var}_{LS^T}\{f_{LS^T}(\underline{a})\} = \mathrm{var}_{LS^T}\{\frac{1}{T}\sum_{i=1}^{T} f_{LS_i}(\underline{a})\} = \frac{1}{T^2}\sum_{i=1}^{T} \mathrm{var}_{LS_i}\{f_{LS_i}(\underline{a})\} = \frac{1}{T}\mathrm{var}_R(\underline{a}) \tag{4.18}$$

since the f_{ls_i} are independent and have the same variance as the original learning algorithm. Thus, averaging over T learning sets divides the variance by a factor T. All in all, the average prediction is less variable than each individual one and its bias remains unchanged. The result is obviously a decrease of the mean squared error at point $\underline{a}$. In this ideal case, it is possible to compute the value of T which leads to a value of error satisfying some initial requirements.

Equivalently, in classification, a set of models can be induced from random learning sets ls_i and the aggregated prediction at each point is the majority class among the set of predictions :

$$f_{ls^T}(\underline{a}) = \arg\max_c \sum_{i=1}^{T} 1(f_{ls_i}(\underline{a}) = c). \tag{4.19}$$

Numbers of interest in this case are the probabilities that this model outputs one particular class c. Let us denote by $P_{LS^T}(c|\underline{a})$ this probability for one class c. Assuming a sufficiently large value of T, we know from the central limit theorem that the proportion :

$$P_c(\underline{a}) \triangleq \frac{1}{T}\sum_{i=1}^{T} 1(f_{ls_i}(\underline{a}) = c), \tag{4.20}$$

of instances of class c appearing in the set of predictions is distributed according to a Gaussian distribution with mean :

$$E_{LS^T}\{P_c(\underline{a})\} = P_{LS}(c|\underline{a}), \tag{4.21}$$

and variance:

$$\mathrm{var}_{LS^T}\{P_c(\underline{a})\} = \frac{P_{LS}(c|\underline{a}) \cdot (1 - P_{LS}(c|\underline{a}))}{T}. \tag{4.22}$$

In a two class case, the probability $P_{LS^T}(c|\underline{a})$ is the probability of $P_c(\underline{a})$ being greater than 0.5, which yields :

$$P_{LS^T}(c|\underline{a}) = \Phi(\frac{0.5 - P_{LS}(c|\underline{a})}{P_{LS}(c|\underline{a}) \cdot (1 - P_{LS}(c|\underline{a}))} \cdot T), \tag{4.23}$$

where Φ is the upper tail of the standard normal distribution (see Figure 3.7, pp. 38). From the definition of Φ, it thus appears that this probability tends towards 1 with increasing values of T if c is the majority class at $\underline{a}$, $f_{LS}^{MAJ}(\underline{a})$, and towards 0 otherwise. So, the classifier f_T converges to the majority vote classifier. Asymptotically, the error of f_{LS^T} is thus equal to Tibishirani's bias and its variance vanishes. Contrary to the regression case, however, as mentioned in the previous chapter, this majority vote classifier is not always better than individual predictions for biased points (in the mean). However, assuming that most of the learning algorithms are mainly unbiased (at least the ones which need such techniques because of their high variance), aggregation is beneficial most of the times. Actually, in the two class case, it can be shown that we have $P_{LS^T}(c|\underline{a}) > P_{LS}(c|\underline{a})$ if $P_{LS}(c|\underline{a}) > 0.5$, whatever the value of $T > 2$ (see [ZC01]). Hence, if the learning algorithm is unbiased at this point, aggregated models $f_{LS^T}(\underline{a})$ are always better than single ones.

The bootstrap approximation

In practice, of course, we do not have access to the true distribution $P(LS)$ but only to a single learning set ls. The most direct idea to find an approximation of $P(LS)$ is to use bootstrap sampling to generate several learning sets from the original one. This is the key idea underlying bagging (for bootstrap aggregating, [Bre96b]). Bagging proceeds as follows:

1. From ls (of size N), derive different bootstrap samples bs_i $(i = 1, \dots, T)$ of size N by random resampling with replacement.

2. From each bs_i produce a model f_{bs_i} using the learning algorithm.

3. Produce a single model, f_{bag}, by aggregating the T models f_{bs_i}:

$$f_{\text{bag}}(\underline{a}) = \text{aggr}\{f_{bs_1}(\underline{a}), f_{bs_2}(\underline{a}), \dots, f_{bs_T}(\underline{a})\} \tag{4.24}$$

The aggregation operator in the context of regression models is of course averaging.

For classification problems, we have the choice between computing the majority class among the T predictions (like (4.19)) or, if the learning algorithm provides class probability estimates, we can average these conditional class probability estimates of individual models and then select the class corresponding to the largest aggregated probability estimate. Note that several authors have carried out experiments with these two bagging variants in classification (e.g. [BK99]) and even if aggregating class-probability estimates appears to be slightly better, there is no clear evidence that this improvement is significant.

4.3.2 Randomized algorithms

Let us assume that we dispose of a randomized learning algorithm which on different runs with a fixed learning sample may output different models. For example, typical neural network learning algorithms behave in this way since they start iterations from random initial conditions which are re-drawn on every run of the algorithm. Similarly, automatic learning methods using genetic algorithms in the optimization part typically will produce randomized results. Also, as we will see, many existing deterministic automatic learning algorithms may easily be modified in order to randomize them. Our aim in this section is to show that the variance introduced by such randomizations should be reduced by using the idea of averaging described in the preceding section in order to improve performance.

Bias/variance decomposition

Let us assume that the randomness of our algorithm can be fully captured by a random vector of parameters which we will denote by ε (e.g. the final model given by neural network learning is a function of the random initial weights). The model produced by this algorithm is thus a function of the two random variables LS and ε and will be denoted by $f_{LS,\varepsilon}$. The true mean error of this algorithm at a point $\underline{a}$ is given by $E_{LS,\varepsilon}\{Err(f_{LS,\varepsilon}(\underline{a}))\}$. Taking into account the additional expectation according to ε, the regression bias/variance decomposition of the mean error becomes (see (3.15)):

$$E_{LS,\varepsilon}\{Err(f_{LS,\varepsilon})\} = \sigma_R^2(\underline{a}) + (f_B(\underline{a}) - \overline{f}_{LS,\varepsilon}(\underline{a}))^2 + var_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\}, \tag{4.25}$$

where:

$$\begin{aligned} \overline{f}_{LS,\varepsilon}(\underline{a}) &= E_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\} & (4.26)\\ var_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\} &= E_{LS,\varepsilon}\{(f_{LS,\varepsilon}(\underline{a}) - \overline{f}_{LS,\varepsilon}(\underline{a}))^2\}. & (4.27)\end{aligned}$$

From the total variance theorem, the variance term may be further decomposed into two terms:

$$var_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\} = var_{LS}\{E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\} + E_{LS}\{var_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\} \tag{4.28}$$

The first term in (4.28) is the variance with respect to the learning set randomness of the average prediction according to ε while the second term is the expectation over all learning sets of the variance of the prediction with respect to ε.

Averaging out the effect of ε

Let us suppose that instead of building one single model $f_{ls,\epsilon}$ for a given learning set with our original random algorithm, we consider a hypothetical algorithm which would produce for the given learning set ls, the expected model $E_{\varepsilon|ls}\{f_{ls,\varepsilon}(\underline{a})\}$.

The bias of this method would actually be identical to the bias of the original random algorithm, since we have:

$$\overline{f}_{LS,\varepsilon}(\underline{a}) = E_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\} = E_{LS}\{E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\}. \tag{4.29}$$

On the other hand, the variance of this algorithm would be equal to the first term of eqn. (4.28).

In practice however, it may be that $E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}$ is difficult to compute exactly. This suggests the following approximation algorithm: Given a learning set ls, draw several values ϵ_i, $i = 1,\ldots,T$ from the distribution $P(\varepsilon|ls)$, build a model f_{ls,ϵ_i} according to the learning algorithm for each value ϵ_i and compute the average model:

$$f_{ls,\epsilon^T}(\underline{a}) = \frac{1}{T}\sum_{i=1}^{T} f_{ls,\epsilon_i}(\underline{a}). \tag{4.30}$$

Transposing the development of the previous section, it is easy to show that the bias of this model is equal to the bias of the original learning algorithm and that its variance is given by:

$$var_{LS,\varepsilon^T}\{f_{LS,\varepsilon^T}(\underline{a})\} = var_{LS}\{E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\} + \frac{E_{LS}\{var_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\}}{T} \tag{4.31}$$

So averaging over T values of ε reduces the second part of the variance by a factor T and leaves the remaining parts of the mean error unchanged. The global reduction of error by averaging is thus proportional to the term $E_{LS}\{var_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\}$:

$$E_{LS,\varepsilon^T}\{Err(f_{LS,\varepsilon^T}(\underline{a}))\} = E_{LS,\varepsilon}\{Err(f_{LS,\varepsilon}(\underline{a})\} - (1 - \frac{1}{T}) \cdot E_{LS}\{var_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}. \tag{4.32}$$

The more variable are the predictions with respect to ε, the more useful it is to use averaging.

Ambiguity decomposition

While equation (4.32) describes the mean impact of averaging, it is possible to relate the error of an average model to the individual errors of its components. Indeed, this latter term is given by:

$$\frac{1}{T}\sum_{i=1}^{T} Err(f_{ls,\epsilon_i}(\underline{a})) = \frac{1}{T}\sum_{i=1}^{T} E_{Y|\underline{a}}\{(Y - f_{ls,\epsilon_i}(\underline{a}))^2\}, \tag{4.33}$$

which may be decomposed into:

$$\frac{1}{T}\sum_{i=1}^{T} Err(f_{ls,\epsilon_i}(\underline{a})) = E_{Y|\underline{a}}\{(Y - f_{ls,\epsilon^T}(\underline{a}))^2\} + \frac{1}{T}\sum_{i=1}^{T}(f_{ls,\epsilon_i}(\underline{a}) - f_{ls,\epsilon^T}(\underline{a}))^2. \tag{4.34}$$

Inverting the decomposition, we obtain:

$$Err(f_{ls,\epsilon^T}(\underline{a})) = \frac{1}{T}\sum_{i=1}^{T} Err(f_{ls,\epsilon_i}(\underline{a})) - \frac{1}{T}\sum_{i=1}^{T}(f_{ls,\epsilon_i}(\underline{a}) - f_{ls,\epsilon^T}(\underline{a}))^2. \tag{4.35}$$

So, the error of the average model is the difference between the mean error of individual models minus the variance of their predictions. This decomposition is due to Krogh and Vedelsby [KV95] and is called the ambiguity decomposition. The last term of equation (4.35) is called the ambiguity:

$$A_r(\underline{a}) = \frac{1}{T}\sum_{i=1}^{T}(f_{ls,\varepsilon_i}(\underline{a}) - f_{ls,\epsilon^T}(\underline{a}))^2. \tag{4.36}$$

As ambiguity is always positive, the main conclusion is that the average model is always better than the individual models in the mean. How much better depends on the ambiguity: the more individual predictions disagree with each other, the better will be the improvement by consensus. Taking the expectation of both sides of (4.35) according to the learning set and ε^T, we retrieve equation (4.32) and by identifying the different terms, we get:

$$(1 - \frac{1}{T}) \cdot E_{LS}\{var_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\} = E_{LS,\epsilon^T}\{\frac{1}{T}\sum_{i=1}^{T}(f_{LS,\varepsilon_i}(\underline{a}) - f_{LS,\epsilon^T}(\underline{a}))^2\} \tag{4.37}$$

which shows that the reduction of mean error is equivalently the mean (over all learning sets and parameters choice) variance of the prediction inside the ensemble produced for each learning sample.

The case of classification problems

While the effect of model averaging according to ε is very simple to study in the context of regression, the problem is much more complex in classification. Assuming a random learning algorithm, Tibishirani's bias/variance decomposition of the mean error rate can be written:

$$E_{LS,\varepsilon}\{Err(f_{LS,\varepsilon}(\underline{a}))\} = Err(\mathrm{maj}_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\}) + \mathrm{var}_T(\underline{a}) \tag{4.38}$$

where $\mathrm{maj}_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\} = \arg\max_c P_{LS,\varepsilon}(c|\underline{a})$ computes the majority class over all values of LS and ε and the term $\mathrm{var}_T(\underline{a})$ is defined such that this additive decomposition holds. The relationship between the mean error of $f_{LS,\varepsilon}$ and the one of the aggregated classifier $\mathrm{maj}_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}$ is difficult to characterize. First, contrary to the regression case, the bias is not necessarily conserved by aggregation. Indeed, in general, we have:

$$\mathrm{maj}_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\} \neq \mathrm{maj}_{LS}\{\mathrm{maj}_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\} \tag{4.39}$$

and hence,

$$Err(\mathrm{maj}_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\}) \neq Err(\mathrm{maj}_{LS}\{\mathrm{maj}_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\}). \tag{4.40}$$

Second, even if the bias is unchanged, as variance can be negative, it is also possible for the error to increase by averaging. The only thing which is certain in classification is that $E_{\varepsilon|LS}\{1(f_{LS,\varepsilon}(\underline{a}) = c)\}$ are better estimates than $1(f_{LS,\varepsilon}(\underline{a}) = c)$ for class probabilities $P(c|\underline{a})$ in the sense of least square since their bias terms are the same and the variance of the latter is reduced by averaging. However, there is no guarantee that this will yield lower misclassification error.

Focusing on non average results, it is also difficult to rely the error of the majority vote classifier to the mean error of its individual constituents. For example, trying to transpose the ambiguity decomposition from the regression case, we get the following definition for the ambiguity in classification:

$$A_c(\underline{a}) = Err(\mathrm{maj}_{i=1\ldots T}\{f_{ls,\epsilon_i}(\underline{a})\}) - \frac{1}{T}\sum_{i=1}^{T} Err(f_{ls,\epsilon_i}(\underline{a})) \tag{4.41}$$

which positiveness is unfortunately not ensured.

4.3.3 General perturb and combine algorithm

Generic algorithm

If we average different models built from several runs of a random algorithm, we know from the previous section that we obtain a more stable algorithm. This idea can not be applied directly to a deterministic algorithm which always returns the same model in response to a fixed learning set. However, a deterministic algorithm can be made random by the introduction of some perturbations in the learning procedure. This gives the idea of the following generic algorithm:

- From a learning set ls, for i going from 1 to T:
  - perturb the learning procedure (e.g. by perturbing its data or settings of its parameters) with a set of random parameters ϵ_i drawn independently from a distribution $P(\varepsilon|ls)$ and apply the perturbed versions to get the models f_{ls,ϵ_i}
- Build the aggregated predictor f_{ls,ϵ^T} defined by:

  $$f_{ls,\epsilon^T}(\underline{a}) = \mathrm{aggr}\{f_{ls,\epsilon_1}(\underline{a}), f_{ls,\epsilon_2}(\underline{a}), \ldots, f_{ls,\epsilon_T}(\underline{a})\}, \tag{4.42}$$

  where, like for bagging, "aggr" stands for average in regression and majority vote or average of probability estimates in classification.

Following Breiman [Bre96a], we will call an averaging technique following this strategy a *perturb and combine* (P&C) algorithm. For example, bagging is a perturb and combine algorithm. The random vector ε is in this case a vector of N (the learning set size) integers randomly and uniformly drawn in $[1, N]$, where each component of ε indexes one element of the learning set selected in the particular random bootstrap sample of size N.

When does it work ?

For the sake of the present discussion let use the following terminology

- "original algorithm" denotes any given automatic learning algorithm (we assume, without any limitation, throughout the discussion that this algorithm is deterministic, say like the standard decision tree induction method);
- "perturbed algorithm" denotes the randomized version of the original algorithm, according to some perturbation technique (say a decision tree method selecting tests in a random fashion);

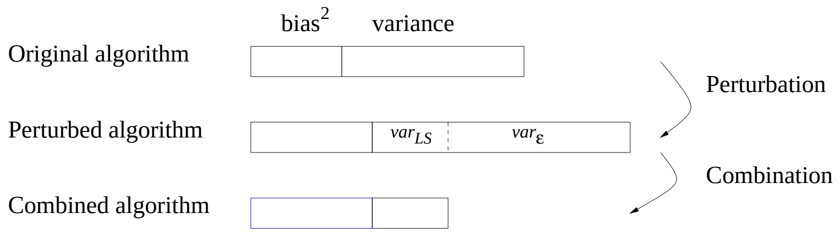


Figure 4.2: Expected evolution of bias and variance by perturbation and combination

- "combined algorithm" denotes the algorithm producing models built by aggregating T different models obtained with the perturbed algorithm.

Our discussion aims at giving conditions under which the *combined* algorithm would perform better (in the usual sense of average error) than the *original* algorithm. Given the formal difficulties that we have already encountered in the context of classification methods when discussing the effect of aggregation, we will focus here on regression.

Figure 4.2 shows the expected evolution of bias and variance during one perturb and combine procedure and is explained below. The perturbed algorithm provides a model $f_{LS,\varepsilon}$ which depends on the original algorithm and on the way the perturbation ε interacts with the induction process of this latter. In regression, the variance of this new algorithm can be decomposed into the two terms given in (4.28):

$$var_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\} = var_{LS}\{E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\} + E_{LS}\{var_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\}. \tag{4.43}$$

At the same time, the square bias of the perturbed algorithm becomes:

$$(f_B(\underline{a}) - \overline{f}_{LS,\varepsilon}(\underline{a}))^2. \tag{4.44}$$

Using the average operator to aggregate T regression models produced in this fashion, we obtain the combined algorithm which, as we already know, produces a model f_{LS,ε^T} with the same bias and lower variance than the perturbed algorithm. As T increases, the variance of the perturbed algorithm converges towards

$$var_{LS}\{E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\}.$$

Thus, asymptotically with respect to T, the whole process of perturbation and aggregation decreases the variance with respect to our original algorithm if:

$$var_{LS}\{E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\} < var_{LS}\{f_{LS}(\underline{a})\}, \tag{4.45}$$

i.e. if the perturbation artificially introduced in the learning procedure captures some part of the intrinsic variance of the initial learning algorithm, given that this artificial variance is canceled by the averaging process.

However, in order to assess the average error of the perturbed algorithm and compare it with that of the initial algorithm, we need also to take into account the effect on bias. Indeed, in general,

$$E_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\} \neq E_{LS}\{f_{LS}(\underline{a})\} \tag{4.46}$$

and hence bias could possibly be increased by the perturb and combine process. Actually, when the original learning algorithm explicitly tries to minimize empirical risk (like decision tree induction without pruning which builds trees of minimal learning set error), it is likely that the perturbed version will disturb the algorithm in this goal. Consequently, both the perturbed and the combined algorithms are likely to have higher bias than the original algorithm. All in all, a perturb and combine approach will be effective if the increase in bias is not larger than the reduction of variance.

It is possible to draw similar conclusions from the ambiguity decomposition (4.35). From this decomposition, Krogh and Vedelsby have emphasized the need for small mean errors of models $f_{LS,\varepsilon}$ and high ambiguity between them for averaging to be efficient. Small mean errors mean small values of $E_{LS,\varepsilon}\{Err(f_{LS,\varepsilon}(\underline{a}))\}$ and high ambiguity means a large variance term $E_{LS}\{var_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\}$. From the decomposition (4.25) which combines these two terms via (4.28), these two requirements can only be satisfied at the same time if the bias of $f_{LS,\varepsilon}$ is kept small and the variance term $var_{LS}\{E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\}$ is also small. So the randomization of the original algorithm should be carried out in order to meet these requirements.

In practice this discussion implies that for the perturb and combine method to be effective in reducing errors, the randomization part of it should essentially produce a highly diverse set of models of small empirical risk. These models will be likely to have small bias, and at the same time have a reduced dependence on the learning sample.

Randomization in classification

Of course, the discussion about classification problems at the end of the previous section can be transposed here. Very little can be said in theory about the effect of perturb and combine algorithms in classification. Nevertheless, several authors have searched for a relationship in classification between the diversity of ensemble members and the improvement obtained by averaging. For example, Ali [AP95] and Dietterich [Die00c] separately define a measure of the diversity of the classification obtained by the models $f_{LS,\varepsilon}(\underline{a})$ for random ε and empirically show a correlation between their diversity measure and the improvement obtained by aggregation. Ho in [Ho98] defines a measure of disagreement between a set of classification models and also attributes better performance of various perturb and combine algorithms to the increase of disagreement they yield in the context of decision trees. Amit and Geman [AD97, AB01], as well as Breiman [Bre01], give an upper bound on the error $E_{\underline{A}}\{Err(E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{A}))\}$ which is a decreasing function of the "strength" of the individual models (which is some measure of their accuracy) and an increasing function of the "correlation" between them (which is some measure of the dependence between their predictions). They also show empirically that strong uncorrelated models give the best improvement when aggregated.

Bias/variance tradeoff

From the preceding discussion as well as from experimental studies[3] it appears that in the P&C algorithms there is a bias/variance tradeoff regulated by the amount of perturbation injected in the learning procedure:

- the stronger the effect of randomization, the lesser the dependence of the perturbed models on the learning sample and the lesser the variance of the combined models;
- the stronger the effect of randomization, the lesser the dependence of output predictions on the input attributes and the higher bias.

Hence, suppose that we can measure the strength of the random effect of the perturbation by a parameter s, then the general techniques described in the previous section to fix s can be transposed here to fix the right level of perturbation.

Note that the increase of variance resulting from the second term of (4.43) does not influence the tradeoff since this term is the one which is canceled by averaging. Nevertheless, larger values of this term will imply slower convergence with respect to the number T of ensemble terms, and thus lead to higher computational requirements in practice.

Discussion

Note that the preceding analysis does not give a practical way to design valid perturb and combine algorithms. In practice, it is difficult to assess otherwise than empirically the effect

[3]See, for instance chapters 7 and 8 for detailed results in the context of decision tree induction.

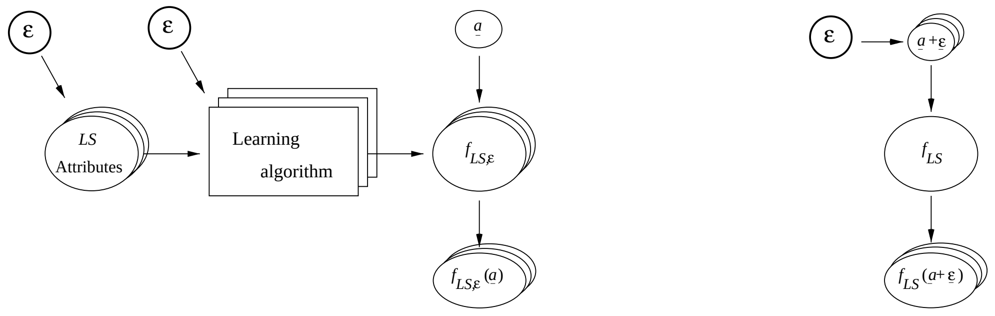


Figure 4.3: Three places where to introduce noise in the learning protocol. Left, with traditional perturb and combine algorithms. Right, the dual perturb and combine approach

of the introduction of a perturbation in a learning algorithm. However, by interpreting the effect of this perturbation in terms of bias and variance, it is somewhat possible to predict its behavior. For example, bagging which perturbs the algorithm by building a model from a bootstrap sample of the actual learning set is likely not to increase very much bias and, at the same time, should capture by this perturbation some of the variance of the initial algorithm. This kind of reasoning will allow us to design at Chapter 7 an extreme perturb and combine algorithm in the context of decision tree induction.

4.3.4 Examples of P&C algorithms

Several algorithms have been proposed in the literature which satisfy the description given in the preceding section. They all introduce some noise in the learning process so as to generate several models. Algorithms differ in the place where the random noise is injected during the learning process. The left part of Figure 4.3 shows two places where perturbations are usually introduced: the learning sample or the learning algorithm. The first type of techniques generate several models by feeding the original learning algorithm with modified versions of the learning sample and hence may be potentially applied to any learning algorithms. The second type of techniques directly perturb the learning algorithm to generate the set of models with the same learning sample. We review below some algorithms which have been proposed in these two categories.

Eventually, we introduce a new idea of perturb and combine method which contrasts with other methods by the fact that perturbations are introduced at the prediction stage. The right part of Figure 4.3 illustrates this idea. In order to generate a set of perturbed predictions, the attribute vector of the tested object is randomized and each of the resulting vectors is propagated in the same model, giving a set of predictions which are then aggregated. In this category, we briefly discuss our "Dual Perturb and Combine" algorithm which will be presented in more detail in Chapter 8.

Perturbing the learning set : bagging and its variants

The most direct idea to emulate the variance coming from the learning set randomness is to perturb the learning set itself. The most direct idea to do this is then to use bootstrap sampling to generate several learning sets from the original one. Starting with a learning set ls of size N, this yields the following general procedure:

1. From ls, derive different "bootstrap" samples ls_i, $i = 1, \ldots, T$, of size M by random resampling with or without replacement.

2. From each ls_i, produce a model f_{ls_i} using the learning algorithm.

3. Produce a single model by aggregating the T predictions of the models f_{ls_i}

Bagging is a particular instance of this algorithm with $M = N$ and resampling with replacement. Its generalization has been studied for example by Sollich and Krogh [SK96] for linear models and by Friedman and Hall [FH00b] in the nonlinear case. Although better results are possible with smaller values of M and/or without replacement sampling, bagging has proven to be very efficient in many applications, and although its efficiency varies from one application to another it almost never decreases accuracy.

Wagging [BK99] is a variant of bagging which requires that the learning algorithm can handle weights associated to learning cases. These weights give differing relative importance to learning cases. All the algorithms presented here can deal with such weights. For example, in neural networks, weights are introduced naturally in the error criterion which becomes a weighted sum of the square difference between the output and model. In decision trees, frequency counts at nodes also can easily take into account such weights. While original learning algorithm uses unit weights for each instance of the learning set, wagging (for weight aggregating) perturbs the learning set by assigning random weights to the learning set cases. For example, in the original formulation of wagging, Bauer and Kohavi [BK99] generate random weights from a Gaussian distribution of mean zero and a fixed standard deviation. Webb's version [Web00] of wagging draws weights from a continuous Poisson distribution. In both variants, wagging has been shown to give very similar results to classical bagging.

Another way to perturb the learning set is to perturb the attribute vector describing each instance. Along this idea, Breiman proposed an approach he calls *output smearing* [Bre00], in which only the output attribute values are perturbed when the auxiliary learning sets are generated. In the case of regression, an instance $(\underline{a}, y) \in LS$ thus becomes $(\underline{a}, y+\sigma_y \cdot \varepsilon)$ where ε is the realization of a Gaussian random variable of mean zero and unit standard deviation and σ_y is the standard deviation of the output Y estimated in the learning set. Classification problems can be handled in this framework, either by transforming them into a regression problem (the output variables of which will correspond to class-indicator variables), or by *flipping* output classes. This latter option consists in changing the output class of an instance for another one according to a probability distribution. The probability $p(j|k)$ of flipping class k to class j is given in his implementation by:

$$p(j|k) = \begin{cases} w \cdot c(j) & \text{if } j \neq k \\ 1 - w(1 - c(k)) & \text{if } j = k \end{cases} \tag{4.47}$$

where w is a parameter and $c(k)$ is the proportion of element of class k in the learning set. The experiments detailed in [Bre00] shows that randomizing output (output smearing or output flipping) actually gives consistently better results than bagging.

Another way to perturb the attribute vector is to randomly hide some attributes to the learning algorithm. This approach is adopted for example in [CC00] in combination with K-NN or in [Ho98] in combination with decision trees.

Perturbing the learning algorithm

If the learning algorithm is random, i.e. may output different models in different runs on the same learning set, we have seen that some part of the variance can be reduced simply by running the algorithm several times on the same learning set and aggregate the predictions. For example, in neural networks, a large part of variance is due to the fact that initial weights are random. A good way to get rid of this variance is to build several models with different random initial weights and aggregate the results. This procedure has been called "simple" in [Han00] and shows improvements comparable to bagging or boosting on several problems. These results are confirmed by experiments in [OM99].

If the learning algorithm is deterministic (or not), several predictions can still be produced by injecting perturbations during learning. In the context of neural networks for example, several models can be produced by using different sets of learning parameters [Alp93], or by considering different architectures [Has97]. To be efficient in terms of variance reduction, these perturbations should be consistent with natural perturbations which comes from the learning set randomness. For example, in decision tree induction, at each node, a test is selected among a set of candidates. In the next chapter, it is shown that the choice of a particular test is highly variable from one learning set to another. This variability can be artificially expressed by randomly selecting one test among the best ones in the set of candidates. This approach is adopted by Dietterich in [Die00c] to build random trees, by selecting at each node a test at random among the twenty best candidate tests. A set of trees is built from the learning set using this random algorithm and their predictions are aggregated. The experiments reported in [Die00c] show that this type of randomization compares favorably to bagging. Several other ways to perturb decision tree induction proposed in the literature will be discussed in more details in chapter 6.

Dual perturb and combine

The two previous algorithms precompute a set of models by injecting perturbations at the learning stage; for prediction these models are used one by one on a new object so as to yield a corresponding number of output values which are then aggregated. In contrast to this "perturbation at the learning stage", one can imagine to build one single model with the initial learning algorithm and learning sample and then perturb the predictions made by this model by introducing at the prediction stage some random variations. For example, in the "dual perturb and combine method" proposed in [Geu01a], perturbations of the predictions of a model f_{ls} at a point $\underline{a}$ are obtained by

$$f_{ls,\underline{\epsilon}}(\underline{a}) = f_{ls}(\underline{a} + \underline{\epsilon}), \tag{4.48}$$

where $\underline{\epsilon}$ is a noise vector of same dimensionality than the input space, generated according to some appropriately chosen distribution, and where the $+$ operator denotes in a generic way the combination of two attribute vectors.

Based on this idea, we propose a new averaging technique which proceeds as follows: A certain number of perturbed versions of the attribute vector $\underline{a}$ of a test instance are produced. The model (induced traditionally from the original learning set) is applied to these vectors resulting in a set of predictions which are aggregated to obtain the final prediction. Considering numerical attributes only, a natural choice to perturb the attribute vector is to add a zero-mean Gaussian noise to it. Thus, dual perturb and combine makes a prediction at $\underline{a}$ with a model f_{ls} according to the following formula:

$$f_{ls}^{DPC}(\underline{a}) = \text{aggr}\{f_{ls}(\underline{a} + \underline{\epsilon}_1), f_{ls}(\underline{a} + \underline{\epsilon}_2), ..., f_{ls}(\underline{a} + \underline{\epsilon}_T)\}, \tag{4.49}$$

where the vectors $\underline{\epsilon}_i$ are obtained from a simple diagonal Gaussian distribution normalized according to the variance of each input variable. In other words, for each input attribute A_i a noise term is generated according to a normal distribution of zero-mean and standard deviation equal to $\lambda\sigma_i$, where σ_i is the standard deviation of the attribute A_i in the learning set and λ is a constant which magnitude reflects the intensity of perturbation.

If the model is smooth, the value of $f_{ls}^{DPC}(\underline{a})$ will be close to the prediction $f_{ls}(\underline{a})$ for small values of λ. On the other hand, if the model is rough, dual perturb and combine makes it smoother. The amount of noise on attributes, measured by λ, thus regulates some bias/variance tradeoff and the best value can be determined by cross-validation for example.

Although less efficient than bagging, we will see in Chapter 8 that this technique reduces very strongly the variance of decision trees. Furthermore, one of its main advantages with respect to other averaging techniques is that it does not require to learn several models and hence intrinsically saves the interpretability and efficiency of the learning algorithm to which it is applied.

4.3.5 Other aggregation techniques

We have described a general framework encompassing perturb and combine algorithms like bagging. Several other techniques have been proposed in the machine learning literature which also rely on the combination of the output of several models. We describe some of them in this section and relate them to the perturb and combine approach. We first review Bayesian model averaging, then boosting, and eventually, several approaches where the combination of the outputs is learned from the data in contrast with perturb and combine algorithms which give equal importance to all models in the ensemble.

Bayesian model averaging

The full Bayesian approach does not advice to select one model from the posterior probability distribution (for example the maximum a posteriori) but rather to compute the average predictions at a point $\underline{a}$ according to the posterior distribution, i.e.:

$$f^*_{LS}(\underline{a}) = \int_{\mathcal{H}} h(\underline{a}) dP(H = h|LS), \tag{4.50}$$

In practice, this integration over all possible models is only tractable in specific cases and, usually, it is replaced by approximate subsampling from the posterior distribution $P(H|LS)$ (for example by Monte-Carlo methods [Nea96]).

With respect to the classical selection of one model, this approach takes into account the uncertainty about the model which remains when we have seen the data. From this point of view, Bayesian model averaging draws similarities with bagging and other perturb and combine approaches. The ultimate goal of perturb and combine algorithms (as discussed in this section) is to capture and eliminate the uncertainty about the model which are caused by the uncertainty about the learning sample. To this end, this uncertainty is approximated by some probability distribution on models $P_{P\&C}(H|LS) = P(H|\varepsilon, LS).P(\varepsilon|LS)$ where $P(H|\varepsilon, LS)$ is often deterministic. This distribution is then sampled to build the models which are finally aggregated. We have seen that in our approach, randomization should not decrease too much the bias which means that the error of the model should remain low on the learning sample. So, the posterior probability distribution with P&C would ideally be centered around the best models on the learning sample. On the other hand, the posterior distribution $P(H|LS)$ with uniform priors is proportional to $P(LS|H)$ and hence should also be centered on very good models (from the empirical risk viewpoint). So, intuitively, perturb and combine may be considered as an approximation of the full Bayesian approach. However, whether or not this relationship gives some explanation of the good behavior of either perturb and combine algorithms or Bayesian approaches is a difficult question. Some discussions about the links between Bagging and Bayesian model averaging may be found in [Dom97b] and [HTF01] for example.

Boosting

Boosting is still another approach which combines the predictions of several classification models. Originally, it has been proposed by Schapire [Sch90] as a way to "boost" the performance on the learning sample of a "weak" learning algorithm, i.e. a learning algorithm which is only slightly (but certainly) better than chance. After several improvements, this algorithm has evolved into the now popular AdaBoost algorithm (for "adaptive boosting", see [FS95]). The "M1" version of this algorithm introduced in [FS96] is given in Table 4.1.

Contrary to perturb and combine algorithms, Boosting builds models sequentially. It assumes that the weak learning algorithm can handle weighted objects in the learning sample (this is the case of all algorithms discussed in this thesis) and sequentially applies this algorithm to the original learning sample with modified weights. Starting with uniform weights, the algorithm sequentially changes them by increasing the weights of the misclassified instances by the previous model in the sequence. So, the learning cases which are difficult to classify by the type of model under consideration are receiving higher and higher weights as the iteration proceeds and the subsequent models are thus forced to focus on these "difficult" instances. Eventually, predictions are carried out by a weighted majority vote among the models according to weights α_i which depend on the accuracy of the corresponding model on the learning sample.

Table 4.1: Boosting algorithm ("AdaBoost.M1")

```
AdaBoost.M1($ls$, $LA$):
(Build a boosted ensemble of models from a learning sample $ls$ using a learning algorithm $LA$)

  • set $w_k = 1/N \ \forall k = 1, \ldots, N$
  • For $i = 1$ to $T$
      – Build a model, $f_{ls,i}(.)$, with $LA$ from the learning sample $ls$ using weights $w_k$.
      – Compute
            $Err_i = \frac{1}{N} \sum_{k=1}^{N} w_k . 1(c_k \neq f_{ls,i}(\underline{a}_k))$,
        and $\alpha_i = 1/2 \log((1 - Err_i)/Err_i)$.
      – Set $w_k = w_k . \exp(\alpha_i . 1(c_k \neq f_{ls,i}(\underline{a}_k)))$, $k = 1, \ldots, N$, and normalize these weights
        such that $\sum w_k = 1$.
  • Return the model
            $f_{Boost}(\underline{a}) = \arg\max_{c} \sum_{i=1}^{T} \alpha_i 1(f_{ls,i}(\underline{a}) = c)$
    .
```

The efficiency of boosting at improving classification learning algorithms has been highlighted in many empirical studies, mainly in combination with decision trees (e.g. [DC96], [BK99]). Actually, boosting with trees has been termed by Breiman [Bre98] as the "best off-the-shelf classifier of the world". Several variants of this original algorithm have been proposed, for classification problem [Bre98] but also for regression problems [FHT00].

Although boosting and perturb and combine algorithms have been compared in many empirical studies (e.g. [Qui96, BK99, Die00c]), the two algorithms are very different. Indeed, boosting is a deterministic algorithm while bagging and perturb and combine algorithms are random. While the main goal of perturb and combine approaches is to reduce the variance of a learning algorithm, the goal of boosting is to reduce the bias as well by focusing on misclassified instances in the learning sample. From our discussion about bias and variance, it is clear that such an algorithm which focuses more and more on misclassified instances in the learning sample should suffer at some point from overfitting. Although it is possible to find situations where the error of boosting starts increasing when further models are added to the ensemble (see [FHT00]), boosting seems nevertheless very resistant to overfitting in practice. Because of its success, many researchers have come with some theory to explain the good behavior of boosting algorithms ([SFBL97, Bre99, FHT00]) but there still remain several open questions to be solved before we can really explain the good behavior of boosting algorithms.

As this algorithm has become a state of the art algorithm especially in combination with decision trees, in subsequent chapters, we will sometimes analyze our results with respect to those obtained by boosting decision trees (using the AdaBoost.M1 version of Table 4.1).

Mixture of models, stacking

In our formulation of perturb and combine, models are averaged using uniform weights. In boosting, even if the weights are changing from one model to another, they are not learned globally to improve the accuracy of the ensemble of models. There exist however several techniques in machine learning which aim at learning the way to combine several models to improve the accuracy of the ensemble. For example, in a mixture of experts [JJNH91], the predictions of several models (the experts) are aggregated by using weights which are function of the points of the input space at which we want to make a prediction. The idea is that some experts may be better than others to predict the output variable at some point of the input space and hence they should receive more importance at this point. The input space is divided by a model which is related to soft decision trees (see Section 8.4.3 of Chapter 8). Another method to learn the combination step is stacking [Wol92]. Stacking consists in using a learning algorithm to build a model which combines the predictions given by several other models (which may be built all with the same algorithm or with different ones). If the combining algorithm learns linear models, then we come up with the weighted predictions of several models where the weights are learned from the data.

With respect to the perturb and combine algorithms, the idea underlying such algorithms is somewhat different. It is argued that the models are complementary and hence that accuracy will be improved by combining their prediction taking into account their complementarity. The idea behind perturb and combine algorithms is that several models are equally good (or bad) given the available learning sample and we have no argument to prefer one over the others. So, learning the weights of the aggregation from the learning sample should reduce the bias but also increase the variance with respect to the uniform aggregation of the same models.

4.4 Conclusion

In this chapter, we have introduced several means to control the variance of a learning algorithm. The first class of methods are means to control the complexity of the hypothesis space so as to find the best tradeoff between bias and variance. The second class of methods, termed perturb and combine algorithms, consist in aggregating the predictions provided by several models built from the same learning sample. The dual perturb and combine algorithm contrasts from other perturb and combine algorithms by the fact that it generates its perturbed predictions at the prediction stage with only one model.

We will exploit these techniques in the subsequent parts of this thesis. In the context of decision tree induction, the first kind of method will be described and validated in Chapter 6. Chapters 7 and 8 are respectively devoted to perturb and combine algorithms and the dual perturb and combine approach. Many of the ideas developed here will also be used in the third part of this thesis in the specific context of learning from temporal data.

Part II

Bias and variance in decision tree induction

Chapter 5

Variance in decision tree induction

In this chapter, we provide a thorough empirical evaluation of decision tree variance. The exact impact of variance on mean error is first studied with respect to several parameters. Then, the possible sources of this variance are enumerated and an experimental evaluation is carried out in order to appraise the relative importance of these sources and their contribution to the total prediction variance. Eventually, we demonstrate through simulations the high variance of the parameters which define tests at the internal nodes of a decision tree (attribute and discretization threshold).

We focus on decision trees, not on regression trees. The main justification of this choice is that tree based methods are used much more in classification than in regression.

5.1 Introduction

The fact that decision (or regression) tree induction is among the automatic learning methods of highest variance has been repeatedly claimed by many researchers (e.g. [DK95] or [Bre96b]). The experiments of chapter 3 have confirmed this on one example where it was found that this variance is actually the main cause of high prediction errors of regression trees. It is thus sensible to develop specific techniques which aim at reducing this variance, but before trying to do so we need first to evaluate more systematically the importance of tree variance, understand where it comes from and how it translates into instability of the different parts of a decision tree. In this chapter, a thorough empirical study is carried out to study this variance in detail. The main goal of this study is, first, to convince the reader that variance is indeed a major drawback in decision tree induction, and, second, to highlight the main sources and consequences of this variance and thus pinpoint the parts of the algorithm which could be improved to reduce this variance. The variance reduction methods presented in the subsequent three chapters have been designed on the basis of the conclusions drawn from these experiments.

As discussed in Chapter 3, variance has two main consequences on a learning algorithm. First, by means of the bias/variance decomposition, we know that it reduces the mean accuracy of the learning algorithm. Second, in the case of high variance, the parameters of the models produced by the learning algorithm are very different from one learning set to another. While the effect on accuracy is undesirable for any learning algorithm, the parameter variance is especially undesirable in the context of decision tree induction which pretends to produce interpretable models.[1] These two types of effect of variance will be appraised in this chapter.

The study is carried out in three steps. First, Section 5.4 focuses on the prediction variance of decision trees. Experiments are carried out to show that variance is an important source of error in decision tree growing. We will study how this variance evolves with respect to the learning sample size and the complexity of the trees.

After having shown the high variance of decision trees, the next step is of course to pinpoint

[1]Instability of tree parameters (choices of tests, number and labels of terminal nodes) translates into instability of the meaning of the decision rules extracted from a tree, and hence makes interpretation uneasy.

the possible sources of variance during tree growing. This is the goal of Section 5.5. From an analysis of the growing algorithm, three main sources are emerging. An empirical study will try to evaluate the importance of each one of these on prediction variance. From this study, it appears that the most important source of variance is related to the discretization of numerical attributes, i.e. the choice of thresholds in the tests on numerical attributes.

The last section is devoted to a study of parameters variance. The parameters we will consider are the attributes which are installed at test nodes and the discretization thresholds for numerical attributes. Among other things, the variance of these two parameters will be evaluated with respect to the score measure which is used to choose the tests during tree induction.

These experiments are preceded by a description of our particular implementation of decision tree induction algorithm in Section 5.2 and by general considerations about the experimental protocol in Section 5.3. Generally, decision tree induction is conducted in two separate steps: growing which aims at selecting the tree structure and then pruning which selects the right complexity. In this chapter, we focus only on the variance of the growing part of the algorithm. Since pruning is a variance reduction technique, it will be discussed in subsequent chapters. Also, we will concentrate on numerical attributes only. This choice is not too limitative since numerical attributes are dominant in most of nowadays applications of automatic learning techniques.

5.2 Description of the algorithm

The paternity of tree based methods is often attributed to the seminal work of Hunt and its concept Learning System [Hun62]. Nevertheless, the method was really popularized by the book of Breiman, Friedman, Olshen and Stone on Classification And Regression Trees (CART, [BFOS84]). Quinlan has also significantly contributed to the field with his ID3 algorithm and its successor C4.5 [Qui86]. Several variants of tree induction algorithms have been proposed which differ from the point of view of the candidate tests and specific aspects of the tree development algorithm. We are not going to discuss in detail here the different peculiarities of each method but mainly concentrate on our particular implementation (for a good review of tree and graph induction techniques, see [ZR00]). The method we are using in our experiment is the one proposed by Wehenkel (see for example [Weh98] for a description of the method).

Tree induction is generally composed of two separate steps:

- tree growing, which aims at selecting the tree structure, tests and predictions at leaf nodes.
- tree pruning, which aims at selecting afterwards the right complexity to avoid overfitting.

In this chapter, we focus on the growing part of the algorithm. The description of the pruning step which is a variance reduction technique will be given in the next chapter. We first give a brief overview of the general principles of decision tree growing algorithms. Then we describe the particular choice underlying our implementation. Finally, we discuss the different score measures which will be compared in our experiments (e.g. in Sections 5.6.1 and 5.6.3).

5.2.1 General principles of decision tree growing

The main goal of decision tree growing is to produce a partition of the input space into regions where the output variable is best estimated by a constant, i.e. into regions where the Bayes model is quasi-constant. Depending on the complexity of the Bayes model and on the level of precision required, this may be possible with a simple tree structure (with a small number of regions), or may require a very large number of regions. Decision tree induction aims at finding such a partition on the sole basis of a learning sample, and uses also the learning sample in order to determine the constant output prediction attached to each region (the labels attached to the terminal nodes of the tree). If we apply the empirical risk minimization principle to this problem, then decision tree induction would aim at finding a tree minimizing the resubstitution error on the learning sample. In other words it would aim at finding a partition of the learning set into subsets of objects which present the same output value, thus yielding an empirical risk equal to zero.

Unfortunately, the straightforward application of the empirical risk minimization principle does not make sense in this particular context[2], because for most problems there is an infinite number of decision trees which classify perfectly any learning sample of finite size, even for very large samples. One way to (at least partially) circumvent this difficulty would consist of restating the objective of decision tree induction as "finding the - or a - least complex tree of minimal resubstitution error". Another way out would be to search for a tree of minimal resubstitution error, of an a priori fixed complexity. Actually these two approaches are included in the following one: *define a tree quality measure as a tradeoff between complexity and empirical risk, and determine the tree of maximal quality.* Unfortunately, in all but trivial cases these alternative formulations lead to intractable problems (NP-completeness [HR76]).

Thus, in practice, since it is neither practical nor effective to consider every possible partition of the learning sample by a decision tree, decision tree growing is usually carried out in a top-down greedy fashion, using a *hill-climbing* algorithm. This algorithm starts with a single node tree corresponding to the complete learning sample and tries to split this node by selecting a test among a set of candidate tests. The algorithm then proceeds recursively to split the successors of this node. The whole process results in a partition of the learning set into smaller and smaller subsets. The development of a branch is stopped when some stop-splitting criterion applies. Eventually, each terminal node of the tree is labeled with a prediction which is computed on the basis of the local subsets of objects which reach this node. The different parts of this algorithm are briefly discussed below.

Score measure

To split a node, a score measure is defined and the test among the - yet to be defined - set of candidate tests which realizes the best score is selected. For the sake of simplicity of the resulting trees, score measures should favor tests which produce successors as pure as possible in terms of the output variable. To this end, the score usually evaluates the ability of the test to reduce the impurity of the output variable within the local learning subset. Thus, the problem is restated as "a score or purity maximization problem", and the choice of the particular score measure to use appears to be the fundamental question. Many different *partition score* measures have been proposed in the literature, most of which are derived from a *set impurity measure*; the most popular ones are those based on Shannon's entropy. In section 5.2.3 we will provide the precise definitions of the score measures used within our experiments, and in our experiments we will be able to appraise the effect of changing the score measure on the performances of decision tree induction.

Candidate tests

The set of candidate tests used in the score maximization procedure defines the actual hypothesis space and the representation power of decision trees of given complexity. Of course, the larger the set of candidate tests the easier it would be to induce simple trees with low empirical risk. On the other hand, in order to keep the score maximization tractable, the set of candidate tests should not be too "large", or at least structured in such a way that the score maximization can be efficiently[3] carried out. A straightforward way to yield scalable score maximization al-

[2]From the viewpoint of statistical learning theory, because the VC-dimension of the set of decision trees of arbitrary complexity is infinite.

[3]Without formalizing this notion, let us say that in automatic learning an algorithm is efficient (or scalable) if its complexity is bounded by a low order polynomial (e.g. linear) function of the database size (product of the number of objects and number of attributes).

gorithms is to restrict the number of candidate tests to be bounded by a low order polynomial function of the number of input attributes and of the size of the learning sample. Another possible approach is to choose the set of candidate tests and score measure in such way that there exists an efficient optimization technique.

In the standard decision tree induction method considered in this thesis, candidate tests involve only one attribute at a time and for each candidate attribute the number of candidate tests is bounded by the number of learning objects[4]. The main reason is computational efficiency, but another reason to use tests involving one single attribute is interpretability. Indeed, decision trees with such elementary tests are generally easier to interprete whereas trees involving complex tests are more black box like. Note that this simplification has important consequences in terms of application scope of the standard method. In particular this method is not well suited for problems where the information is shared among a large number of attributes (e.g. some of the time-series classification problems considered in the third part of this thesis).

In the standard method, node splitting is thus carried out in two stages: the first stage selects for each input attribute an optimal test and the second stage selects the optimal attribute. The type of candidate tests considered depends on the type of the attribute:

- **Symbolic attributes:** when the attribute takes its values in a finite set $\mathcal{A} = \{a_1, a_2, \ldots, a_k\}$, tests consist in partitioning $\mathcal{A}$ into a fixed number, p, of subsets and creating a successor node for each of these subsets. In the literature, generally partitions into singletons ($p = k$) or into two subset have been considered. The first approach proposes a single test for each discrete attribute while the second approach considers binary tests of the form $[A \in V\,?]$ where V is a subset of $\mathcal{A}$. We thus need some way to determine the subset V which realizes the optimal score. As the number of subsets V is 2^k, exhaustive search is only possible for small values of k.[5] But when the discrete set of values is ordered, the set of candidate tests can be reduced by taking into account this ordering: the natural candidate subsets V are the subsets $\{a_1, \ldots, a_i\}(i = 1, \ldots, n-1)$ which correspond to the tests $[A < a_{i+1}](i = 1, \ldots, n-1)$ compatible with the predefined ordering of $\mathcal{A}$;

- **Numerical attributes:** in the standard method, tests on numerical attributes at decision tree nodes are also of the form, $[A < a_{th}\,?]$, where a_{th} is a discretization threshold for the attribute A. Although it is possible to pre-discretize numerical attributes and handle them as ordered symbolic attributes, the most common approach is to incorporate the discretization process in the node splitting subroutine. To locally determine the optimal threshold value, normally an exhaustive search procedure is used so as to maximize the score measure. This algorithm is described below in the context of our implementation.

Stop splitting criterion

There are several possible stop-splitting criteria. First, it is certainly adequate to stop the development of a branch when the output variable is constant in the local subset or when all input attributes are constant (in which case impurity can not be further reduced). Another possibility is to compute the impurity of the output variable in the subset and stop splitting if this value is lower than a given threshold. It is also possible to stop induction when the number of instances in the current subset is too small. Since complex decision trees will suffer from variance, more elaborate stop-splitting criterion have been designed in order avoid overfitting. They will be discussed in the next chapter together with tree pruning methods.

Node labeling

In the context of classification, the subsample of objects which reach a terminal node gives some information about the distribution of the classification output variable in the corresponding

[4]Hence, the number of candidate tests actually varies from one test node to another.

[5]Heuristics have been proposed in the literature to cope with this problem (see [BFOS84] for example)

Table 5.1: Tree growing algorithm

```
Treegrow($LS$):
(Build a tree from the set $LS$.)

• Create a node $\mathcal{N}$ and set $LS(\mathcal{N})$ to $LS$;
• Set $L = \{\mathcal{N}\}$, the list of open nodes.
• While $L$ is not empty:
    – Select and remove a node $\mathcal{N}$ from $L$;
    – If stopsplitting($LS(\mathcal{N})$) is true then
        * Compute the vector of conditional class probability estimates $\hat{P}(C|\mathcal{N})$ from
          $LS(\mathcal{N})$ and attach it to the node $\mathcal{N}$;
    – Else
        * Select the attribute $A_{i^*}$ and threshold $a^*_{th}$ which maximize
          Score($LS(\mathcal{N}), [A_{i^*} < a^*_{th}]$);
        * Compute the subsamples $LS_l = \{(\underline{a}, y) \in LS(\mathcal{N}) | a_{i^*} < a^*_{th}\}$ and $LS_r = \{(\underline{a}, y) \in$
          $LS(\mathcal{N}) | a_{i^*} \geq a^*_{th}\}$;
        * Create two nodes $\mathcal{N}_l$ and $\mathcal{N}_r$ and set $LS(\mathcal{N}_l) = LS_l$ and $LS(\mathcal{N}_r) = LS_r$;
        * Insert $\mathcal{N}_l$ and $\mathcal{N}_r$ in $L$.
```

region of the input space. From this subsample, it is first possible to compute an estimation of the conditional class probabilities in this region. A maximum likelihood approach suggests to use class frequency counts to estimate these probabilities but more sophisticated estimation techniques are possible. From these probability estimates, it is possible to give a condensed prediction on the form of a class. The prediction which minimizes the resubstitution error rate is the class which maximizes the conditional class probability in this node, i.e., using the maximum likelihood estimates, the majority class in the local subsample.

5.2.2 Particular implementation

We give now the detail of the implementation of the decision growing algorithm we have used in our experiments. As already mentioned, we do not consider the case of symbolic attributes, so this part of the algorithm will be skipped in our description.

Structure of the algorithm

The particular form of the algorithm we used to build a decision tree from a learning sample LS is given in Table 5.1. This algorithm follows the top-down hill-climbing strategy described earlier. It works with a list L of open nodes which initially contains a single node: the top-node of the tree corresponding to the complete learning set. As long as the list of open nodes L is not empty, an open node is picked in L at each iteration step and a numerical attribute and a threshold are selected according to the score measure to split the local subset of objects reaching the node into consideration. The two successor nodes are created and they are inserted in L (together with the corresponding sub-samples of objects). The development of a node is stopped when a stop-splitting criterion applies. In this case, the (leaf) node is labeled with conditional class probability estimates computed by frequency counts in $LS(\mathcal{N})$.

We still need to precise the score measure, candidate test definition, and the stop-splitting criterion.

Score measure

The reference score measure we used in our experiments is based on a normalization of Shannon information (see [Weh98]). For a sample S and a test T, this measure is given by:

$$Score_W(S,T) = \frac{2.I_C^T(S)}{H_T(S) + H_C(S)}, \tag{5.1}$$

where $H_C(S)$ is the entropy of the classification in S, $H_T(S)$ is the test entropy and $I_C^T(S)$ the mutual information of the test and the classification. How these terms are computed from a sample will be explained more precisely in Section 5.2.3.

Discretization of numerical attributes

For a given attribute A, the determination of the threshold a_{th} which maximizes the score is determined by an exhaustive search. If we denote by $\mathcal{A}(LS(\mathcal{N})) = \{a_1, a_2, \ldots, a_n\}$, the set of different values of the attribute A which appear in the local subset $LS(\mathcal{N})$ sorted by increasing order, each threshold a_{th} between two consecutive values in $\mathcal{A}(LS(\mathcal{N}))$ induces the same division of $LS(\mathcal{N})$ and thus obtains the same score. So, our discretization technique considers only thresholds of the form $\frac{a_i+a_{i+1}}{2}(i = 1, \ldots, n-1)$. The local learning sample is first sorted by increasing values of the attribute, and the scores corresponding to these $n-1$ thresholds are evaluated each one in turn, and the best threshold value is retained.

This procedure is repeated for each input attribute and the best pair attribute-threshold is selected to split the node.

Stop splitting and development strategy

Two variants in terms of the stop-splitting criterion will be used in our experiments:

- **Fully grown trees:** These trees are obtained by the minimal stop-splitting criterion which stops the induction only when the output variable is constant in the local subset corresponding to the node, or when all input attributes are constant. In this case, because the stop-splitting criterion is *local*, the order by which nodes in L are processed is not important. Our implementation uses a *depth first* strategy, i.e. the first node in L is removed for splitting and new nodes are placed at the top of the list (L is thus managed like a stack).

- **Trees of limited complexity:** Sometimes, it is interesting to study the properties of trees of a priori fixed complexity. To generate such trees, we stop the development of any new node as soon as the number of terminal nodes has reached the desired complexity. The depth first approach which first develops at full a branch is not adapted to this *global* stop-splitting criterion. To give a more balanced sequence of trees, in this case, we need to fix the exact order in which the open nodes in L are considered. In our experiments, we use an approximate[6] *best first strategy* which consists in developing at each step the node which presents the highest impurity. This impurity is computed by the entropy of the classification variable in the learning sample times the size of the local learning set ($|LS(\mathcal{N})| \times H_C(LS(\mathcal{N}))$). So, at the beginning of each iteration step of the algorithm of Table 5.1, the node of highest impurity is selected for splitting from the list L of current open nodes.

In our experiments, we will sometimes consider minimal trees composed of only one test. Such trees are built by the best first strategy by limiting the complexity to two terminal nodes. These simple trees are called "decision stumps" in the machine learning literature (see [IL92]).

[6]A true best first strategy would select the node and test which together yield the largest increase in quality.

5.2.3 Alternative score measures

The score measure is responsible for most of the choices during the induction of the trees. Since we are studying in what follows the variance of decision trees, it is justified to wonder how far the choice of a score measure influences this variance. So, we review here several score measures which have been proposed in the context of decision tree induction. Three are derived from Shannon's entropy (including our reference measure), one is based on the Kolmogorov distance and the last one is based on the misclassification error.

Entropy based measures

The most used impurity measure is certainly Shannon's entropy. Entropy measures the uncertainty associated to a discrete variable. For example, the entropy of the classification variable, C, in the set S is given by:

$$H_C(S) = -\sum_{c=1}^{m} \frac{N_c}{N} \log_2 \frac{N_c}{N}, \tag{5.2}$$

where N is the size of the sample S and N_c is the number of objects of class c in S. The uncertainty about the class variable, $H_C(S)$, is maximum when $\frac{N_c}{N} = \frac{1}{m}$ for all c and is minimum when only one class appears in S. In this context, the mean reduction of impurity of a test T, that partitions the subset S into p subsets denoted by $S_i, i = 1, \ldots, p$, is denoted by $I_C^T(S)$ and computed by:

$$\begin{aligned} I_C^T(S) &= H_C(S) - \sum_{i=1}^{p} \frac{N_i}{N} H_C(S_i) && (5.3)\\ &= H_C(S) - H_{C|T}(S), && (5.4) \end{aligned}$$

where N_i is the size of S_i and $H_{C|T}(S)$ is the mean conditional entropy of the classification given the test. $I_C^T(S)$ is the mutual information between C and T. It reaches its maximal value $H_C(S)$ when the uncertainty about the class is null in every subset S_i and is null when $H_C(S_i) = H_C(S), \forall i$. In this latter case, the test does not provide any information about the classification. The first score measure we will consider in our experiments is the mutual information:

$$\text{Score}_{\text{info}}(S,T) = I_C^T(S). \tag{5.5}$$

Different variants of this score measure are obtained by normalizing it in different ways. In our tests, we consider two different normalizations. The first one has been proposed by Quinlan:

$$\text{Score}_Q(S,T) = \frac{I_C^T(S)}{H_T(S)}, \tag{5.6}$$

where $H_T(S)$ is the entropy of the test outcome in S:

$$H_T(S) = -\sum_{i=1}^{p} \frac{N_i}{N} \log_2 \frac{N_i}{N}. \tag{5.7}$$

The score (5.6) is called the "gain ratio" and the test entropy is called "split information" by Quinlan. The second normalization is proposed by Wehenkel [Weh98] and gives a symmetric score:

$$\text{Score}_W(S,T) = \frac{2.I_C^T(S)}{H_T(S) + H_C(S)}. \tag{5.8}$$

This latter score measure is the reference measure that we use in our experiments unless otherwise stated. For a discussion of entropy based score measures and normalizations, the interested reader can refer to [DM91] and [Weh96].

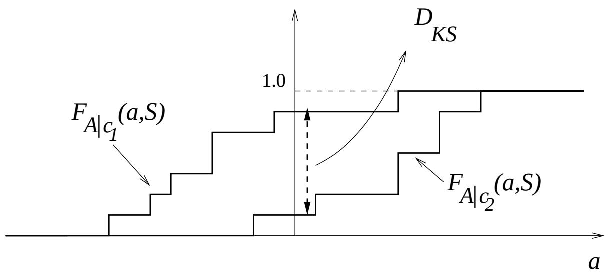


Figure 5.1: Class conditional cumulative distributions of an attribute and the Kolmogorov distance between these distributions

Kolmogorov-Smirnoff distance

For the discretization of numerical attributes, Friedman [Fri77] has proposed very early a quite different criterion which is based on the Kolmogorov-Smirnoff distance. Let us denote by $F_{A|c_i}(a,S)$ the empirical class conditional cumulative distribution of the attribute A in S. $F_{A|c_i}(a,S)$ is the proportion of instances of class c_i in S which correspond to a value for the attribute A lower than a:

$$F_{A|c_i}(a,S) = \frac{1}{N_i} \sum_{o \in S | C(o)=c_i} 1(A(o) < a). \tag{5.9}$$

The Kolmogorov distance between two distributions $F_{A|c_i}(a,S)$ and $F_{A|c_j}(a,S)$, for two classes c_i and c_j, is defined by:

$$D_{KS}(c_i, c_j, S) = \max_{a'} |F_{c_i}(a',S) - F_{c_j}(a',S)|. \tag{5.10}$$

This distance is illustrated in Figure 5.1. In the two-class case, Friedman proposes to choose the threshold which realizes the Kolmogorov-Smirnoff distance and to choose the attribute which maximizes this distance. This amounts at using a score measure defined in the two-class case by:

$$\text{Score}_{KS}(S,[A < a']) = |F_{A|c_1}(a',S) - F_{A|c_2}(a',S)|, \tag{5.11}$$

for a test T which compares the attribute A to a threshold a'. The standard extension to non-binary classes consists in using instead the maximal Kolmogorov-Smirnoff distance between one class against the group formed by all other classes. This yields the following score measures:

$$\text{Score}_{KS}(S,[A < a']) = \max_{c} |F_{A|c}(a',S) - F_{A|\neg c}(a',S)|, \tag{5.12}$$

where $F_{A|\neg c}(a',S)$ is the cumulative empirical distribution of A for instance of class other than c in S. This is the variant we will use in our experiments.

Empirical misclassification error

Since the goal of decision tree growing is to induce accurate rules in terms of error rates, an a priori logical choice for the score measure is the reduction of (resubstitution) error produced by the splitting of the current subset S using the test T:

$$\text{Score}_{Err}(S,T) = 1 - \max_{c} \frac{N_c}{N} - \sum_{i=1}^{p} \frac{N_i}{N}(1 - \max_{c} \frac{N_{c,i}}{N_i}) \tag{5.13}$$

$$= 1 - \max_{c} \frac{N_c}{N} - (1 - \frac{1}{N} \sum_{i=1}^{p} \max_{c} N_{c,i}), \tag{5.14}$$

where $N_{c,i}$ denotes the number of objects of class c in the subset i. The quantity $N \times \text{Score}_{Err}(S,T)$ is equal to the difference between two empirical errors :

Table 5.2: Data set summaries

Data set	Nb. attributes	Nb. classes	$\sigma_C(\%)$	$\#PS$	$\#LS$	$\#VS$	$\#TS$
Gaussian	2	2	11.85	14000	100	2000	4000
Waveform	21	3	14.0	3000	300	1000	1000
two-norm	20	2	2.3	7000	300	1000	2000
Omib	6	2	0.0	14000	500	2000	4000
Satellite	36	6	-	3435	500	1000	2000
Pendigits	16	10	-	5494	500	2000	3498
Dig44	16	10	-	9000	500	2000	4000

- $N - \max_c N_c$, which is the number of misclassified states in the local subset if all the states are classified into the majority class of the current node.
- $N - \sum_{i=1}^{p} \max_c N_{c,i}$, which the number of errors if the set is first split by the test T and then classified using the majority rule in each subset.

Even though this is intuitively the most evident score measure, it is well-known that this measure is pathological. Indeed, it is not sensitive to variations in the repartition of objects among subsets S_i, only to the total number of misclassified objects in S before and after the split[7]. Hence, other measures are generally preferred to this one. We nevertheless include this measure in our study, essentially because it behaves very differently with respect to the other ones.

5.3 Experimental protocol

Experiments of this chapter and the following ones are carried out on seven classification problems which are fully described in Appendix B. The parameters of each problem are summarized in Table 5.2. These datasets have been essentially chosen for their large size which allows reliable estimation of bias and variance terms. All attributes are numerical. In Table 5.2, σ_c denotes the minimal attainable error rate (the error rate of the Bayes classifier) for those datasets where this value is known; the last four columns show the number of objects of different subsamples derived from the dataset, which are used for different purposes as described below.

The bias/variance profile of a learning algorithm will be analyzed using Tibshirani's bias and variance terms, denoted bias_T and var_T, and Kohavi and Wolpert's variance term, denoted by var_{KW} (see Chapter 3 for the formulas). As decision trees give conditional class probability estimates, we will often complete our analysis by the bias/variance decomposition of the square error on these estimates. As we do not know the distribution $P(C|\underline{A})$, we will define the error on these probability estimates by:

$$Err_{\hat{P}} = \frac{1}{m} \sum_{i=1}^{m} E_{\underline{A},LS,C}\{(Y_C^i - \hat{P}_{LS}(c_i|\underline{A}))^2\}, \tag{5.15}$$

where we sum over the classes. This error is equal to the true square error on probability estimates only when there is no noise. However, this error shares its variance term with the true square error on probability estimates. This latter variance is given by:

$$\text{var}_{\hat{P}} = \frac{1}{m} \sum_{i=1}^{m} E_{\underline{A},LS}\{(\hat{P}_{LS}(c_i|\underline{A}) - E_{LS}\{\hat{P}_{LS}(c_i|\underline{A})\})^2\}. \tag{5.16}$$

All bias and variance terms are estimated using the bootstrap method described in Section 3.2.4. The size of the pool (PS) and test (TS) sample in each problem are given in Table 5.2.

[7]From a mathematical point of view, this is due to the fact that the error rate is not a strictly convex function of class probabilities [BFOS84].

Table 5.3: Average percentage of error due to variance

	var_T	var_{KW}	$\text{var}_{\hat{P}}$	σ_C
Full DT	54%	61%	61%	> 17%

For each dataset, a reference learning sample size is defined in the sixth column of the same table which is the sample size used during all experiments unless otherwise stated. In all cases, 50 learning samples are drawn to estimate expectations over the learning sample. As we do not know the Bayes model for all problems, bias values reported in tables and figures actually include the residual error (as discussed in Section 3.2.4). Nevertheless, the fourth column of Table 5.2 gives an idea of the residual error in classification for the problems where it is known.

5.4 Prediction variance

Note. Within the following presentation we will provide systematic studies over the seven datasets just described. For the sake of clarity, the detailed simulation results on each database are not provided within the main text. They are collected in the form of tables in Appendix C where the reader can look at them if desired. The results provided here are either average results over all seven problems, or some specific results for one or two datasets (generally in the form of graphics) selected in order to serve our purpose.

5.4.1 Percentage of error due to variance

Experiments of chapter 3 show that decision tree induction is one of the methods which suffers the most from variance while at the same time its bias is small. To appraise the exact impact of variance on the accuracy of decision trees, we estimated on all data sets bias, variances and error rates for a given size of learning sets (relatively small with respect to the pool set size, see Table 5.2). Table 5.3 summarizes these results in terms of average percentage of error due to variance for the three decompositions (for fully grown trees): Tibshirani's var_T, Kohavi and Wolpert's var_{KW} and variance of probability estimates, $\text{var}_{\hat{P}}$. The last column gives a lower bound on the part of the error rates which is due to the residual error. As already mentioned in previous chapter, this error is difficult to estimate in real problems. Nevertheless, taking into account residual error values when they are known (see Table 5.2) and assuming no noise when they are actually unknown, the percentage of error due to residual error in this case is already about 17%.

Thus, on these problems, more than 50% of the error can be attributed to variance, whatever the adopted decomposition. At the same time, taking into account the residual error, less than 30% of the error can be attributed to bias. Variance is thus the main source of error on these problems. Two factors that influence the bias/variance compromise are the learning sample size and the complexity of the tree. It is interesting to see how the relative effect of variance on error evolve with respect to these two parameters.

5.4.2 Variance as a function of learning set size

Figures 5.2 show, for two problems, the evolution of the different bias and variance terms for increasing learning set sizes and fully grown trees. As expected, when the learning set size increases, both bias and variance terms decrease leading to a reduction of error. However, the decrease is quite slow and variance always remains the most important source of error whatever the learning set size. We can observe that for the larger sample sizes the variance term really predominates with respect to bias.

To check the consistency of this analysis on other problems, Table 5.4 shows the mean effect on all seven datasets of using increasing learning set size (N denotes the reference size for LS given in Table 5.2). The first number in each cell of the table is again the percentage of the total error which is due to variance; the number between parentheses is the average

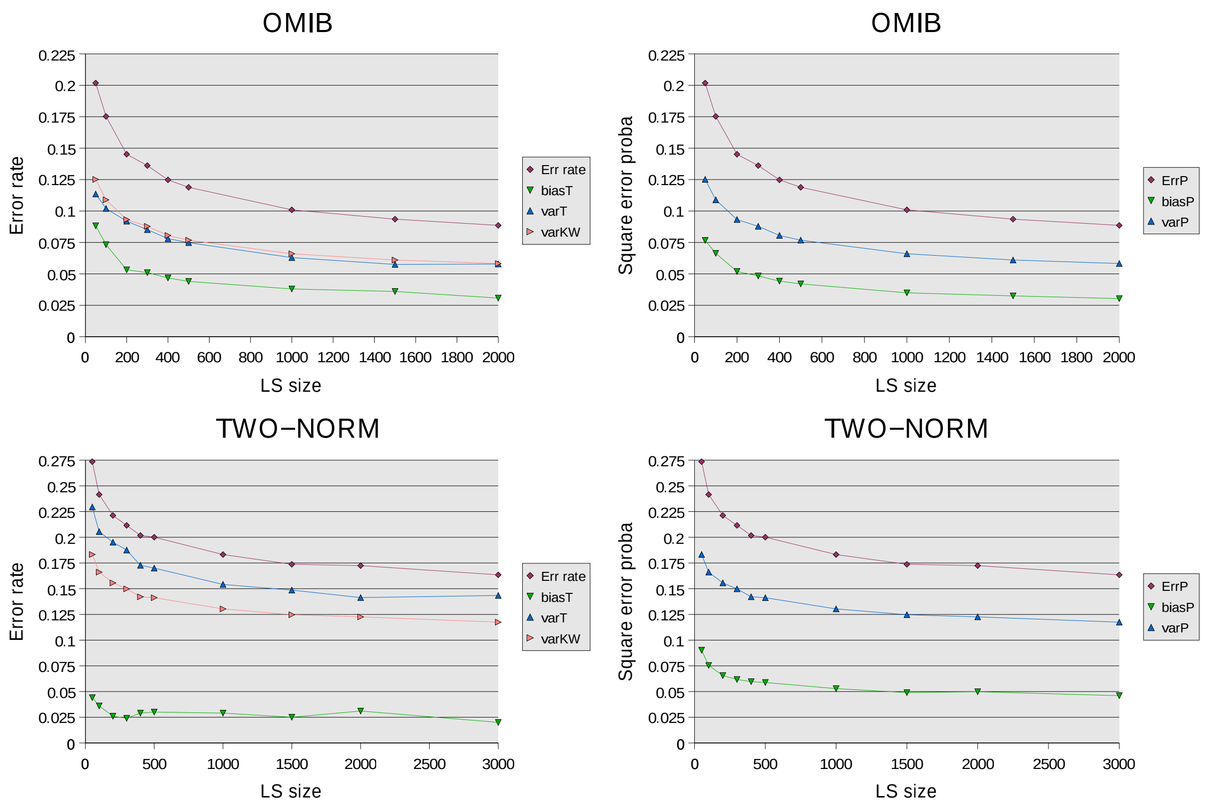


Figure 5.2: Evolution of different bias and variance terms with the learning set size. Top on the omib problem, bottom on the two-norm problem. Trees are fully grown.

Table 5.4: Percentage of error due to variance

LS size	$\text{var}_T(\%)$	$\text{var}_{KW}(\%)$	$\text{var}_{\hat{P}}(\%)$	$\sigma_C(\%)$
N	54 (100)	61 (100)	61 (100)	> 17
$2 \cdot N$	54 (90)	60 (88)	60 (88)	> 18
$4 \cdot N$	51 (75)	59 (78)	59 (78)	> 19

percentage of absolute variance which remains with respect to the reference size N. We can see that multiplying by 2 (resp. 4) the learning sample size reduces the variance by about 10% (resp. 25%). Thus, when the sample size increases variance decreases, but rather slowly, and its contribution to the error rate does not decrease significantly. On the other hand, bias decreases so that for larger sample sizes the error rate tends to be shared mostly between variance and residual error (which effect increases due to the reduction of bias). Asymptotically, we would expect both variance and bias to vanish, so that the residual error would contribute 100% to the error rate.

5.4.3 Variance as a function of tree complexity

It could also be argued that variance comes from the fact that trees are fully grown and that the variance of small decision trees is certainly much smaller. Figure 5.3 plots bias and variance curves for increasing tree complexity (the number of test nodes), again on the omib and two-norm problem. Here, the trees of a priori fixed complexity are grown using the best-first strategy described earlier. In all cases, bias is a decreasing function of complexity. We observe that on the omib problem, var_T is increasing with the complexity and reaches its maximum value very early in the sequence (around ten test nodes) although the bias is still decreasing. Kohavi and Wolpert's variance first increases for low complexity values and - surprisingly - decreases when the complexity further increases. On the other hand, bias and variance of probability

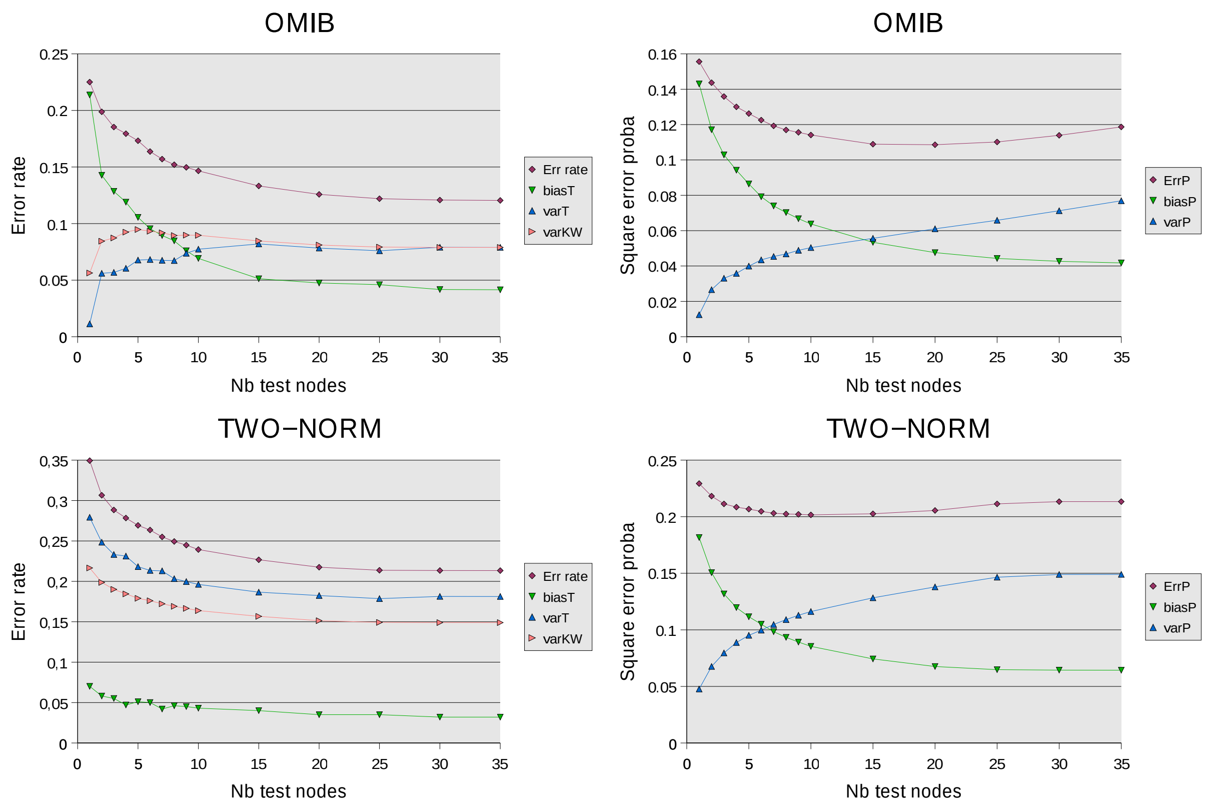


Figure 5.3: Evolution of different bias and variance terms with the tree complexity. Top on the omib problem, bottom on the two-norm problem

estimates follow the expected trajectories described in the previous chapter. On the two-norm problem, classification variance decreases monotonically with increasing complexity right from the beginning. Although counter-intuitive, this decrease of classification variance with increasing tree complexity can be explained as follows. When the complexity is low, terminal nodes of the tree are not pure and hence probability estimates are far from extreme values (zero or one). In these cases, even small variance of the estimates can make the majority class at a node flip from one class to another; the distribution $P_{LS}(C|\underline{A})$ is close to a uniform distribution which maximizes classification variance. This phenomenon is stronger when tests at shallow nodes are not able to reduce impurity very significantly. For example, in the two-norm problem, all 20 attributes influence the classification frontier in the same way. Thus, a test on a single attribute can reduce impurity only very marginally, and for this reason, impurity remains high within the first few levels of the tree. Hence classification variance remains also high in spite of the fact that the variance of probability estimates is low.

Again, to check the consistency of these results on other problems, Table 5.5 gives average results over the seven datasets obtained for 4 different levels of tree complexity. The first line of the Table corresponds to decision stumps, i.e. trees with one single test node (root split). The second and third lines correspond to the case where the complexity is fixed for each problem to respectively 25% and 50% of the average fully grown tree size.[8] The last line of Table 5.5 reports the result with fully grown trees already given in Table 5.4 (first line). The first number in each cell provides the average percentage of error due to variance and the number between parentheses provides the percentage of variance with respect to the variance of fully grown trees.

The analysis depends on the chosen variance measure. For Tibshirani's variance, simple decision stumps (with one node) already capture 40% of the variance of full trees (which in

[8]$\overline{C(\mathcal{T})}$ was determined for each problem by building 50 fully grown trees using the 50 randomly selected learning samples used in our bootstrap estimates.

Table 5.5: Percentage of error due to variance

Nb tests	$\mathrm{var}_T(\%)$	$\mathrm{var}_{KW}(\%)$	$\mathrm{var}_{\hat{P}}(\%)$	$\sigma_C(\%)$
1 (stump)	15 (40)	40 (145)	11 (21)	> 14
$0.25 \cdot \overline{C(\mathcal{T})}$	40 (83)	54 (101)	37 (54)	> 18
$0.5 \cdot \overline{C(\mathcal{T})}$	49 (93)	58 (98)	49 (70)	> 18
Full	54 (100)	61 (100)	61 (100)	> 17

average and in the mean over the seven datasets contain 44 test nodes), even if variance in this case is responsible for only 15% of the error. This number raises to respectively 83% and 93% for 25% and 50% complexity. This shows that variance appears very quickly during tree growing and is not restricted to terminal nodes. The effect is similar although less important on $\mathrm{var}_{\hat{P}}$. With var_{KW}, we also get in average the counter-intuitive result that variance actually increases when trees are smaller even if its global effect on error is slightly reduced.

5.4.4 Discussion

The conclusion of these experiments is that prediction variance in decision trees is high and it is high even when trees are small and even if learning sets are large. Another important conclusion is that, contrary to regression problem, variance may decrease when trees are more developed. This observation highlights the very different interaction between bias and variance in classification and in regression. In the next section, we will try to separate different variance sources and assess their impact on global variance.

5.5 Variance sources in decision tree induction

Variance appears in a learning algorithm, when it comes to estimating some values or to take some decisions from the learning set. In decision trees, such decisions are numerous:

- During node splitting:
  - The choice of the attribute used to split a node is dependent on the learning set.
  - If this attribute is numerical, we have also to induce a discretization threshold from the local learning set. The choice of this threshold is certainly subject to variance.

  The variability of the chosen test at a tree node is expected to increase at deeper nodes of the tree. Indeed, because of the recursive partitioning, the learning set size inevitably decreases when going down into the tree. Also, the variability at a node causes some variability of the subsets of objects which are propagated to the left and right successors of this node. Hence, the variance at shallow nodes of the tree can be further amplified at deeper nodes.

- The exact structure of the tree (the number of nodes and the depth of the branches) is determined from the learning set by means of the stop splitting criterion. Some variability may thus result from different learning set choices.

- Probability estimates at leaf nodes are based on frequency counts in the local learning set. Since learning sets at terminal nodes are often small, the probability estimates corresponding to a particular location in the input space are also certainly subject to high variance (specially, when the terminal sets are pure by construction).

Empirical evaluation

The effect on accuracy of each potential variance source is very difficult to evaluate since there is a strong interaction between them and this interaction is quite complex. However to give an idea of the relative importance of each source, we propose to carry out the following experiment:

Table 5.6: Source of variance in decision tree induction (omib problem, fully grown trees)

Variant	Mean error	compl	bias_T	var_T	var_{KW}	$\text{var}_{\hat{P}}$
Normal	0.1194	74	0.0480	0.0714(100)	0.0772(100)	0.0772(100)
Att. fixed	0.1071	88	0.0445	0.0626(88)	0.0688(89)	0.0688(89)
All fixed	0.0932	137	0.0768	0.0165(23)	0.0380(49)	0.0304(39)

Table 5.7: Source of variance in decision tree induction (omib problem, 10 test node trees)

Variant	Mean error	compl	bias_T	var_T	var_{KW}	$\text{var}_{\hat{P}}$
Normal	0.1508	21	0.0763	0.0746(100)	0.0909(100)	0.0489(100)
Att. fixed	0.1469	21	0.0822	0.0646(87)	0.0871(96)	0.0464(95)
All fixed	0.1330	21	0.1268	0.0063(8)	0.0302(33)	0.0032(7)

- First, we compute the overall variance of the growing algorithm using learning sets of size N.

- Then, we repeat the same experiment but this time by fixing the attribute choice at each node of the tree. To this end, we build a tree from a very large sample of size $M >> N$ (for example, from all the available data) and we modify the growing algorithm so as to use at each node the attribute detected during the induction of this big tree. Note that, as N is much smaller than M, the trees generated for the estimation of variance will be much smaller than the big reference tree, and thus the attribute choice is well defined in any case. In this simulation, learning essentially amounts at finding discretization thresholds and estimating conditional class probabilities at terminal nodes. Hence, the resulting variance will be imputed to these two parts of the learning algorithm.

- The last variant consists in fixing both attribute choice and discretization threshold. To this end, again, a big tree is built and all its internal choices are used to guide the growing procedure. The learning set is only used to choose when to stop the induction of the tree and to estimate class probabilities at the leaves. The variance estimates emerging from this experiment are then inputed to the class-probability estimation part of the algorithm.

Table 5.6 shows the results obtained on the omib problem following this experimental protocol with $N = 500$ and $M = 16000$. Results are reported in terms of mean error rate, mean complexity of the tree, bias_T, var_T, var_{KW}, and variance of class probability estimates, $\text{var}_{\hat{P}}$. The number in parenthesis beside each variance level is the variance expressed as the percentage of variance with respect to that of the standard algorithm. So for example, by fixing the attribute choice, Tibshirani's variance is reduced by 12%. If thresholds are furthermore fixed, there only remains 23% of the initial variance. So Tibshirani's variance can be roughly decomposed into three parts: 23% which comes from probability estimates and stop splitting, 65% which comes from discretization of numerical attributes and 12% which comes from attribute selection at test nodes. This decomposition becomes 49%, 40% and 11% for var_{KW} and 39%,50%, and 11% for $\text{var}_{\hat{P}}$. Clearly, the largest part of the variance (more than 55%) comes from the discretization procedure.

However, by fixing successively the attribute choices and the discretization thresholds, we build more and more complex trees (see the third column of Table 5.6) and this lack of balance in terms of complexity between the three variants could possibly hide some variability. So, it seems more fair to compare the three variants while controlling the complexity separately. To this end, the same experiment was repeated, while fixing the complexity of the trees to ten test nodes (using the best first strategy). Results on the omib problem are reported in Table 5.7. This time, the variance which remains when everything is fixed is much lower in each case (8% of var_T for example) and thus even more variance has to be attributed to discretization.

Appendices report the detailed results obtained from similar experiments carried out on the other problems. Overall, these results are very consistent with those observed here on the omib problem. They are summarized in Table 5.8 in terms of the decomposition of each variance

Table 5.8: Summary of variance sources in several problems

	$\mathrm{var}_T(\%)$	$\mathrm{var}_{KW}(\%)$	$\mathrm{var}_{\hat{P}}(\%)$
Full trees			
Attribute choice	21	12	12
Discretization	58	45	60
Proba estimate	21	43	28
Fixed complexity (10 test nodes)			
Attribute choice	34	18	27
Discretization	56	51	64
Proba estimate	10	31	9

term into the three sources. From this table, it appears that in the mean over all problems more than half of the variance is due to discretization, whatever the particular measure of variance and whatever the complexity of the trees. The decomposition of the remaining variance is however strongly dependent on the used variance measure. When the trees are fully developed, the variance due to probability estimates dominates that due to attribute selection in var_T and $\mathrm{var}_{\hat{P}}$. However, when the complexity is limited, the variance due to attribute choice is greater than that of probability estimates. In var_{KW} on the other hand, the variance due to probability estimates is always the greatest.

Although limited, our experiments in this section definitely show that the greatest part of decision tree variance comes from the node splitting algorithm (attribute selection and discretization of numerical attributes) while the remaining part is due to recursive partitioning and probability estimates at leaf nodes. Because the variance of node splitting is especially large, we will study it in further detail and from a complementary point of view in the next section.

5.6 Parameter variance: node splitting

Choices during node splitting are thus responsible for most of the variance in decision tree induction. An interesting question is how the prediction variance relates to the variance of the parameters of the decision tree which is induced. Does high prediction variance mean high parameter variance ? Several authors (e.g. [DK95] or [Fri96]) have claimed that the parameters chosen during tree induction are very unstable. What we propose here is to study quantitatively how large this variance is. This issue is important since one of the main advantage of decision trees is their interpretability which will be put into question if the variance of the parameters appears to be also high. As they are responsible for most of the prediction variance, we will focus here on the parameters that define tests: discretization thresholds for numerical attributes and the attributes which are selected to split the nodes.

5.6.1 Discretization threshold

A large amount of prediction variance has been imputed from our experiments to the discretization of numerical attributes. In this section, we propose to study how variable are the thresholds on numerical attributes which are selected at decision tree nodes. Several experiments aim at showing and analyzing this variance in depth.

Score curves

According to the algorithm of Table 5.1, the threshold which is locally selected is the one which realizes the maximum score. Figure 5.4 represents the relationship between Score_W and the discretization threshold for one particular attribute of the Omib problem. Each curve corresponds to a different learning sample, left for learning sets of size 100 and right for larger learning sets of size 1000. The histograms beneath the curves correspond to the sampling

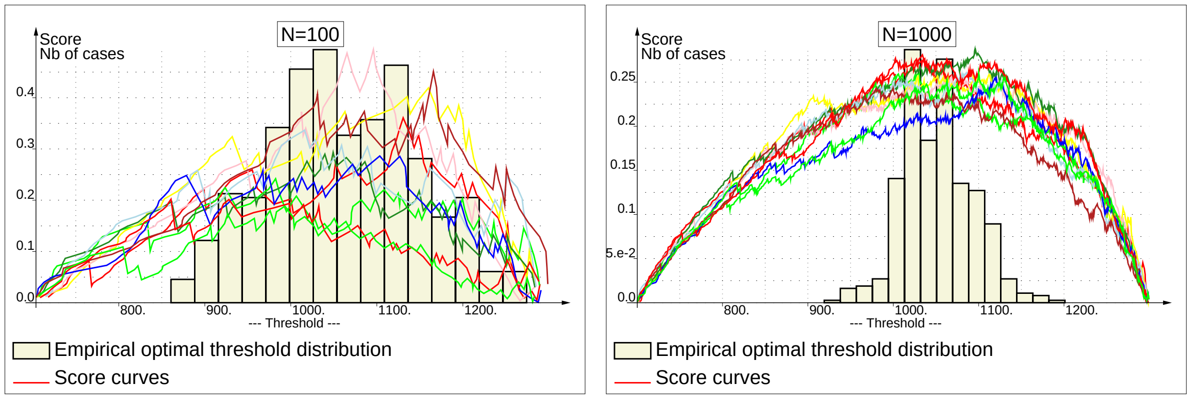


Figure 5.4: 10 score curves and empirical optimal threshold distribution for learning sets of size 100 (left) and 1000 (right)

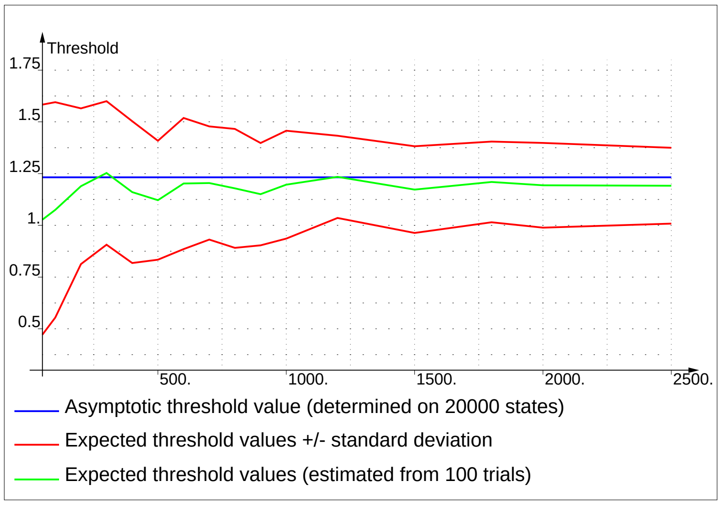


Figure 5.5: Expected threshold values and standard deviation for increasing learning set sizes on the Gaussian problem

distribution of the global maxima of these curves (i.e. the threshold selected by the classical discretization method). The very chaotic nature of these curves (especially for small learning set sizes) is responsible for the flatness of the histograms. One observes that even for large sample sizes (right hand curves), the variance of the "optimal" threshold determined by the classical method remains rather high (in comparison with the range of variation of the variable itself).

Influence of the learning sample size

To evaluate quantitatively this variance with respect to the learning set size, the following experiment is carried out. 100 sets $ls_1, \ldots, ls_{100}$ of size N are drawn from the Gaussian dataset for sample sizes growing from $N = 50$ (typical for deep nodes) to $N = 2500$ (shallow nodes). For each ls_i, the threshold maximizing Score_W is computed. The graph of Figure 5.5 plots the averages ($\pm$ standard deviation) of these 100 numbers as a function of N. The horizontal line is the value of the optimal threshold determined from the whole dataset which is used as a reference. This graph clearly highlights how slowly threshold variance decreases with sample size.

The first line of Table 5.9 shows, on the omib problem and for the attribute which yields the best score on the pool sample, the numerical value of the standard deviation of the threshold for three learning set sizes as well as the average difference between this threshold and the

Table 5.9: Discretization variance for the Omib problem, $A_{th}^{\infty} = 1060$, $\sigma_A = 170$

score	$N = 50$		$N = 500$		$N = 2000$	
	σ_{Th} (%)	Thres. bias	σ_{Th} (%)	Thres. bias	σ_{Th} (%)	Thres. bias
Score$_W$	96.5026(56)	6.2134	47.3234(28)	-7.0138	35.8138(21)	-3.7539
Score$_{\text{med}}$	41.6863(24)	-52.6901	12.9687(8)	-60.6760	6.9955(4)	-60.3774

Table 5.10: Mean relative standard deviation of discretization threshold (in %)

Score	$N = 50$	$N = 500$	$N = 2000$
Score$_W$	49,71	24,57	14,14
Score$_{\text{med}}$	20,86	6,43	2,86

asymptotic threshold, A_{th}^{∞}, determined from the whole dataset (the threshold bias). The number in parenthesis beside the standard deviation is the relative standard deviation (in percent) with respect to the standard deviation of the discretized attribute in the dataset. For comparison, the second line of the same table reports the results obtained by discretizing at the median of the attribute value in the local learning sample. This amounts at using a score, Score$_{\text{med}}$, which attributes the maximal score to the median in the local subset, independently of the class distribution. With our score, the standard deviation of the threshold for small sample sizes is more than half of the standard deviation of the attribute in the dataset and even for the largest sample size, it is still about 20% of the attribute standard deviation. With the median on the other hand, the threshold variance is very small and vanishes almost completely when the learning set size increases. The high bias with respect to the classical information theoretic score translates the fact that they search for different thresholds.

Table 5.10 reports average results on all datasets for the same sample sizes. For each problem, the considered attribute is the one which realizes the best score on the whole pool set. Tabulated values are the mean relative standard deviation of the threshold (in percent, with respect to the standard deviation of the attribute). One can observed that average results over our 7 problems are again similar to those obtained on the omib problem: for small sample size, almost half of the standard deviation of the attribute appears on the discretization threshold and even for large sample size, the remaining variance is not negligible; on the other hand, selecting the discretization threshold in an unsupervised fashion by the sample median gives much more stable results.

Influence of the depth in the tree

In the previous section, it has been claimed that threshold variance should be amplified when going down into the tree. The following experiment tries to verify this statement. To concentrate on discretization variance only, a big tree is built using Score$_W$ which fixes all attribute choices during the induction. Then, thresholds of this tree are determined from 50 different learning sets. The mean standard deviation of the discretization threshold normalized by the standard deviation of the corresponding attribute is averaged among all test nodes at a given level of the tree. These average values are plotted as a function of the tree level in Figure 5.6 with Score$_W$ and with the median discretization. Only test nodes which have been developed in more than 5 trees among the 50 ones are taken into account for building this curve. With Score$_W$, the standard deviation first increases and reaches at the third level a very important value of 70% of the standard deviation of the attribute. However, after the third level, it starts decreasing and stabilizes at about 50% of the global standard deviation. On the other hand, the median is monotonically increasing for deeper and deeper levels. The decrease of variance observed with Score$_W$, although at first sight surprising, can be explained as follows: in local subsets attached to deep nodes in the tree, the ranges of variation of the attributes which have already been chosen at higher nodes may become narrower and hence, the standard deviation of the discretization threshold may also decrease. Note that, qualitative behaviors are similar on other datasets.

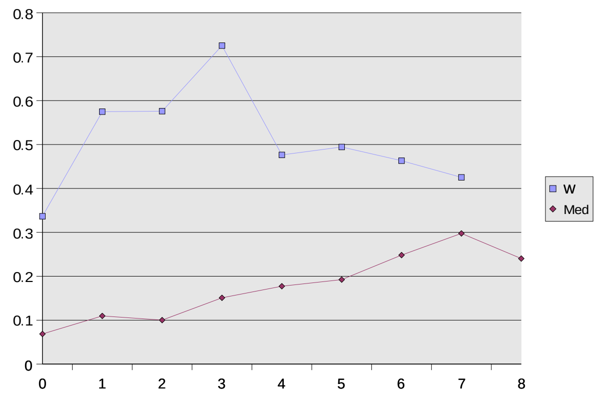


Figure 5.6: Relative standard deviation versus tree depth on the omib problem

Table 5.11: Impact of score measure on discretization variance (learning sample size of 500).

SCORE	OMIB $A_{th}^{\infty} = 1060, \sigma_A = 170$		PENDIGITS $A_{th}^{\infty} = 24.5, \sigma_A = 35.8$	
	σ_{Th} (%)	Thres. bias	σ_{Th} (%)	Thres. bias
Score$_W$	47.3234(28)	-7.0138	5.4438(15)	-1.2950
Score$_{\text{info}}$	43.1451(25)	-18.7450	4.5207(13)	-2.2250
Score$_Q$	93.8516(55)	78.9720	14.6218(41)	23.2850
Score$_{\text{med}}$	12.9687(8)	-60.6760	1.9878(6)	-15.5350
Score$_{KS}$	26.3415(15)	-16.1272	20.9179(58)	15.6000
Score$_{Err}$	27.4454(16)	56.1267	11.3976(32)	-12.5350

Influence of the score measure

In section 5.2.3, we have reviewed several score measures proposed in the literature to evaluate candidate thresholds. To see how far the particular score measure used in a method is responsible for the variance of the threshold, we have carried out the same experiments with the four additional score measures introduced in section 5.2.3: Score$_{Info}$, Score$_Q$, Score$_{KS}$, and, Score$_{Err}$. The impact of each score measure on threshold variance is different from one data set to another. Table 5.11 compares all score measures left on the omib problem and right on the pendigits problem, for a fixed learning sample size of 500 in both cases. The column labeled σ_{th} provides the standard deviation of the threshold and its relative value with respect to the standard deviation of the attribute (in percent) and threshold bias is the average difference between the learned threshold and the asymptotic threshold determined by the classical method with Score$_W$. On both problems, it is clear that all measures are not equal in terms of threshold variance. The high bias of every score with respect to Wehenkel's score also confirms that they search for different thresholds. On the omib problem, Score$_{KS}$ and Score$_{Err}$ are very close to each other and better than entropy based measures. Among these latter, Quinlan's normalization is the worst and Wehenkel's score is slightly inferior to the raw information gain. On the pendigits problem, Score$_{KS}$ appears to be inferior to other score measures in terms of threshold variance. Also, while Score$_{Err}$ is competitive with other scores on the omib problem, its variance is higher on this new problem especially for larger sample sizes. Among entropy criteria, Quinlan's normalization gain is by far the worst. It is nice to see that among the entropy based measures, information quantity which from a purely theoretical point of view is probably the most founded choice, also has the best behavior in terms of variance. Note that the normalized measures have some advantages (not relevant in the present context) in the context of induction of decision trees of variable branching factors (see e.g. [Weh96], for a discussion of these considerations).

To check the consistency of these conclusions with the tree depth, variance versus level curves like the one of Figure 5.6 are plotted for the different score measures in Figure 5.7, left

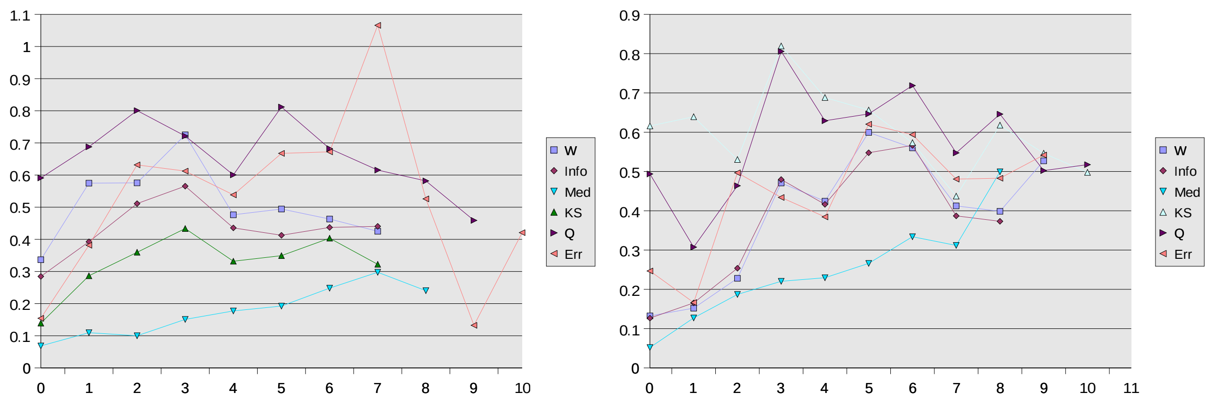


Figure 5.7: Relative standard deviation versus tree depth on the omib (left) and pendigits (right) problems with all score measures

Table 5.12: Influence of the number of classes on score performance

score	All	$m \leq 3$	$m > 3$
Score_W	24,57	30,75	16,33
$\text{Score}_{\text{info}}$	21,29	25,5	15,67
Score_Q	69	67,25	71,33
$\text{Score}_{\text{med}}$	6,43	6,5	6,33
Score_{KS}	22,86	15,5	32,67
Score_{Err}	21,14	16,5	27,33

on the omib problem and right on the two-norm problem. The guiding structure is the same for all score measures and is determined with Score_W. These curves partially confirm the results of table 5.11. On the omib problem, Kolmogorov-Smirnoff distance gives the lowest threshold variance at each level and Score_Q is the worst. However, Score_{Err} which is good at the top level of the tree increases the variance at deeper levels. On the pendigits problem, Score_W and $\text{Score}_{\text{info}}$ give more stable threshold than other score measures.

A more in depth analysis shows that, among our seven datasets, the behavior on the omib problem is observed in all problems where the number of classes is no more than 3 and the behavior of pendigits is observed when the number of classes is greater than 3. This dependence on the number of classes is highlighted in Table 5.12 which compares average relative standard deviations for $N = 500$, left on all the seven problems, center, on the four problems corresponding to no more than 3 classes and right, on the third remaining problems with more than 3 classes. In average over the seven problems, all score measures are very close to each other except for Quinlan's score which is significantly worse and median, which somewhat trivially is the most stable one. On the other hand, for values of $m \leq 3$ (first 4 problems of Table 5.2), the Kolmogorov-Smirnoff distance outperforms the entropy based measures by about a factor 2. On the contrary, for large numbers of classes ($m > 3$), Score_{KS} yields an increase in the relative standard deviation by a factor 2 with respect to entropy based measures.

To give a finer analysis, several score curves are plotted in Figures 5.8 and 5.9 respectively on the omib ($m = 2$) and dig44 ($m = 10$) problems, left for Score_Q and right for Score_{KS}. These curves should be compared to the left one of Figure 5.4. First, the normalization introduced by Quinlan increases the score at the boundaries of the attribute range and reduces it in the middle of the interval. This actually flattens the top of the curve and hence the position of the global maximum is more instable. On the other hand, Kolmogorov-Smirnoff distance gives smoother curves than the two other scores and hence this explains the reduction of variance on the omib problem. Nevertheless, even if smoother, the same score curve for the dig44 problem actually presents several local maxima which may be selected according to the particular learning sets and thus increases the variance. The presence of the two local maxima comes from the non natural handling of several classes by the Kolmogorov-Smirnof score. All possible divisions of

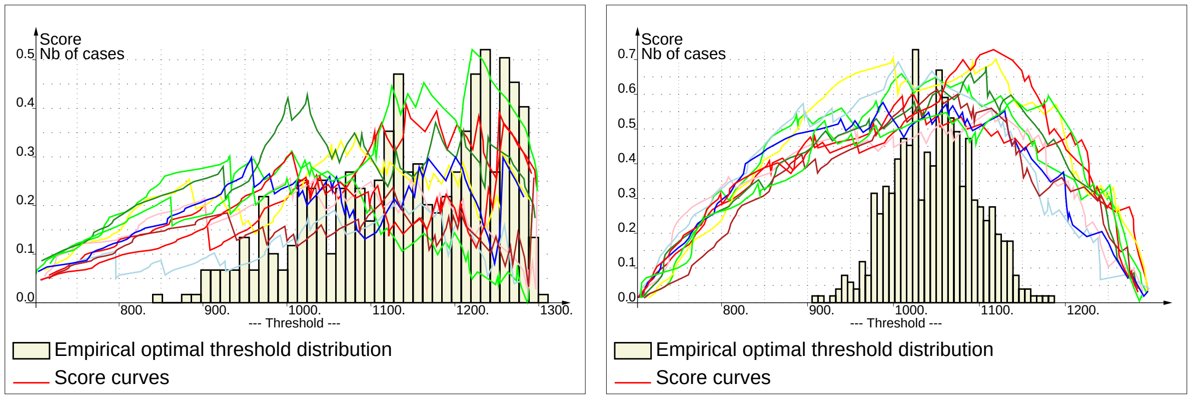


Figure 5.8: Score curves for samples of size 100 on the omib problem. left with Score_Q, right with Score_{KS}

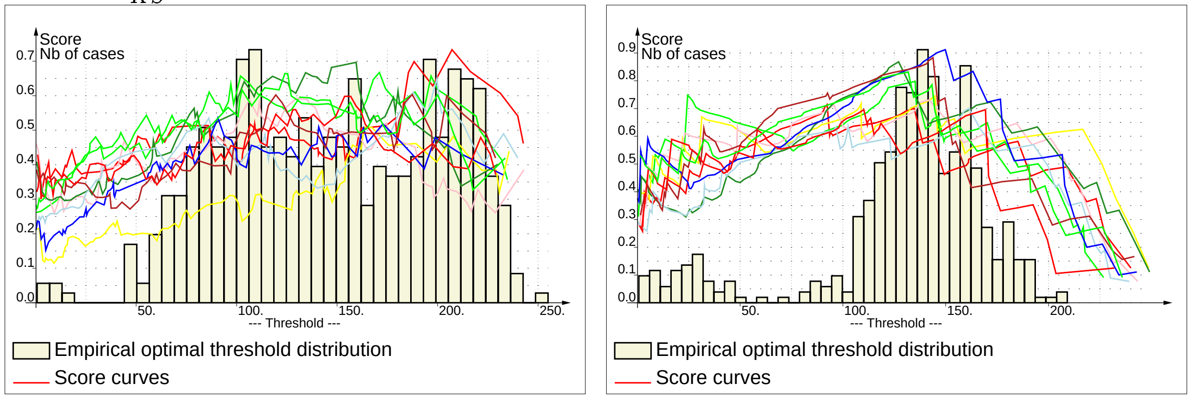


Figure 5.9: Score curves for samples of size 100 on the dig44 problem. left with Score_Q, right with Score_{KS}

one class against the other are considered and are responsible each one for a different maximum.

Discussion

In conclusion, irrespectively of particular score measure used, the discretization procedure produces thresholds with a very high variance, even for large learning sample sizes. This variance increases at deeper levels in the tree and is responsible of more than 50% of the high prediction variance of decision trees. Clearly, the high discretization variance leads also to a reduction of interpretability of decision trees, since these thresholds form part of the knowledge that can be extracted from a database by decision tree induction.

Although all the score measures which have been studied here produce high variance, they are not equal according to this criterion. This conclusion contrasts strongly with previous studies published in the decision tree literature about the impact of score measures. These studies concluded generally that the choice of a particular measure is not important, because these studies merely focused on overall impact on accuracy. This latter statement will be confirmed on our problems in Section 5.6.3 but the present investigation shows however that in terms of variance of the thresholds, some measures are much better or much worse than others. In particular, the gain-ratio proposed by Quinlan (normalization of entropy-based measure by the split-entropy) is not a good idea since we have seen that it increases the variance of the model. On the other hand, in problems with few classes, it seems interesting to prefer KS distance to other score measures. If we want a robust measure in all cases, Shannon information seems to be the best average choice, which we are pleased to say !

5.6.2 Attribute choice

The uncertainty about the attribute which is selected at a particular tree node is responsible for about 35% of the variance. To evaluate this uncertainty at a node, one interesting measure is the entropy of the choice. If the attribute $A_i (i = 1, \ldots, n)$ is selected by the score maximization procedure with the relative frequency f_i at a node $\mathcal{N}$, then this entropy is given by:

$$H(\underline{A}, \mathcal{N}) = -\sum_{i=1}^{n} f_i \log_n(f_i), \tag{5.17}$$

which is null when the same attribute is selected whatever the learning set and (because of $\log_n$ equals to one when all the attributes appear at this node with the same frequency, i.e. $f_i = \frac{1}{n}$. To evaluate this uncertainty at different levels of decision trees, we carried out the following experiment: from 50 learning sets randomly drawn from the pool set, we build 50 decision trees according to the usual growing algorithm. Then, we compute at each position of the tree, the frequency f_i estimated from these 50 trees. If some trees are not developed until this position, they simply do not participate in the counting. Only test nodes which appear in at least 5 trees are taken into account. Entropy values at the same level are averaged to give a single value for each depth. Left graphs of Figure 5.10 show three different behaviors respectively on the omib, two-norm, and pendigits problems for the different score measures. Ideally, the curve should be a horizontal line at 0.

Like for discretization threshold standard deviation, we do not observe a monotonic behavior with increasing depth level. On the omib problem, the attribute tested at the top node (depth 0) is always the same whatever the score measure (entropy is close to 0). Then, the entropy increases and becomes maximum at the second level before it starts decreasing. pendigits presents similar curves. On the other hand, on the two-norm problem, the entropy is already high at shallow nodes and, except for Quinlan's score, it also decreases at lower levels. The non monotone behavior may be explained by two phenomena. First, when it is necessary to combine several attributes to provide satisfying classification and when the impact of these attributes individually on classification are equivalent, the particular order in which these attributes appear in a branch is very dependent on the learning set and hence entropy is high. Such a variance is thus caused by the hill-climbing algorithm which is not adapted to this kind of problems. Second, when some attributes have been intensively used at top levels, they are not likely to bring new information at deeper levels (their values are more or less constant in local subsets). So, their f_i will be close to 0 and the sum (5.17) is lower even if the choice among remaining attributes is very uncertain. This may explain the reduction of the entropy at deeper levels.

One problem of the measure we have defined so far is that the entropy at a node is not conditional to the choice at the previous node and hence even if the total number of structures considered by the growing algorithm is very small, the entropy of the choice at a node independently of its parents and successors can be high. Another way to measure this variance consists in computing the most frequent structure, i.e. the structure which appears most often (as it) in the trees induced from the learning set. This structure is defined recursively from a set of trees by the following procedure which starts at the top node:

- if the set of trees contains only one tree, then stop;
- else, search for the attribute A_i which appears the most frequently at the current location (Ties are broken arbitrarily). Build a new node, attach the attribute A_i and its frequency of appearance f_i to this node.
- remove from the set of trees, the trees which do not use the most frequent attribute at the current location.
- proceeds recursively to build the left and right successors of this node from the new set of trees.

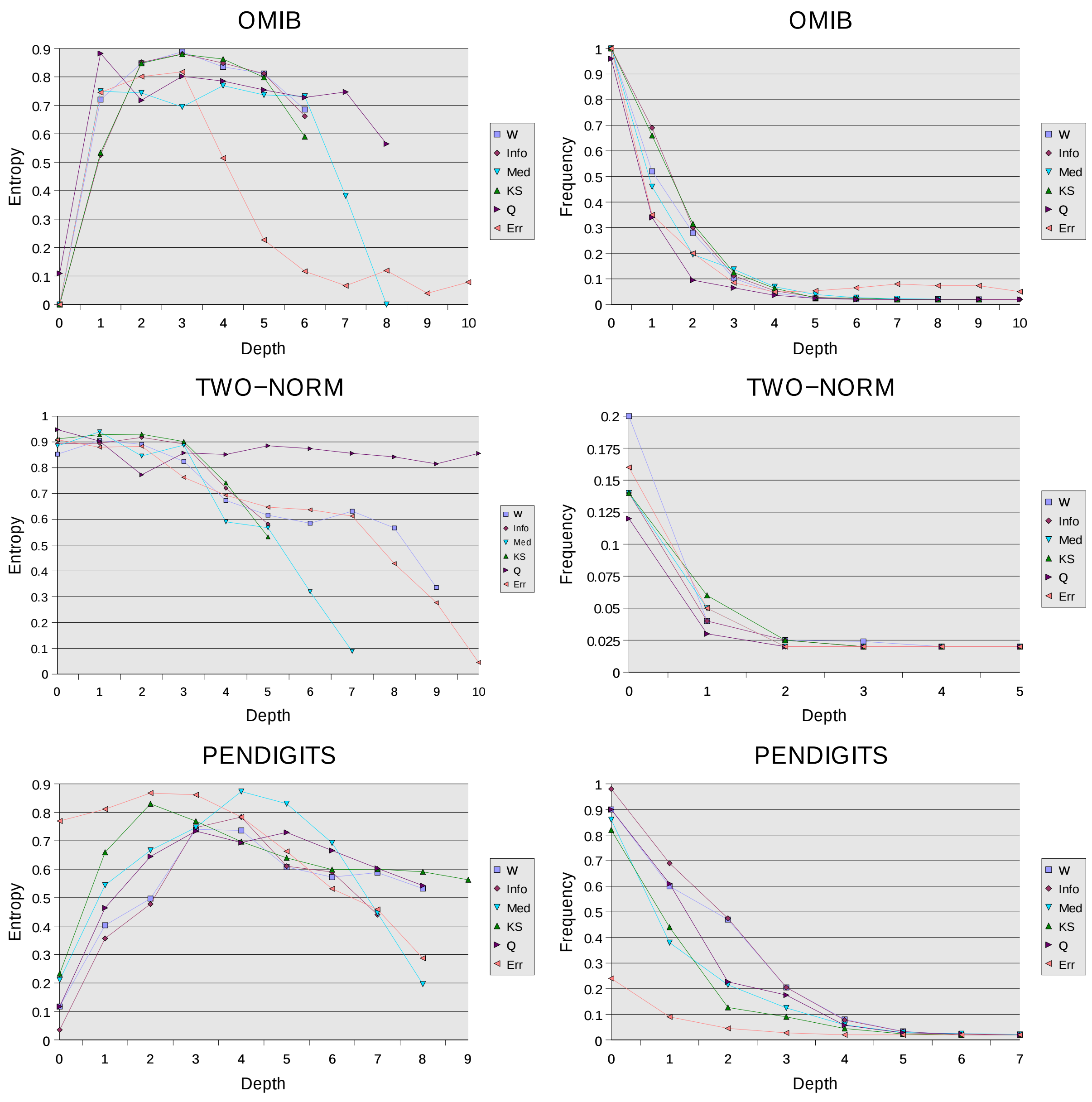


Figure 5.10: Attribute choice variance with increasing tree depth, on the omib (top), two-norm (center) and pendigits (bottom) problems.

Table 5.13: Global effect of score measure (average on all datasets)

	Mean error	compl	bias_T	var_T	var_{KW}	$\text{var}_{\hat{P}}$
Score_W	0,2141	87	0,0935	0,1206	0,1316	0,0767
$\text{Score}_{\text{info}}$	0,2126	85	0,0973	0,1152	0,1301	0,0750
Score_Q	0,2206	109	0,1056	0,1150	0,1329	0,0764
$\text{Score}_{\text{med}}$	0,2315	178	0,1056	0,1259	0,1417	0,0769
Score_{KS}	0,2218	98	0,0968	0,1250	0,1368	0,0759
Score_{Err}	0,2415	276	0,1080	0,1335	0,1476	0,0860

Right graphs of Figure 5.10 plot the average frequencies for all nodes at the same level in the structure induced in this way. For example, a value of 0.4 for level 4 means that in average over all paths to the nodes of the fourth level, 40% of the trees choose the same attributes until one node of the fourth level in the tree. The ideal case corresponds to a horizontal line at 1 (all trees share the same structure). On Omib and Pendigits, the proportion of trees which share the top level test is quite large but this proportion decreases very quickly. Less than 50% of the trees share a second level test whatever the score measure. On the two-norm problem, even the first most frequent test is only present in 20% of the trees.

From the point of view of the score measures, there is not a marked trend like for the discretization threshold. On the omib and two-norm problems, all measures are more or less equivalent, except for quinlan's score which produces the most unstable choices. On Pendigits however, Score_{KS} and especially Score_{Err} show worse behaviors than other score measure.

In conclusion, the attribute which is selected to split a tree node is very uncertain whatever the score measure used. However, this uncertainty is very dependent on the way the attribute intervene in the Bayes model and thus is very problem dependent.

5.6.3 Global effect of the score measure on decision trees

In this section, we have analyzed the impact of the score measure on the stability of the model. One conclusion of this study is that, depending on the problem, all measures are not equivalent according to this criterion. Large experimental studies in the literature however have shown that the particular choice of a score measure rarely seems to affect the final accuracy of the trees ([Min89b, MS94]). To be complete, we thus computed for each score measure mean error rate, complexity and various bias and variance terms for fully grown trees. We have included in this comparison the median discretization. In this variant, the attribute which is chosen to split the node is the one which maximizes, together with the median, Score_W. Table 5.13 reports average results on all datasets.

These results do not invalidate previous statements in the literature. Indeed, mean error rates are very close to each other. Surprisingly, even discretizing at the median gives competitive results with other score measures, even though the threshold is not related to the classification. Concerning bias and variance terms, it is difficult to draw any conclusions from Table 5.13, all values being very close. While we would expect that the median discretization gives the smallest variance (and highest bias), actually this method presents comparable variance and bias with respect to other scores. Nevertheless, this variance can be explained by the increase of complexity which counterbalances the effect of the stabilization of thresholds at shallow nodes. In terms of complexity of the resulting trees, Score_W and $\text{Score}_{\text{info}}$ are better than other score measures. Score_{Err}, because of its pathology, produces the most complex trees.

Tables 5.14 and 5.15 further provide average results for the two different cases underlined previously, which corresponds to small and large numbers of classes. Kolmogorov-Smirnoff's distance and median discretization which are slightly better than other score measure when the number of classes is low are significantly worse when the number of classes is greater. Entropy based scores seem to be more robust with respect to the number of classes.

Table 5.14: Global effect of score measure ($m \leq 3$)

	Mean error	compl	bias_T	var_T	var_{KW}	$\text{var}_{\hat{P}}$
Score_W	0,2026	57	0,0867	0,1159	0,1235	0,1089
$\text{Score}_{\text{info}}$	0,1992	54	0,0919	0,1073	0,1201	0,1059
Score_Q	0,2068	78	0,1033	0,1034	0,1230	0,1079
$\text{Score}_{\text{med}}$	0,2058	121	0,0988	0,1070	0,1222	0,1051
Score_{KS}	0,1987	59	0,0926	0,1061	0,1192	0,1047
Score_{Err}	0,2322	329	0,1076	0,1246	0,1380	0,1222

Table 5.15: Global effect of score measure ($m > 3$)

	Mean error	compl	bias_T	var_T	var_{KW}	$\text{var}_{\hat{P}}$
Score_W	0,2295	130	0,1027	0,1268	0,1425	0,0337
$\text{Score}_{\text{info}}$	0,2304	127	0,1045	0,1259	0,1434	0,0339
Score_Q	0,2391	151	0,1087	0,1304	0,1462	0,0345
$\text{Score}_{\text{med}}$	0,2657	254	0,1146	0,1510	0,1676	0,0393
Score_{KS}	0,2526	152	0,1024	0,1501	0,1603	0,0376
Score_{Err}	0,2539	204	0,1086	0,1454	0,1604	0,0378

5.7 Conclusion

The main conclusion of the experiments of this chapter is that decision tree variance is high, and this is true whatever the method we adopt to effectively evaluate this variance.

First, prediction variance is high. It is responsible for more than 50% of the error on our problems. Variance appears very quickly during the development of the tree and remains the dominant source of error even when the learning set size increases. This variance is mainly due to the fact that the algorithm must choose discretization thresholds for numerical attributes.

A second expression of variance is that the parameters which define the tests are very unstable. The discretization threshold which is chosen during node splitting presents a high variance. The attribute which is selected is also very unstable. This instability is probably due to the hill-climbing algorithm which is not fully adapted to problems where several attributes have to be combined to give a reliable classification rule.

Although we could have pursued our study in several directions (e.g. to study the variance of probability estimates or of the tree complexity), we do not believe this would change our conclusions.

The consequence of this high variance are two-fold. First, decision trees are suboptimal. Their accuracy could be further improved if we could reduce their variance. Second, their interpretability is questionable. We can not really trust the choice of the attributes and thresholds. Therefore, three complementary directions of improvements will be explored in the following chapters. First, chapter 6 is dedicated to techniques that try to improve the interpretability of decision trees. From the experiments of this chapter, the main part of the algorithm which needs improvement in this prospect is the local discretization algorithm. Several techniques will thus be proposed and validated that stabilize discretization thresholds. On the other hand, Chapters 7 and 8 will be devoted to techniques that mainly try to improve the accuracy of decision trees by reducing their prediction variance. The high variance of discretization thresholds will inspire us two different variance reduction techniques. The first one simulates the discretization variance by really selecting a threshold at random and then reduces this variance by averaging several decision trees grown in this random fashion. The second method handles the problem of discretization variance with one model by softly propagating an object along the different branches of a decision tree, and by weighting the resulting predictions corresponding to the different terminal nodes according to some probability distribution, recursively computed by the propagation scheme.

Chapter 6

Reducing the variance of decision trees

In this chapter, we consider two variance reduction techniques which do not improve or change the representation bias of decision trees. The first set of experiments concerns the classical way of optimizing the bias/variance tradeoff through the post-pruning algorithm which operates on the tree complexity to reduce the variance while increasing bias. The second group of experiments concern several new methods which we propose in order to reduce the high variance of discretization thresholds during tree growing. The limitations of such techniques are discussed in our conclusions.

6.1 Introduction

In the previous chapter, several experiments highlighted the high variance of the decision tree growing algorithm. In this chapter and in the next two, we will experiment with several algorithms which aim at decreasing this variance. Variance reduction methods can be divided into two categories: methods which do not change the representation bias (or hypothesis space) of decision trees and method which do, i.e. which produce models which can not be represented by simple standard decision trees. This chapter is devoted to methods of the first kind. To save the hypothesis space of decision tree is especially interesting since it saves the interpretability of the models which are induced. However, since the possibility to represent a model by a simple decision tree is a very strong constraint, we expect the methods of the first kind to produce less improvement in terms of accuracy than methods of the second kind, which have the freedom to change this representation bias.

Two approaches will be considered in this chapter: first a classical approach, namely pruning which operates only on the tree complexity to reduce the variance; second, a more original approach which considers test stabilization techniques, especially by reducing the variance of discretization thresholds, but also by acting on the attribute selection.

6.2 Pruning

From previous experiments, we already know that bias is a decreasing function of the model complexity and variance (at least the variance of probability estimates) on the other hand is an increasing function of complexity. There thus exists a tradeoff between these two sources of error. In the context of decision trees, this tradeoff is regulated by the number of terminal nodes. The more the tree is developed, the lower is the bias but the higher is the variance. Therefore, letting the tree grow until all terminal nodes are pure, as we did in some of the experiments of the preceding chapter, is often suboptimal. On the other hand, fixing the complexity a priori would be even worse, since the optimal complexity is actually problem and sample size dependent, and it is impossible to guess this value a priori. Pruning aims at adjusting the complexity in some optimal fashion (to be defined) in a problem and sample dependent way.

In this section, we briefly study pruning algorithms which have been proposed to control the complexity of decision trees. These algorithms are important for two reasons. First, they may improve accuracy by reducing prediction variance and second, they improve the readability of the trees by reducing their complexity, and by removing those parts which are meaningless. We start by describing the particular pruning algorithm(s) we use (taken from [Weh98, Weh93]) and then carry out experiments on our seven classification problems with this method.

6.2.1 Description of pruning algorithms

There are two alternative ways to control the complexity during decision tree induction:

- Pre-pruning or stop-splitting: which is based on a sensible stop-splitting criterion designed so as to measure the relevance of a test and to stop developing a branch as soon as the best test found to develop the branch is not good enough according to this criterion.

- Post-pruning: which consists in first developing a fully grown tree, then generating from this tree a sequence of nested trees of decreasing complexity and finally selecting one of these trees so as to minimize the error, on the basis of an unbiased error estimator (e.g. holdout or cross-validation error estimate).

The description below provides the details of the two methods which have been used in our experiments and shows also how these approaches are related.

Pre-pruning or stop-splitting

A first way to implicitly prune a tree is to stop splitting a node if it is impossible to find a test which is considered significant enough to split the current node. Mainly two approaches have been investigated to test the significance of a test: hypothesis testing and detection of a local maximum of a tree quality measure.

If we have information about the sampling distribution of the score when the attribute is actually independent of the classification in the given subset, then a simple hypothesis testing approach gives a procedure to check the significance of a test:

- search for the test which realizes the best score,

- determine from the local sample the probability of getting a value of the score greater than this best score (under the hypothesis of independence between test outcome and output class),

- accept the independence hypothesis if this probability is larger than an a priori fixed threshold α, consider the score as "not significant" and let the node become a leaf.

In this approach, a choice of $\alpha = 1.0$ corresponds to fully developed trees and to $\alpha = 0.0$ correspond only trivial trees (with one single node). Intermediate values would lead to more or less strongly pre-pruned decision trees.

Let us consider two examples of this approach, related to the two main score measures considered in the preceding chapter (information quantity and Kolmogorov-Smirnoff distance).

The empirical distribution of the Kolmogorov-Smirnoff distance is well-known in statistics and has been used for hypothesis testing during tree induction by Rounds [Rou80]. As concerns information quantity, assuming the independence of the test with respect to the classification in the learning sample, Kvålseth ([Kvå87]) has shown that the following quantity :

$$G^2 \stackrel{\triangle}{=} 2N. \ln 2. \hat{I}_C^T(S), \tag{6.1}$$

follows a χ-square distribution with $(m-1)(p-1)$ degrees of freedom, where m is the number of classes and p is the number of test issues. Thus, on this basis, we would choose not to split a node if the probability of G^2 being greater than the observed value is larger than α. Stop-splitting based on the G^2 criterion has been proposed in [WVCRP89].

A second approach to evaluate the significance of a test is to define a global quality measure of a tree and to reject the addition of a new test if it results in a decrease of global quality. For example, in [Weh98], decision tree quality is defined by:

$$Q_\beta(\mathcal{T}, LS) = N I_C^{\mathcal{T}}(LS) - \beta C(\mathcal{T}), \tag{6.2}$$

where $I_C^{\mathcal{T}}(LS)$ is the information provided by the tree $\mathcal{T}$ about the classification estimated from the learning sample and $C(\mathcal{T})$ is the tree complexity which is defined as the number of terminal nodes of the tree minus one. An interesting property of this quality measure is its additivity in terms of the subtrees of a given tree. Indeed, the increase of quality resulting from the addition of a test T to a terminal node $\mathcal{N}$ is given by:

$$\Delta Q_\beta(\mathcal{N}, T) = N_{\mathcal{N}} I_C^T(LS(\mathcal{N})) - \beta(p-1), \tag{6.3}$$

where p is the number of outcomes of the test and $N_{\mathcal{N}}$ is the size of the subset $LS(\mathcal{N})$. The variation of quality is also the quality of the subtree with root node $\mathcal{N}$ and only one test T. Using this measure, the development of a branch is stopped at a node $\mathcal{N}$ if the best test T^* is such that $\Delta Q_\beta(\mathcal{N}, T^*) \leq 0$. In this case, the level of development depends on the value of the parameter β: to $\beta = 0$ correspond fully developed trees, and to $\beta = +\infty$ correspond trivial trees.

If the number of test outcomes is constant throughout the whole tree growing (for example if we consider only binary tests), this approach as well as the approach based on the G^2 statistic amount to comparing at each node the product $N_{\mathcal{N}} I_C^{T^*}(LS(\mathcal{N}))$ to a threshold. Indeed, for each value of α, there exists a value of β which yields exactly the same tree. Although both approaches are equivalent, fixing a priori the value of α is easier because of its statistical interpretation (typical values used in practice are in the range of $[0.0001-0.01]$, and correspond in the 2-class case to values of $\beta \in [5-20]$.)

Post-pruning

A potential drawback of stop-splitting is that it may be trapped in a local maximum of the quality of the tree. Indeed, even if the current test is not significant, it it possible that subsequent tests at deeper nodes actually improve the quality sufficiently to compensate the reduction of quality of the present test. Another drawback of the stop-splitting approach is to require the user to fix a priori the value of a meta-parameter (α or β, in the preceding discussion) which value may strongly affect the resulting tree complexity. These two problems are addressed by the post-pruning approach. In post-pruning, a tree $\mathcal{T}$ is first fully grown (i.e. without activating any special stop splitting criterion) and a posteriori some of its test nodes are pruned to become the leaves of a new tree. The following text provides an explanation of the principle behind the pruning method used in our experiments. Overall, the method is similar to so-called "cost-complexity pruning" given in the well-known reference on CART [BFOS84]. The reader not interested in these details may proceed directly to the validation in Section 6.2.2.

The pruning algorithm used in our experiments is based on the tree quality measure defined in eqn. (6.2). The algorithm has two levels of optimization: at the inner level it extracts for a given value of β a tree of optimal quality; at the outer level, it selects the optimal value of β in order to minimize the generalization error of the resulting tree.

Let us first explain how, for a given value of β, the inner level of the pruning algorithm computes a pruned version of the full tree $\mathcal{T}$ which has maximal quality (we call this a $\beta-$optimal pruning of $\mathcal{T}$). Let us denote by $\mathcal{T}(\mathcal{N})$ the subtree of the full tree with root node $\mathcal{N}$ and by $LS(\mathcal{N})$ the local subset of learning samples reaching this node. Let us further denote by

$\mathcal{T} - \mathcal{T}(\mathcal{N})$ the tree obtained from $\mathcal{T}$ by contracting the node $\mathcal{N}$ (replacing it by a terminal node). Then, due to the additivity of the quality measure we have (for any value of β)

$$Q_\beta(\mathcal{T}, LS) = Q_\beta(\mathcal{T} - \mathcal{T}(\mathcal{N}), LS) + Q_\beta(\mathcal{T}(\mathcal{N}), LS(\mathcal{N})). \tag{6.4}$$

In other words, the quality of a tree is the sum of the qualities of its component trees with respect to the samples belonging to the root nodes of these component trees. Note also that the quality of a trivial tree (composed of a single node) is equal to zero, for any value of β. So, denoting by $P_\beta(\mathcal{T}, LS)$ the $\beta-$optimal pruning of $\mathcal{T}$ on the learning sample LS, one of the following two statements must be true:

- $P_\beta(\mathcal{T}, LS)$ is trivial (reduces to the root node), in which case $Q_\beta(P_\beta(\mathcal{T}, LS), LS) = 0$.
- $P_\beta(\mathcal{T}, LS)$ is non-trivial, but all its subtrees are optimally pruned (w.r.t their subsample) versions of the corresponding subtrees of $\mathcal{T}$; in this case $Q_\beta(P_\beta(\mathcal{T}, LS), LS) \geq 0$.

It may be the case that for some particular value of β there exist more than one optimally pruned versions of β. In such a case, we would prefer the version of minimal complexity, so as to ensure continuity on the right with respect to β. Putting all these considerations together, it is easy to check that the following recursive algorithm computes $P_\beta(\mathcal{T}, LS)$:

- If $\mathcal{T}$ is trivial, then $P_\beta(\mathcal{T}, LS) = \mathcal{T}$.
- Otherwise, first call the pruning algorithm recursively on every successor of the root of $\mathcal{T}$, and let $\mathcal{T}'$ denote the result of this operation.
  - If $Q_\beta(\mathcal{T}', LS) \leq 0$, then $P_\beta(\mathcal{T}, LS) = \mathcal{T} - \mathcal{T}$ (the trivial tree).
  - Otherwise, $P_\beta(\mathcal{T}, LS) = \mathcal{T}'$.

This recursive algorithm is efficient and the optimally pruned trees have several nice properties (see [Weh93] for a detailed derivation of these properties). The most important property is that for increasing values of β the algorithm generates a *nested* sequence of trees starting with $P_0(\mathcal{T}, LS) = \mathcal{T}$ (provided that information quantities are strictly positive for any subtree of $\mathcal{T}$, which is normally the case), and ending with $P_\infty(\mathcal{T}, LS) = \mathcal{T} - \mathcal{T}$ (the trivial tree). Because the trees are nested, the total number of different non-trivial trees in such a pruning sequence is at most equal to the number of test nodes of the original tree. These nested trees correspond to a sequence of increasing critical values of β, which can be determined for a given tree and learning sample in an efficient recursive way (not explained here).

Thus, in order to determine the optimal value of β the outer level of the pruning algorithm can be implemented along the general scheme already discussed in Chapter 4:

1. consider the growing sequence $\beta_0(=0), \beta_1, \beta_2, \ldots$ of critical values of β determined for the tree $\mathcal{T}$ on the basis of the learning-sample LS;
2. then generate the sequence $\mathcal{T}_0 = \mathcal{T}, \mathcal{T}_1 = P_{\beta_1}(\mathcal{T}, LS), \mathcal{T}_2 = P_{\beta_2}(\mathcal{T}, LS), \ldots$ of (nested) $\beta-$optimally pruned trees;
3. for each $\mathcal{T}_i$ estimate its error in some unbiased way (in our experiment we use an independent sample called the pruning cross-validation sample, to this end).
4. select the tree $\mathcal{T}_*$ of minimal error estimate as the overall optimally pruned tree.

Note that we skip here some implementation details which allow us to carry out the whole procedure in very efficient way, in particular using incremental updating formulas to derive one tree in the sequence from its predecessor without restarting from scratch. With these sophistications, the resulting algorithm becomes linear in the number of test nodes of $\mathcal{T}$ and in the number of objects contained in the pruning cross-validation sample.

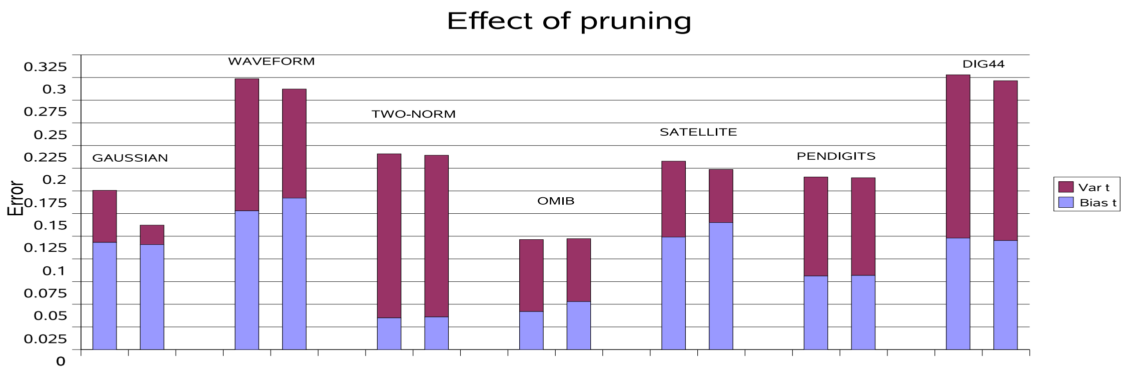


Figure 6.1: Effect of pruning on error rates. Left, without pruning, right with pruning

Table 6.1: Average effect of pruning

	Mean error	compl	bias_T	var_T	var_{KW}	$\text{var}_{\hat{P}}$
Full tree	0.2161	90	0.0966	0.1195	0.1326	0.0771
Pruned tree	0.2065	56	0.1020	0.1046	0.1201	0.0576

6.2.2 Validation of pruning

To assess the effect of pruning on bias and variance, we applied the post-pruning algorithm described in section 6.2.1 on our seven problems. In each dataset, a quite large validation set is reserved to estimate generalization error of the trees of the sequence (its size is given in Table 5.2 pp. 83 under the name VS) and hence to select an optimal value for β. Notice that in practice, such a validation set is often unavailable and either the growing set must be split or k-fold cross-validation used. However, in this section, the focus being more on what can be gained by reducing the complexity of tree, we venture to use such validation set. Figure 6.1 shows Tibshirani's bias/variance decomposition for all datasets without and with pruning. Table 6.1 reports average results in terms of percentage of error rates, complexity, bias, and variances with respect to fully grown trees.

Although the reduction of complexity is important (from 90 to 56 terminal nodes in average), we observe a rather small reduction of variance and a (smaller) increase in bias. The global effect is thus only a slight reduction of error rates. Looking at individual results, the only very important decrease of error rates appears on the Gaussian problems where pruning reduces variance by a factor 2.5 while leaving bias mostly unchanged.

6.2.3 Discussion

These results are not surprising with respect to our analysis of decision tree variance. First, we have seen that variance already appears in the top nodes of the trees and even that the classification variance may be higher for small decision trees. As pruning does not make any change to the node splitting algorithm, it is tributary upon the variance of the growing algorithm. So the effect of pruning on decision tree accuracy is very limited. Although our study is carried out with a particular implementation of pruning, it does not give contradictory results with respect to large study about pruning algorithms (see for example [Min89a]). Nevertheless, the significant reduction of complexity fully justifies the need for such techniques at least for interpretability and simplicity reasons.

6.3 Discretization threshold stabilization

In the previous chapter, we have observed the high variance of the parameters that define decision trees and claimed that this variance may jeopardize the interpretability of the model.

We have also shown that a large part of this variance is due to the discretization of numerical attributes.

Therefore, if we want to further reduce the variance of decision trees, it is necessary to act on this part of the algorithm. The goal of this section is to improve the node splitting subroutine in order to stabilize the choice of the discretization threshold. In this chapter, as we do not want to destroy interpretability, we do not put into question the choice of a single attribute and threshold to split the node. We will propose in Chapter 8 a technique that amounts to selecting several discretization thresholds instead of one.

In Section 6.3.1, five variants of the local discretization algorithm are considered in order to determine thresholds with reduced variance. In Section 6.3.2, we evaluate empirically the effect of these techniques on the variance of discretization thresholds. The resulting effect on global decision tree accuracy is then assessed in Section 6.3.3. Anticipating on these results, let us say that we will find that the effect is more pronounced on small trees while it is almost canceled by recursive partitioning when trees are fully grown. We show also that the effect on global accuracy is blurred by the variance on the attribute choice. Hence, two techniques are proposed and evaluated in Section 6.3.4 which try to reduce this variance as well. We end with some discussion in Section 6.3.5.

6.3.1 Stabilization methods

The algorithm which produces a discretization threshold from a local learning sample LS at each test node of the tree can be considered as a regression learning algorithm which tries to estimate the asymptotic optimal threshold, i.e. the threshold which realizes the best score on the universe. The classical algorithm described in Chapter 5 does this by maximizing the estimation of the score computed from the local sample (just like a learning algorithm which minimizes the resubstitution error). Denoting by a_{th}^{∞} the asymptotic threshold and by $a_{th}(LS)$ the threshold determined by some algorithm from a random learning set LS, the bias/variance decomposition of the mean square error of this algorithm is written:

$$E_{LS}\{(a_{th}(LS) - a_{th}^{\infty})^2\} = (E_{LS}\{a_{th}(LS)\} - a_{th}^{\infty})^2 + E_{LS}\{(a_{th}(LS) - E_{LS}\{a_{th}(LS)\})^2\}. \quad (6.5)$$

For example, for learning sample of size 500 on the omib problem, the (squared) bias of the classical algorithm is 169 and its variance is 3169 (cf. Table 6.2 p.107). As it was already shown in Chapter 5, the variance of the classical algorithm is very important.

The goal of the stabilization methods proposed below is to improve the classical discretization algorithm by reducing the variance term of this decomposition without increasing too much its bias. Five variants are proposed.

- **Smoothing.** We have seen in the previous chapter that the very unstable nature of score curves is responsible for the high variance of the threshold. The smoothing approach consists in applying a moving-average filter (of fixed window size) to the score curve before selecting its maximum. If $[a_1, a_2, a_3, \ldots, a_{N_{th}}]$ is the ordered list of candidate thresholds, then the smoothed score value for each candidate is given by:

$$\widetilde{\mathrm{Score}}_{ws}([A < a_i], GS) = \begin{cases} \frac{1}{ws} \sum_{j=-\frac{ws-1}{2}}^{\frac{ws-1}{2}} \mathrm{Score}([A < a_{i+j}], GS) \text{ if } \frac{ws-1}{2} < i \leq N_{th} - \frac{ws-1}{2} \\ \frac{1}{i+\frac{ws-1}{2}} \sum_{j=1}^{i+\frac{ws-1}{2}} \mathrm{Score}([A < a_j], GS) \text{ if } i \leq \frac{ws-1}{2} \\ \frac{1}{N_{th}-i+\frac{ws-1}{2}+1} \sum_{j=i-\frac{ws-1}{2}}^{N_{th}} \mathrm{Score}([A < a_j], GS) \text{ if } i > N_{th} - \frac{ws-1}{2} \end{cases}, \quad (6.6)$$

  where ws is the window size (which is supposed to be odd). To make ws dependent on the number of thresholds, we take it as the lowest odd integer greater than $\lambda_{ws}.N_{th}$ where λ_{ws} is some user defined parameter in $[0, 1]$. A score curve on the omib problem and its smoothed version with $\lambda_{ws} = 0.2$ are reproduced in Figure 6.2. We expect the variance to be a decreasing function of the parameter λ_{ws} while bias should increase with λ_{ws}.

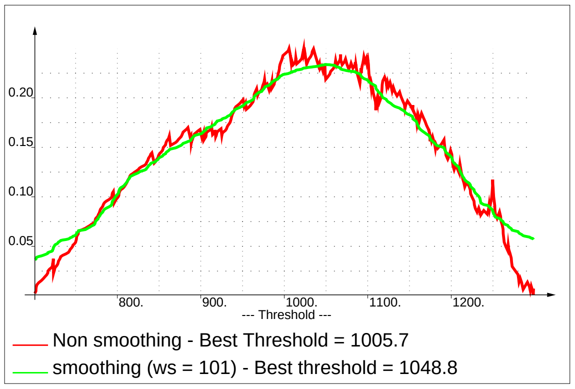


Figure 6.2: A score curve and its smoothed version, on the omib problem with $N = 500$ and $\lambda_{ws} = 0.2$

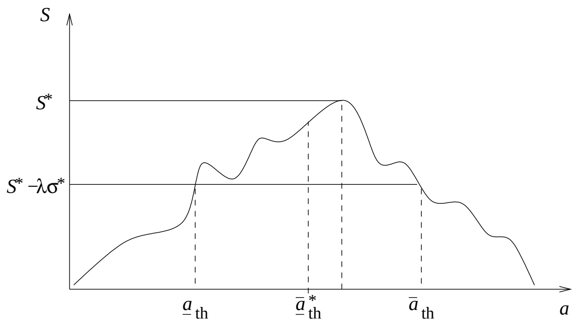


Figure 6.3: Illustration of the averaging technique

- **Aggregation.** We aggregate the n_a best thresholds according to the score measure using an arithmetic average[1]. Denoting by $\{a_1^*, a_2^*, \ldots, a_{n_a}^*\}$ the n_a best thresholds, this yields the following discretization threshold:

$$a_{aa}^* = \frac{1}{n_a} \sum_{i=1}^{n_a} a_i^* \tag{6.7}$$

$$\tag{6.8}$$

  Again, to make n_a dependent on the number of thresholds which are considered during node splitting, we take it as $\lambda_{n_a}.N_{th}$ where N_{th} is the number of different thresholds and λ_{n_a} is some user-defined parameter in $[0, 1]$. The more thresholds are averaged, the smaller will be the variance. In the extreme case where $\lambda_{n_a} = 1$, the threshold a_{aa}^* corresponds to the average value of the attribute in the local learning sample.

- **Averaging.** This method has been proposed by Wehenkel in [Weh97]. The idea here is similar to the so-called "one-standard error rule" used in decision tree pruning (not explained in our thesis). More precisely, the idea is to estimate the standard deviation of the sample estimates of the score at the best threshold and use this information in order to avoid overfitting the sample while selecting the discretization threshold. In this context, the main advantage of the score measure Score_W is the existence of an analytical formula for estimating its standard deviation [Weh97]. Thus "averaging" works according to the following procedure: (i) the score curve and the optimal threshold are first computed, yielding test T^*, the score estimate $\text{Score}_W(T^*, GS)$ and its standard deviation estimate $\hat{\sigma}_{T^*}$; (ii) a second pass through the score curve determines the smallest and largest threshold values $\underline{a}_{\text{th}}$ and $\overline{a}_{\text{th}}$ yielding a score larger than $\text{Score}_W(T^*, GS) - \lambda_{avg}\hat{\sigma}_{T^*}$, where λ_{avg} is a tunable parameter; (iii) finally the discretization threshold is computed as :

$$\overline{\underline{a}}_{\text{th}}^* = \frac{\underline{a}_{\text{th}} + \overline{a}_{\text{th}}}{2}. \tag{6.9}$$

  The method is illustrated in Figure 6.3. When λ is small, the chosen threshold is close to the best score threshold. On the other hand, when λ increases, it becomes closer to the center of the subsample and hence its variance should decrease and its bias increase.

[1]In other experiments not described here, we have also used a weighted by the score average but this does not show significant difference with a simple arithmetic average.

- **Local bootstrap.** This technique is inspired by bagging. The procedure is as follows : (i) draw by bootstrap[2] (i.e. with replacement) n_b learning sets from the original local learning subset; (ii) use the classical procedure on each subsample to determine n_b threshold values; (iii) determine discretization threshold as the average of these latter.

- **Global bootstrap.** The previous techniques are local in the sense that they improve the local node splitting algorithm only. With bootstrap, we could imagine a global technique which tries to stabilize all thresholds at once. To this end, a tree is build in the traditional way from the growing set. Then, n_b bootstrap samples of the complete sample are used to determine n_b trees which follow the choice of attribute of the first tree. Threshold values found this way are averaged to give a more stable tree and the whole growing set is propagated through this new tree in order to re-estimate conditional class probabilities at leaf nodes. While the stabilization of the top node threshold is similar to the one obtained by local bootstrap, we expect global bootstrap to reduce the variance of deeper test nodes better than the previous local techniques. Indeed, since the choices of thresholds in each bootstrap are conditional to the choice of their parent thresholds, averaging also takes into account the variability which results from the recursive partitioning of decision trees.

In the context of the first three techniques, it is clear that the parameter λ regulates some bias/variance tradeoff. So, the impact of such parameters will be studied in our experiments. Bootstrapping with finite n_b, on the other hand, computes a sample estimate of the average value of the threshold over all bootstrap samples and we know from statistical theory that such estimates are unbiased. So, the number n_b of bootstrap samples is not regulating any bias/variance tradeoff. The larger it is, the lower is the error on the threshold. For this reason, the value of n_b will be fixed to 50 throughout our experiments.

While we restrict our evaluation to the score measure Score_W (see equation 5.8), all these procedures could in principle be applied to other measures. Note however that the averaging needs an estimate of the score standard deviation[3]. Note also that further sophistications of these methods could be considered, but we will see that the above techniques provide already very interesting results. Finally, let us stress the fact that these methods, except for the bootstrap, generate only a very small extra computational cost with respect to the basic classical method.

The validation of these methods is carried out in two steps. We analyze first their effect locally on discretization threshold variance and then, their global effect on error rates, bias and variance of decision trees.

6.3.2 Validation on threshold variance

As the first goal of the proposed methods is to reduce the variance term of the decomposition (6.5), we carry out here experiments to verify how well each method succeeds in this goal. To estimate the different terms of the decomposition, we draw several samples (here, 100) of a given size from the pool, and run the particular version of the discretization algorithm on each of them. The asymptotic threshold is estimated by the threshold which is determined by the classical algorithm on the whole pool set. In all experiments, the attribute which is discretized is selected as the one which maximizes the score on the pool sample.

In smoothing, aggregation, and averaging, a parameter regulates some bias/variance tradeoff. Before comparing all methods, let us first see on one problem the evolution of bias/variance with respect to these parameters. Figure 6.4 shows the evolution of bias, variance and the square error on the threshold with the three methods for increasing values of the parameter. These curves are obtained on the omib problem and with learning samples of size 500. As expected, in

[2]We use here a stratified by class bootstrap in order to avoid pure samples (in term of class) which do not give a threshold value.

[3]If no analytical formula is available, a bootstrap technique can be used which is however computationally more demanding.

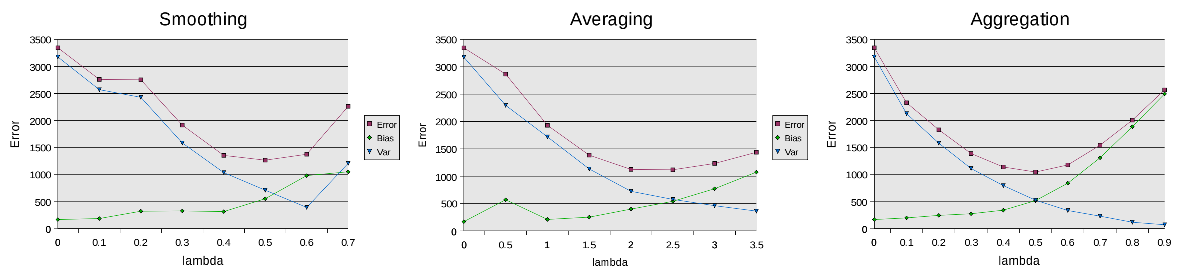


Figure 6.4: Evolution of error, bias and variance of the threshold with smoothing, aggregation and averaging (on the Omib problem with sample of size 500).

Table 6.2: Discretization variance for the Omib problem ($\sigma_A = 171.1327$, $A_{th}^{\infty} = 1060.695$)

method	$N = 50$			$N = 500$			$N = 2000$		
	$\sqrt{Err.}$	bias	σ_{Th}	$\sqrt{Err.}$	bias	σ_{Th}	$\sqrt{Err.}$	bias	σ_{Th}
Classic	108.2	23.1	105.7	57.8	-13.0	56.3	35.8	1.6	35.7
Smoothing (λ_{ws})	108.2 (0.0)	23.1	105.7	35.6 (0.5)	-23.6	26.7	17.8 (0.3)	-7.3	16.2
Aggregation (λ_{n_a})	43.5(0.6)	-23.4	36.7	32.4 (0.5)	-22.8	23.0	17.2 (0.3)	-6.2	16.0
Averaging (λ_{avg})	56.1 (2.5)	-44.3	34.4	33.4 (2.3)	-23.3	24.0	17.2 (2.0)	-7.3	15.6
Bootstrap	63.5	29.2	56.3	40.0	-6.7	39.4	23.3	0.7	23.3

each case, the variance decreases and the bias increases leading to a minimum in the square error for some value of the parameter. With smoothing, the reduction of error is not as important as with the two other techniques. On this problem, aggregation seems to work best.

The bias/variance profiles corresponding to the error minimum of each method are reported in Table 6.2 with the results of the classical discretization algorithm and the bootstrap method for increasing learning sample sizes. For each sample size, the bias/variance decomposition is provided in terms of the square root of the error, the bias (not the squared bias), and the standard deviation of the threshold. The best values for λ_{n_a} and λ_{ws} were searched in $\{0.0, 0.1, 0.2, \ldots, 1.0\}$ and the best value for λ_{avg} was searched in $\{1.0, 1.5, 2.0, 2.5, 3.0, 3.5\}$[4]. The values of the parameters leading to the minimum error in these subsets are reported in parenthesis near the error. On this problem, aggregation is the best method. It reduces the threshold standard deviation by more than a factor 2. Averaging is very close. Smoothing is also very good for the larger sizes but is useless on learning set of size 50. With respect to other techniques, bootstrap does not decrease so much the standard deviation but on the other hand, it keeps the bias low.

Table 6.3 summarizes the average results obtained on all datasets using the same experimental conditions. Before computing the average, all values are normalized by the standard deviation of the discretized attribute in the dataset. The average value of the parameter for smoothing, aggregation and averaging is reported in parenthesis near the average error. Aggregation is the best method with a reduction of average relative standard deviation by more than 50% for each size of learning sets. Averaging is always very close to aggregation. Smoothing does not work well on the smallest learning set size. It only reduces the standard deviation by 13%. For the two other sizes, however, it is competitive with aggregation and averaging. The effect of bootstrap is more limited but still remains interesting (more than a 30% reduction of standard deviation for each size). The average optimal value of λ_{n_a} is decreasing with the size of the learning set, while the value of λ_{avg} is less dependent. As the standard deviation of the score is also a decreasing function of the size of the learning set, this shows that more averaging is allowed for smaller sample sizes.

[4]These values were selected as the most sensible ones for each parameter from previous experiments not reported here.

Table 6.3: Average results on discretization with all techniques (in %)

method	$N = 50$			$N = 500$			$N = 2000$		
	$\sqrt{Err.}$	bias	σ_{Th}	$\sqrt{Err.}$	bias	σ_{Th}	$\sqrt{Err.}$	bias	σ_{Th}
Classic	51.5	9.3	50.4	25.4	5.8	24.6	16.3	4.8	15.2
Smoothing (λ_{ws})	46.2 (0.23)	12.2	44.0	16.0 (0.39)	8.0	13.5	8.5 (0.31)	5.3	6.2
Aggregation (λ_{n_a})	24.3 (0.69)	14.0	18.8	12.7 (0.54)	8.1	8.5	8.3 (0.33)	4.7	6.1
Averaging (λ_{avg})	29.5 (2.36)	11.9	25.5	14.2 (2.71)	8.5	11.0	9.2 (2.07)	5.4	7.0
Bootstrap	33.0	12.1	30.1	17.9	5.5	16.8	11.7	5.3	9.8

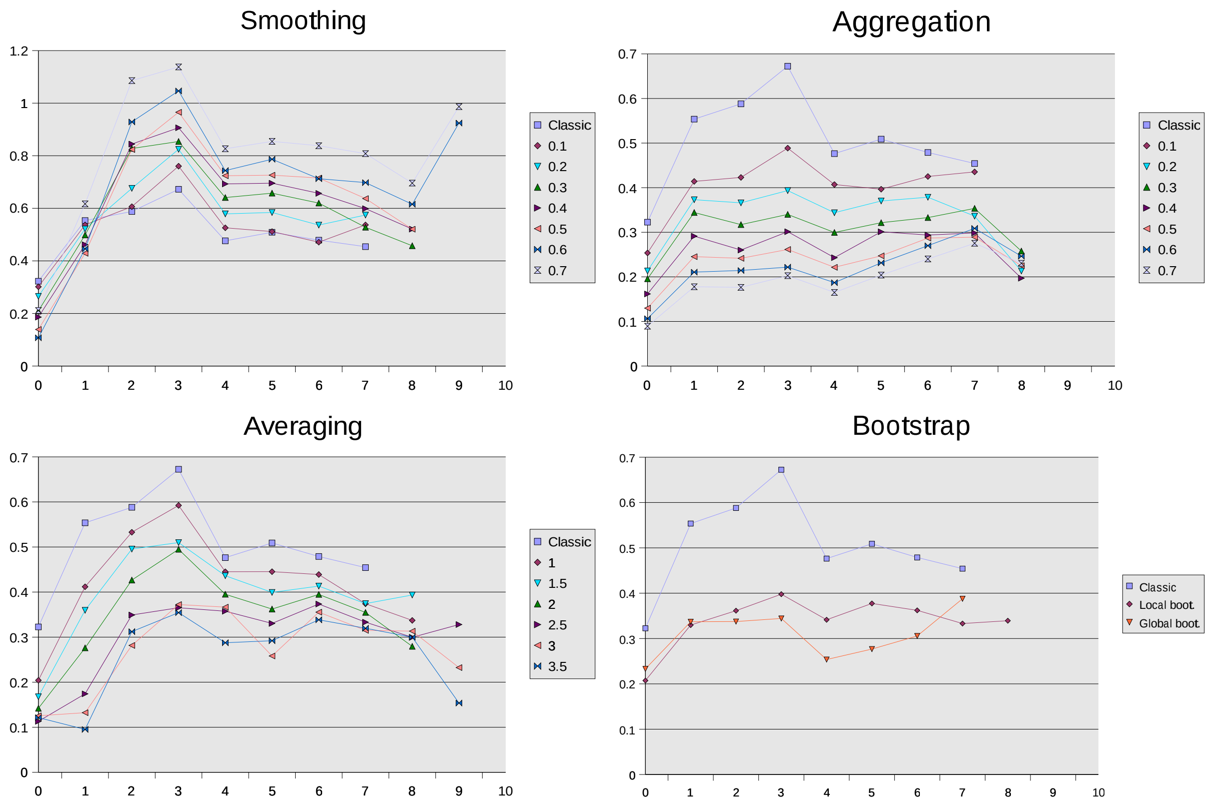


Figure 6.5: Relative standard deviation of the threshold versus tree depth on the omib problem

Influence of the tree depth

It is interesting to check the effect of variance reduction at deeper and deeper levels in the tree. To this end, we use the same experimental setup as in the previous chapter when studying score measures. A maximal tree is built from the pool sample and is used to guide the induction from smaller learning sets. The mean standard deviation of the discretization threshold (normalized by the standard deviation of the attribute) is averaged among all test nodes at a given level of the tree and these average values are plotted for increasing level values in Figure 6.5 (again on the omib problem with the reference learning sample, 300). For each method, different values of the parameter are considered. If smoothing can reduce the variance at shallow nodes, the effect on variance at deeper node is however very chaotic. Aggregation on the other hand, is very efficient. The more we increase λ_{n_a}, the more the variance decreases at each level of the tree. Averaging is also efficient at each level although its variance reduction seems to saturate faster than aggregation. Although very efficient at all levels, bootstrap can not go as far as averaging or aggregation in terms of the variance reduction. Now, the difference appears between the local and the global approaches. Although both approaches are similar at shallow nodes, as expected, the global one emphasizes the variance reduction at deeper nodes. Notice that the behavior of all algorithms is consistent from one data set to another.

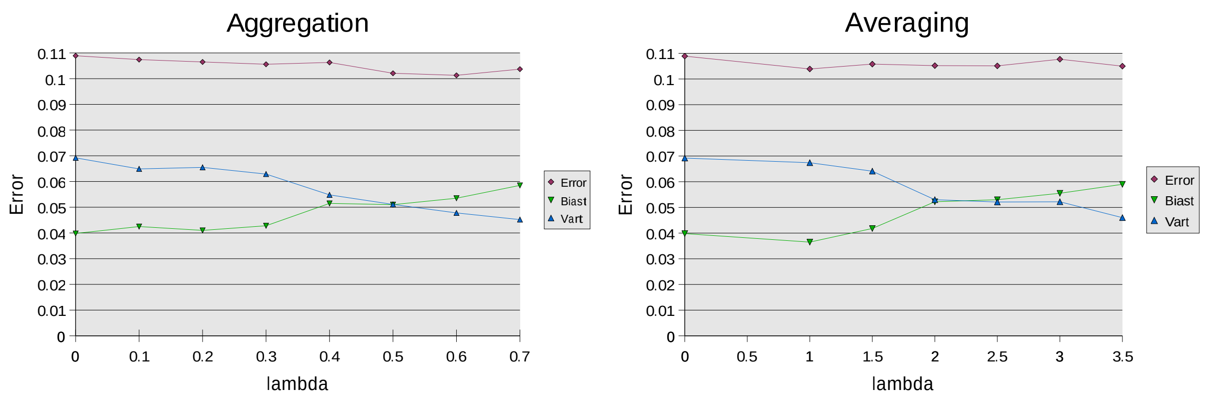


Figure 6.6: Aggregation and averaging with fixed attribute choice and fully grown trees on the omib problem

6.3.3 Validation on global accuracy

Except for smoothing, these simple techniques can reduce significantly the variance of the threshold which is chosen at decision tree nodes and they do it at each level of the tree. However, the impact of these techniques on decision tree accuracy has still to be evaluated. For the methods which depend on a parameter, the same value will be used at each node of the same tree. To determine the best value for this parameter, the only available criterion (without a priori knowledge of the thresholds) is the accuracy of the resulting trees which can be estimated either by cross-validation or from a separate validation set. However, the focus of the experiments of this section is more on the impact of these parameters on bias and variance and no algorithm will be validated that determines automatically the optimal value of the parameter. At the end of this section, we will nevertheless give some advice about a practical use of threshold stabilization techniques.

In our experiments, we will show that two factors influence the impact of threshold stabilization methods on the accuracy of decision trees: the variance on the choice of the attribute and the complexity of the trees. To separate the different effects and facilitate the analysis, we carry out experiments in several steps. First, the choice of attributes is fixed by a maximal tree and small trees (of limited complexity) are compared to fully grown trees. Second, the choice of attributes is freed and this time, small trees are compared to pruned trees.

The discussion is illustrated by the results obtained on the omib problem but they are similar on the other problems. As smoothing is not satisfactory at reducing threshold variance and also for the sake of brevity, we will exclude this method from the discussions of this section. Nevertheless, the interested reader may take a look at the tables of Appendix C for the results of smoothing and on other datasets.

Fixed attribute choice and fully grown trees

To focus on discretization threshold only, the techniques are first experimented in a fixed attribute choice setting. As usual, to fix the attribute choice, we build a maximal tree from the pool sample and used it to guide the induction with smaller learning samples. When building fully grown trees, the effect of increasing values of λ with aggregation and averaging is described on the omib problem in Figure 6.6 in terms of classification bias and variance and mean error rates. In both cases, the more λ is increased, the more variance decreases and bias increases. The bias/variance decompositions corresponding to the error minimum with respect to λ are compared to other methods in the top part of Table 6.4. In the case of aggregation and averaging the optimal value of λ is reported in parenthesis in the first column. With aggregation, the reduction of variance at the minimum of mean error (corresponding to $\lambda_{n_a} = 0.6$) is quite important but the bias increases and so the result is only a slight reduction of error. With av-

Table 6.4: Global accuracy of all techniques with fixed attribute choice and fully grown trees

	Mean error	compl	bias_T	var_T	var_{KW}	$\text{var}_{\hat{P}}$
OMIB						
Classic	0.1089	78	0.0398	0.0692	0.0709	0.0657
Aggr. ($\lambda_{n_a} = 0.6$)	0.1013	153	0.0535	0.0478	0.0593	0.0503
Avg. ($\lambda_{avg} = 1.0$)	0.1039	107	0.0365	0.0674	0.0673	0.0609
Local boot.	0.1109	144	0.0473	0.0637	0.0701	0.0634
Global boot.	0.1183	61	0.0670	0.0513	0.0686	0.0397
Average results on all datasets						
Classic	0.1996	105	0.1184	0.0812	0.1114	0.0550
Aggr. ($\overline{\lambda_{n_a}} = 0.3$)	0.1987	146	0.1238	0.0749	0.1078	0.0522
Avg. ($\overline{\lambda_{avg}} = 1.29$)	0.1988	132	0.1167	0.0820	0.1103	0.0532
Local boot.	0.2012	154	0.1242	0.0770	0.1100	0.0557
Global boot.	0.1978	86	0.1396	0.0582	0.0973	0.0311

eraging, the mean error rate goes through a minimum for a value of $\lambda = 1.0$ which corresponds only to a very small reduction of variance with respect to the classical algorithm. Nevertheless, from Figure 6.6, we see that the error rate is mainly unaffected by the reduction of threshold variance even for larger values of λ. Local and global bootstrap both reduce the variance but at the same time increases the bias and hence the error rate slightly goes up.

Average results on all datasets are reported in the bottom part of Table 6.4 with the best choice of parameters on each problem (the average value of the parameter is reported in the first column). With every method, variance decreases and bias increases and this leads to no significant improvement or deterioration of average error rates. Only global bagging reduces average variance by 28%. Nevertheless, this reduction of variance is emphasized by the fact that this method produces smaller trees than other methods (since thresholds are reestimated globally from bootstrap sample which are smaller than the original learning sample) and thus can not be fully attributed to the method itself.

Fixed attribute choice and limited complexity

The reduction of global prediction variance is thus not very important. Experiments with small trees may help to understand the reasons why this is so. To this end, we further limited the complexity of the trees to a given dataset dependent value[5] using the best first strategy discussed in the previous chapter. Effect of increasing values of λ_{n_a} and λ_{avg} are drawn in Figure 6.7. With respect to the experiments with fully grown trees, the variance decreases much more strongly but the bias is increasing to the same extent. The top part of Table 6.5 describes the results obtained with all methods on the omib problem. With respect to the experiment with full trees, the variance decreases more strongly with all methods but again the effect is an increase of the bias and hence a rather slight improvement of the mean error rate. Average results are summarized in the lower part of Table 6.5. This time, the minimum of error rate with averaging and aggregation corresponds to a significant reduction of average variance (respectively by 40% and by 50% for var_T). Local and global bootstrap are very close to each other and also decreases the variance significantly (by about 40%).

So, it appears that, with fixed attribute choice, threshold variance reduction indeed stabilizes the prediction of small trees but at the same time increases their prediction bias, yielding nevertheless a slight reduction of error rates. However, all the effort at reducing the variance at shallow nodes of the tree is more or less canceled when trees are fully developed. In this case, error rates, bias and variance are mainly unaffected by threshold variance reduction techniques whatever their intensity. The occurrence of variance in the tree is thus only delayed to the deeper levels but will appear nevertheless.

[5]This value is fixed at 50% of the average complexity of pruned trees on the same dataset

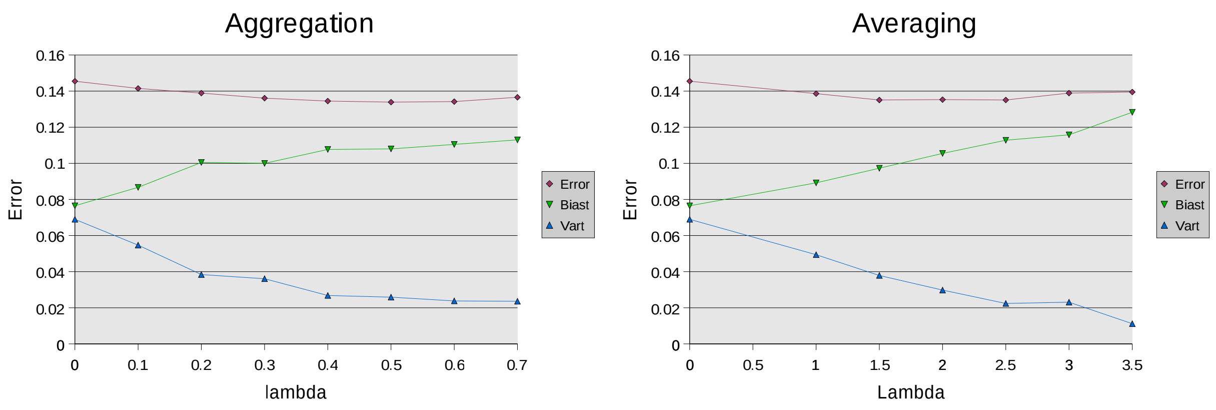


Figure 6.7: Aggregation and averaging with fixed attribute choice and limited complexity on the omib problem

Table 6.5: Global accuracy of all techniques with fixed attribute choice and limited complexity

	Mean error	compl	bias_T	var_T	var_{KW}	$\text{var}_{\hat{P}}$
OMIB						
Classic	0.1455	25	0.0765	0.0690	0.0843	0.0463
Aggr. ($\lambda_{n_a} = 0.5$)	0.1339	26	0.1080	0.0259	0.0558	0.0206
Avg. ($\lambda_{avg} = 2.5$)	0.1351	25	0.1128	0.0224	0.0532	0.0188
Local boot.	0.1342	29	0.0940	0.0402	0.0669	0.0307
Global boot.	0.1359	25	0.0877	0.0481	0.0701	0.0305
Average results on all datasets						
Classic	0.2399	27	0.1812	0.0587	0.1059	0.0280
Aggr. ($\overline{\lambda_{n_a}} = 0.33$)	0.2309	28	0.2014	0.0295	0.0820	0.0152
Avg. ($\overline{\lambda_{avg}} = 1.79$)	0.2316	28	0.1965	0.0351	0.0845	0.0171
Local boot.	0.2304	29	0.1965	0.0339	0.0882	0.0192
Global boot.	0.2301	27	0.1907	0.0395	0.0893	0.0190

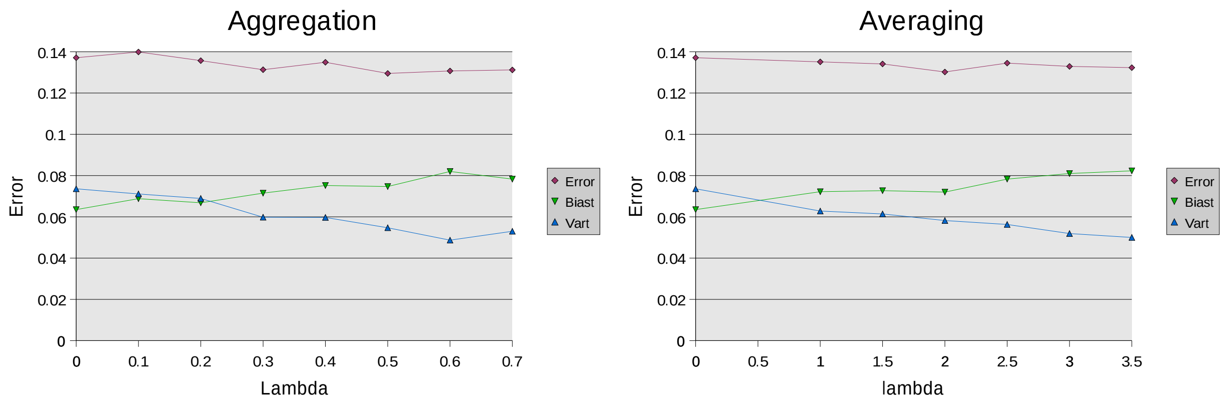


Figure 6.8: Aggregation and averaging with free attribute choice and limited complexity on the omib problem

Table 6.6: Global accuracy of all techniques with free attribute choice and limited complexity

	Mean error	compl	bias_T	var_T	var_{KW}	$\text{var}_{\hat{P}}$
OMIB						
Classic	0.1371	29	0.0635	0.0736	0.0852	0.0546
Aggr. ($\lambda_{n_a} = 0.5$)	0.1295	29	0.0747	0.0547	0.0728	0.0351
Avg. ($\lambda_{avg} = 2.0$)	0.1302	29	0.0720	0.0582	0.0740	0.0367
Local boot.	0.1408	29	0.0798	0.0610	0.0827	0.0415
Global boot.	0.1385	28	0.0803	0.0582	0.0797	0.0364
Average results on all datasets						
Classic	0.2284	29	0.1220	0.1064	0.1288	0.0472
Aggr. ($\overline{\lambda_{n_a}} = 0.40$)	0.2252	29	0.1278	0.0974	0.1238	0.0376
Avg. ($\overline{\lambda_{avg}} = 1.71$)	0.2256	29	0.1237	0.1020	0.1249	0.0409
Local boot.	0.2271	29	0.1287	0.0985	0.1252	0.0406
Global boot.	0.2288	29	0.1311	0.0977	0.1251	0.0365

Table 6.7: Global accuracy of all techniques with free attribute choice and pruned tree

	Mean error	compl	bias_T	var_T	var_{KW}	$\text{var}_{\hat{P}}$
OMIB						
Classic	0.1189	57	0.0485	0.0704	0.0763	0.0691
Aggr. ($\lambda_{n_a} = 0.7$)	0.1089	89	0.0558	0.0532	0.0662	0.0569
Avg. ($\lambda_{avg} = 1.5$)	0.1107	80	0.0428	0.0680	0.0710	0.0654
Local boot.	0.1234	77	0.0553	0.0682	0.0781	0.0675
Global boot.	0.1291	44	0.0727	0.0564	0.0768	0.0426
Average results on all datasets						
Classic	0.2066	56	0.1046	0.1020	0.1205	0.0576
Aggr. ($\overline{\lambda_{n_a}} = 0,47$)	0.2012	68	0.0996	0.1016	0.1176	0.0558
Avg. ($\overline{\lambda_{avg}} = 2,07$)	0.2015	73	0.1052	0.0963	0.1161	0.0523
Local boot.	0.2061	63	0.1065	0.0997	0.1192	0.0552
Global boot.	0.2121	54	0.1136	0.0985	0.1204	0.0420

Free attribute choice and limited complexity

In practice, of course, test attributes are selected from the local subsample. Hence, let us see how evolve the bias/variance terms of trees of limited complexity when we let the algorithm choose the attributes (i.e. with the classical growing algorithm). Figure 6.8 shows in this case the effect of aggregation and averaging. With respect to Figure 6.7 in the case of fixed attribute choice, the effect of threshold variance reduction is more mitigated. Table 6.6 reports results with all methods on the omib problem and in average. Threshold variance reduction techniques still reduce the classification variance but the result is much less pronounced than with fixed attribute choice. When the reduction of var_T by aggregation (the best method here) was by about 60% on the omib problem and 50% in average on all problems with fixed attribute choice, it is only by 25% and 10% respectively in this case. So, it seems that the additional variance coming from the attribute choice somehow cancels the impact of threshold variance reduction on prediction variance.

Free attribute choice and pruned trees

As we have seen that the impact of discretization depend on the complexity of the tree, the obvious candidate method to adapt the complexity is pruning. Table 6.7 gives the result of all methods with pruned trees, top on omib and bottom in average on all datasets. The pruning algorithm is the post-pruning algorithm described in the previous section. On each problem, the large validation set was used.

In terms of error rate, bias and variance, there is not a very marked trend. If the mean error does not really increase (except with local bootstrap), it does not decreases significantly. The fact that bias and variance are also mainly unaffected by threshold variance reduction

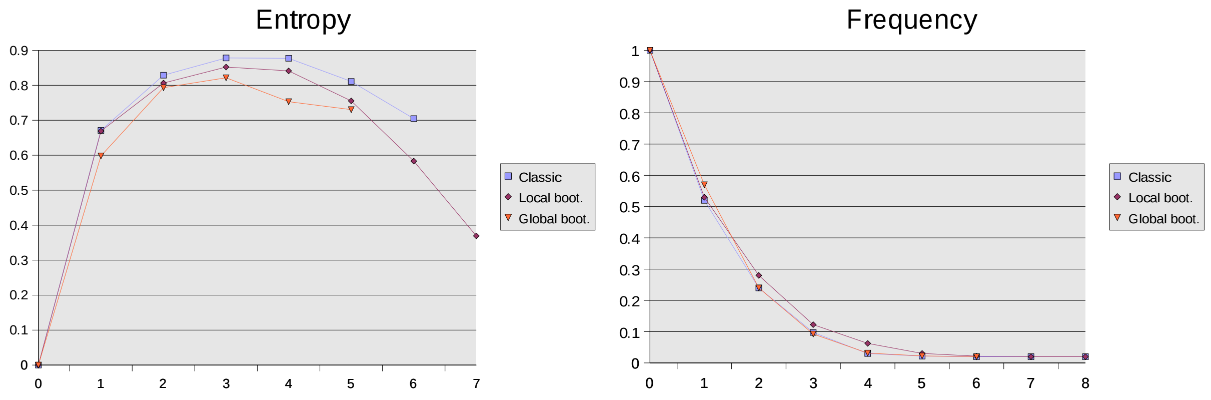


Figure 6.9: Attribute choice variance with increasing depth on the omib problem

Table 6.8: Global accuracy of attribute choice stabilization techniques with full trees

	Mean error	compl	bias_T	var_T	var_{KW}	$\text{var}_{\hat{P}}$
Classic	0.2154	90	0.0969	0.1185	0.1320	0.0765
Local boot.	0.2256	136	0.1047	0.1209	0.1379	0.0794
Global boot.	0.2274	58	0.1227	0.1047	0.1286	0.0394

techniques can be explained by the effect of pruning which gives more complex trees in these cases than in the classical algorithm. Nevertheless, the average optimal value of the parameters λ_{n_a} and λ_{avg} (determined from the test set) are quite high and thus from the analysis on the discretization threshold variance, they should correspond to an important reduction of the variance of the thresholds used to define the trees.

6.3.4 Attribute selection

In Chapter 5, we have shown that the choice of the attribute at a tree node is rather unstable and the preceding experiments about discretization threshold stabilization show that, from the point of view of global accuracy, the attribute selection variance actually cancels the reduction of threshold variance which could be obtained by the techniques just discussed. It is therefore worth trying to reduce the attribute selection variance as well.

The bootstrap approach to the stabilization of the threshold can be also applied to the stabilization of the attribute choice. Like for the threshold, we evaluated a global and a local approach:

- **Local bootstrap** At each node, draw by bootstrap n_b learning sets from the original local learning subset. Use the classical procedure on each subsample to determine an attribute and a threshold for this attribute. Among the n_b pairs, select the most frequently chosen attribute.

- **Global bootstrap** Again, draw n_b bootstrap samples from the growing set, build a tree from each of these samples according to the classical tree growing algorithm. Then, compute the most probable structure from this set of trees (according to the procedure given in Section 5.6.2) and relearn discretization thresholds from the growing set following this structure.

In both cases, the threshold is determined using the equivalent bootstrap approach described in the previous subsection.

Both of these algorithms are tested in the same conditions as in Section 5.6.2. Left part of Figure 6.9 shows the entropy of the attribute choice for increasing level in the tree obtained with both approaches on the omib problem. The right part of this figure shows the frequency of appearance of the most probable structure as defined in Section 5.6.2. The improvement on attribute choice entropy is very slight but still greater with the global approach. However, there is almost no change on the frequency curve. Average results with full trees in Table 6.8 reflects the small change on these curves. There is no significant difference in terms of variance if we compare these results with those obtained in the preceding experiments considering the threshold variance reduction alone. However, the bias increases slightly and hence globally the accuracy is even slightly deteriorated.

6.3.5 Discussion

From the point of view of accuracy, the results of our threshold stabilization techniques are not very encouraging. Although some further refinements are possible (for example, we could adapt the parameters to the size of the local subset), in the light of our experiments, we do not believe they will succeed in reducing errors in an impressive way. Indeed, the accuracy of decision tree is tied by two phenomena:

- First, if we let the tree grow in order to reach optimal accuracy on the learning set and hence low bias, then the variability still appears at deep nodes and our techniques have almost no effect on global accuracy.
- Second, if we limit the size of the tree, then model stabilization techniques can arbitrarily reduce the prediction variance but the price to pay is an increase of the bias. So, at best, the effect on accuracy is only a very slight improvement of the error.

In spite of this somewhat negative conclusion, the techniques introduced in this section are still able to reduce significantly the variance of the threshold which is chosen to split a decision tree node. The reduction of the variance appears at all levels of the tree and is obtained, for some of the methods, at low computational cost with respect to the classical algorithm. Thus, these techniques indeed improve the interpretability of decision trees without deteriorating their accuracy.

Among the five techniques, the most interesting ones are certainly aggregation and averaging. Smoothing does not really work and bootstrap is not as good at reducing threshold variance and furthermore it increases significantly computation times. On the other hand, aggregation and averaging both succeed in reducing threshold variance and their impact on computation times is almost null (in the case of averaging, provided that an estimation for the standard deviation of the score is available at negligible computational cost). Practically, the choice of the parameters for these two methods can be done by cross-validation. However, from our experiments, a rule of thumb could be to take $\lambda_{n_a} = 0.4$ and $\lambda_{avg} = 2.0$.

Although we have focused here on local (node by node) discretization philosophies, it is clear from our results that global discretization must show similar variance problems (since this variance is present even for large sample sizes) and that some of the ideas and methodology discussed here might be fruitfully applied to global discretization as well. More broadly, all machine learning methods which need to discretize continuous attributes in some way could take advantage of improvements.

The work on discretization presented in this section is complementary to most previous work on discretization which has been devoted exclusively to the improvement of global characteristics of trees (complexity, predictive accuracy, and computational efficiency). With this goal, one part of the publications in this domain investigates the global vs local discretization question (e.g. [FW99]) and another part focuses on the choice of a good measure (e.g. [DKS95]), or on the search problem (e.g. [ZRF99]). Except for [Weh97], we have no knowledge of previous work on stabilization of decision tree models.

6.4 Conclusion

The very conclusion we draw from the investigations reported in this chapter is that pruning and threshold stabilization essentially only improve interpretability of decision trees, on one hand, by simplifying them and, on the other hand, by stabilizing their tests. However, their effect on prediction accuracy is small. Although the stability of the induction algorithm can certainly still be improved (for example, by stabilizing the node labeling), we believe that our investigations clearly show the limitations of what can be done in terms of accuracy improvement by further refining decision tree induction without relaxing its intrinsic representation bias.

These intrinsic limitations of decision trees may be illustrated by an example. With small trees built from learning sets of 500 objects on the omib problem, we get a mean error rate of 13.7% which is decomposed into a bias of 6.35% and a variance of 7.35% (see the first line of Table 6.6 pp. 112). On the other hand, a small tree built from the 16000 objects of the pool set yields an error rate of 12.15%, thus an improvement of only 1.55%. With learning sets 32 times smaller than this learning set, it is very unlikely to get a model as accurate as this model whatever the variance reduction technique. So, if we could reduce the variance by more than 1.55%, it must increase the bias in some way. For example, with averaging and $\lambda = 3.5$, we get a decrease of variance by 2.35%, but in the mean time, the bias increases by 1.875%. So, it is wrong to believe that it is possible to reach a mean error rate close to the bias of 6.35% by solely reducing the variance and keeping the representation bias of decision trees unchanged.

A further significant step would need a relaxation of this representation bias. This step is made by the averaging techniques studied in the next chapters. As we will see, because of the implicit extension of the hypothesis space provided by these approaches, a reduction of prediction variance is this time possible without increasing too much the bias.

Chapter 7

Perturbing and combining decision trees

In this chapter, perturb and combine algorithms are discussed in the context of decision tree induction. We start our presentation with the description of Breiman's now classical bagging method and provide some experimental results to show its value in the context of the test problems used in the preceding chapters. We then briefly discuss some other approaches along the same idea which have been proposed in the literature, and end up with a new proposal which goes one step further than existing P&C techniques. This is an extreme random algorithm which builds trees randomized on purpose which, when averaged, yield significantly better results than bagging. Although this algorithm is still in its infancy, we will provide a certain number of indications about the possible extensions of this idea which would allow one to cope efficiently with very large datasets and we argue that this technique can be viewed as a bias reduction algorithm.

7.1 Introduction

Experiments of the previous chapter have shown that the accuracy of a single decision tree is regulated by a very constraining tradeoff between bias and variance. Reducing the variance of the decision tree induction algorithm inevitably results in an increase of the bias and hence yields only slight improvement of accuracy. On the other hand, we have seen in Chapter 4 that perturb and combine algorithms allow to go beyond this intrinsic tradeoff in the context of a learning algorithm by averaging the predictions given by several models (and hence extending the hypothesis space of the method). Actually, most of the general perturb and combine (P&C) techniques described in chapter 4 have been introduced to improve decision trees. The application of P&C techniques in combination with decision trees is appealing for two reasons. First, since decision tree variance is high, averaging often gives impressive improvements with respect to the classical induction algorithm. Second, tree growing is fast and thus building several models is not too costly. This chapter is devoted to a study of P&C algorithms in the context of decision tree induction.

To give a first idea of what can be gained with P&C algorithms applied to decision trees, we first experiment with the popular bagging algorithm on our seven classification problems. Note that while bagging is a general purpose wrapper technique which can in principle be combined with any automatic learning method, we focus here on its use in the context of decision tree induction. Then, we discuss several P&C techniques specifically dedicated to decision trees which have been proposed in the literature and explain their success in the light of the analysis of our previous chapters. From this discussion and our results on decision tree variance, we propose a new algorithm which goes one step further than existing algorithms by building trees almost fully randomly. We show that this technique gives better results than bagging and boosting. Eventually, we conclude with a discussion about interesting future work directions in the context of P&C algorithms, notably in the direction of a reduction of the bias.

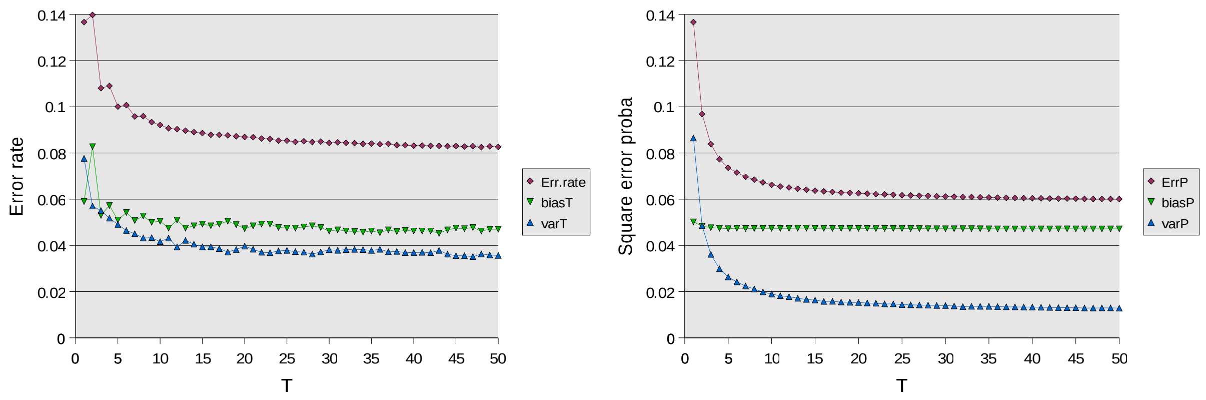


Figure 7.1: Evolution of bias/variance with the number of bootstrap samples with bagging on the omib problem. Left, with error rate, right with square error of probability estimates.

Table 7.1: Effect of bagging (average over 7 problems)

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
Single tree	0.2161	90	0.0966	0.1195	0.1326	0.1257	0.0486	0.0771
Bagging (25 trees)	0.1402	1631	0.0969	0.0434	0.0689	0.0624	0.0502	0.0121

7.2 Bagging

Bagging was one of the first ensemble methods proposed in the literature and its first validation was made on tree based algorithms [Bre96b]. In classification, bagging explicitly tries to find an approximation of the majority vote classifier by simulating sampling from the universe by sampling with replacement from the learning set. If the approximation is good, it should result in a decrease of variance while leaving the bias unchanged.

We kindly refer the reader to section 4.3.1 for the description of this method. We merely note here that since bagging is a (pure) variance reduction technique, it is generally applied to unpruned decision trees (which have a smaller bias than pruned ones). Let us note however, that bagging, especially with a finite number of bootstrap samples, will not reduce the variance to zero. Hence, it is often possible to obtain slightly better results by post-pruning the ensemble of trees in an ad hoc way. We refer the interested reader to reference [GW00] which reports some of our work in this context.

7.2.1 Experiments

The left part of Figure 7.1 shows the evolution of the mean error rate with respect to the number of bootstrap samples on the omib problem (with fully grown trees and the reference learning sample size). Predictions are made according to average of conditional class probability estimates at leaf nodes. The right part of the same figure draws the evolution of the square error of these probability estimates. As expected, the bias of probability estimates is essentially constant whatever the number of trees which are averaged and the regression variance is monotonically decreasing. While Tibshirani's bias is slightly decreasing[1], the main effect on error rates is also a reduction of variance.

Mean results on all datasets are reported in Table 7.1. They are obtained by averaging probability estimates of 25 fully grown trees. On each problem, the reference learning sample size was used (see Table 5.2 pp. 83). The reduction of variance is very impressive (almost a factor 3 for var_T and a factor 6 for regression variance). In the mean time, biases only very slightly increase. Average error rate thus goes from 21,6% to 14%, a relative reduction of 35%.

[1]The short increase in bias with two trees appears because many ties occur in this case and, in our implementation, they are always broken in favor of the same class.

Table 7.2: Bagging on the omib problem

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
Single tree	0.1213	75	0.0420	0.0793	0.0789	0.1213	0.0424	0.0789
One bootstrap	0.1371	56	0.0492	0.0878	0.0887	0.1371	0.0483	0.0887
Bagging (25 trees)	0.0842	1386	0.0475	0.0367	0.0483	0.0611	0.0466	0.0144

7.2.2 Bias/variance interpretation

In chapter 4, bias/variance analysis shows that good perturb and combine techniques should introduce some noise ε in the algorithm catching some of the variance coming from the learning set randomness but at the same time should not introduce too much bias. From the previous chapter, it also appears that both discretization variance and attribute selection variance was particularly high. So, bagging is a good idea to introduce such a noise in the context of decision tree induction. Indeed, inducing a tree from a bootstrap sample of the learning set is not going to increase very much the bias since it contains in average more than 60%[2] of the points of the learning set. In the mean time, as decision tree variance is high, even such a small perturbation of the learning set causes very variable models. For example, Table 7.2 compares, on the omib problem, single trees from the whole learning sample, single trees from bootstrap replicates of the learning sample, and, bagging of 25 trees. Building trees from random bootstrap samples slightly increases both regression and classification bias and variance with respect to the (non random) classical algorithm. When averaging such trees, the part of the regression variance which is captured by randomizing the learning set is here given by:

$$\begin{aligned} \text{var}_{LS}\{f_{LS}(\underline{a})\} - \text{var}_{LS}\{E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\} &= 0.0789 - 0.0144 && (7.1) \\ &= 0.0645 && (7.2) \end{aligned}$$

i.e. 82% of the initial decision tree variance. The effect on the decomposition of the mean error rate is also a strong reduction of var_T (by 64% in the mean) and a slight increase of bias_T.

7.2.3 Geometrical interpretation

Besides the classical bias/variance interpretation of bagging, the good behavior of bagging can also be interpreted in a geometrical way.

An ensemble of several decision trees can always be represented by a much more complex tree, as it also cuts the input space into hyper-rectangular regions where the prediction is constant. For example, the left part of Figure 7.2 shows the partitioning of a bi-dimensional input space done by a single tree trying to separate empty circles from full circles and the right part of the same figure shows the partitioning done by bagging of ten such trees. From this experiment, the finer cutting of bagging is obvious and this also shows clearly the high variance of decision trees (if all trees were identical, the two figures would also be identical). On Figure 7.3, we can also see the regions on which depend the classification of three points (marked by a cross) of the same input space: left with a single tree, right with bagging. In this latter case, each rectangular region corresponds to the terminal node of one of the ten trees averaged by bagging which is reached by the point. The classification regions with bagging are more extended than those obtained by a single tree and also overlap between each other even if the points are far from each other. This implies that the prediction at a particular point in the input space depends on a much larger number of learning objects on the one hand, and on the other hand, that the influence of a learning object on the prediction at a particular point tends to decrease progressively as the distance between the two attribute vectors increases. We are thus far from the recursive partitioning induced by a single tree.

[2]An instance of the learning sample has probability $1 - (1 - 1/N)^N$ of being selected at least once in a bootstrap sample. For large N, this value converges to $1 - 1/e = 0.632$ and hence a bootstrap sample contains about 63.2% of unique instances of the learning sample [BK99].

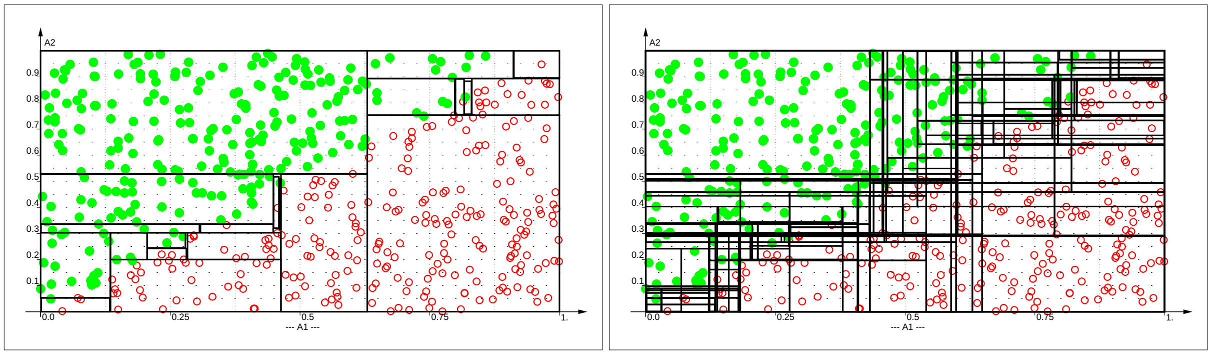


Figure 7.2: Partition of a bi-dimensional input space by a single tree (left) and ten bagged trees (right)

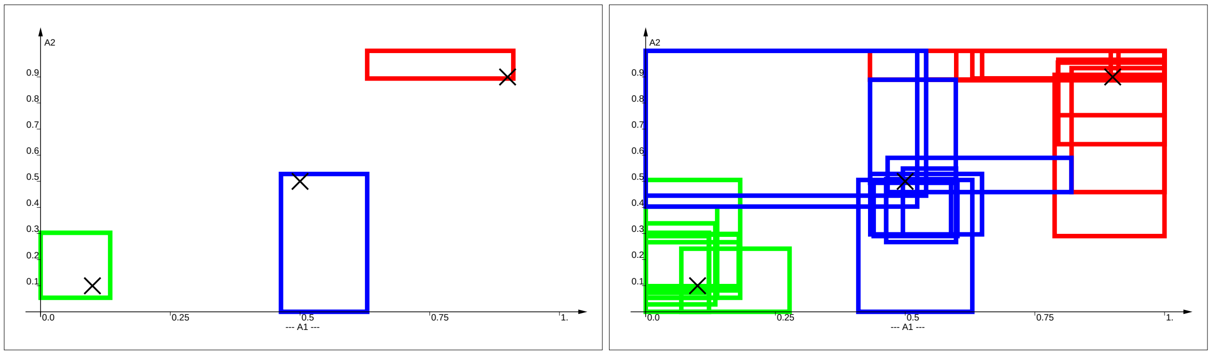


Figure 7.3: Classification regions at three points, single tree (left) and ten bagged trees (right)

Also, while the prediction of a single tree is very irregular, the prediction given by bagging is, for the same reason, much "smoother" (although still piecewise constant). Indeed, because of averaging, when we make a small move in the input space, only a small proportion of the terminal nodes reached by our test point changes and thus the perturbation of the aggregated prediction is reduced with respect to the perturbations of individual predictions. Decision tree bagging is thus closer in its behavior to the k-nearest-neighbor method than decision trees. The main advantages with respect to k-NN are the adaptation of the number of neighbors to the position in the input space (the size of the regions) and the implicit automatic induction of a distance measure which takes into account as much as possible only relevant attributes for classification.

Note that, although illustrated in the case of bagging, this analysis remains valid for other perturb and combine techniques in the context of decision trees.

7.3 Specific P&C methods for decision tree induction

Bagging is not the only P&C technique available in the context of decision tree induction. Besides the general P&C techniques presented in Chapter 4 (most of which have been validated exclusively with decision trees), there exist several algorithms specifically designed for tree-based models.

One of the first averaging algorithm is the option trees introduced by Buntine [Bun92] in a Bayesian framework (see section 4.3.5). The goal of option trees is to represent in the same structure several decision trees which posterior probability is high. For example instead of considering only one test at the first level, the most significant tests are all selected and used to build a tree. The process is repeated at all pairs of successor nodes resulting from the different splits. This yields a complex structure which grows exponentially with the depth of the tree. The prediction at a node is obtained by averaging the predictions of all trees starting from this node weighted by the posterior probability of each tree. Although this method was mainly designed to approximate a full Bayesian approach (by keeping only a finite set of highly probable trees), the final result is very similar to a bagged set of decision trees. The initial idea of Buntine was later studied and extended by others (e.g. [OD95] or [KK97]).

Several people have proposed to randomize the induction algorithm of decision trees to obtain a set of different models. For example, Kwok and Carter [KC90] propose to randomize the top node split of a decision tree and their algorithm shows improvement with respect to classical trees on two problems. Along the same idea, Ali and Pazzani [AP95, AP96] propose to select a test at random among the best tests. If S^* is the score of the optimal test, the set of tests is defined as the tests which have a score greater than $S^* - \beta.S^*$, where β is some constant between 0 and 1. The probability of selecting a test among best ones is proportional to its score. A comparison of this algorithm with bagging and other averaging techniques shows that it gives similar improvement as bagging. Dietterich [Die00c] uses a very similar approach to build random trees: during node splitting, a test is simply selected at random among the 20 best splits. He compares his method with bagging and boosting on several datasets. While not as robust as bagging to noise, his random trees give occasionally better results than bagging.

A number of authors have proposed to randomize decision trees by randomizing the set of candidate attributes. For example, Ho [Ho95, Ho98] builds decision trees from randomly selected subsets of attributes. In a second stage, random trees are obtained by locally (instead of globally) perturbing the subset of attributes used to split a node. In a character recognition application [AD97], Amit and Geman define a very large set of candidate attributes corresponding to different geometrical features computed from images and handle such a large set by selecting random subsets of it at decision tree nodes. In a more recent publication [Bre01], Breiman builds random trees in two ways: first like Ho and Amit and Geman by locally selecting a random subset of attributes, second by constructing locally a set of random linear combinations of attributes and searching among them for the best split. These two variants are then shown to compare favorably with boosting[3] techniques on several classification and regression problems.

7.4 Extra-trees : extremely randomized trees

The good behavior in practice of the randomization techniques described in the previous section is not surprising in the light of our analysis of decision tree variance. All decisions taken during tree induction are significantly random, especially the choice of tested attributes and discretization thresholds. We have seen how small perturbations of the learning set translate into strong perturbations of the tests which are chosen at tree nodes. All the perturbation techniques discussed in the previous sections are essentially trying to simulate such perturbations. For example, if several tests have close scores, then it is likely that they will be selected in turn depending on the particular learning set choice. This variability can be simulated by selecting one of them at random during induction. However, even in this case, the choice of the best tests still depends on the learning set and hence some (possibly reducible) variance remains.

One interesting question we would like to answer in this section is: how much randomness can be introduced in the tree growing algorithm without decreasing accuracy of the aggregated model, and even more ambitiously, how much randomness should be introduced so as to maximize this accuracy ? From the development of section 4.3, the condition for preserving accuracy is simply not to increase too much the bias when perturbing the learning algorithm. One condition to ensure low bias is to keep the resubstitution error low. So, while previous authors have been quite timid, we propose to go one step further and randomize both discretization thresholds and attribute choices. The algorithm we propose is first described and then compared empirically to bagging on our seven test problems. Eventually, we discuss the potential of such a random algorithm to handle very large datasets.

[3]Boosting is often better than bagging.

Table 7.3: Extra-tree growing

```
EXTRAtreegrow($LS$):
(Build an extremely random tree from the set $LS$.)

  • Create a node $\mathcal{N}$ and set $LS(\mathcal{N})$ to $LS$;
  • Set $L = \{\mathcal{N}\}$, the list of open nodes.
  • While $L$ is not empty:
      – Select and remove a node $\mathcal{N}$ in $L$;
      – If stopsplitting($LS(\mathcal{N})$) is true then
          ∗ Compute conditional class probability estimates $\hat{P}(C|\mathcal{N})$ from $LS(\mathcal{N})$ and attach it to the node $\mathcal{N}$;
      – Else
          ∗ Select an attribute $A_i$ at random from the set of candidate attributes
          ∗ Compute the empirical mean, $\overline{A_i}(LS(\mathcal{N}))$, and standard deviation, $\sigma_{A_i}(LS(\mathcal{N}))$, of this attribute in $LS$.
          ∗ Draw a threshold, $a_{th}$ at random according to the distribution $N(\overline{A_i}(LS(\mathcal{N})), \sigma_{A_i}(LS(\mathcal{N})))$
          ∗ If $\text{Score}_W(LS(\mathcal{N}), [A_i < a_{th}]) < s_{th}$ return at the first step and select another attribute
          ∗ Compute the subsamples $LS_l = \{(\underline{a}, y) \in LS(\mathcal{N}) | a_i < a_{th}\}$ and $LS_r = \{(\underline{a}, y) \in LS(\mathcal{N}) | a_i \geq a_{th}\}$;
          ∗ Create two nodes $\mathcal{N}_l$ and $\mathcal{N}_r$ and set $LS(\mathcal{N}_l) = LS_l$ and $LS(\mathcal{N}_r) = LS_r$;
          ∗ Insert $\mathcal{N}_l$ and $\mathcal{N}_r$ in $L$.
```

7.4.1 The algorithm

Our modified tree growing algorithm is described in Table 7.3 (see Table 5.1 pp. 79 for a reminder of the non random algorithm). The splitting attribute is first selected at random, then a threshold is drawn from a Gaussian distribution with mean and standard deviation given by the corresponding empirical values in the local subset. To avoid very bad tests, the attribute choice is rejected if the score of the test is lower than a given threshold (the value of s_{th} was fixed at 0.1 in the experiments below, which is 10% of the maximal possible value of the score) and another attribute is selected. The development of a branch is stopped only when the node is pure in terms of the classification output or when all attributes have constant values in the local subset. One important characteristic of this algorithm is that it still produces perfect trees on the learning set (at least to the same extent than the classical algorithm), although much more complex than trees induced classically, and thus it should preserve low bias.

Of course, whether this randomized method is "extreme" in the sense questioned above is an open question. It is however not far from a totally random algorithm, which would build a tree structure without even looking at the learning sample, and use this sample only to determine which nodes are terminal and what labels to attach to them.

With respect to existing random algorithms discussed above, the main originality of our extra-trees is the combination of the randomization of attribute and discretization threshold and the fact that the discretization threshold is really selected at random, i.e. not on the basis of any score (except for the filtering of very bad tests).

Table 7.4: Extra-trees on the omib problem

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
Single tree	0.1213	75	0.0420	0.0793	0.0789	0.1213	0.0424	0.0789
Single extra-tree	0.1299	221	0.0415	0.0884	0.0854	0.1299	0.0445	0.0853
Extra-tree averaging (25 trees)	0.0676	5450	0.0370	0.0306	0.0381	0.0525	0.0434	0.0090

Table 7.5: Decision stump averaging on the waveform problem

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
1 Decision stump	0.4492	3	0.3910	0.0582	0.1805	0.1882	0.1640	0.0242
bagging with stump (25)	0.3589	75	0.3250	0.0339	0.1328	0.1657	0.1592	0.0065
Random stump averaging (25)	0.2903	75	0.2510	0.0393	0.1219	0.1780	0.1766	0.0014

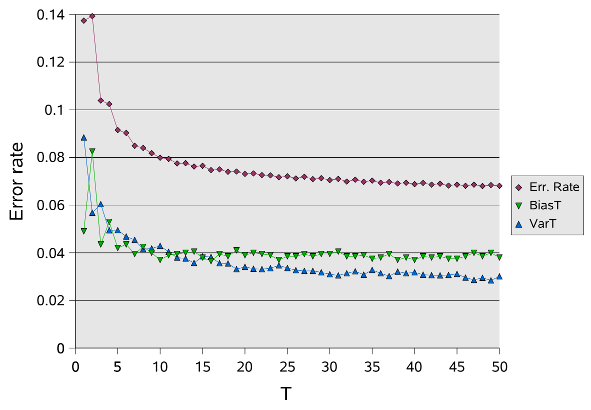
7.4.2 Empirical validation

Table 7.4 compares on the omib problem classical single trees with single extra-trees and the aggregation of 25 extra-trees (by averaging of probability estimates) built from the same learning sample. As expected, randomization increases the variance of probability estimates but at the same time keeps their bias low. However, the increase in variance with respect to classical trees is surprisingly very small on this problem. It announces that the variance artificially introduced by our algorithm is very close to the original variance of decision trees and further that aggregating such trees will give good results. Indeed, we observe from the last line of the table that averaging of these random trees yields a very impressive reduction of variance and no significant change in bias. The part of the regression variance which is captured by randomization is thus here $0.0789 - 0.0090 = 0.0699(88\%)$ which is greater than the reduction obtained by bagging on the same problem.

In terms of classification bias, we get the astonishing result that Tibshirani's bias actually decreases slightly, but this can be explained by a similar argument than the one we used in Section 3.3 to explain the decrease of bias with the small learning sets. Indeed, randomization allows to take into account more combinations of tests and hence increase the representation power of a set of decision trees. This phenomenon is more pronounced and easier to explain in the case of decision stumps. With a decision stump, only one attribute can be taken into account. If the choice of the top node attribute is relatively stable with respect to the learning set randomness, then the majority vote classifier will depend merely on this attribute. On the other hand, the random algorithm of Table 7.3 allows every attribute to appear during induction and hence the majority vote classifier can be more flexible. To illustrate this, Table 7.5 shows on the waveform problem the bias/variance profile of a single (non random) decision stump, of the average of 25 stumps obtained by bagging, and of the average of 25 random stumps. Both bagging and random stumps are reducing the bias but the reduction of bias is significantly more important with random test choices than with bagging. Although both methods reduce the variance, the reduction of error is in this experiment is mostly due to the reduction of bias. Since, the size of the learning set limits the size of the trees, similar arguments apply when dealing with fully grown trees: randomization may induce more flexible majority vote classifiers and hence may reduce bias with respect to bagging.

Figure 7.4 further shows the evolution of the different bias and variance terms as the number of averaged extra-trees increases, again on the omib problem. The behavior of this method is very similar to the one of bagging in Figure 7.1, but with a more important reduction of variance. The bias of probability estimates is essentially constant whatever the number of extra-trees. After some fluctuations for small numbers of terms, Tibshirani's bias also remains constant when the number of extra-trees increases and the reduction of error rates is mainly due to a reduction of variance.

Classical decision trees, bagging, and, aggregated extra-trees are then compared on all datasets in Figure 7.5, top in terms of the square error of probability estimates, bottom in terms of the mean error rates. Average results are summarized in Table 7.6 for the same



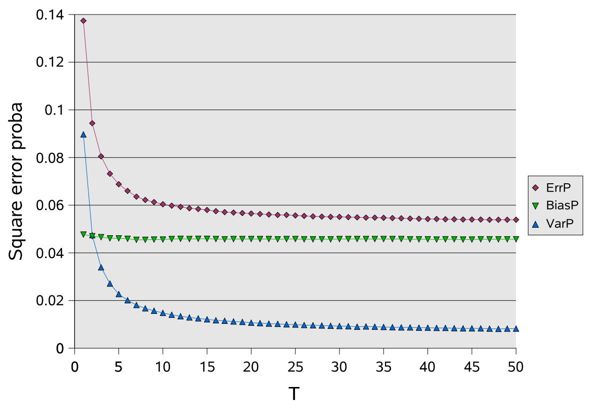


Figure 7.4: Evolution of bias/variance with the number of averaged extra-trees on the omib problem. Left, with error rate, right with square error of probability estimates.

Table 7.6: Bagging and aggregated extra-trees on average on all datasets

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
Single tree	0.2161	89	0.0966	0.1195	0.1326	0.1257	0.0486	0.0771
Bagging (25 trees)	0.1402	1630	0.0969	0.0434	0.0689	0.0624	0.0502	0.0121
Single extra-tree	0.2672	311	0.0925	0.1748	0.1712	0.1401	0.0520	0.0881
Extra-tree averaging (25 trees)	0.1184	7776	0.0845	0.0339	0.0555	0.0592	0.0505	0.0087

three methods as well as single extra-trees. They confirm the results obtained on the omib problem. Extra-tree averaging strongly reduces the regression variance (by 88% in the mean) and slightly increases the regression bias. On the other hand, it reduces both classification variance (by 70% in average) and bias (by 12%) on all datasets. The improvement with respect to bagging is significant and is attributed to both the bias and the variance reductions. Another interesting result is the surprisingly not so bad results of the extra-tree growing algorithm without averaging. In average on all datasets, it only increases the error rates by 5% with respect to the classical decision tree algorithm. On two-norm, it even yields a slight improvement; this further highlights the high variance of decision trees.

7.4.3 Discussion

Even if this algorithm produces better results than bagging (at least on our problems), it suffers one drawback. As thresholds and attributes are not selected on the basis of their discriminative power, induced trees are much more complex than classical trees and ensembles can be very big. In average, the complexity of ensembles of extra-trees is 5 times greater than the complexity of bagged ensembles (from 1630 nodes to 7776 nodes in average).

Hopefully, this increase of complexity still comes with a strong decrease of the computation times because of the strong simplification of the node splitting subroutine. Indeed, local learning sets do not have anymore to be sorted according to all attribute values and only one test has to be evaluated at each test node. To give an idea of what can be gained, we built and tested several models on the two-norm problem on a learning sample of size 5000 and a test sample of size 5000. Results are reported in Table 7.7 in terms of complexity, computation times[4] for growing and testing (in second) and error rates. The construction of an extra-tree which is nevertheless 3 times more complex requires 10 times less seconds than the construction of a tree with the classical algorithm. Building an ensemble of 25 extra-trees is 6 times faster than bagging of 25 trees while the complexity is 5 times higher. Despite the increase in tree complexity, the testing time is also slightly smaller with extra-trees than with bagged ones.

[4]The system is implemented in Common Lisp and runs on a AMD athlon 1GHz microprocessor with 256Mo of main memory. Computation times does not include the time spent in the garbage collector.

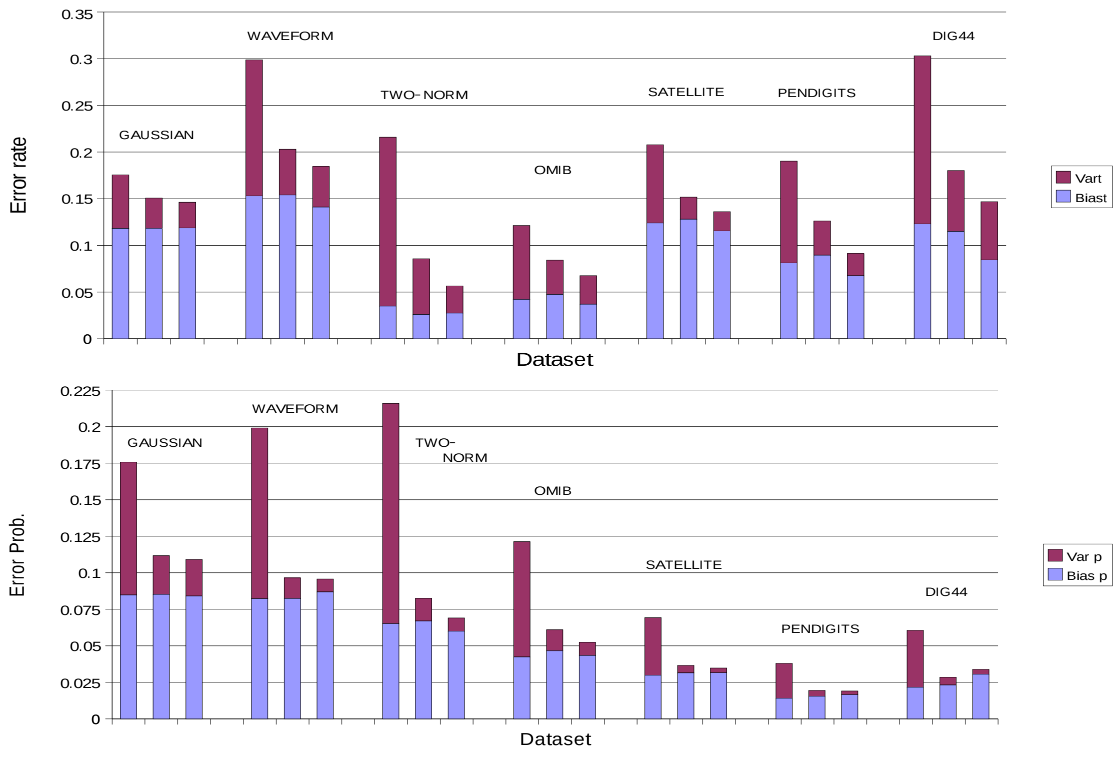


Figure 7.5: Comparison of single trees (left), bagging (center), and, extra-tree averaging (right) on all datasets. Top, in terms of error rates, bottom in terms of the square error of probability estimates.

Table 7.7: Computation times of several variants on the two-norm problem, LS size 5000 and TS size 5000

Model	compl	Growing (s)	Testing (s)	Error(%)
Classical tree	671	19.4	0.4	15.84
Extra-tree	2217	1.7	0.3	16.18
Bagging (25 trees)	11809	290.4	5.5	4.3
Extra-tree averaging (25 trees)	54997	43.5	4.7	4.08

We explain this by the fact that the extra-trees tend to be more balanced than classical trees (because the threshold is always chosen around the mean in the local subset) and hence paths from the root node to terminal nodes are shorter in average.

Taking into account the fact that this algorithm may be easily parallelized (since models are induced independently of each others), this makes such a method a good candidate for tackling very large data sets. In this prospect, it would be interesting to consider further simplifications of the node splitting subroutine to realize good compromise between computation times and accuracy. For example, the computation of the mean (and hence one pass over the data) could be avoided by selecting the threshold value at random among attribute values appearing in the learning sample. Note that on our (non optimized) implementation of extra-trees, it is still less time consuming to build ten random trees than one single classical tree. Hence, while random tree averaging is certainly more efficient than bagging, it also constitutes an interesting alternative to single tree induction, more efficient and more accurate at the same time.

7.5 The question of bias reduction with P&C algorithms

Given the excellent results of tree averaging in terms of variance reduction, bias is now the dominant source of error of this algorithm. So, if we want to further improve accuracy of decision trees, it is necessary to develop approaches able to fight against bias as well as reducing the variance. Although the random tree algorithm does slightly reduce bias, this is only a side-effect of the increase of flexibility provided by randomization. One of the main sources of bias in decision trees is the limitation of the number of attributes which can be taken into account along a branch. Indeed, because of the recursive partitioning, the number of tests is dependent on the size of the learning set. If many[5] attributes contribute to the classification frontier, decision trees will suffer from high bias. There exist extensions of decision trees which improve the representation power of the candidate tests, for example by taking into account several attributes in one single test. The main drawbacks of these approaches are to deteriorate the computational efficiency of the tree growing algorithm and also to increase variance. From our experiments with random tree averaging algorithm, the solution is obvious: further increase the flexibility of decision trees by building a very large set of functional candidate attributes and search among only a small random subset of them for the best one to split each tree node. The questions of the increase of variance will (hopefully) be solved by aggregation and the question of computational efficiency will be (partially) solved by randomization. We have already cited some literature [AD97, Bre01] proposing methods along these lines. Now the success of these approaches is easy to understand from the preceding analysis. In the third part of this dissertation, we will suggest an application of this idea for time-series classification.

Clearly, the very impressive results obtained on our test problems would encourage further research in the context of random tree building. For example, from the theoretical point of view, we would suggest to study the relationship with support vector machines (see [Bur98] for an introduction). Indeed, we see the following analogy between the two approaches: in support vector machines, the problem of bias is solved by applying a transformation from the initial input space into a new potentially very large (often infinite) functional input space where classes are linearly separable. The problem of computational efficiency is solved by considering only destination input space where similarity between objects are in fact easy to compute and expressing the separating hyperplane in the new input space as a function of this similarity measure. On the other hand, the problem of overfitting is solved by keeping the classification *margin* as large as possible. With respect to this approach, the advantage of random tree growing would be that it does not include the delicate a priori choice of a transformation.

Another well known approach to bias reduction is the boosting algorithm. For comparison, we have applied decision tree boosting[6] on the omib dataset. Figure 7.6 plots the evolution of bias, variance and error rate with respect to the number of iterations. These curves should be compared with the same curves obtained with bagging (on Figure 7.1) and with extra-trees (on Figure 7.4). Contrary to these latter methods, we see that boosting indeed reduces both classification bias and variance. On the other hand, it increases the bias of probability estimates. Average results on all datasets are reported in Table 7.8. We observe that the reduction of bias is more important than the reduction of bias by random tree averaging. However, the reduction of variance is also less important and in average, there is no significant difference between the two algorithms in terms of error rate. On the other hand, in terms of computational efficiency the random tree averaging method has the same advantages over boosting than it has over bagging.

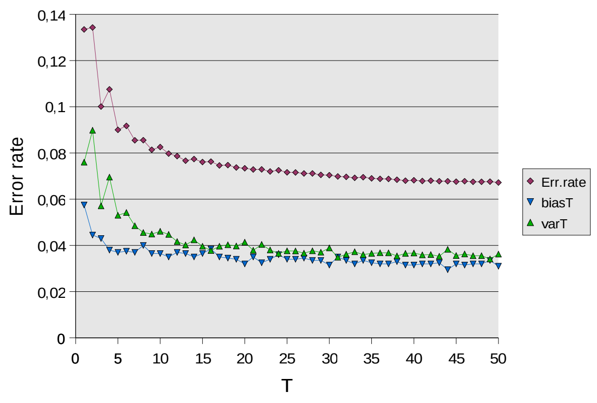
Let us however mention that there exist several variants of boosting (see [Bre96a, HTF01]) which could possibly give an even stronger reduction of bias, and also that randomization could be combined with boosting so as to further reduce variance.

7.6 Conclusion

We have clearly seen that with perturb and combine algorithms, it is possible to reduce very strongly the prediction variance of decision trees. Bagging is not the optimal solution in this goal and it is possible to randomize much more the tree induction algorithm to further improve

[5]i.e. many more than average tree depth for the given learning sample size

[6]Contrary to bagging which can cope with fully grown trees, the boosting algorithm works only with pruned trees. In our trials, we have used the stop-splitting criterion based on the G^2 statistic with $\alpha = 0.005$.



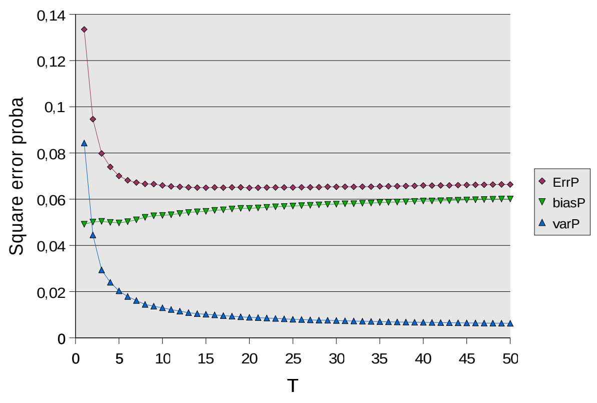


Figure 7.6: Evolution of bias/variance with the number of boosting iterations on the omib problem. Left, with error rate, right with square error of probability estimates.

Table 7.8: Boosting vs bagging and extra-tree ensembles (average over all datasets)

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
Bagging (25 trees)	0.1402	1630	0.0969	0.0434	0.0689	0.0624	0.0502	0.0121
Extra-tree averaging (25 trees)	0.1184	7776	0.0845	0.0339	0.0555	0.0592	0.0505	0.0087
Boosting (25 trees)	0.1217	1065	0.0815	0.0403	0.0615	0.0639	0.0546	0.0093

the accuracy by averaging. Our experiments show that, if accuracy is the only criterion, we can almost fully randomize the tree structure, provided that the tree is sufficiently developed so that its terminal nodes correspond to class pure learning sub-samples (which ensures that the empirical error of these trees is as low as possible). Nevertheless, if our experiments show that very much randomness may be introduced in the decision tree induction algorithm and still improves accuracy, they do not answer to the question of how much randomness should be introduced to optimize accuracy. For example, we have not yet shown that our algorithm is better in this prospect than existing (less) random approaches discussed in Section 7.3. Also, many other perturbation techniques could be imagined that would maybe improve upon our algorithm. For example, we could experiment with different means to randomize the attribute choice or the discretization thresholds, and also study the impact of the threshold s_{th} which determines the strength of randomization. In the context of bagging different variants can also be imagined for example by using smaller learning set sizes for bootstrap sampling and/or using sampling without replacement (like in [GW00] or [FH00b] for example). A further possibility would be to combine bootstrap sampling with randomization (like in [Bre01]).

Despite the very impressive improvement of accuracy, perturb and combine techniques still suffer two drawbacks: they do not preserve the efficiency and the interpretability of decision trees. The increase of computation times depends on the particular technique considered (e.g. in the case of extra-trees, the fact that a certain number of trees must be built is nicely counterbalanced by the fact that growing a single extra-tree is much faster than growing a classical tree), and it is also important to realize that the parallelization of bagging and random tree growing is almost trivial. To circumvent the problem of interpretability, some authors have tried to build a tree which does the same job as a set of trees ([Dom97a]). Others [Bre01, HTF01] have proposed some means to extract interpretable information (such as a ranking of the attributes according to their relevance for classification) from such a set. But, intrinsically, a decision tree ensemble is not interpretable. This is the case especially with our extra-tree algorithm where test choices are not really related to the classification.

At this stage of this dissertation, we have thus developed, on one hand, a way to improve the interpretability of decision trees by stabilizing its parameters but which does not improve their accuracy, and, on the other hand, a way to improve their accuracy by averaging several perturbed models but which deteriorates interpretability and efficiency (to some extent). In the next chapter, we will show that the dual perturb and combine algorithm actually bridges the gap between these two extrema, i.e. improves the accuracy of decision trees but at same time saves interpretability and efficiency by doing a prediction with only one model. This approach will thus lead to a compromise between the two approaches already considered.

Chapter 8

Dual perturb and combine or soft decision trees

Dual perturb and combine is a new algorithm which, rather than generating several perturbed models at the learning stage, consists in producing the perturbed predictions at the prediction stage using only one single model derived from the learning sample. In this chapter, we develop and validate a closed form implementation of this algorithm in the context of decision trees, which yields soft decision trees. Some connections are drawn between dual P&C and other P&C algorithm like bagging. Then, the combination of these soft trees with bagging is assessed. We conclude this chapter by discussing related work on other approaches to build soft tree models.

8.1 Introduction

The generic dual perturb and combine algorithm presented in chapter 4 consists in generating perturbed predictions by perturbing the attribute vector corresponding to a test point and using a single model to produce a prediction for each of these perturbed vectors. It was introduced in Chapter 4 as an efficient alternative to bagging which needs only one model. In the present chapter, we analyze this method in the context of decision trees. We will see that with respect to other P&C techniques, dual P&C has the further advantage of preserving the interpretability of decision tree by making a prediction with only one model.

In Section 8.2.2, we show that the general dual P&C algorithm recalled in Section 8.2.1 is, in the case of decision trees, equivalent to a method that softens the decisions at test nodes. Instead of taking the "hard" (or crisp) decision of propagating an object to the right or to the left successor of a test node, this object is propagated with dual P&C in both directions with some probability. This vision allows to compute efficiently the asymptotic prediction corresponding to an infinite number of perturbations of an attribute vector. Depending on the value chosen for the noise level (the meta-parameter of the dual P&C algorithm), the sharpness of these soft decisions may be adjusted. The automatic choice of the noise level is thus discussed in Section 8.2.3 and the resulting method is validated in Section 8.3. The soft tree vision also allows to draw connections between dual P&C and traditional perturb an combine techniques. However, despite these connections, our experimental results show that dual P&C and bagging have also some complementary behavior and hence their combination may be a further interesting step; this idea is validated in Section 8.4.1. The last section of this chapter provides a discussion of other soft decision tree methods found in the literature.

8.2 Proposed algorithm

8.2.1 Generic dual P&C algorithm

The idea behind dual P&C is very simple: a certain number of perturbed versions of the attribute vector $\underline{a}$ of a test instance are produced. The model (induced in the standard way from the original learning sample) is applied to these vectors resulting in a set of predictions which are aggregated to obtain the final prediction. Considering numerical attributes only, a natural choice to perturb the attribute vector is to add a zero-mean Gaussian noise to it. Thus, from a model f_{ls}, the prediction of dual perturb and combine at a point $\underline{a}$ is given by:

$$f_{DPC}(\underline{a}) = \text{aggr}_{i=1}^{T} f_{ls}(\underline{a} + \underline{\epsilon}^i), \tag{8.1}$$

where aggr is one of the aggregation operators also used in the context of other P&C methods (averaging or majority vote) and $\underline{\epsilon}^i$ are realizations of a random vector $\underline{\varepsilon} = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_m)$, where ε_i are independent random variables distributed according to a Gaussian law $\varepsilon_i \sim N(0, \lambda_i.\sigma_i)$, with σ_i the standard deviation of the attribute A_i in the learning set and λ_i the noise level on this attribute. In our experiments, we have simplified this by using the same level of noise, $\lambda_i = \lambda$, for all the attributes, which has the advantage of minimizing the number of meta-parameters of the algorithm. Although many other perturbation schemes could be imagined, this is the only one considered in this dissertation.

In regression, this procedure with finite T actually computes an approximation of:

$$f_{DPC}(\underline{a}) = E_{\underline{\varepsilon}}\{f_{ls}(\underline{a} + \underline{\varepsilon})\} \tag{8.2}$$

In classification, the majority class for T growing to infinity is given by:

$$f_{DPC}(\underline{a}) = \arg\max_c E_{\underline{\varepsilon}}\{1(f_{ls}(\underline{a} + \underline{\varepsilon}) = c)\}, \tag{8.3}$$

and, when the aggregation operator consists in averaging conditional class probability estimates $\hat{P}_{LS}(c|\underline{a})$, the asymptotic value of 8.1 is:

$$f_{DPC}(\underline{a}) = \arg\max_c E_{\underline{\varepsilon}}\{\hat{P}_{LS}(c|\underline{a} + \underline{\varepsilon})\} \tag{8.4}$$

In the next section, we will present a closed-form implementation of this algorithm in the context of decision tree induction which computes exact expectations according to $\underline{\varepsilon}$ under some simplifying hypothesis.

8.2.2 Closed-form dual P&C of decision trees

The general form of dual P&C can be particularized when models f_{ls} are decision trees. Denoting by $\mathcal{L}_j$ $(j = 1, \ldots, L)$ the leaves of a tree and by g_j, $(j = 1, \ldots, L)$ some numerical predictions associated to these leaves, the prediction given by this tree at a point $\underline{a}$ can be written as:

$$g(\underline{a}) = \sum_j 1(\underline{a} \to \mathcal{L}_j) \cdot g_j, \tag{8.5}$$

where $1(\underline{a} \to \mathcal{L}_j)$ is the characteristic function of the region of the attribute space covered by $\mathcal{L}_j$. On the other hand, the asymptotic prediction given by dual perturb and combine is:

$$\begin{aligned} g_{DPC}(\underline{a}) &= E_{\underline{\varepsilon}}\{g(\underline{a} + \underline{\varepsilon})\} & (8.6) \\ &= \sum_j E_{\underline{\varepsilon}}\{1(\underline{a} + \underline{\varepsilon} \to \mathcal{L}_j)\} \cdot g_j & (8.7) \\ &= \sum_j P_{\underline{\varepsilon}}(\underline{a} + \underline{\varepsilon} \to \mathcal{L}_j) \cdot g_j, & (8.8) \end{aligned}$$

where $P_{\underline{\varepsilon}}(\underline{a} + \varepsilon \to \mathcal{L}_j)$ is thus the probability that a perturbation of $\underline{a}$ reaches the leaf $\mathcal{L}_j$. Denoting by T_1,T_2,...,T_{N_j} the tests along the path connecting the root node to the leaf $\mathcal{L}_j$, the probability that a perturbed case $\underline{a}$ reaches this leaf is the probability that all tests are verified:

$$P_{\underline{\varepsilon}}(\underline{a} + \underline{\varepsilon} \to \mathcal{L}_j) = P_{\underline{\varepsilon}}(T_1(\underline{a} + \underline{\varepsilon}) \wedge T_2(\underline{a} + \underline{\varepsilon}) \wedge \ldots \wedge T_{N_j}(\underline{a} + \underline{\varepsilon})). \tag{8.9}$$

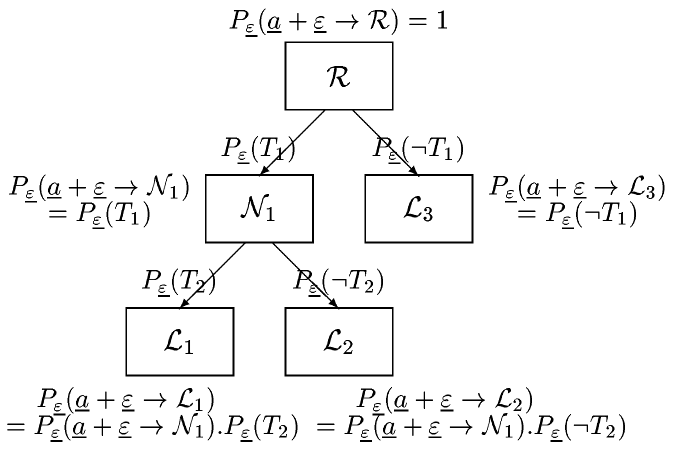

$P_{\underline{\varepsilon}}(\underline{a}+\underline{\varepsilon} \to \mathcal{R}) = 1$
$\mathcal{R}$
$P_{\underline{\varepsilon}}(T_1)$
$P_{\underline{\varepsilon}}(\neg T_1)$
$P_{\underline{\varepsilon}}(\underline{a}+\underline{\varepsilon} \to \mathcal{N}_1) = P_{\underline{\varepsilon}}(T_1)$
$\mathcal{N}_1$
$\mathcal{L}_3$
$P_{\underline{\varepsilon}}(\underline{a}+\underline{\varepsilon} \to \mathcal{L}_3) = P_{\underline{\varepsilon}}(\neg T_1)$
$P_{\underline{\varepsilon}}(T_2)$
$P_{\underline{\varepsilon}}(\neg T_2)$
$\mathcal{L}_1$
$\mathcal{L}_2$
$P_{\underline{\varepsilon}}(\underline{a}+\underline{\varepsilon} \to \mathcal{L}_1) = P_{\underline{\varepsilon}}(\underline{a}+\underline{\varepsilon} \to \mathcal{N}_1).P_{\underline{\varepsilon}}(T_2)$
$P_{\underline{\varepsilon}}(\underline{a}+\underline{\varepsilon} \to \mathcal{L}_2) = P_{\underline{\varepsilon}}(\underline{a}+\underline{\varepsilon} \to \mathcal{N}_1).P_{\underline{\varepsilon}}(\neg T_2)$

Figure 8.1: Example leaves distribution computation

But tests are based on only one attribute and attribute values are perturbed independently (ε_i are independent random variables). So, assuming there is no more than one test using the same attribute along each branch of the tree, equation (8.9) can be factorized into:

$$P_{\underline{\varepsilon}}(\underline{a}+\underline{\varepsilon} \to \mathcal{L}_j) = P_{\underline{\varepsilon}}(T_1(\underline{a}+\underline{\varepsilon})) \cdot P_{\underline{\varepsilon}}(T_2(\underline{a}+\underline{\varepsilon})) \cdots P_{\underline{\varepsilon}}(T_{N_j}(\underline{a}+\underline{\varepsilon})). \tag{8.10}$$

Furthermore, each test T is of the form $[A_i < (\geq) a_{th}]$ where A_i is some attribute and hence the probability of the test being true for a perturbation of the vector $\underline{a}$ is computed by:

$$P_{\underline{\varepsilon}}(T(\underline{a}+\underline{\varepsilon})) = P_{\underline{\varepsilon}}(a_i+\varepsilon_i < (\geq) a_{th}) = P(Z < (\geq) \frac{a_{th}-a_i}{\lambda\sigma_i}), \tag{8.11}$$

where Z is a $N(0,1)$ random variable.

For each test node, the probability expressed in (8.11) can be obtained by elementary table look-up. On the other hand, the computation of $E_{\underline{\varepsilon}}\{g(\underline{a}+\underline{\varepsilon})\}$ can be done efficiently using a forward-backward propagation scheme (see Figure 8.1) : in the forward pass information about the attributes of an object is sent from the root node to the leaves in the form of probabilities of belonging to the traversed nodes, starting with a probability of 1.0 at the root and multiplying this probability by the probabilities associated to the arcs which are traversed; in the backward pass information about the predicted output variable is sent back towards the root with test-nodes aggregating information received from their successors. The complexity of this algorithm (which recursively factorizes equation (8.8)) is thus proportional to the tree complexity.

Substituting g_i for the appropriate predictions in (8.6), this algorithm allows to compute efficiently the values of (8.2), (8.3), or (8.4). For example, in the next section, we will make predictions according to the average of conditional class probability estimates. So the probability of class c for a case $\underline{a}$ will be estimated by:

$$\hat{P}_{DPC}(c|\underline{a}) = E_{\underline{\varepsilon}}\{\hat{P}_{ls}(c|\underline{a}+\underline{\varepsilon})\} = \sum_j P_{\underline{\varepsilon}}(\underline{a}+\underline{\varepsilon} \to \mathcal{L}_j).\hat{P}(c|\mathcal{L}_j), \tag{8.12}$$

where $\hat{P}(c|\mathcal{L}_j)$ is the class c probability estimate at the leaf $\mathcal{L}_j$.

This closed-form implementation has several advantages over the general dual perturb and combine algorithm which consists in averaging a finite number T of predictions obtained from a set of perturbed attribute vectors. First, it makes dual perturb and combine deterministic while the general form is dependent on the random choice of perturbed attribute vectors. Since the general dual P&C algorithm computes a sample estimate of the expectation which is computed by the closed-form algorithm, the prediction given by the general algorithm for finite T will suffer from a larger variance than the closed-form algorithm and hence its average accuracy should be lower, especially if T is small. In terms of computational efficiency, the computation of (8.12) (which is proportional to the tree complexity) requires more time than the computation

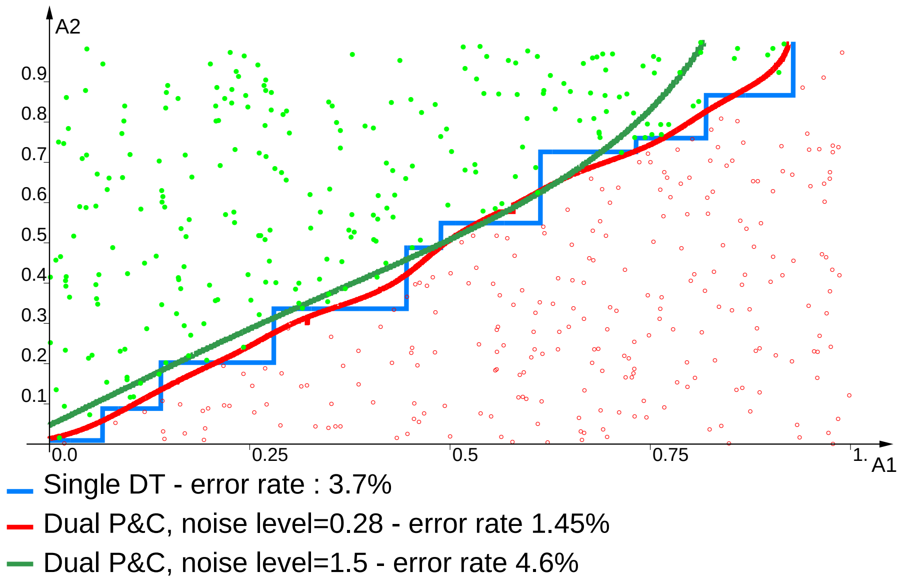


Figure 8.2: Decision tree boundaries with different noise levels

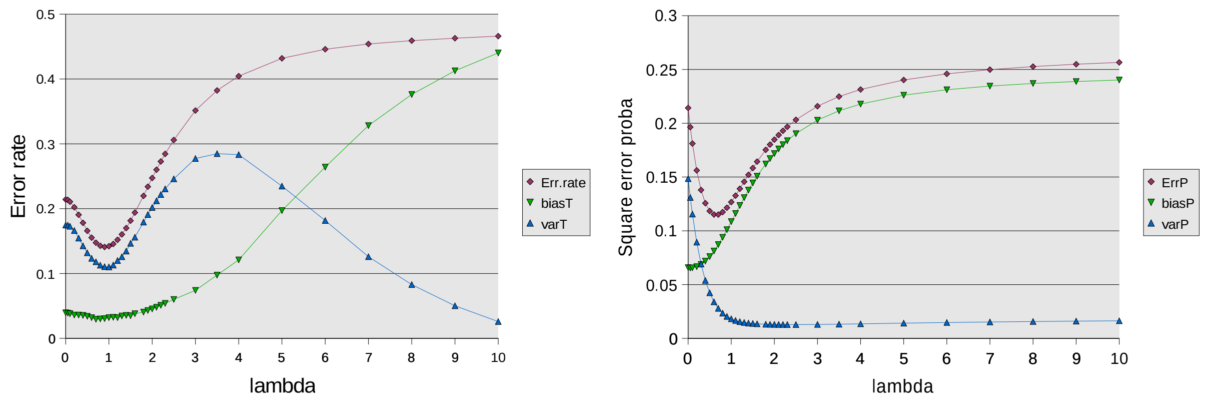


Figure 8.3: Evolution of error, bias and variance with the noise level on the two-norm problem. Left, with error rate, right with square error.

of one prediction with a classical tree (which is proportional to the tree depth). For small T, the general approach could be more efficient while the second one is better when large values of T are considered. Although it would be interesting to study this tradeoff between computational efficiency and accuracy with respect to T, we will nevertheless focus in our experiments on the closed-form implementation which is more appealing in terms of accuracy.

8.2.3 Impact of the noise level on bias and variance

Before discussing the way to choose an appropriate noise level, let us see on a simple example what is the effect of dual P&C with different values of the noise level. Suppose we want to separate empty circles from full circles on Figure 8.2. This figure shows the partitioning done by a single tree and what this partitioning becomes when using dual P&C of the same tree with an optimal noise level of $\lambda = 0.28$ and with a higher noise level of $\lambda = 1.5$. The smoothing ability of dual P&C is pretty clear from this figure and also the over-smoothing when increasing too much λ.

The effect of the noise level may also be approached by a study of bias and variance. Figure 8.3 shows the effect of the noise level λ on the two-norm problem, left on Tibshirani's bias and variance terms and right on regression bias and variance of probability estimates. These graphs are representative of other problems as well. In terms of regression, the bias is monotonically increasing with the value of λ while the variance is monotonically decreasing and this happens whatever the datasets. Indeed, the more we extend the region over which the predictions are averaged, the less flexible is the regression function which can be represented and hence bias increases and variance decreases (at some point, it starts slightly increasing). This compromise is very similar to the compromise which appears in k-NN with the number k of neighbors.

The evolution of Tibshirani's bias and variance terms is more difficult to explain. First, small values of λ reduce both variance and bias, thus resulting in an overall decrease of error rates. When λ further increases, both bias and variance reach a minimum, and start increasing again. Then, while bias is monotonically increasing, the variance goes through a maximum and then starts decreasing. This apparently strange behavior can nevertheless be explained. As trees are unpruned, class probability estimates, $\hat{P}_{DPC}(C|\underline{a})$, given by dual P&C with no noise are either 0 or 1 (tree leaves are mostly pure). When the noise increases, probability estimates move away from these extreme values. For points of the input space which are well classified by the original tree, the regression bias inevitably increases while the classification bias remains unchanged as long as the estimates do not go on the other side of the decision boundaries (corresponding to 0.5 in the two class case). On the other hand, some biased points become unbiased and both their regression and classification bias decrease. As bias is rather small and thus the number of biased points is much smaller than the number of unbiased points, the overall effect is an increase of the regression bias. On the other hand, at the beginning, the classification bias decreases until the number of newly biased points exactly balances the number of newly unbiased points. When the probability estimates get near 0.5, even small regression variance of the probability estimates can make the classification change from one tree to another and hence the classification variance increases. When we further increase λ, probabilities $P_{\underline{\varepsilon}}(\underline{a} + \underline{\varepsilon} \to \mathcal{L}_j)$ and hence $\hat{P}_{DPC}(C|\underline{a})$ becomes more and more independent of the object. Indeed, asymptotically, we have $P_{\underline{\varepsilon}}(\underline{a} + \underline{\varepsilon} \to \mathcal{L}_j) = (0.5)^{N_j}$, where N_j is the depth of the leaf $\mathcal{L}_j$. Hence, $\hat{P}_{DPC}(C|\underline{a})$ becomes an average of the classification over all the leaves and the value of the majority class is highly dependent on the exact tree but is constant allover the input space. Thus, there is a high number of biased points and hence bias is dominating the error. On the other hand, as the "variance" of biased points is in fact negative, the overall effect is a decrease of variance with large value of λ.

All in all, this study shows that there exists an optimal value of the noise level. Many experiments on several datasets (not reported here for the sake of brevity) show that the optimal value of λ is highly dependent of the induced model and learning task and also not related to the learning set size. So, one possible way to determine its value is to devote a validation set to this task. As the curve of error rate is typically convex with only one minimum (like in Figure 8.3), a simple minimization method will be appropriate to determine the optimal value of λ. In our experiments, we use a simple optimization by dichotomy.

8.3 Empirical study

The next step is of course to carry out experiments with dual P&C on our seven classification problems. To this end, we use the closed-form implementation of dual P&C described in section 8.2.2 and average of conditional class probabilities (equation 8.4). In these experiments, the trees are not pruned, so as to minimize bias, and the optimal value of λ is determined individually for each tree from a separate validation set which size is given in Table B.1. Again, even if the assumption of availability of a validation set is not realistic in practice, the goal of these experiments is mainly to study the effect of dual P&C on decision trees in an optimal setting. Figure 8.4 compares dual P&C with bagging and single trees on all datasets. Table 8.1 shows average results.

Dual perturb and combine decreases significantly mean error rates. This decrease of error rates comes from a decrease of variance (by about 33%) and a very slight decrease of the classification bias. In average, bagging does better than dual perturb and combine in reducing the variance. Dual P&C decreases the variance of probability estimates almost as well as

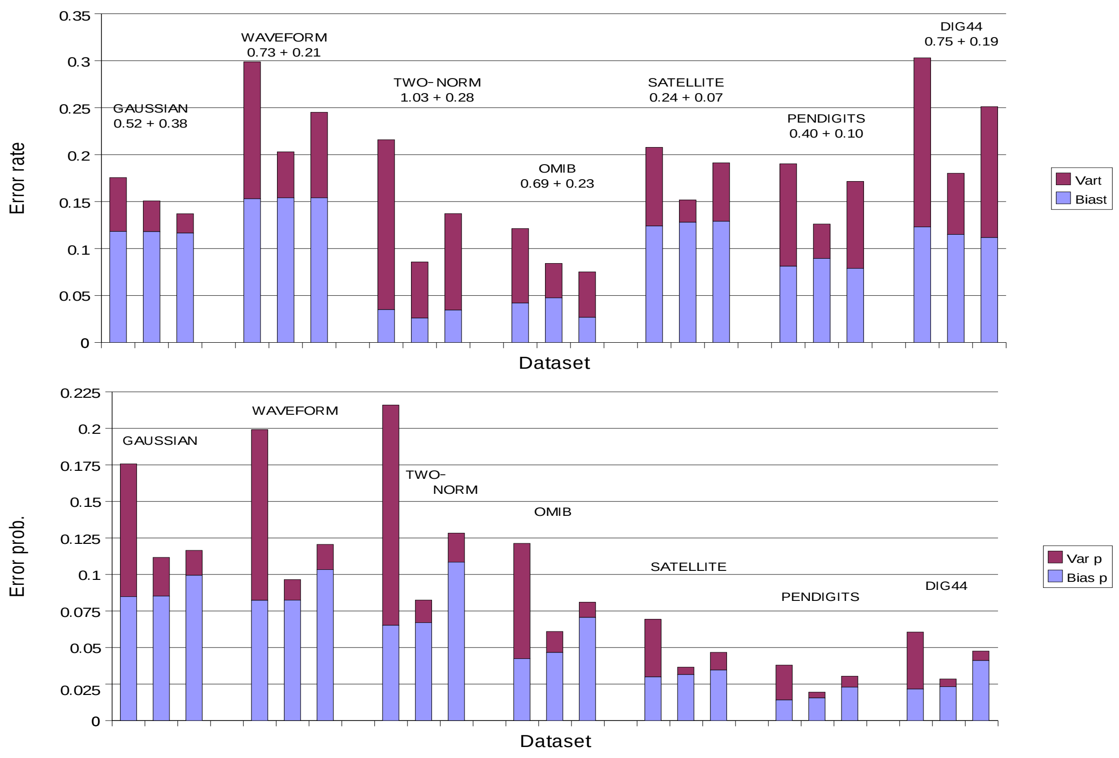


Figure 8.4: Comparison of single trees (left), bagging (center), and, dual P&C (right) on all datasets. Top, in terms of error rates, bottom in terms of the square error of probability estimates.

Table 8.1: Average effect of dual P&C

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
Single tree	0.2161	90	0.0966	0.1195	0.1326	0.1257	0.0486	0.0771
Dual P&C	0.1726	90	0.0931	0.0796	0.0989	0.0816	0.0686	0.0130
Bagging (25 trees)	0.1402	1631	0.0969	0.0434	0.0689	0.0624	0.0502	0.0121

Table 8.2: Computation times of several variants on the two-norm problem, LS size 5000 and TS size 5000

	compl	Growing (s)	Testing (s)	Error(%)
Classical (full) tree	671	19.4	0.4	15.84
Bagging (25 trees)	11809	290.4	5.5	4.3
Dual P&C	671	49.4	13.3	6.72

bagging but at the same time, it increases very strongly the regression bias and thus yields also less accurate estimates of conditional class probabilities than bagging. On two problems however (Gaussian and Omib), dual P&C gives lower error rates than bagging mainly because of a stronger reduction of variance. In the next section, we will try to explain in which situations dual perturb and combine may be superior to bagging.

The average optimal value of λ determined from the validation set and its standard deviation are given beside the name of the dataset in Figure 8.4. Mean values are very different from one dataset to another and on some problems, the standard deviation is quite large, showing that the optimal value is also very dependent on the particular decision trees. Actually, the optimal λ, just like decision tree, suffers from a high variance in some cases.

In terms of computational efficiency, there is an overhead during learning to determine the optimal value of λ. The determination of this value requires several tests of the tree (with different values of λ) using the forward-backward algorithm of Section 8.2.2. The computation of a prediction according to this latter algorithm takes also more time than the traditional propagation of an object in a decision tree (corresponding to $\lambda = 0$). Indeed, the former is generally proportional to the tree complexity while the latter is proportional to the average tree depth. To give an indication of the computation times, Table 8.2 compares dual P&C with a single tree and a set of 25 bagged trees in terms of growing and testing times. Like in the previous chapter, the model is built on the two-norm problem from a learning sample of 5000 cases and tested on a test sample of the same size. In the case of dual P&C, the time for growing includes the construction of the decision tree (which, according to the first line, takes 19.4s) and the determination of the noise level on an independent sample of size 1000 (which, in this particular case, requires 10 tests of the tree). This Table shows that testing a tree with dual P&C is much slower than testing a classical tree. On this problem, the determination of the value of λ is actually the dominant part of the induction stage. Nevertheless, the resulting method is still much faster than bagging and in this particular case, it decreases error rates substantially with respect to a single tree. Furthermore, even if the determination of λ is slow, it is not very demanding in terms of computer resources since objects can be considered sequentially.

8.4 Relationship with other methods

In this section, we discuss the relationships which exist on one hand between dual P&C and other P&C algorithms and, on the other hand, between dual P&C and other soft tree models. The first connection gives some explanation of the good behavior of the dual perturb and combine algorithm (which works like a model averaging technique) and the second one gives a possible explanation of the good behavior of soft tree models (which, like dual P&C, have lower variance than decision trees).

8.4.1 Connection with model averaging techniques

As tree tests on numerical attributes are of the form $[A < a_{th}]$, adding Gaussian noise to the attribute vector before propagating a test case in the tree is more or less equivalent to adding Gaussian noise to the discretization thresholds and leaving the test case unchanged. Actually, this is strictly equivalent if there is no more than one test on the same attribute along each branch of the tree (otherwise, thresholds on the same attribute should be perturbed in the

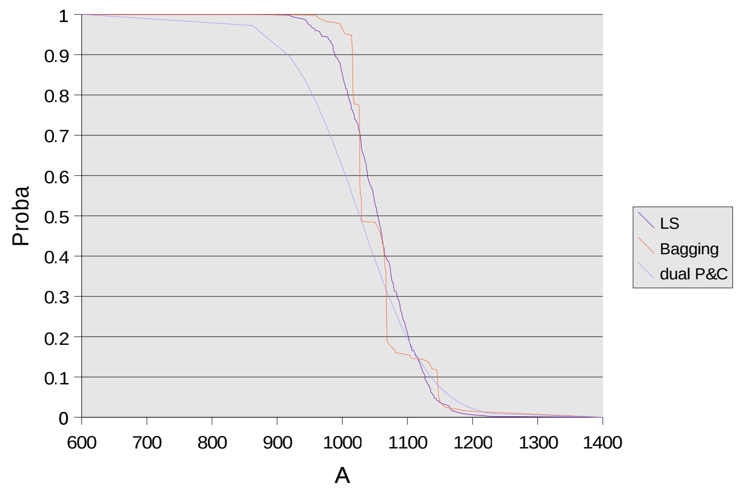


Figure 8.5: Discretization threshold distributions on the omib problem (attribute PU)

same way for the two approaches to be equivalent). Hence for decision trees, dual P&C is very similar to a traditional perturb and combine algorithm restricted to randomizing discretization thresholds.

On the other hand, focusing on discretization thresholds, bagging amounts to generate different decision trees, using the different samples generated by the bootstrap technique, which differ mainly in the precise value of the thresholds. Thus, the bootstrap sampling distribution induces a discretization threshold sampling distribution at each test-node. Consequently, when averaging the predictions of the set of trees created in this way, we essentially average predictions according to the joint sampling distribution of the thresholds at the different test-nodes of the tree.

To illustrate this idea, let us consider the case of a decision stump in a system with only one input attribute A. Neglecting the variance of probability estimates in this example amounts to assume that all the predictions attached to the left nodes (resp. right nodes) are independent of the particular instance of the bootstrap sample (and hence identical to those obtained with the full sample LS). Denoting by g_l and g_r, these (assumed) common values, the aggregated prediction of the bagged stump is given by:

$$g_{BAG}(\underline{a}) = \frac{1}{T}\sum_{i=1}^{T}[1(a \leq a_{th}(ls_i^b)).g_l + 1(a > a_{th}(ls_i^b)).g_r], \tag{8.13}$$

where ls_i^b denotes the i^{th} bootstrap sample from ls and $a_{th}(ls_i^b)$ is the threshold induced from this sample. So, as T grows to infinity, this expression converges to:

$$g_{BAG}(\underline{a}) = P_{LS^b}(a \leq a_{th}(LS^b)).g_l + P_{LS^b}(a > a_{th}(LS^b).g_r, \tag{8.14}$$

where P_{LS^b} denotes the probability according to a bootstrap sample. This equation has to be compared to the prediction of dual perturb and combine particularized to this simple case (from equation (8.8) and (8.11)):

$$g_{DPC}(\underline{a}) = P_{\underline{\varepsilon}}(a \leq a_{th} + \underline{\varepsilon}).g_l + P_{\underline{\varepsilon}}(a > a_{th} + \underline{\varepsilon}).g_r. \tag{8.15}$$

So at a first (rough) approximation, both bagging and dual perturb and combine appear to replace a crisp decision tree by a soft one. In the case of bagging, the thresholds are softened according to their sampling distribution estimated by bootstrap. In the case of dual perturb and combine, thresholds are softened according to a Gaussian distribution. For illustration, Figure 8.5 draws three distributions on the omib problem: $P_{LS^b}(a \leq a_{th}(LS^b))$ given by bagging, $P_{\underline{\varepsilon}}(a \leq a_{th} + \underline{\varepsilon})$ given by dual P&C, and also $P_{LS}(a \leq a_{th}(LS))$, i.e. the true threshold

Table 8.3: Comparison of structure fixed bagging and dual P&C

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
Dual P&C	0.1726	90	0.0931	0.0796	0.0989	0.0816	0.0686	0.0130
Bagging (25 trees)	0.1402	1631	0.0969	0.0434	0.0689	0.0624	0.0502	0.0121
Str. fixed Bag.	0.1861	2168	0.0970	0.0883	0.1084	0.0766	0.0496	0.0270

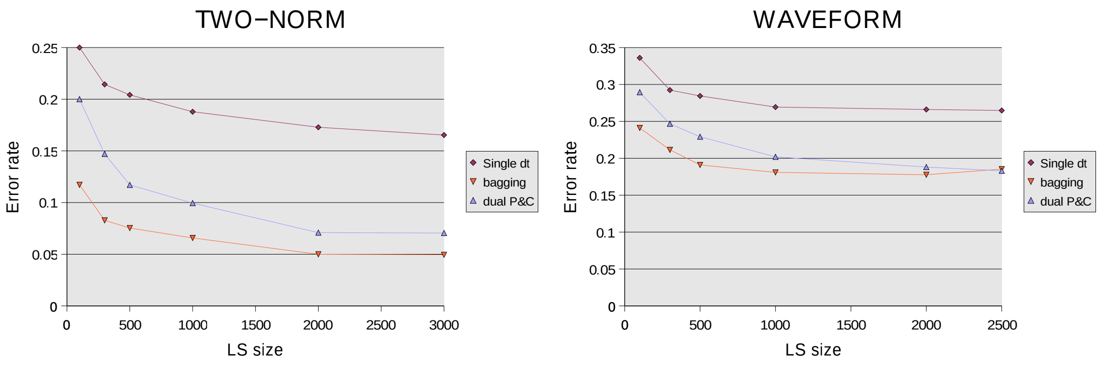


Figure 8.6: Dual P&C versus bagging with respect the the LS size. Left, on two-norm, right, on waveform

distribution (estimated from a high number of learning sets drawn from the pool). Note that both the distribution obtained with bagging and the one obtained by dual P&C are quite far from the true distribution. While the closed-form of dual P&C gives a perfectly continuous distribution, the one induced by bagging remains discontinuous even for a large number T of bootstrap samples since the number of candidate thresholds is limited by the size of the learning set.

Bagging and dual P&C thus work the same way if the tree structure is fixed and probability estimates are not changed. Of course, this is a rough approximation and, in the case of bagging predictions at leaf nodes may change from one bootstrap sample to another and the structure of the trees is also highly variable. To confirm this statement, we carried out another experiment with a variant of bagging which leaves the structure fixed. A maximal tree is first built from the learning set ls and its structure is used to guide the induction from 25 bootstrap samples. The average results obtained with this method on all datasets are compared in Table 8.3 with bagging and dual P&C (under the name "structure fixed bagging"). The mean error of this variant is not as good as bagging but it is close to the mean error of dual P&C. Their classification bias and variance are also quite close. Nevertheless, dual perturb and combine works better than structure fixed bagging. Indeed, in dual P&C, the noise level on discretization is adapted to the learning set while bagging introduces this noise by bootstrap sampling and hence is less flexible. The adaptation of the noise level allows to exchange some regression bias for a reduction of the regression variance which leads to a more important reduction of the classification variance. The counterpart is that structure fixed bagging gives slightly more reliable probability estimates essentially because it only slightly increases the regression bias.

So, contrary to bagging dual P&C only fights variance coming from the discretization threshold. This analysis suggests that dual perturb and combine will give similar results to bagging if the number of relevant attributes is small and the attribute choice variance is small. In our experiments, the two problems where dual perturb and combine is better than bagging, Gaussian and omib, precisely correspond to the two problems with the smallest number of attributes. On the Gaussian problem, there are only two candidate attributes and the classification frontier is symmetric in the input space. On the omib problem, two attributes among the six ones are especially useful for classification. Another consequence is that it is also likely that the error of dual perturb and combine should come closer to the error of bagging if the learning set size is increased since larger learning set size allows deeper trees and thus allows to take into account

Table 8.4: bagging with dual P&C

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
Bagging (25 trees)	0.1402	1631	0.0969	0.0434	0.0689	0.0624	0.0502	0.0121
Bagging + dual P&C	0.1150	1569	0.0889	0.0260	0.0468	0.0770	0.0727	0.0043

more attributes. To check this, we carried out experiments on the two-norm and waveform datasets for increasing learning set sizes. Figure 8.6 plots average error rates of single tree, dual P&C and bagging for increasing learning set size. For each size, the plotted values are the mean error (estimated from the test sample) over 10 learning sets drawn from the pool sample (the same samples are used by each method). On both problems, the error rates of dual P&C get near to the error rates of bagging as the learning set size increases. On the waveform problem, dual P&C even yields better results than bagging for larger samples. These results are also confirmed by the results presented in [Geu01a] and [Geu01c] where we use larger learning sample sizes and indeed dual P&C compares more favorably to bagging.

8.4.2 Combination of dual P&C and bagging

In the previous section, dual P&C and bagging both are described as a way to soften decision trees. However, it is shown by a comparison between structure fixed bagging and dual P&C that the latter technique is better in this goal essentially because it builds really smooth models and also because of its parametric form. So, it could be a good idea to consider a combination of the two methods.

There are several possible combinations of dual P&C with bagging. For example, we could build several trees from bootstrap samples and optimize a different value of λ for each one of these trees before aggregating their predictions, i.e. apply bagging on the top of dual P&C. In this section, we rather choose to apply dual P&C on the top of bagging because it requires to optimize only one parameter λ.

When the output of the model is numerical (regression variable or probability estimates), the prediction given by applying dual P&C on the top of bagging is the following:

$$E_{\underline{\varepsilon}}\{\frac{1}{T}\sum_{i=1}^{T} f_{ls_i}(\underline{a}+\underline{\varepsilon})\} = \frac{1}{T}\sum_{i=1}^{T} E_{\underline{\varepsilon}}\{f_{ls_i}(\underline{a}+\underline{\varepsilon})\} \tag{8.16}$$

and thus is equivalent to applying bagging on the top of dual P&C trees which use the same value of the perturbation level. However, in classification with the majority vote aggregation operator, there is no such equivalence since generally:

$$\begin{aligned} &\arg\max_c E_{\underline{\varepsilon}}\{1([\arg\max_c \sum_{i=1}^{T} 1(f_{ls_i}(\underline{a}+\underline{\varepsilon})=c)]=c)\} \\ \neq\ &\arg\max_c \sum_{i=1}^{T} 1([\arg\max_c E_{\underline{\varepsilon}}\{1(f_{ls_i}(\underline{a}+\underline{\varepsilon})=c)\}]=c) \end{aligned}$$

So, in our experiment, we will continue to aggregate predictions by averaging conditional class probability estimates, since this will allow us to use the closed-form implementation of dual P&C in the context of decision trees. Like for single trees, the optimal value of λ is determined by cross-validation from the validation set whose size is described in Table 5.2.

Figure 8.7 compares bagging with bagging in combination with dual P&C on all datasets. Table 8.4 summarizes average results. On all datasets, the combination improves results with respect to bagging (less significantly on the satellite dataset). The improvement comes from a decrease of both classification bias and variance. The decrease in classification variance is by 35% with respect to bagging alone and it is more than 75% with respect to single trees. Unsurprisingly, the combination gives worse probability estimates than bagging because of an increase of the bias. Nevertheless, the regression variance of these estimates is now almost null

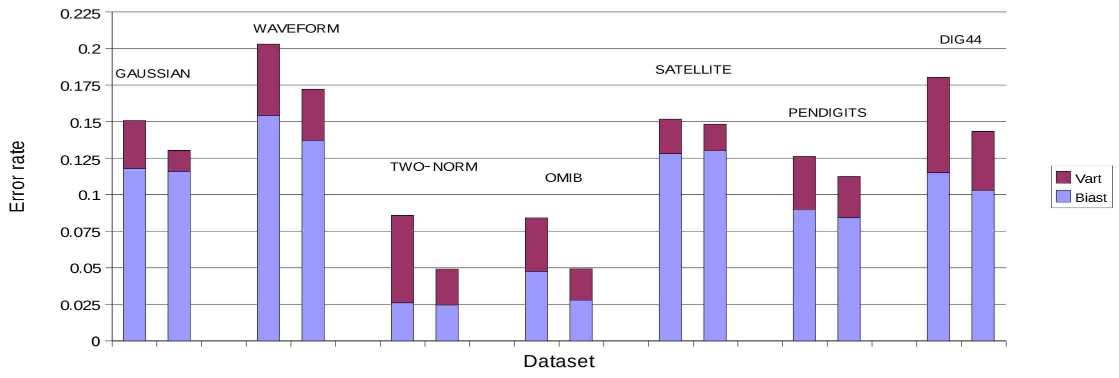


Figure 8.7: Comparison of bagging (left), and, bagging with dual P&C (right) on all datasets in terms of error rates.

with respect to their bias. So it is the ability of dual P&C to increase the regression bias for the benefit of the regression variance that makes the two approaches actually complementary.

8.4.3 Other soft tree models

The soft tree model which results here from the application of dual P&C to decision trees is not new in machine learning. Actually, several methods have been proposed that are somehow related to soft trees.

In the machine learning community, Carter and Catlett [CC87] noticed that a decision tree misclassifies more often examples which are close to the discretization thresholds and so propose to propagate such examples to both successors of a test node and weight the predictions of the corresponding subtrees according to the distance of these examples to the discretization threshold. Quinlan [Qui86] following them provides in its C4.5 algorithm a (quite rudimentary) way to soften discretization thresholds. Dietterich and Kong [DK95] observe that this variant of C4.5 indeed reduces the classification variance of decision trees while increasing their classification bias. In [Fri96], Friedman points out the main drawbacks of the recursive partitioning of decision tree growing: there are more errors at the classification boundaries, few attributes can be taken into account along a branch because of the fast decrease in learning sample size, and the parameters of the cutting are subject to high variance. To circumvent these problems, he proposes a method which recursively divides the learning sample into overlapping subsets and uses voting schemes to aggregate competing predictions (just like soft trees).

Markov tree models [Jor94] and hierarchical mixtures of experts (HME) [JJNH91] use a probabilistic framework to justify soft decisions. In such models, a test object goes to the left and right successors of a test node with probabilities that depends on the input attributes. In HMEs, tests are extended to take into account linear combinations of several attributes and linear models are further attached to terminal nodes. Because of the soft probabilistic decisions, the prediction (a probability distribution on the output variable) given by such a model is continuous with respect to variations of its parameters and hence optimization can be carried out by techniques that use gradient computation (in this probabilistic context, this optimization algorithm takes the form of the popular Expectation-Maximization or EM algorithm). These authors however do not provide an algorithm that learns the structure of the HME and often, a classical tree growing algorithm is used and the resulting tree is softened before optimization.

Soft trees have also been introduced in the context of fuzzy logic under the name fuzzy decision trees. In a fuzzy decision tree [BW99, MCSL01], the weight used to propagate an object to the left and right successors of a test node is interpreted as a fuzzy membership degree to a fuzzy set. Two approaches have been studied to build a fuzzy decision tree. Like for HME, it is possible to build a crisp tree, soften the threshold using fuzzy logic and then globally optimize the parameters of this model (e.g. [MCSL01]). There exist also algorithms which directly integrate fuzzy techniques during the growing stage (e.g. [OW02]) by dividing at each step the learning sample into overlapping subsets (like in [Fri96]).

Eventually, some connections exist also between soft tree models and neural networks. Indeed, the probability of our soft test (8.11) may be represented by a function:

$$P(Z < \frac{a_{th} - a_i}{\lambda\sigma_i}) = \Phi(\frac{1}{\lambda\sigma_i}.a - \frac{a_t h}{\lambda\sigma_i}), \tag{8.17}$$

which is the output of a perceptron with a integrated Gaussian activation function. It is thus possible to combine such simple units into several layers of a neural networks to give a model which computes the same function as a soft tree. Some researchers have proposed to transform a decision tree to a neural networks and then use optimization techniques available in the context of neural networks [Set95]. This parallel between soft trees and neural networks has stimulated the use of optimization techniques developed in the neural networks community (which are also related to the EM algorithm) to optimize fuzzy decision trees (see [MCSL01] or [OW02]).

With respect to all these works, the originality of our approach is to present soft trees as the particularization to decision tree of a general variance reduction technique. With respect to all these works, the next obvious step to improve the accuracy of our algorithm is to independently optimize the noise levels and discretization thresholds used to propagate objects. Nevertheless, good care should be taken in this generalization step. First, although the increase of flexibility provided by a full optimization of the parameters certainly reduces the bias, it is not sure that this will not increase the variance with respect to our algorithm which only makes use of one parameter and still improves very significantly the accuracy of decision trees. And, indeed, we know for example that complex neural networks also suffer from a high variance. Second, with further sophistications, soft trees begin to share several shortcomings of neural networks type of models: the parameters resulting from a global optimization are instable, models are not interpretable anymore, and global optimization is very computationally demanding. In this prospect, we believe that algorithms which still grow soft trees in a recursive and efficient way (for example [OW02]) are more appealing that global optimization techniques.

8.5 Conclusion

Dual P&C is an interesting alternative to bagging in the context of decision tree induction. Although not as good as bagging, it still gives substantial improvements of error rates. At the same time, it saves most of the efficiency of the method and the interpretability of decision trees. Furthermore, it can be combined with bagging where it gives the best results of all the methods considered in this thesis on our classification problems. Nevertheless, the algorithm requires the determination of (only) one parameter, the noise level. Since the optimal noise level seems to be very dependent on the particular model and hence learning sample, a separate validation set seems to be the only reasonable way to select its value.

There remain several open questions about this method. First, in combination with decision trees, we could study the impact of pruning on accuracy. The method could also be combined with extra-trees or boosted trees which are better than bagging. More interestingly, it would be useful to find a way to learn the noise level directly from the learning sample, or at least to give another interpretation of the noise level which would make it less dependent on the particular learning sample. More generally, we could consider different ways of perturbing the attribute vector, for example by using different noise levels for different attributes. Knowledge about the application problem and hence about the interpretation of the attributes may also be used to imagine different perturbation schemes. We will give an application of this idea in Chapter 11 in the context of time-series classification with the segment and combine algorithm. Another interesting direction of research is the application of the general dual perturb and combine algorithm in combination with other learning algorithms. Preliminary studies show that this method can improve the accuracy when applied on the top of neural networks for example. In this context, it could be interesting to consider the derivation of a closed-form implementation similar to the one derived here for decision trees.

Chapter 9

Closure of Part II

In this chapter, we provide a synthetic overview of the methods which have been proposed and studied in the preceding chapters. After a summary of the proposed algorithms, we give some comments about our experimentation protocol and the conclusions which can be drawn on the basis of our empirical studies. Then, a relative ranking of the methods is given, first from the viewpoint of accuracy, second from the viewpoint of computational efficiency. We conclude with some general comments about complementary investigations.

9.1 On the classes of methods investigated

After a careful evaluation of the variance of standard decision tree induction, the main theme of the research described in this part of the thesis has been the design of methods able to reduce this variance. The two main motivations for reducing the variance of an automatic learning method are, on the one hand, the search for higher accuracy and, on the other hand, the improvement of interpretability.

The ideal method would of course yield improvements simultaneously on both criteria. Thus, we started our investigation of variance reduction techniques by a class of methods which intrinsically preserve or improve interpretability, and assessed how much they were able to also improve accuracy. Among the methods studied in this class we find the by now classical post-pruning methods and two novel techniques based respectively on threshold averaging and on threshold softening (the dual P&C method applied to decision trees). The main conclusion we can draw about these methods is that their capability of improving accuracy, while significant, is only rarely competitive with the performances of model averaging techniques.

Hence, in a second step of our research we have relaxed the objective of interpretability, and have focused on the sole objective of reducing the prediction variance. Naturally, we have started our investigation with the analysis of the classical bagging and boosting methods, and then proposed two other approaches based respectively on extremely randomized trees and on the bagging of soft trees. The main conclusion drawn from these investigations is that there is still much room for improvements in this area, be it from the point of view of accuracy, or be it from the point of view of computational efficiency. For example, the extra-tree method outperforms boosting and bagging by far in terms of computational complexity, while remaining fully competitive in terms of accuracy. On the other hand, the combination of bagging and threshold softening yields an improvement in terms of accuracy, which, while small in average in our tests, is still quite significant in some cases.

9.2 On the criteria and their evaluation methodology

Given the motivations of our research, it is natural that the main criterion used to analyse the studied methods has been variance. Of course, since bias and variance are often anti-correlated, we naturally also have assessed the effect of the different modifications on the bias. In order to be able to evaluate variance and bias in empirical studies, it is however necessary to simulate the generation of random learning samples of a given size and build models for several such samples. Therefore, we have chosen a moderate number of test problems for which large enough datasets were available or could be produced by simulation. For the same reason and in order to limit the computational requirements to a reasonable amount, we also carried out the majority of our trials with moderate sample sizes (about 500 objects). Remind that the assessment of one method on one problem is obtained by applying the method to 50 randomly selected learning sets, so as to estimate variance, bias and average error rate with sufficient accuracy. For a single assessment of an ensemble method, like bagging or boosting, this eventually leads to the growing of 1250 different trees.

The systematic obedience to this protocol, while cumbersome in some occasions (e.g. when studying methods such as boosting or bagging), allows us to be confident in the conclusions drawn from these simulations. Of course, it is always possible to argue that the test problems used in our experiments are maybe not representative of all possible types of practical problems or that the medium sample sizes used in our assessments are not representative of many real situations where databases can contain millions of objects. Nevertheless, it is important to note that the experiments reported in this thesis are only a small subset of all the experiments which lead us to our final algorithms (for example, a different methodology and also other problems were used in [Geu01a] to validate the dual P&C algorithm). Eventually, we hope that our experimentation protocol is described with sufficient transparency so as to let the reader decide whether our experiments have been designed with sufficient care to avoid the drawing of overoptimistic conclusions.

The other two evaluation criteria, which were considered for each method, concern respectively interpretability and computational efficiency. These are even more difficult to assess in a precise way than variance and accuracy. In particular, interpretability is a fuzzy and subjective notion and computational efficiency depends a lot on software optimization. For this reason, we have discussed interpretability in a qualitative way only, and concerning computational efficiency we have restricted our evaluation to elementary assessments of the computational complexity and efficiency of our own implementations of these algorithms.

9.3 On the relative ranking of methods

9.3.1 From the viewpoint of bias, variance and average accuracy

Figure 9.1 provides a graphical ranking of the main methods investigated in the second part, by decreasing order of average mean error rate over the seven test problems used in our empirical studies.

The left part of the figure considers the bias/variance decomposition (with Tibshirani's measure) of the error rates, starting in the top with the standard fully developed trees, and ending in the bottom with the combination of bagging and the dual perturb and combine technique. The new methods, proposed in this thesis, are highlighted in bold face so as to distinguish them easily from the reference techniques developed by other authors. The threshold stabilization method represented in this figure is aggregation and it was combined with pruning. Looking at the error bars, we observe that the reduction of error rates is in average about 50% from the worst to the best method. In terms of bias reduction the best method remains boosting, whereas in terms of variance reduction the best technique is the combination of bagging and soft trees, which is also the best method in terms of error rates. Quantitatively, the variance of these methods decreases from 0.1195 in average (fully grown trees) to 0.0260 (Bagging + Dual P&C), which is a reduction of about 80%. Although boosting reduces classification bias more than other variants, the effect is actually only marginal compared to its effect on variance.

The right part of the figure concerns the performances of the different methods when they are used to estimate conditional class probabilities. Again, in terms of variance, we observe a very strong reduction (from 0.0771 for fully grown trees to 0.0043, a reduction of about 95%),

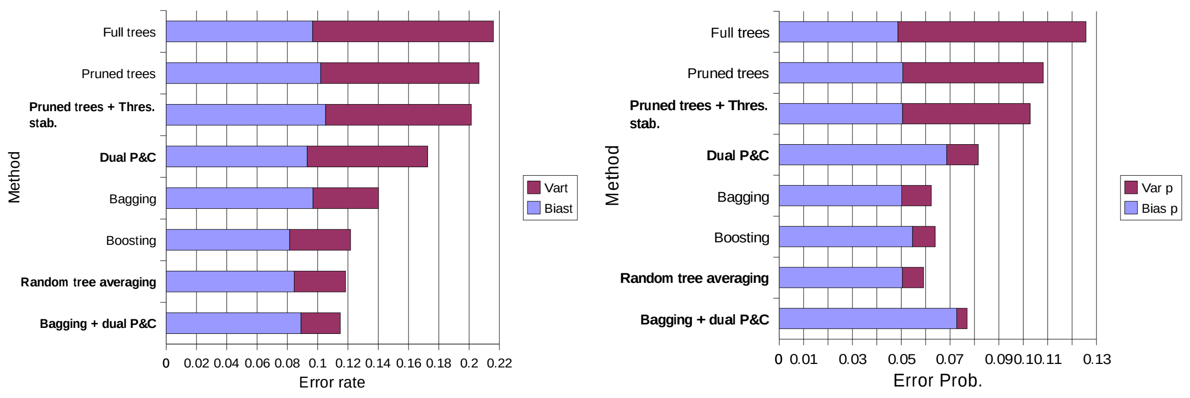


Figure 9.1: Summary of average results with all methods. Left, in terms of error rate and classification bias and variance, right in terms of square error of probability estimates and regression bias and variance.

even stronger than in the case of classification errors. On the other hand, we observe that the dual P&C based methods lead to a significant increase of bias, as well as boosting.

The most important gap in terms of error rates between groups of methods appears between dual P&C and bagging (although we have seen that in some problems, dual P&C can be competitive with bagging). This gap also corresponds to the frontier between methods based on one model and methods that average several of them. It thus corresponds to the limit between interpretable and non interpretable models. From the discussion of Section 8.4.3, we believe that this gap could be made thiner by the introduction in this table of “real” smooth trees (built for example by a fuzzy approach).

Of course, it is certainly possible to get further improvements in terms of variance and to a lesser extent bias[1], but we feel comfortable with the experiments presented in this part of the thesis since they already present a quite broad spectrum of techniques and yield also a broad spectrum of improvements.

In order to check the consistency of these average results throughout the problems, Table 9.1 provides a synthetic view in terms of error rates of the eight approaches on each one of the seven test problems considered in our experiments. The lowest error is highlighted in each column. The table recalls also whenever available the minimal attainable error rates (the error rate of the Bayes classifier corresponding to each problem), so as to assess how far our improvements could possibly be outperformed by other automatic learning methods (say multi-layer perceptrons, support vector machines or k-nearest-neighbor) which evaluation falls out of the scope of this thesis. In terms of relative ranking, in most of the cases, these results are coherent with the average trends depicted in Figure 9.1. Still, there are a few notable exceptions. On the Gaussian problem, Pruning alone is better than most averaging techniques but averaging techniques still give significant improvement with respect to fully grown trees. On the last three problems, bagging with dual P&C, while better than bagging and dual P&C alone, does not go as far as boosting or random tree averaging. In terms of absolute error reduction, we notice that on the first four test problems, the best variance reduction techniques are able to provide more than 60% of the ultimately achievable improvements (given by the error of the Bayes classifier) with respect to standard non-pruned trees.

9.3.2 From the viewpoint of computational efficiency

Eventually, the comparison of several learning algorithms is not complete without a comparison of their computational performance. The question of computational performance optimization

[1]From more detailed results given in the appendices, it appears, for example, that for the first 3 problems the bias is already negligible.

Table 9.1: Synthetic comparison of different algorithms on seven databases (error rates)

Method	Gaussian	Waveform	Two-norm	Omib	Satellite	Pendigits	Dig44
Full trees	0.1736	0.2959	0.2161	0.1213	0.2119	0.1877	0.3016
Pruned trees	0.1397	0.2890	0.2162	0.1189	0.2008	0.1870	0.2947
Pr. trees and Thres. stab.	0.1379	0.2889	0.2064	0.1089	0.1933	0.1773	0.2975
Dual P&C	0.1371	0.2451	0.1372	0.0752	0.1913	0.1715	0.2511
Bagging	0.1507	0.2030	0.0857	0.0842	0.1517	0.1260	0.1802
Boosting	0.1539	0.1960	0.0634	0.0696	0.1388	**0.0900**	**0.1405**
Random tree avg.	0.1463	0.1845	0.0566	0.0676	**0.1361**	0.0912	0.1468
Bagging and dual P&C	**0.1303**	**0.1721**	**0.0492**	**0.0492**	0.1482	0.1124	0.1433
Bayes classifier	0.1185	0.14	0.023	0.0	-	-	-

is by itself a very broad and difficult subject which we have only very marginally addressed in our research, and which definitely deserves further work.

However, in terms of computing times the spectrum of the investigated methods is actually very broad, with a factor of 100 between the fastest and the slowest method among those investigated. Thus, it is still possible to give a gross ranking of the methods in terms of computational efficiency. All in all, and not so astonishingly, the computational burden is an increasing function of effectiveness in terms of variance reduction. The most notable exception to this rule is the Extra-tree method, which is simultaneously among the fastest and the most accurate methods. Among the other methods, we can distinguish among three classes: single crisp trees, ensembles of crisp trees, and Dual P&C based methods.

- **Single crisp trees.** The fastest method is also the least sophisticated one, namely standard fully grown trees. However, pruning is very fast and hence causes only a slight (negligible) overhead with respect to the tree growing algorithm. Similarly, threshold stabilization by aggregation can also be done in negligible time with respect to the original discretization algorithm.

- **Ensembles of crisp trees.** Both bagging and boosting essentially amount to building several standard trees (in our experiments this number is in the order of 25). In the case of bagging, the trees are fully developed but grown on the basis of a bootstrap sample; in the case of boosting, the trees are pre-pruned and built on the basis of the full learning sample with changing weights. So the relative performance of these two methods is essentially problem dependent, but compared to standard full tree growing these algorithms were about 15-25 times slower in our applications.

- **Dual P&C based method.** The use of soft splits and the determination of the degree of softness lead to an increase in CPU of a factor of about 2-4 depending on the method with which this technique is combined. Thus the combination of dual P&C with standard trees is about 3 times slower than the standard trees, whereas the combination with bagging is about 100 times slower than standard trees.

- **Ensembles of random trees (Extra-trees).** Although it builds several models, we have seen in Chapter 7 that the construction of one extra-tree was very fast and that, in our implementation, ten extra-trees are built in less time than one classical tree. Hence, this method certainly constitutes the most promising approach from the point of view of computational efficiency, especially for handling very large datasets.

9.4 About complementary investigations

As we have seen, the design of automatic learning algorithms is a complex multiobjective optimization problem. Among the main (conflicting) objectives in this context, we have average accuracy over a large class of problems, interpretability of models, computational efficiency and scalability, but also flexibility and uniformity of performances. By flexibility, we mean the possibility of easily adapting (or customizing) a method to problem specific features, such as non-standard data representations, missing values, non-uniform mis-classification costs... By uniformity of performances, we mean the fact that a method should not only be good in average over a large class of problems, but it should also continue to work under worst-case conditions.

It is probably not necessary to say that it is impossible to find a method which would be uniformly better than all other (possible, or even existing) automatic learning methods. Thus, given the non-existence of such a free-lunch possibility, we would suggest to continue research in order to determine more in depth the strengths and weaknesses of the most interesting methods among those investigated.

In particular, the dual perturb and combine method and the extra-tree approach deserve further validation and extensions so as to make them fully exploitable in the context of the high diversity of automatic learning problems, and in particular in the context of large scale complex data mining applications. Above all, a very challenging and interesting question which is partially adressed in the last part of this thesis, concerns the adaptation of these methods in order to handle time-series and image data. Indeed, more and more databases containing such data become available for automatic learning and there is still a lot of work to be done before these datasets can be handled in an efficient and fully automatic way. As concerns more specific further developments needed around these automatic learning methods, we postpone their discussion to the last chapter of this thesis.

Part III

Knowledge extraction from time series

Chapter 10

Supervised learning with temporal data

In this chapter, we first give our motivation for studying automatic learning in the context of temporal data. In particular, we discuss the practical relevance of the topic and argue intuitively why automatic learning has some specific problems in this context. We also position this problem as a particular case of the more general problem of handling complex structured data in automatic learning. Then, we formally define the problem of supervised learning with temporal data, particularize it to time series classification, and explain more in detail why automatic learning in this context is difficult. Before concluding we provide a description of datasets used in the sequel, and give a short guide to related literature on time series classification.

10.1 Motivation

The basic supervised learning problem is defined in Chapter 2 as the problem of learning a function of several input attributes which predicts as well as possible the value of an output attribute. Both input and output attributes were defined as random variables taking their values either on the real axis or in a discrete set of symbolic values. Most mature supervised learning algorithms are actually designed for solving such problems. However, in recent applications of supervised learning, algorithms are facing much more complex data (as input attributes but also at the output of the model) which structure does not fit easily to the simple attribute-value framework used in the first part of this thesis. For example, the field of text mining studies methods that extract information from texts and, in particular, techniques to classify documents automatically according to some user's preferences [Coh96]. In bioinformatics [BB98], automatic learning algorithms must be able to handle data in the form of DNA sequences, for example in order to locate or classify genes. In network security, researchers try to build intrusion detection algorithms which aim at identifying abnormal behaviors of users on the basis of a temporal sequence of discrete system events triggered by the user. In this case, the desired output of the model is thus a time varying classification of the user as a normal user or as an intruder [LS98]. Many other applications of automatic learning are also intended to handle images. For example, automatic learning is used in [BAS^{+}98] to detect volcanoes on Venus from satellite images. In [Sen99], the goal is to detect and follow faces in a video sequence, and the desired output of a model is the sequence of positions of different faces in the successive video frames. In all these examples, the input information which must be processed, and in many cases also the output information provided by the model, are clearly more complex than a single numerical or symbolic value.

From the automatic learning point of view, these problems share some common characteristics that we want to highlight. At the basic level, data corresponding to these problems is represented by a large number of low-level attributes (characters in the case of text, pixel values in the case of images, instantaneous values in the case of time series, elementary alleles in the case of genetics, ...) structured in a uni-,bi- or tri-dimensional space. For example, in the case of digital image processing, a 1024×1024 high resolution grey level image would be represented by 1024×1024 low-level numerical attributes. In general, in these applications the original low-level attributes independently give almost no information about the output variable; rather it is the way they interact among each other which defines the global properties on the basis of which classification or prediction models could be formulated easily. For example, in an e-mail filtering application, it is clear that nothing interesting can be said from the fact that an "a" appears in position 10 in a text. Rather, it is the way the characters are combined to form words, and words to form sentences which is important in terms of global interpretation. The high number of low-level attributes is responsible for the main difficulties found when applying automatic learning to these problems. First, if automatic learning algorithms are applied directly on the basis of the low-level representation they need to build very complex models combining in some fashion the large number of low-level attributes in order to extract information from them. As a result, these methods will almost surely suffer from a very high variance.[1] Also, these problems often correspond to very large datasets (especially if the variance is to be kept low) and for computational efficiency reasons, some techniques can not be applied to these problems. Another difficulty is to produce interpretable rules in such cases. Indeed, even supposing that it would be possible to build an accurate decision tree or rule using the low-level attributes (say pixels) to classify objects (say images) the resulting model is likely to be very intricate and essentially impossible to interprete.

Practically, these difficulties can nevertheless be overcome by introducing a priori information about the problem. For example, in a document classification problem, we know from our knowledge of the language that a text can be segmented into words and that it is a good idea to characterize different texts on the basis of the frequency of occurrence of some well chosen words. From this prior knowledge, the goal of supervised learning can thus be reduced to the simpler problem of the choice of the words to combine to give an accurate classification rule. As suggested by this example, we can apply the standard automatic learning methods to these problems, provided that we are able to change the representation of objects by defining new attributes (or *features*) which can be computed from the low-level ones and are more directly related to the output variable. This is quite frustrating, since the most difficult part of the job is in this way carried out during the feature extraction step, which in practice relies highly on human expertise (a priori knowledge about the problem and about the automatic learning method used) and is difficult to formalize. Also, in a growing number of applications such a priori information is not available, or there is not enough time (or money) available to let a human expert work out the synthetic features. In such domains, what is missing is general approaches which would be able to directly use the low-level attributes and without requiring other information would be able to infer the models. Such methods could either explicitly combine an automatic feature generation mechanism with a general purpose automatic learning method, or directly use the low-level features to formulate the models. In this latter case, we would need automatic learning methods which variance does not increase significantly with the number of attributes.

The preceding discussion gives some ideas and motivations around current topics in automatic learning from complex structured data. In our research, we have considered the particular problem where input data are time series and the output attribute is a constant symbolic classification variable. We call this problem the time series classification problem. The possible application domains for time series classification are numerous. In speech recognition, an acoustic speech signal must be mapped to a set of words. In a medical application, we are interested in detecting heart diseases from electrocardiograms (ecg) of patients. Intrusion detection systems aim at classifying user event sequences as normal or not. In all these application fields (especially speech recognition), there exist working systems but these systems make many a priori hypotheses about the application domain and tend to be of low flexibility. In this thesis,

[1]Because of the high number of low-level attributes, it is easy to find models which perfectly separate learning samples but which in terms of generalization are meaningless.

we are interested in designing general, possibly more flexible approaches to handle such problems. Note that we are aware that our goal is quite ambitious, and also that our contribution is only a modest first step towards this goal.

Before focusing on the time series classification problem, we will in the remaining of this chapter define the framework and notation for supervised learning with temporal data, in general, and discuss several typical temporal classification problems so as to circumscribe more precisely the scope of what we call time series classification. Then, in Section 10.3, we discuss more in depth the peculiarities of this problem that could prevent naive approaches from working. Note that (part of) this discussion can more generally be applied to other problems such as image and text classification. The last section of this chapter describes the datasets that will be used throughout the experiments of the other two chapters of this part of the thesis. Note that two of these test problems have been specifically designed to make appear the peculiarities discussed in Section 10.3.

10.2 Problem statement

In this section, we first generalize the framework given in Chapter 2 to handle temporal data in supervised learning. Then, we propose a classification of the problems according to the exact nature of the output attribute. The particular problem which will be tackled in this part of the thesis is the problem of time series classification. This problem and our simplifying hypothesis are described in Section 10.2.3.

10.2.1 General framework for temporal supervised learning

When dealing with temporal data, an object from the universe $\mathcal{U}$ denotes a dynamic system trajectory (or episode, scenario)[2]. In order to describe such a trajectory, the notion of attribute must be extended by allowing time functions as attribute values. A temporal attribute is thus a function of object and time, defined on $\mathcal{U} \times [0, +\infty[$[3]:

$$A(.,.) : \mathcal{U} \times [0, +\infty[\mapsto \mathcal{A}, \tag{10.1}$$

where $\mathcal{A}$ is the codomain of the function $A(o,.)$. In what follows, we denote by $A(o,t)$ the value of the attribute A at time t for the object o. $A(.,t)$ for each $t \in [0, +\infty[$ is thus a random variable and the time function $A(.,.)$ denotes what is called a random or stochastic process. In what follows, we will sometimes drop the o variable from the notation and use respectively $A(t)$ and $A(.)$ to denote these two random values. Of course, non temporal attributes can still be mixed with temporal ones to describe scenarios; to simplify our notation, they will be considered as a particular case of a temporal attribute where the function $A(o,.)$ is constant for any o. In a supervised learning problem, a particular attribute denoted by Y is the output variable and input attributes are denoted by a vector $\underline{A}$, which is now a vector of random processes.

As in Chapter 2, mainly two types of temporal attribute values may be considered:

- Symbolic values: the codomain of the time function is a finite set $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$ of symbolic values. This kind of temporal attribute models a sequence of discrete events (for example, the evolution of the on/off status of some switch or the sequence of discrete actions performed by a user on a computer).

- Numerical values: the codomain of the time function is the real axis $\mathbb{R}$. A numerical temporal attribute may be used to model the continuous evolution of some measurement on the system.

[2]In what follows, we will indifferently use the terms object and scenario to denote an element of $\mathcal{U}$.

[3]We assume that time is continuous, and without loss of generality, that the starting time of a scenario is always 0.

A learning sample is a sample of N objects randomly drawn from $\mathcal{U}$ and described by their attributes. Each object of the learning set is further defined on a finite interval of time $[0, t_f(o)]$ during which the values of its attributes have been observed. Note that, in general, observation times may be different from one scenario to another. Denoting by $A_{[0,t']}(o,.)$ the restriction of the attribute A to the interval $[0, t']$ and by $\underline{A}_{[0,t']}(o,.)$, the same restriction applied to the vector of input attributes, we have:

$$LS = ((\underline{A}_{[0,t_f(o_1)]}(o_1,.), Y_{[0,t_f(o_1)]}(o_1,.)), \ldots, (\underline{A}_{[0,t_f(o_N)]}(o_N,.), Y_{[0,t_f(o_N)]}(o_N,.))). \qquad (10.2)$$

The goal of supervised learning is to induce from a learning set ls a function of object and time $f(.,.)$ which approximates as well as possible the output attribute $Y(.,.)$ for objects of $\mathcal{U}$. Of course, the model $f(o,t)$ must satisfy the following property:

$$f(o,t) = g(\underline{A}(o,.), t), \qquad (10.3)$$

i.e. compute a prediction at time t for an the object o which is a function of its input attributes only.

Discussion

The preceeding formalization has been made in a rather general way so as to cover a broad spectrum of temporal learning problems.

In particular, we have assumed that the time axis is continuous and that temporal attribute values may change continuously through time, because in some applications it is difficult to fix a priori a sampling period (in particular for the output of the models produced by automatic learning). In terms of computer representation, even if the functions are defined on the continuous time axis, they will actually be represented by the combination of a finite number of parameters and some implicit interpolation or function generation mechanism; these parameters could be temporal samples, or expansion coefficients of some set of basis functions, for example.

The output produced by automatic learning is a function of time, in the general case, and this function is not necessarily generated in a causal way from the temporal attribute values. Further restrictions can be introduced when the framework is particularized to some practical (on-line) applications (see below).

10.2.2 Classification of temporal supervised learning problems

Depending of the type of the output attribute Y and the form of the function f, very different problems may be fitted to our general framework. A classification of these problems is given below.

Constant vs temporal output

Besides the distinction between symbolic or numerical output variable, there is a more important distinction in the presence of temporal data:

- $Y(o,t) = Y(o,t'), \forall t, t'$, i.e. $Y(o,t)$ is constant over time. It is the classical supervised problem but now with temporal attributes. The output variable may describe in some way the state of the system at the end or at the beginning of the scenario which should be determined from its observation during some finite period of time. For example, in speech recognition, the goal is to determine the word which has been pronounced from the observation of the acoustic voice signals.

- $Y(o,t)$ is a time function. When $Y(o,t)$ takes its values on the real axis, one common application is time series prediction. In this case, there is only one temporal attribute A and the output is the time shifted version of this attribute $(Y(o,t) = A(o,t+\triangle t))$, where

$\triangle t$ defines the time horizon for the prediction) which has to be approximated at time t from only present and past attribute values. One possible application is the prediction of the evolution of prices in a financial market. When $Y(o,t)$ is discrete, we are facing a segmentation and labeling problem. Indeed, the time interval has to be divided into segments where the classification is constant and each segment has to be labeled with the correct class. One common application is again speech recognition, where it is sometimes needed to segment a phrase before labeling each of its constituting words. Note that segmentation is an application where it is difficult to predict in advance at what times the output may change, so it fits better in a continuous time setting than in a discrete time setting.

Off-line vs on-line model

Another distinction between problems appears when it comes to apply the model in practice. Sometimes, the model must be applied on-line, i.e. be able to receive measurements on the system in real-time and process this information to give a prediction (e.g. forecasted output) at each time step. In such cases, the output at time t must take into account only past and present attribute values, i.e. be of the form (causality):

$$f(o,t) = g(\underline{A}_{[0,t]}(o,.),t). \tag{10.4}$$

In other situations, the model is used off-line to analyze the data (which is supposed to be complete) and then, the desired model can have the more general form:

$$f(o,t) = g(\underline{A}_{I(t)}(o,.),t), \tag{10.5}$$

where $I(t)$ is a subinterval of $[0,+\infty[$ and $\underline{A}_{I(t)}(o,.)$ is the restriction of the attribute vector to this interval. The difference is not only esthetic. Indeed, in the first case, some interesting techniques simplifying the problem are not applicable. For example, normalization of the length in time of the signals and classical Fourier transformation do not make any sense with partially observed signals.

A practical model is sometimes a combination of these two kinds of models. For example, in a real-time speech recognition system, a first model may segment the time series on-line into words and then each of these segments are processed by an off-line model which gives the output word corresponding to the data in this segment.

Early detection problems

In some cases, even though the desired output variable is constant over time, it makes sense to use an automatic learning method which will produce a time varying output. Indeed, for a non temporal output variable, an on-line temporal model could actually produce predictions which may evolve with time as the available information about the system behavior increases. For example, consider an intrusion detection system and suppose that objects represent the sessions of an FTP sever, in terms of sequences of system calls generated by the server. The goal would be to build an algorithm observing a stream of such events (for a given session) and if the session corresponds to an attempt to intrude into the system provide an alarm. In this application, objects are sessions and for the purpose of supervised learning we would use a subset of normal sessions and a subset of intrusion attempts. The objective of automatic learning would be to build a causal function of the input stream of system events, which as early as possible would switch from "normal" to "alarm", so as to cancel the session before the intruder has started taking advantage of his intrusion. Notice that in this application it would be very difficult to determine in advance how many system events would be necessary to generate an alarm (in the case of intrusion); also the time required to detect intrusions in a reliable fashion would most probably vary in practice from one intrusion attempt to another.

The preceding example highlights some specific features of the "early detection problem". In general, the early detection problem corresponds to applications where one seeks to induce a monitoring system that can trigger emergency procedures of a system. Other examples are patient monitoring in a medical application [KCC01] or anticipation of power systems black-out [GW98]. One way to handle this problem by automatic learning is to use a constant (over time) desired output, a causal temporal model, and to incorporate into the search procedure used by the automatic learning method a preference with respect to models which carry out the detection as early as possible. Note that in these problems, accuracy and earliness of detection are usually competing objectives, because in many detection problems the "undesired" behavior tends to become sooner or later obvious to detect. For example, in the work reported in [GW98], we have designed a method based on temporal decision trees for the early on-line detection of instability in electric power systems; in this application the objective was to detect (or predict) instability when it is still time to prevent it, i.e. before its undesired consequences start to show up. In this problem, instability sooner or later translates into a monotone and strong decrease of voltage levels in the power system, and, without bias towards early detection, the automatic learning method would have produced useless models detecting instability when it is already far too late to react.

10.2.3 Time series classification

From the above analysis, it is clear that the introduction of the time dimension in the supervised learning problem increases very much the complexity and the diversity of the encountered problems. In this thesis, we focus on the particular problem of temporal signal classification. Every scenario, o, is classified into one class, $C(o) \in \{c_1, \ldots, c_m\}$ and the goal of learning is to find a function $f(o)$ which is as close as possible to the true classification $C(o)$. This function should depend only on attribute values, i.e. $f(o) = g(\underline{A}(o, .))$, and it should not depend on absolute time values. A consequence of this latter property is that the model should be able to classify every scenario whatever its duration of observation.

Furthermore, we restrict ourselves to numerical attributes and we assume a discrete time representation of such attributes:

$$A(.,.) : \mathcal{U} \times \{t_0, t_1, t_2, \ldots\} \mapsto \mathcal{A} \subset I\!R. \tag{10.6}$$

Each scenario is described by a sequence of vectors: $(\underline{A}(o, t_0), \underline{A}(o, t_1), ..., \underline{A}(o, t_{n(o)-1}))$, where the number of time-points $n(o)$ is possibly object dependent. We further assume that the discretization step of time is constant over time and object, i.e. $t_i = i. \triangle t, \forall i$. This is not a too strong assumption since continuous signals usually are sampled for representation in computer memory and, if the discretization step is not constant, it can be changed by resampling the time signal at the lowest time scale. Usually, in the literature, the term "time series" is devoted to such discrete time and constant discretization step representation of temporal signals.

As the models we will develop here are independent of absolute time values, time instants can most of the time be forgotten. So, in what follows, we will denote by $(a_0, a_1, \ldots, a_N)$ a vector of instantaneous values defining a time series, without explicit reference to the time instants t_i, and we will identify elements of $I\!R^N$, time series of N time steps, and temporal signals defined on the bounded interval $\{t_0, t_1, \ldots, t_{N-1}\}$. Also, although we are essentially dealing with the time series representation, we will however sometimes use the term temporal signal to denote a time function corresponding to an attribute.[4]

10.3 Peculiarities of time series classification problems

A large diversity of problems can be imagined with temporal data in terms of the goal of learning. Even restricting ourselves to time series classification, there are still many different

[4]Moreover, we will (non-orthodoxly) mix sequence, vector, and signal notation where we find it appropriate.

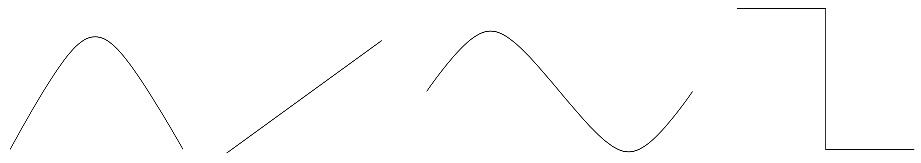


Figure 10.1: Four examples of patterns

possible behaviors that can characterize scenarios of one class with respect to other classes and it is probably too ambitious to aim at building a single learning algorithm which would be able to handle effectively every such kind of behavior. Therefore, we will build up our reasoning here on a class of problems for which we suppose that it is possible to discriminate between classes by extracting some elementary more or less local *patterns* from the temporal signals corresponding to attributes and by combining these patterns into decision rules able to express temporal relationships among them (from a simple temporal ordering of the patterns to a strict temporal constraint on their relative start times).

More precisely, we define a pattern as a limited support signal that satisfies some properties specified a priori (say in the form of a dictionary of elementary signals and a pattern generation mechanism). A few examples of such patterns are given in Figure 10.1: a local maximum, an increasing trend, a cycle, a decreasing step... These patterns play a similar role to the words in the context of text mining, and we assume that it is possible to discriminate between different classes of signals by expressing rules or models of moderate complexity formulated on the basis of a small number of such local patterns.

Note that there are many problems that do not satisfy this locality requirement. For example, a system that discriminates scenarios on the basis of the integration of a temporal attribute (or, in the discrete case, by counting the number of occurrences of a symbol) is impossible to model with only local characteristics. Nevertheless, many of these problems may be solved by introducing new temporal attributes which are functions of the original ones and by detecting some local patterns among these latter. For example, the integration or counting problems may be solved by introducing a new attribute which computes the integration of new functional attribute

$$a'(o,t) = \int_0^t a(o,\tau)d\tau$$

and which would present discriminative local characteristics for the problem under consideration.

What we do not propose at this stage is a technique that would be able to automatically build such functional attributes. Rather, we rely on the user for an a priori good choice of attributes.

Even with this further restriction, the problem of temporal signal classification may be difficult to solve directly with existing standard learning algorithms. What we call here a direct (or naive) approach consists in sampling temporal signals at different moments and using these newly extracted non temporal attributes to induce a model without taking into account their temporal ordering. The limits of this naive approach will be evaluated in the next chapter but, before looking at these results, let us analyze what would prevent such techniques to yield good results with temporal data or equivalently what makes classification with temporal data a challenging task (see also [Kad98] for a similar discussion).

- **Variable number of features.** We have seen that scenarios may be described on different intervals of time ($t_f(o)$ and $n(o)$ are in general variable from one object to another) and thus sampling may result in a different number of features from one scenario to another, which is impossible to tackle with most learning algorithms. Some solutions must be imagined (e.g. truncation of larger signals or normalization of time before sampling).

- **Large number of features.** The number of non temporal attributes extracted by sampling is equal to the number of temporal attributes times the number of time instants.

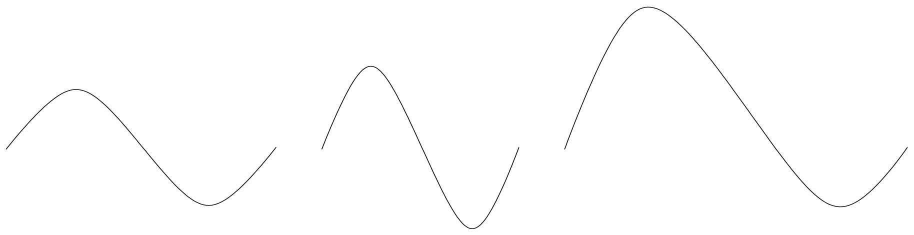


Figure 10.2: Different transformations of the same pattern

If the dynamics of the system are fast, the sampling period must be small in order not to loose information. Hence the number of potential attributes may be high, thereby increasing variance and overfitting.

- **Different time or amplitude scale.** Discriminative patterns may differ in amplitude and duration but still represent the same local properties in the signal. For example, it is clear that the three signals of Figure 10.2 represent similar behaviors (a maximum immediately followed by a minimum) but the second one is obtained from the first one by a linear transformation of amplitude and time scale while the third one is obtained by a non linear transformation of these two axes. Thus, these signals may correspond to very different attribute vectors (in terms of the euclidian distance) when discretized although they may characterize scenarios of the same class.

- **Shift invariant properties.** If we are interested in an on-line model, the absolute location in time of a pattern in a signal should not be relevant for classification (only the relative distance between different patterns may be important). The representation of such shift invariant properties is difficult with the naive approach.

- **Temporal constraints.** Pattern detections may need to be relied by strong temporal constraints for classification purpose. These relationships may also be difficult to model using traditional learning algorithms.

Because of their universal approximation properties, learning algorithms like decision trees and neural networks are theoretically able to model the last three properties of temporal problems. However, this is at the price of very complex models and thus only possible if the learning set is large and if good care is taken to reduce the variance. Before proposing real temporal approaches, the next chapter is devoted to approaches based on sampling which do not really take into account the temporal ordering of attributes.

Note that the characteristics presented here are not the monopoly of temporal problems. For example, image classification models are certainly facing the same problems, further amplified in this domain by the additional dimension.

10.4 Test problems

While there are very large repositories of data sets which can be used to validate classical algorithms (for example [Bay99]), there is not such a large choice of datasets in the temporal domain, and hence many researchers rely on artificial problems. In this thesis, we use six problems for the validation of our algorithms among which four are artificial. These datasets are summarized in Table 10.1 in terms of the number of temporal attributes, number of classes, number of time points (minimum and maximum if it is variable) and the total number of scenarios. Notice that the literature on time series classification via automatic learning is for the time being much more scarce than the literature on standard supervised learning, so we have also incorporated in our datasets some which have been used by other researchers, so as to allow us to compare some of our results with those obtained by others.

Table 10.1: Temporal data sets summary

Data set	Nb. attributes	Nb. classes	Nb. instants	$\#DB$
CBF	1	3	128	796
CBF-tr	1	3	128	5000
Two-pat	1	4	128	5000
CC	6	1	60	600
Auslan	8	10	32-101	200
JV	12	9	7-29	640

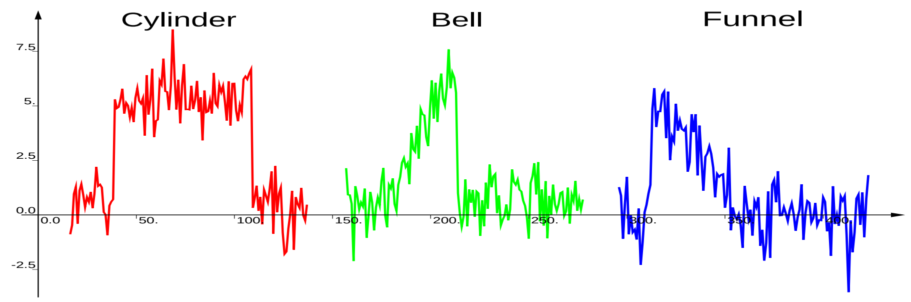


Figure 10.3: An example of each class from the CBF dataset

The first four problems are the artificial ones, specifically designed to present properties typical of temporal problems. With respect to the enumeration of the previous section, CBF is mainly characterized by amplitude and time scale invariant properties. CBF-tr is a variant of this problem which emphasizes the shift invariant property (already present in the CBF problem). In Two-pat, the combination of several patterns is necessary for classification. These problems are described below in quantitative terms. The exact way datasets are generated is reported to Appendix B.

Cylinder-Bell-Funnel (CBF)

This problem was first introduced in [Sai94] and then used in [Man97, Kad99, AR00] for validation of time series classification systems. There is only one temporal attribute and the goal is to separate three classes of objects: cylinder(c), bell(b), and funnel(f). Figure 10.3 shows an example of each class. Each class is characterized by a specific pattern. The cylinder[5] class is characterized by a plateau, the bell class by an increasing (linear) ramp followed by a sharp decrease, the funnel class by a sharp increase followed by a decreasing ramp. To catch some typical properties of the time domain, the start time, the duration, and the amplitude of the pattern are randomized from one scenario to another. A random additive independent Gaussian noise is added to each time point. Note that because of this noise, the residual error in this problem is not necessarily null. As in [Man97], we generate 266 examples for each class.

CBF-translated

Although the start times of patterns are randomized in the CBF problem, the pattern is still always more or less centered in the time interval. In order to enable us to study the robustness of some of our algorithms to cope with shift-invariant properties, we have generated another version of the same dataset by emphasizing the temporal shifting of patterns. In this variant, the durations of patterns are less variable than in CBF but their start time is much more randomized. Figure 10.4 shows several examples of the bell class. This problem should be harder for non temporal approaches than the original CBF problem.

[5]To our knowledge, the names of the class do not refer to some application; they probably are related to the geometrical shape of the signals.

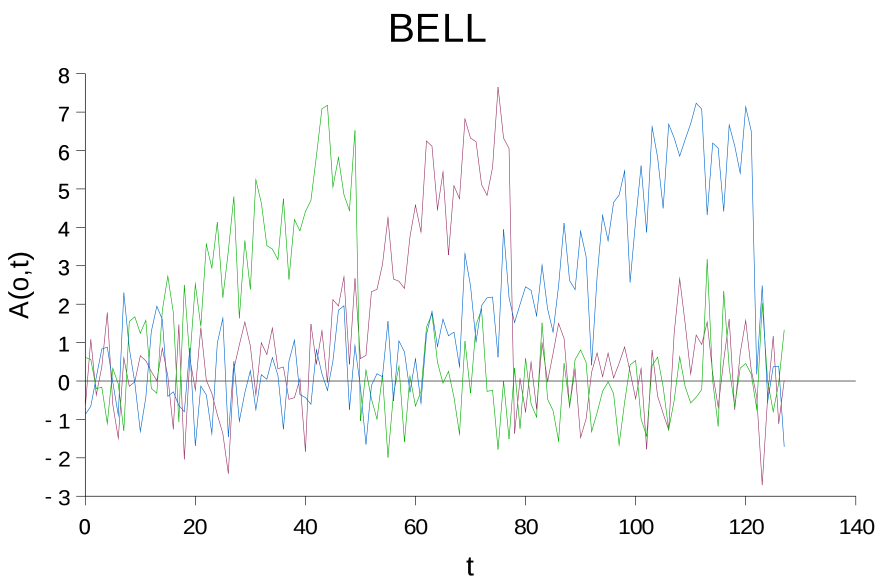


Figure 10.4: Three instances of the bell class in CBF-tr problem

As this dataset will be used for bias/variance computation, we draw a large dataset of 5000 cases which will be divided into a pool set of size 4000 and a test set of size 1000.

Two pattern (Two-pat)

The CBF and CBF-tr problems present some typical properties of the temporal domain but the classification of temporal signals is based on the detection on single simple patterns. In some problems however, it is necessary to combine the occurrence of several patterns for classification. To validate our algorithms with respect to such cases, we designed a simple artificial problem where a class is characterized by the occurrence of two patterns in a definite order. We will call this problem the Two-pat problem.

The problem is characterized by a single temporal attribute and four classes of signals. Figure 10.5 shows one element of each class. Two different patterns are used to define classes: an upward step (which goes from -5 to 5) and a downward step (which goes from 5 to -5). Class UU corresponds to the succession of two upward steps, UD to the succession of an upward step and a downward step, etc. By defining classes in this way, it is necessary to detect the strict succession of patterns to distinguish elements of the class DU from elements of the class UD. The positions and durations of the patterns are randomized (but in such a way that they never overlap). Around patterns, the signal is characterized by a independent Gaussian noise (of mean 0 and unit standard deviation). Note that in this problem, because there is no noise on the pattern and because the noise level is rather small with respect to pattern levels, the residual error is negligible. However, we expect this problem to be especially difficult to handle with classical (non-temporal) learning algorithms.

Like CBF-tr, this problem will be used for bias/variance computation and hence a large dataset of 5000 scenarios is generated.

Control chart (CC)

This data base was proposed in [AM99] to validate clustering techniques and used in [AR00] for classification problems. Objects are described by one temporal attribute and classified into one of six possible classes: normal, cyclic, increasing, decreasing, downward and upward. Figure 10.6 shows two scenarios of each class.. What makes this problem challenging for clustering or classification algorithms in the temporal domain is that some pairs of classes are difficult to separate (cyclic and normal, upward and increasing, downward and decreasing).

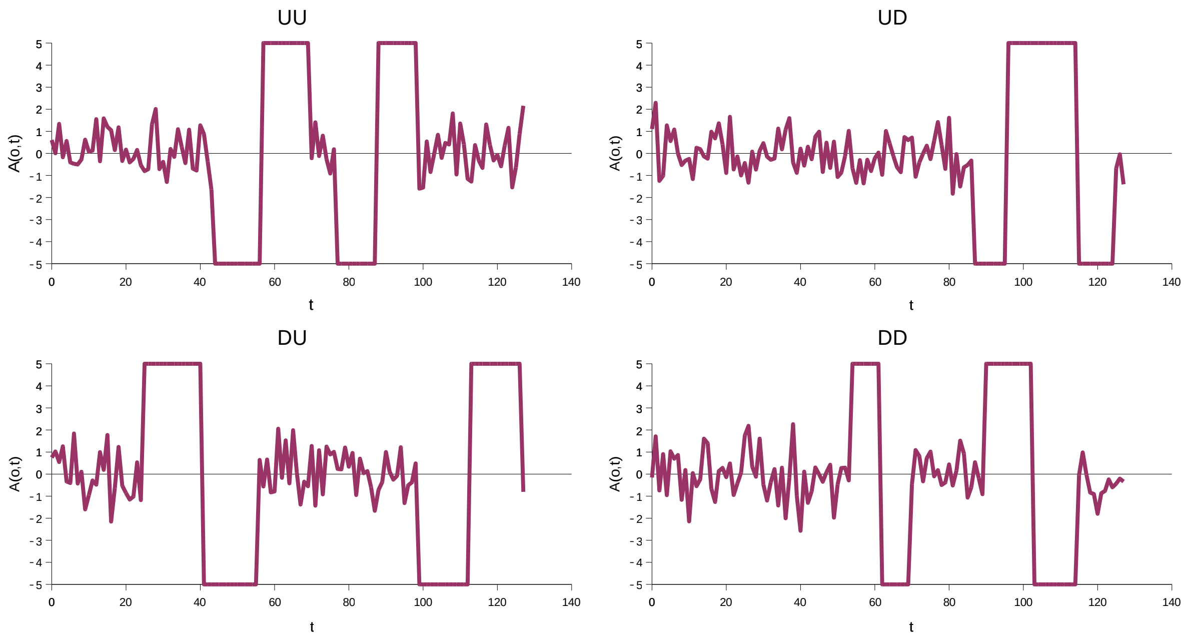


Figure 10.5: An example of each class from the Two-pat dataset

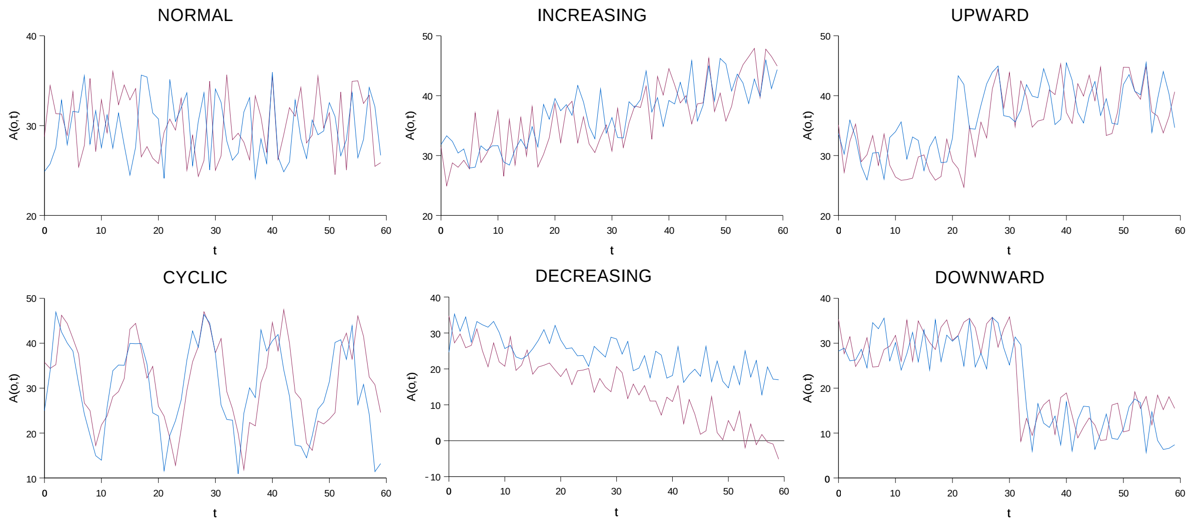


Figure 10.6: Two examples of each class from the CC dataset

The dataset was obtained from the UCI KDD Archive [Bay99] and contains 100 objects of each class. Each time series is defined by 60 time points.

Japanese vowels (JV)

This dataset is also available in the UCI KDD Archive and was built by Kudo et al. [KTS99] to validate their multidimensional curve classification system. The dataset records 640 time series corresponding to the successive utterance of two Japanese vowels by nine male speakers. Each object is described by 12 temporal attributes corresponding to 12 LPC cepstrum[6] transform coefficients. Each signal is represented in memory by between 7 to 29 time points. The goal of machine learning is to identify the correct speaker from this description.

Australian sign language recognition (Auslan)

This dataset was generated by Kadous [Kad99] to validate a time series classification system and used by [AR00]. He recorded instances of the Australian sign language (the language of the Australian deaf community, called Auslan) using an instrumented glove. There are ten classes corresponding to ten different words. Each example is described by 8 temporal numerical attributes: the x, y, and z positions of the hand, wrist roll, thumb, fore, middle, and ring finger bend. The number of time points in each example is variable and ranges between 32 and 101.

10.5 Existing approaches for time series classification

The problem of time series classification has been approached by many researchers from a very broad spectrum of origin: statistics, machine learning but mainly engineering community. However, most of the literature in this domain is applicative and proposes techniques which do not generalize directly to other domains or which need a lot of preprocessing to be applied. Because of this very large spectrum of techniques, the literature is difficult to appraise. So, the brief review given below is at best superficial and certainly incomplete. The goal of this discussion is more to give some idea of the different approaches that exist but that we have not deepened in our search for a general method. Approaches more directly related to our work will be discussed at the end of each chapter.

A very large spectrum of methods for time series classification originate from the speech recognition community. Another application domain for time series classification which presents similar problems and hence solutions is handwriting recognition (where time series record the time varying positions of the hand or the pencil when writing). In these fields, the approach generally adopted is to model the probability density function of the attributes in each class. It is assumed that the signal in each class has been generated by some stochastic process. A model for this stochastic process is learned for each class and then to classify a new scenario, the probability that it has been generated by the model corresponding to each class is computed and the one which has the most probably generated the signal is chosen. Generally, a parametric form is chosen for the model and the parameters of this model are learned from the data. There exist many different techniques to model stochastic processes. The most widely used models in speech recognition are certainly hidden Markov models (see [Rab89] for a tutorial). In such models, it is assumed that the signal at each time step is generated from a distribution conditional of a hidden state random variable which value evolves in time following a first order Markov hypothesis (the state at time t directly depends only on the state at time $t-1$). Much of the recent works in this domain tries to circumvent the limitation of the hypotheses of a simple hidden Markov model (for recent development in HMM-based models see [Ben99]). To model stochastic processes, another method is to take advantage of the many predictive models which have been proposed for time series prediction. These models give a probability distribution on the next value of a time series given the previous values and hence are also

[6]This is a special transform often used in speech recognition.

models of a stochastic process (for a review of predictive models and their use for speech recognition see [Gal98]). Generally, the design of models for stochastic processes uses a lot of a priori information (see [Ben99]). When the model is complex (contains a lot of parameters), this information is even necessary to keep learning algorithms efficient and avoid the problem of overfitting.

Although natural for problems like speech recognition, this approach is not economic since it supposes to model the whole process in each class. On the other hand, discriminative approaches focus directly on a modeling of the classification frontier. The usual approach for time series classification is to extract a set of features and use existing learning algorithms to solve the classification problem. Since using the initial low-level attributes corresponding to each time point is often not appropriate (see the discussion of Section 10.3 and experiments of the next chapter), some techniques may be used to reduce the dimensionality of the problems by selecting good attributes for classification purpose. The simplest way is to use prior information about the problem to manually define useful attributes for classification. This approach is for example adopted in [SLCM98] for a robot learning problem. There exist however general feature extraction techniques such as principal component analysis (PCA) which can be used for time series classification. For feature extraction, one can also take advantage of the many techniques of time series analysis which compute global characteristics of a time series in terms of various statistics and parameters of predictive models (see [BJ76, WG94]). For a review of feature extraction and selection for time series classification, see for example [Has00] and the references therein. For time ordered data, a common approach is also to apply some transform like Fourier transform [Shu82] or wavelet transform [Sai94] and use the transformed signal for classification. According to the particular application domain, one or several of these feature extraction techniques may be useful for classification and often it requires a lot of trials and errors to find the appropriate feature subsets. In this phase, one can also use automatic feature selection techniques (like in [KJ97]) but these techniques do not tell us which initial (large) set of features to choose.

With respect to all this literature, the problem of time series classification has received little specific attention from the machine learning community. Few people have tried to design learning algorithms to solve the time series classification problem. There exists nevertheless some recent works [Man97, Kad99, KTS99, AR00, Ols01a]. With respect to the approach described above, the main difference is logically the focus on the induction of interpretable models and the use of as little as possible knowledge about the application problem. The way to satisfy these objectives is generally to automatically extract features which are used by a standard automatic learning algorithm. So as to yield a general approach, these features are typically automatically adapted in some way to the data. As they are more strongly related to our work, these approaches will be discussed individually in the related work section of Chapter 12.

10.6 Conclusion

In this chapter, we have introduced the time series classification problem and the underlying hypothesis of our approach in this thesis. Our goal in this thesis is to design general approaches, which do not rely on any a priori knowledge about the application domain. In the next two chapters, we tackle this objective in two ways:

- Chapter 11 is devoted to techniques which use only low-level attributes while making very few hypotheses about the ordering of these values. We first apply in this context the variance reduction techniques discussed in the second part of the thesis. Then, we propose a new technique to reduce bias and variance in the context of time series data. The goal of these experiments is to show that the time series classification problem is indeed difficult and its solution needs specific algorithms.

- Chapter 12, on the other hand, proposes to extend the set of candidate tests used in decision tree induction so as to make possible the explicit detection of temporal patterns in time series. The resulting algorithm gives interpretable rules but the price to pay is a decrease of accuracy with respect to dedicated variance reduction techniques.

Although they already provide interesting results, both these approaches only constitute first steps in the goal of building accurate and interpretable methods for time series classification. The main goals of the research presented in this part of the thesis is to open up the domain and give interesting directions for future research. In our conclusion we will give some idea about how we hope to extend our results to the treatment of other complex structured data in machine learning.

Chapter 11

Time series classification with low-level attributes

In this chapter, we tackle the time-series classification problem by the use of classical learning algorithms with low-level features. First, several algorithms are tested on non temporal attributes extracted from time-series by simple sampling approaches. Then, a new method is investigated which reduces both bias and variance by classifying subsequences instead of the whole time-series.

11.1 Introduction

In this chapter, we will experiment with methods that handle directly low-level features with classical learning algorithms. This is certainly the simplest and most direct approach since it does not demand to build new automatic learning algorithms or to extend existing ones. The motivations for such a study are twofold: first, show that temporal problems are often more difficult than non-temporal problems and thus justify the interest in them, second highlight the shortcomings of these approaches and thus narrow our future research in this domain in consequence.

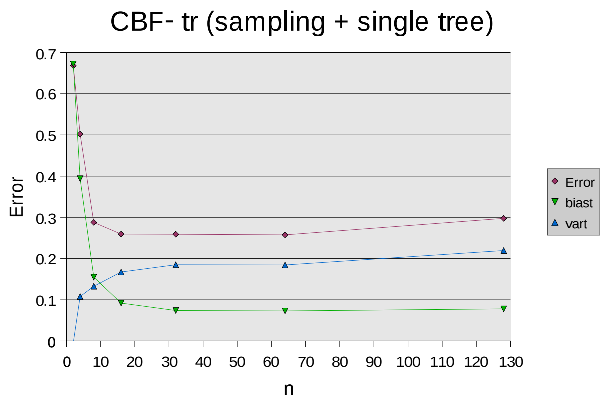
We consider two ways to use a classical learning algorithm with numerical attributes extracted from time-series:

- Simple sampling approaches: the simplest way to extract numerical attributes is certainly to sample the time series at equally spaced instants. Another method is proposed which divides the time series into non overlapping equal length segments and uses the average on each segment as candidate attributes.
- Segment and combine: this method starts by segmenting the original time-series into *overlapping* subseries of a given length, then builds a model for the classification of such subseries (using low-level attributes and standard automatic learning methods), and then classifies a new scenario by aggregating the predictions given by this model for all its subseries. Although this approach uses low-level features as well, it takes advantage to some extent of the temporal structure of the problem; we will show that it improves results very significantly with respect to the first class of methods.

11.2 Simple sampling approaches

The simplest way to use classical automatic learning methods for temporal signal classification is to extract some simple scalar feature sets from signals and then consider them as candidate input attributes for learning a classification model. In this section, experiments are carried out with two very simple sets of features:

- Sampled values of the time series at equally spaced instants. To handle time series of different durations, time instants are taken relative to the duration of the scenario. More



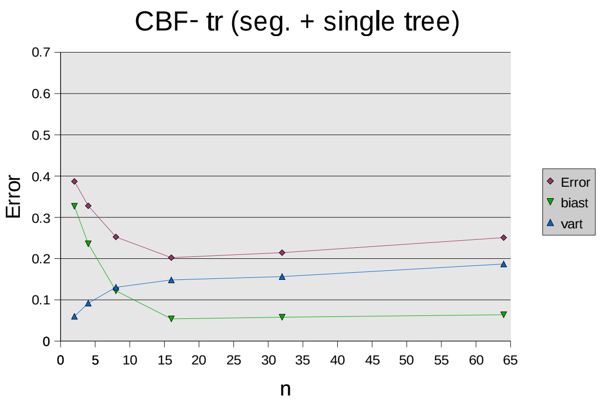


Figure 11.1: Sampling and segmentation on the CBF-tr problem with single trees

precisely, if n is the number of time instants to take into account, each temporal attribute A gives rise to n scalar attributes given by: $A(o, t_f(o)\frac{i}{n-1})$, $i = 0, 1, \cdots, n-1$. In practice, to compute these values from the time series representation, the signals are interpolated in a linear fashion between two successive time points t_i.

- Segmentation of the time interval and averaging (also proposed in [Kad99]). The time axis of each scenario is divided into n segments of equal length and the average value of each temporal attribute along these segments are taken as attributes. In practice, a time series $(a_0, a_1, \ldots, a_{N-1})$ is described by the n following numbers:

$$\bar{a}_i = \frac{n}{N} \sum_{j=i\frac{N}{n}}^{(i+1)\frac{N}{n}-1} a_i, i = 0, 1, \ldots, n-1, \tag{11.1}$$

  when N is a multiple of n (otherwise the last segment receives the remaining time points).

These two approaches give $n \cdot m$ scalar attributes from m temporal ones. Note that the second approach exploits the temporal ordering of values to compute the average on each segment and so is doing some noise filtering.

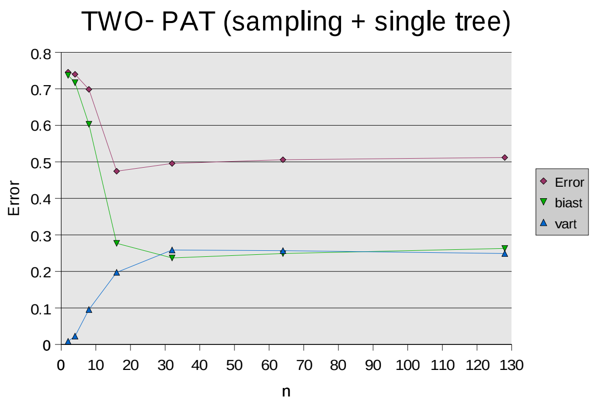
In the first group of experiments, the effect of these two sets of low-level attributes in terms of bias and variance are computed on the CBF-tr and Two-pat problems. Then, further experiments are carried out on the remaining problems. Three classes of methods are tested: single decision trees, variance reduction techniques (boosting and extra-trees averaging), and the one-nearest neighbor rule.

11.2.1 Bias/variance study

Decision trees

We first build decision trees with these two feature sets. Figures 11.1 and 11.2 plot Tibshirani's bias and variance terms for increasing number n of non temporal attributes, respectively on the CBF-tr and Two-pat problems, left with sampling and right with segmentation. In each problem, 50 learning samples of size 300 are drawn from a pool sample of 4000 case and bias and variance terms are estimated from a test sample of size 1000. Trees are fully grown.

One can observe from the figures that bias monotonically decreases with n and variance increases, except for the Two-pat problems and the attributes obtained by segmentation where bias increases for large values of n. Overall, the figures show that there exists an optimal value of n which realizes the best tradeoff between bias and variance. The bias/variance profile corresponding to this optimum is reported in Table 11.1 in each case (the value of n corresponding to this optimum is in the first column). The approach by segmentation is in each case significantly better than the approach by sampling, mostly from the point of view of bias. In both problems,



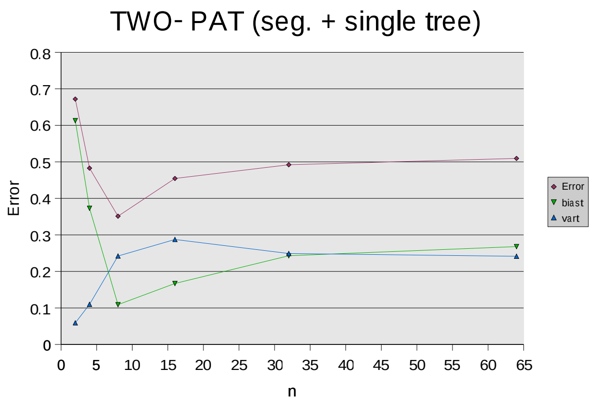


Figure 11.2: Sampling and segmentation on the Two-pat problem with single trees

Table 11.1: Both sampling strategies with single trees on CBF-tr and Two-pat

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
CBF-tr (single trees)								
sampling ($n = 64$)	0.2575	62	0.0730	0.1845	0.1735	0.1717	0.0560	0.1157
Segmentation ($n = 16$)	0.2023	71	0.0540	0.1483	0.1378	0.1349	0.0431	0.0918
Two-pat (single tree)								
Sampling ($n = 16$)	0.4744	139	0.2770	0.1974	0.2637	0.2372	0.1054	0.1318
Segmentation ($n = 8$)	0.3512	128	0.1090	0.2422	0.2193	0.1756	0.0660	0.1096

the variance is high. But the bias is also high, especially on the Two-pat problem. This high bias is also present for the highest value of n, although, in this case, all the information available in the dataset is provided to the decision tree learner.

The high bias of decision trees is here due to the fact that on the one hand the relationship between the low-level attributes and classes can be represented accurately only by very complex decision trees, and on the other hand the rather small sample sizes used here (300) strongly bind the complexity of trees induced by learning. Note that the shift invariant properties of the two problems are certainly the first responsible of the complexity of the relationship between these low-level attributes and classes (we will further discuss this issue in Section 11.2.3).

Extra-trees and boosting

Since variance and bias are high, it seems to be a good idea to repeat the experiment with extra-tree averaging and boosting (which can reduce both bias and variance). Since segmentation is better than sampling, we report here only the results obtained with this approach. Table 11.2 shows the bias/variance profile corresponding to the optimum of error with respect to the number n of segments. In both cases, 25 trees are averaged.

With respect to a single decision tree, the improvement of error with the two techniques is important (especially on the CBF-tr problem) and is mostly due to a reduction of variance.

Table 11.2: Segmentation with extra-tree averaging and boosting on CBF-tr and Two-pat

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
CBF-tr (segmentation)								
Single tree ($n = 16$)	0.2023	71	0.0540	0.1483	0.1378	0.1349	0.0431	0.0918
Extra-tree avg. ($n = 32$)	0.0666	5766	0.0390	0.0276	0.0376	0.0576	0.0506	0.0070
Boosting ($n = 32$)	0.0863	857	0.0410	0.0453	0.0538	0.0710	0.0621	0.0089
Two-pat (segmentation)								
Single tree ($n = 8$)	0.3512	128	0.1090	0.2422	0.2193	0.1756	0.0660	0.1096
Extra-tree avg. ($n = 8$)	0.2357	8833	0.0810	0.1547	0.1526	0.0908	0.0777	0.0131
Boosting ($n = 8$)	0.2621	953	0.1070	0.1551	0.1645	0.1045	0.0943	0.0102

Table 11.3: Segmentation with 1-NN on CBF-tr and Two-pat

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
CBF-tr (segmentation)								
Extra-tree avg. ($n = 32$)	0.0666	5766	0.0390	0.0276	0.0376	0.0576	0.0506	0.0070
1-NN ($n = 32$)	0.0429	-	0.0170	0.0259	0.0277	0.0286	0.0101	0.0185
TWO-PAT (segmentation)								
Extra-tree avg. ($n = 8$)	0.2357	8833	0.0810	0.1547	0.1526	0.0908	0.0777	0.0131
1-NN ($n = 8$)	0.2222	-	0.0780	0.1442	0.1439	0.1111	0.0392	0.0720

Notice that, like in our experiments of Chapter 7, extra-trees yield also here lower error rates than boosting. However, even with this strong reduction of variance, bias and hence error remains quite high on both problems. We conclude that these techniques do not succeed in decreasing sufficiently the high bias of decision trees for these problems, because they are still limited by the small size of the induced trees in comparison to the number of low-level attributes which would be needed to classify these signals.

1-NN

One method which could possibly cope with a large number of relevant low-level attributes is the nearest neighbor technique. Indeed, provided that the vectors of low-level attributes corresponding to different classes of signals are farther away from each other than the vectors of the same class, this method is not directly sensitive to the dimension of the input space. So, it is interesting to try this method in this context. Table 11.3 compares 1-NN[1] with extra-tree averaging (which yields the best result so far). Again, the value of n reported in the table is the one which minimizes the error. The improvement with respect to extra-trees is quite important and is almost only due to a reduction of the bias. Still, the error rates obtained in the case of the Two-pat problem are very high (22%, to be compared with a residual error of 0%).

11.2.2 Further experiments

These two sets of attributes have been tried on the other four problems described in section 10.4 as inputs to the same four learning algorithms. Results in terms of error rates are summarized in Table 11.4. Each line corresponds to a dataset and one sampling approach. Error rates were obtained by ten-fold cross-validation for the CBF, CC and Auslan problems (in which case the result is given in terms of the mean error rate and the standard deviation of this error) and by validation on an independent test set of size 370 for JV[2]. For each method, the error which is reported is the minimum of error with respect to n[3]. The value of n which realizes this error is also reported in the table for each method. The best result in every row is boldfaced.

There are several things to say about this experiment. Like for the first two problems, there exists an optimal value of n which corresponds to the best tradeoff between bias and variance and this value is different from one method to another on the same problem. For the CBF and CC problems, segmentation rather than simple sampling is highly beneficial while on the two other problems, the difference is less pronounced. This can be explained by the presence of an independent Gaussian noise on the CBF and CC problems which is somehow filtered out by segmentation.

Decision trees do not work well but extra-trees averaging and boosting are quite effective. 1-NN is still the best method on CC and CBF but it is beaten by extra-tree averaging on JV and Auslan. On CC, CBF, and JV, the best error rates we get with this simple approach are very good with respect to previously published results on these problems (i.e. with dedicated

[1]With respect to the algorithm described in Chapter 2, we do not normalize attribute values before computing the distance.

[2]This division was suggested by the donors of this dataset [KTS99].

[3]n is chosen in $\{3, 5, 10, 30, 60\}$ on CC, in $\{8, 16, 32, 64, 128\}$ on CBF , in $\{2, 3, 5, 7\}$ on JV, and in $\{2, 4, 8, 10, 16, 24\}$ on Auslan.

Table 11.4: Results with simple sampling methods (error rates in % ± the standard deviation)

Datasets	Single trees		Rand. tree avg.		Boosting		1-NN	
	Err.	n	Err.	n	Err.	n	Err.	n
CC - sampling	6.83 ± 2.29	60	1.67 ± 0.75	30	**1.50 ± 1.17**	60	1.83 ± 1.38	60
CC - segment.	3.50 ± 2.52	10	1.17 ± 1.50	10	1.50 ± 1.38	10	**0.50 ± 1.07**	10
CBF - sampling	7.33 ± 2.49	64	2.25 ± 1.34	32	2.17 ± 1.83	64	**1.16 ± 1.30**	128
CBF - segment.	2.67 ± 1.33	16	0.88 ± 1.26	32	0.67 ± 1.11	16	**0.50 ± 0.76**	16
JV - sampling	14.59	3	**2.97**	2	5.14	3	3.24	2
JV - segment.	12.97	3	4.05	2	6.22	5	**3.51**	2
Auslan - sampling	16.00 ± 8.31	8	**6.50 ± 5.50**	16	7.50 ± 4.03	10	11.00 ± 5.83	16
Auslan - segment.	21.50 ± 9.50	4	**5.50 ± 3.50**	8	8.50 ± 4.50	4	8.00 ± 6.00	4

temporal approaches). On JV, the fact that the best results is obtained by sampling with only two values (measured at the beginning and the end of each scenario) shows that in this problem, the trajectory of the signal is not important for classification and hence improvement will be difficult to obtain with temporal approaches.

Notice that in our reporting of experiments, we have selected the best value of n on the basis of the results on the test sample. Note that, in practice, however, this value should be fixed, either by cross-validation from the learning sample or from an independent validation sample, and certainly not on the basis of the test sample.

11.2.3 Discussion

The results with the simple sampling approaches are not plainly satisfactory. In some problems, we get very good results but in other problems, like CBF-tr and Two-pat, which present typical properties of the temporal domain by construction, the results are far from good even when applying boosting and extra-tree averaging. Also, the best method in terms of accuracy is in several problems the 1-NN method, which has several other well-known drawbacks (in particular the necessity to store the database for prediction and the large computation times required during prediction) which reduce the practical interest of this technique in the context of automatic signal classification tasks which often have some strong real-time constraints.

Another drawback of these approaches based on low-level attributes is that they do not provide interpretable models, which would be useful in order to help a human expert to understand the differences between the different classes of signals. Indeed, among the four studied methods, only decision trees would in principle be able to provide such interpretable results. However, in the present context, their interpretability is very questionable because of the choice of low-level attributes. Indeed, how to understand for example a rule like "if $a(o, 32) < 2.549$ and $a(o, 22) < 3.48$ then $c(o) =$ bell" which was induced from the CBF dataset ? Although very simple and accurate, this rule does not make obvious the temporal increase of the signal peculiar to the bell class (see Figure 10.3).

The lack of accuracy and interpretability with decision trees comes from the fact that some very simple and common temporal features are not easily represented as a function of such attributes. For example, consider a set of sequences of n random numbers in $[0, 1]$, $\{a_1, a_2, ..., a_n\}$, and classify a sequence into the class c_1 if three consecutive numbers greater than 0.5 can be found in the sequence whatever the position. With a logical rule inducer (like decision trees) and using a_1, a_2,...,a_n as input attributes, a way to represent such a classification is the following rule: if ($a_1 > 0.5$ and $a_2 > 0.5$ and $a_3 > 0.5$) or ($a_2 > 0.5$ and $a_3 > 0.5$ and $a_4 > 0.5$) or ... or ($a_{n-2} > 0.5$ and $a_{n-1} > 0.5$ and $a_n > 0.5$) then return class c_1. Although the initial classification rule is very simple, the induced rule has to be very complex. This representation difficulty will result in a high variance of the resulting classifier and thus in low accuracy and

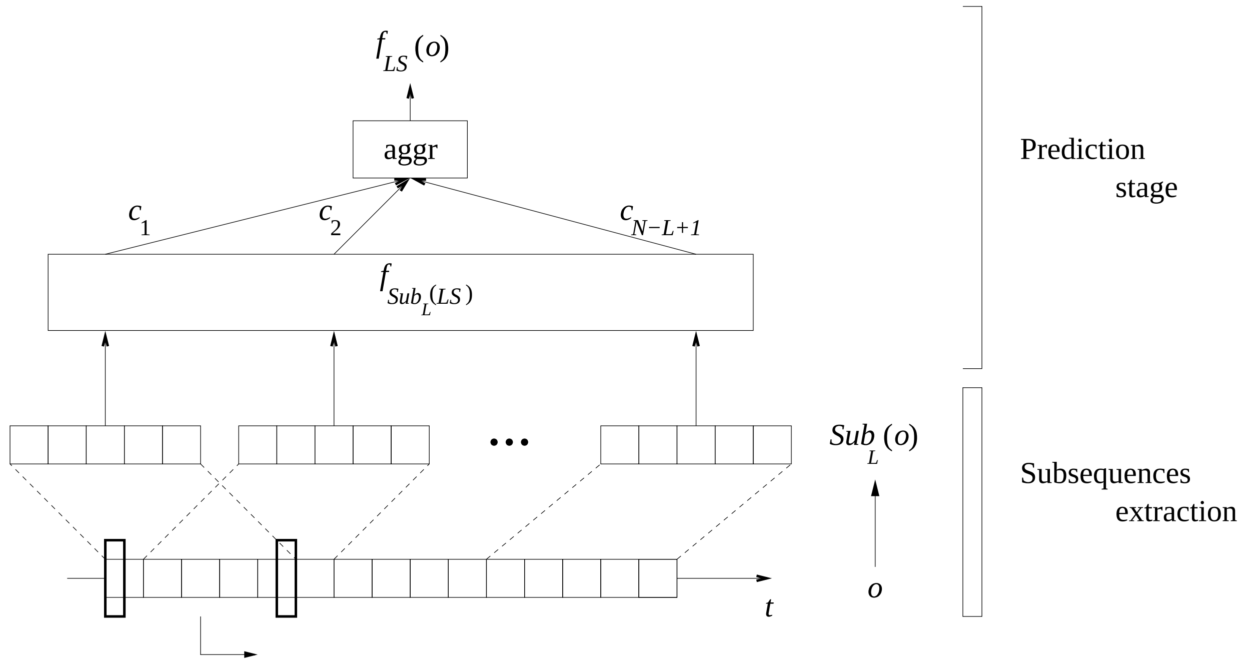


Figure 11.3: Illustration of the segment and combine approach for time series classification (with L=5)

poor interpretability. The use of variance reduction techniques like boosting or extra-trees will not be enough to restore the accuracy. In the next section, we propose a new variance reduction technique which is more adapted to temporal data.

11.3 Segment and combine approach

There are two problems in using low-level features with decision trees. When the number n of features is small, the variance is small but the bias is very high. On the other hand, when n is large, the variance is high but the bias is also high because it is difficult to represent scale and shift invariant properties with low-level attributes and further the tree complexity is limited by the learning sample size. So, all in all, the optimal value of n does not give good classifiers.

In this section, we propose another technique to control the bias/variance tradeoff which is based on subsignals classification. The idea is to segment a scenario into all its overlapping subsignals of a given length L, derive a model to classify these subsignals on the basis of low-level attributes and then combine the predictions given by this model for all the subsignals of a scenario to make a prediction. We call this approach "segment and combine". The algorithm is described in Section 11.3.1 and then validated on our test problems in Sections 11.3.2 and 11.3.3. In Section 11.3.4, we show that some interpretable results may be extracted when subsignals are classified by decision trees.

11.3.1 Algorithm

Learning stage

Table 11.5 describes the algorithm which builds a model for subsequences from a learning sample. This algorithm explicitly assumes that temporal attributes are represented by time series with a constant discretization step $\triangle t$. A window of L time points is defined and moved along the time axis to extract all subsequences of L time points from every scenario of the learning sample (see Figure 11.3). Each subsequence receives the classification of its parent scenario and a learning algorithm is used to build a model of this relation. Any learning

Table 11.5: learning the subsignals classification model

Given a learning set LS, a length of subsignals L and a learning algorithm:

- Compute the set of all subsequences of size L appearing in each scenario $o \in LS$:

$$Sub_L(o) = \{(\underline{A}(o,t_i), \underline{A}(o,t_{i+1}), ..., \underline{A}(o,t_{i+L-1}))|i = 0,1,\dots,n(o)-L\}, \quad (11.2)$$

  and the set of all subsequences of size L appearing in the learning set:

$$Sub_L(LS) = \bigcup_{o \in LS} Sub_L(o) \quad (11.3)$$

  Each object generates $n(o) - L + 1$ subsequences. Each subsequence is fully described by $m \cdot L$ different numbers (where m is the number of temporal attributes)

- Give to each subsequence of $Sub_L(LS)$ the classification of the source scenario.

- Use the learning algorithm to build a model $f_{Sub_L(LS)}$ which reproduces this latter classification using the $m \cdot L$ numbers as input attributes.

Table 11.6: Test procedure with the segment and combine approach

Given a scenario o and a classification model $f_{sub_L(LS)}$ for subsignals of length L

- Compute the set $Sub_L(o)$ of all subsequences of size L appearing in the scenario $o \in LS$;

- Classify each subsignals s in $sub_L(o)$ with the model and aggregate these predictions:

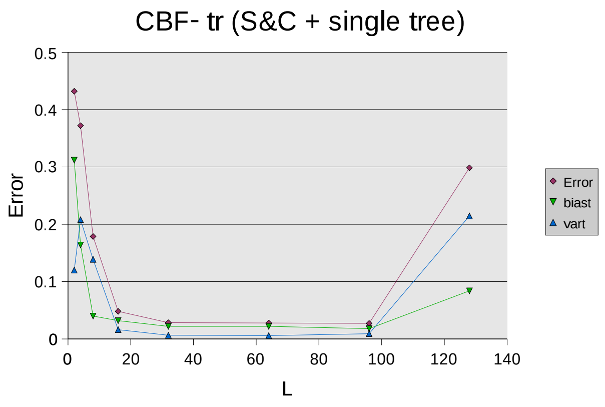
$$f_{LS}(o) = \mathrm{aggr}_{s \in Sub_L(o)}\{f_{Sub_L(LS)}(s)\} \quad (11.4)$$

algorithm could be used in this goal. Input attributes used by this algorithm are all the $m \cdot L$ numbers which describe the subsequence in the time-series representation.

Prediction stage

This model is then used to classify new scenarios as described in Table 11.6. The tested scenario is splitted into all its subsignals of length L and its classification is obtained by aggregating the predictions given for each of these subsignals (see Figure 11.3). Possible aggregation operators are majority vote among the predictions of subsignals or average of conditional class probability estimates assuming the learning algorithm provides such estimates. In our experiments with decision trees, we will only use averages of probability estimates.

With respect to simple sampling approaches, subsequences are here always represented by low-level features at the highest resolution (corresponding to the time step of the time-series representation). The variance is regulated by modifying the length of subsequences. Indeed, when L decreases, the number of features which is proportional to L decreases and the number of subsequence forming the new learning sample increases. Both these effects should reduce the variance. On the other hand, bias should increase when the length of the subsequence decreases. In the limit case where L is equal to the number of time points, only one subsequence is extracted from each scenario and this method is strictly equivalent to the sampling approach with the



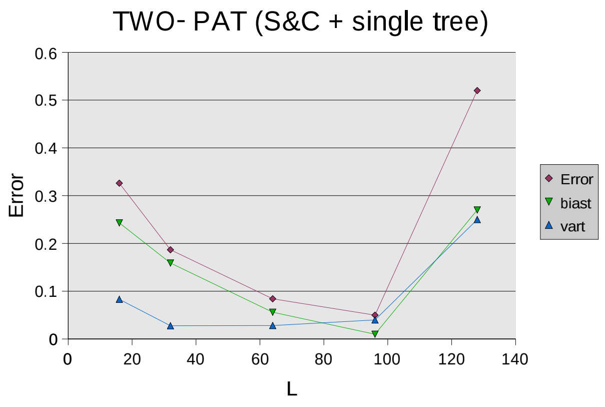


Figure 11.4: bias/variance evolution with the subsequence length L left on the CBF-tr problem, right on the Two-pat problem

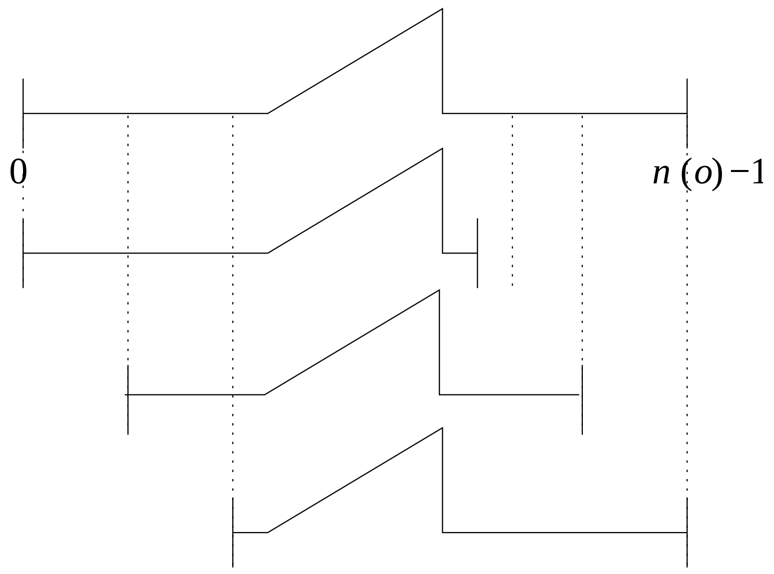


Figure 11.5: A signal and three subsequences extracted from it.

maximum number of time points. Notice that, although this method yields a computational overhead with respect to standard trees, its prediction stage remains intrinsically very efficient.

11.3.2 Bias/variance study

We first carried out experiments on the CBF-tr and Two-pat problems in computing bias and variance. As subsequence classification algorithm, we use fully grown trees. Since the set of all subsequences, $Sub_L(LS)$, may be very large (especially when L is small), its size is limited to 5000 in our experiments. So, If the number of subsequences exceeds 5000 cases, then 5000 samples are randomly drawn from the whole set to build the model.

Figure 11.4 plots Tibshirani's bias and variance terms with respect to the subsequence length L, left on the CBF-tr problem, right on the Two-pat problem. As expected, the bias decreases when the subsequence length increases. However, even for large values of L (close to the maximum number of time points), it is much lower than the bias which is obtained when L is maximum (i.e. exactly matches the number of time points).

The evolution of the bias may be explained by an example. Let us suppose that a scenario is characterized by the pattern at the top of Figure 11.5 which actually may appear at every position in the signal. Figure 11.5 shows three smaller subsequences extracted from the initial scenario. These subsequences correspond to three different positions of the pattern. Segment

Table 11.7: Best results with the S&C approach on CBF-tr and Two-pat

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
CBF-tr								
Extra-tree avg. ($n = 32$)	0.0666	5766	0.0390	0.0276	0.0376	0.0576	0.0506	0.0070
1-NN ($n = 32$)	0.0429	-	0.0170	0.0259	0.0277	0.0286	0.0101	0.0185
S&C ($L = 96$)	0.0272	622	0.0180	0.0092	0.0135	0.0280	0.0244	0.0036
Two-pat								
Extra-tree avg. ($n = 8$)	0.2357	8833	0.0810	0.1547	0.1526	0.0908	0.0777	0.0131
1-NN ($n = 8$)	0.2222	-	0.0780	0.1442	0.1439	0.1111	0.0392	0.0720
S&C ($L = 96$)	0.0499	1015	0.0100	0.0399	0.0390	0.0400	0.0320	0.0080

and combine is thus a way to augment the learning sample with shifted versions of the initial signals. This increase of the learning sample allows to build more complex decision trees which are more tolerant to the shifting of patterns and hence this translates to a reduction of the bias. When L is small, subsequences do not contain enough information about the patterns and the bias is more important and when L is very large, subsequences are very close to the initial scenario and hence the bias increases as well. Note that on the CBF-tr problem, the minimum of the bias appears earlier than on the Two-pat problems. Indeed, from our knowledge of the problem, it is clear that the CBF-tr may be resolved by looking only to small portions of the scenario. However, for the Two-pat problem, a larger portion of the signal must be observed (at least some parts of the two patterns must be seen to classify a scenario).

On the other hand, the variance decreases when L decreases but at some point it also starts increasing. Indeed, for small values of L, it is difficult to separate subsequences of different classes and the model gives class probability estimates which are close to the decision frontier. So, the final decision for a tested scenario may easily goes from one class to another according to the particular learning set choice and this yields a high classification variance.

All in all, there exists an optimal value of L which realizes the best bias/variance tradeoff. The bias/variance profile corresponding to this value on both problems is reported in Table 11.7. For comparison, the best errors obtained by extra-tree averaging and 1-NN are reported in the same table. The improvement obtained with segment and combine is very impressive, especially on the Two-pat problems. It comes from the reduction of both bias and variance. Another advantage of this technique is that the decision tree that classifies subsequences is very small with respect to the size of the set of extra-trees. With respect to 1-NN, a scenario is also classified in much less time and this classification needs only access to a rather compact set of trees.

11.3.3 Further experiments

We carried out further experiments on the other problems, with fully grown trees and with extra-tree averaging. The same limitation to 5000 of the number of subsequences for learning was applied. Table 11.8 reports the best result according to L[4] for each problem and method.

Like for the CBF-tr and Two-pat problems, the value of L which realizes the best compromise on each problem is close to the maximum length of the scenarios. Thus, from our experiments it looks as if it was sufficient to take the length L a little bit (say 30%) smaller than the maximal length of the scenarios to improve significantly the results. Whether this remains true in a broader context remains to be investigated.

With respect to the direct use of low-level attributes with single decision trees, the improvement with segment and combine is very important on all problems. On CC and Auslan, segment and combine with single trees also improves the accuracy with respect to extra-trees and 1-NN with sampling approaches. On CC, the improvement is not very important since the error was already close to the minimum with 1-NN. On Auslan, however, the improvement is important

[4]L is chosen in $\{3, 5, 10, 30, 45, 60\}$ on CC, in $\{8, 16, 32, 64, 96, 128\}$ on CBF , in $\{2, 3, 5, 7\}$ on JV, and in $\{2, 4, 8, 10, 16, 24\}$ on Auslan.

Table 11.8: Results with S&C on small datasets (error rates in % ± the standard deviation)

Datasets	S&C with single trees		S&C with Extra-tree avg.	
	Err.	L	Err.	L
CC	0.33 ± 0.67	30	0.66 ± 0.82	45
CBF	1.50 ± 1.46	96	1.25 ± 1.37	96
JV	7.84	5	2.16	2
Auslan	3.00 ± 4.00	16	1.00 ± 2.00	24

(from 5.5% to 3% error rate). On the other hand, on CBF and JV, extra-trees or 1-nn with sampling give better results than segment and combine with decision trees. Nevertheless, a more powerful learning algorithm can be used in combination with segment and combine to further improve the results. To see if something can be gained this way, we ran segment and combine with extra-tree averaging (of 25 trees) in the same conditions on all problems. On CBF and CC where we already get almost perfect score with simple sampling or segment and combine with single trees, the use of extra-trees does affect accuracy. On JV, with subsequences of length 2, the error rate becomes 2.16%. On Auslan, the error rate was reduced to $1 \pm 2\%$ with a subsequence length of 24. This shows that further improvements are still possible with this algorithm.

11.3.4 Interpretability

Some authors [Oat99, DLM+98] have proposed methods to extract a small subset of relevant subsequences from a set of time series (or a long one) which then can be used for different purposes, e.g. classification or rule discovery (i.e. finding small groups of subsequences which frequently appear together in a time series). As the number of such subsequences can be very huge, these authors have proposed to first use clustering to find relevant prototypes among subsequences. They fix a subsequence length, L, and use a distance measure (e.g. Euclidian or dynamic time warping, which is a distance measure between time series invariant to change in time scale) and a clustering algorithm (e.g. k-means) to cluster all time series subsequences of length L. The relevance of these prototypes for a given purpose is then assessed a posteriori. If we want to find useful subsequences for classification, there are two drawbacks with this approach: we do not know the right number of clusters and we are not ensured to find discriminative patterns with non-supervised clustering.

Actually, a side result of segment and combine with decision trees is that it provides a clustering of subsequences. Indeed, to some extend, a decision tree gives a partition of its input space into regions where the classification is constant. In the context of segment and combine, objects which are classified by the decision trees are subsequences of a given length L. So, each branch of the tree tries to characterize a set of subsequences which are representative of one class. Therefore, at no additional cost, the decision tree classifier gives clusters of subsequences which are as pure as possible in terms of the output classification variable. To visually inspect these clusters, a prototype can be computed by averaging the subsequences which reach a terminal node (attributes per attributes and time points per time points).

To illustrate this idea, we have built a tree for classifying subsequences of length 32 on the CBF problem. This tree was fully grown from 2500 subsequences extracted from a learning sample of size 500 and then pruned from 2500 other subsequences. Figure 11.6 shows the four prototypes corresponding to the four terminal nodes which contain the largest number of subsequences. The majority class in this node is given on the top of the prototype, as well as the number of subsequences of this class and the total number of subsequences in the node. When the node is not pure, only elements of the majority class have been used to compute the prototype. The first prototype corresponds to a “garbage” node which contains actually

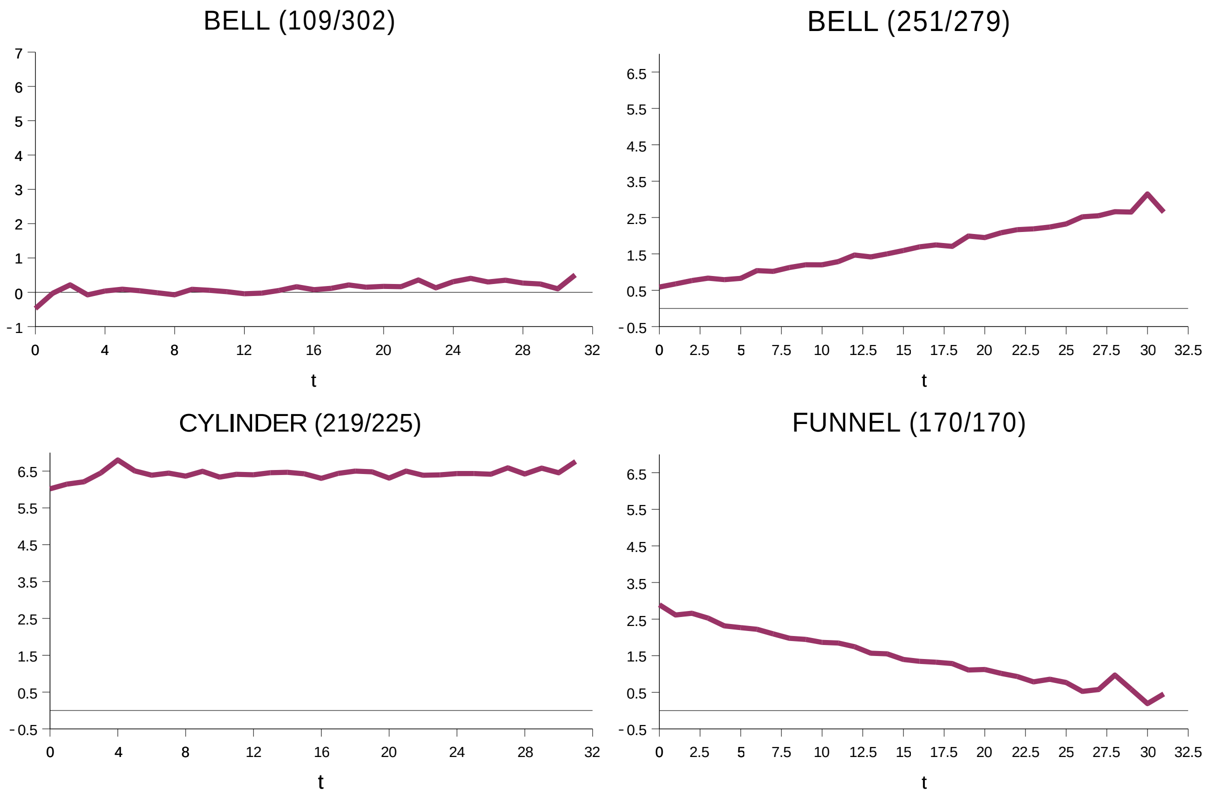


Figure 11.6: Four patterns extracted from a decision tree on the CBF problem

about the same number of elements each one of the tree classes; the bell class happens to be the majority class in this node, but the pattern corresponding to the subsequences arriving at this node is characteristic of no particular class. The interpretation of the other three terminal nodes is straightforward. Each one of them corresponds to one of the three classes and the degree of purity is very high. Also, the prototypes extracted from the corresponding subsequences exactly correspond to the patterns which are used to define the different classes in this problem (see Section 10.4 and Figure 10.3).

Although prototypes in Figure 11.6 are interpretable on this problem, this technique nevertheless assumes that there exist patterns typical of one class. For problems like Two-pat where several patterns have to be combined to classify scenarios, this technique does not give interpretable results.

11.3.5 Related work

The algorithm proposed here can be seen as a combination of jitter and the dual perturb and combine algorithm proposed in the first part of this thesis. Indeed, every subsequence appearing in a time-series may be seen as a perturbation of the initial time-series (at least for large value of the length L). Like in jitter, the learning sample size is increased by the introduction of new perturbed learning cases. Like in dual perturb and combine, the prediction of a new case is obtained by aggregating the predictions given for several perturbations of it. Eventually, like for these two methods, the noise level (here, the length of the subsequence) regulates some bias/variance compromise.

At least two recent papers propose similar algorithms both of them for image classification in the context of handwriting recognition:

- in [FH00a], a new scheme for character classification is proposed which consists in moving a small rectangular window over a pixel-array representing the character image, classify every such part of the array and then combine these predictions to give a final classification of the image. Among other things, they analyze the effect of the window size on accuracy. Although, in this work the window classification is apparently not using decision trees, the overall idea is similar in spirit to our "segment and combine" method transposed to the bidimensional signal classification case;

- in [DKN01], in order to make a classification model independent of some transformations of the input, the learning sample is augmented with new learning cases which are obtained from existing ones by applying such transformations. When testing, a "virtual" test sample is constructed by applying these transformations to the test case and the predictions given by a unique model for these new cases are aggregated to give the final prediction. In their application to handwritten digits classification, transformations are ± 1 pixel shifts of images in x and y directions. Our algorithm can be seen as an instance of their method in the context of time-series prediction, where the transformation is the shifting of patterns.

In [Die00a], a divide and conquer manifesto is discussed as the proper way to handle complex data of which time-series are a particular case. For classification problems, this manifesto suggests to divide the complex data into smaller subparts which individual classifications are merged to provide a solution to the initial classification problem. The paper focuses on examples where the merge step is carried out by models that exploit the relative positions of subparts in the complex object (such as hidden Markov models for time series). Nevertheless, our "segment and combine" method can be considered as a particular instance of this divide and conquer framework where time series are divided into subsequences which classifications are aggregated by a simple (non temporal) averaging of conditional class probability estimates.

11.3.6 Discussion

On our test problems, the segment and combine approach for time series classification gives very good results. It works by reducing both bias and variance. Furthermore, the technique is rather general since it makes few explicit hypotheses about the problem. Its complexity depends on the automatic learning method which is used to classify subsequences and the number of subsequences which are used to learn this model.

All possibilities of this method have not yet been explored. For example, other learning algorithms may be used instead of decision trees or extra-tree averaging. Whatever the basic learning algorithm used in the approach, this algorithm may take advantage of the generally large number of subsequences which are available after the segmentation of time series. For example, some part of these subsequences may be used as a validation sample to estimate some meta-parameters of the base learner (the method used to derive models from subsequences). In the context of decision trees, this validation sample may be used for instance for post-pruning.

Given our analysis on bias and variance, it seems that this method mainly works by making the classification more robust to shift invariant properties. Following the work in [DKN01], other methods could be imagined to perturb scenarios that would make the method robust to other temporal peculiarities. For example, random non linear time axis transformations could be applied in order to make the classification model invariant to changes in time scale. Notice that in many automatic learning problems (not necessarily temporal ones) such invariance properties are known to hold from a priori information about the problem and the approach just discussed provides a generic way to incorporate such prior knowledge in the learning process. This is certainly a very interesting field for further research.

Another research direction concerns the study of alternative methods to combine the predictions of subsequences, for example by taking into account their temporal ordering. Actually, the segment and combine method is able to classify scenarios of any length greater than L. It could be applied in an on-line scheme by feeding the segment classifier with a window of size L moving forward as time progresses, and thus provide a classification that would evolve with time. Such a time varying output could then be used to build an on-line model which would classify a scenario as soon as the probability of one class goes above a given threshold. We can even use it to solve the early detection problem discussed in the first chapter. To this end, the model could be forced to favor the classification of early subsequences by assigning more important weights to them when learning the subsequence classifier.

11.4 Conclusion

In this chapter, we have explored different ways to exploit low-level features to solve the time-series classification problem. First, simple sampling methods which extract scalar attributes from temporal ones have been used in combination with several algorithms. Second, we have proposed a "segment and combine" approach and have validated and analyzed its performance on the six test problems at our disposal. This method consists in building a model for subsequences of a time-series of a given length, and then aggregating the predictions of each subsequence to classify time-series. The appropriate length of the subsequences can be automatically adjusted by cross-validation so as to optimize the resulting bias/variance tradeoff.

On several problems, the results obtained by simple sampling methods are already very good. On the problems where this approach is suboptimal, the segment and combine algorithm improves significantly the results. Although our experiments are limited to six problems, they show that it is possible to get good results by using low-level attributes, even when problems present typical properties of the temporal domain.

The use of low-level attributes with classical learning algorithms however suffers from one important drawback: induced models are often not interpretable. Even if it is possible to interpret in some way the results given by the segment and combine algorithm with decision trees, this possibility remains limited and has still to be investigated in problems where classes are characterized by the occurrence of more than one single pattern.

Since the bias/variance problem seems to be solved in a satisfactory way at this stage (at least for our test problems), while the problem of interpretability remains to some extent, our goal in the next chapter is to design an algorithm which would help us to extract interpretable knowledge from time series data. Anticipating on this, let us say that the adopted method will consist in extending the vocabulary of candidate tests used by decision tree induction so as to enable such tests to detect patterns in time-series. Nevertheless, given the unavoidable compromise which exists between interpretability and accuracy in automatic learning, we do not have the ambition of improving, with interpretable methods, the very good results obtained with the segment and combine approach.

Chapter 12

Time series classification by pattern extraction

In this chapter, we propose a new decision tree based method for time series classification which aims at providing interpretable rules. The method is based on the extension of the set of candidate tests so as to enable them to detect local shift invariant properties or patterns in time series. These patterns are extracted from a piecewise constant representation of time series derived automatically by a regression tree induction method from the time series provided in the learning sample. The accuracy and the interpretability of the method is verified and discussed on the basis of our six test problems.

12.1 Introduction

The conclusion of the previous chapter is that in the context of time series classification there is a stronger need for interpretable models than for better accuracy since some of the techniques (e.g. "segment and combine") already yield very good results in terms of accuracy. From the discussion of section 11.2.3, it appears also that the lack of interpretability partially comes from the fact that some characteristic patterns of a class of signals are generally not easily representable with simple tests involving only one low-level attribute at the time. In this chapter, we thus propose to extend decision tree classifiers by explicitly allowing them to detect local shift invariant properties or patterns in time series. The underlying hypothesis is that it is possible to identify these patterns from the learning sample and to classify the scenarios by combining in a more or less simple way such pattern-detection tests.

In what follows, we first define the algorithm which is used to search for patterns at decision tree nodes. Then, we validate the method in terms of accuracy and interpretability on our test problems. Eventually, we discuss related work and some future works directions.

12.2 Proposed algorithm

The proposed algorithm is similar to classical decision tree induction, where the set of candidate tests are extended from elementary tests based on scalar attributes (e.g. $a(o) < a_{th}$) to more complex tests defined on temporal attributes. All other parts of the standard decision tree algorithm are kept unchanged (i.e the score measure, top-down induction, stop-splitting rules,...) and like they are defined in Chapter 5 and 6. Thus, in what follows we will essentially focus on the mechanism proposed to generate these new candidate tests and how they are used to classify objects described by temporal attributes. Before going into the details of the algorithm, we start with an intuitive presentation of the ideas which have motivated our extension of decision tree tests for handling the time series classification problem.

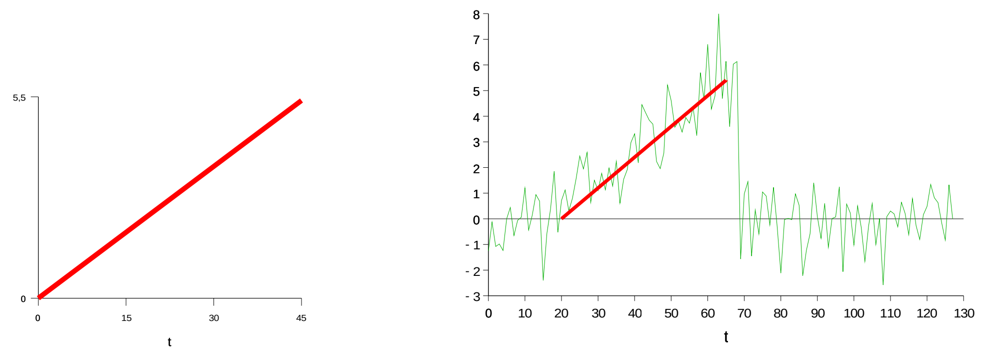


Figure 12.1: A pattern which characterizes the bell class in the CBF problem and its detection in one scenario

12.2.1 Intuitive presentation

With low-level features, tests at decision tree nodes are of the form $[A(o,t') < a_{th}]$. The use of such simple tests makes it difficult to represent temporal peculiarities such as shift invariant properties, and above all, leads to complex decision trees which are hardly interpretable. If we want to provide interpretable rules, we need to extend the representation power of decision trees by taking into account the temporal ordering of low-level attributes, in such a way that the individual tests can be translated into some easily understandable properties characterizing the signals. We limit our ambition here to extend tests at tree nodes to detect local shift invariant properties in temporal attributes. Furthermore, like in standard decision tree induction and for the same reasons (interpretability and efficiency), we restrict our method to tests involving only one attribute at a time.

Let us introduce our test definition by an example. In the CBF problem, scenarios of the bell class are characterized by an increasing ramp which may appear at any temporal position in the signal. It would be convenient to be able to represent such characteristics with one single decision tree test in order to produce understandable rules, like: “if there is a portion of the signal which looks like an increasing ramp then it is very likely to be a scenario of the bell class”. What defines the pattern here is the description “increasing ramp”. To determine such patterns during decision tree induction, we could provide several predefined types of patterns (local minimum, maximum, increasing ramp, decreasing ramp,...) together with an algorithm able to detect these latter in a signal. One drawback of this approach is that it would need prior information about a problem in order to define the dictionary of candidate patterns in an appropriate way.

For the sake of flexibility, we propose here to let the shape of patterns free. Thus, an *elementary* pattern is a limited support signal of any shape, and we consider that such a pattern is present in a signal, if there exists a portion of this signal which is close enough to the pattern, in other words if the *euclidian distance* between the pattern and some portion of the signal is smaller than a given threshold. So, decision tree tests could be defined by the choice of a temporal attribute, an elementary pattern, and a threshold on the distance. An object would be propagated to the left successor if some portion of the given attribute signal is close enough to the pattern according to the threshold and to the right successor otherwise. For example, a pattern defining the bell class is given in Figure 12.1. Its detection in a particular scenario is also plotted in Figure 12.1. With this definition, a pattern is easily interpretable since it can be plotted like any signal.

Unfortunately, this definition of patterns turns out to be too restrictive and sometimes unable to uncover the temporal structure of a problem. Indeed, with this definition there is no way to represent in a simple way temporal constraints among patterns, like for example the fact that two patterns must appear in a given order. Therefore, we further extent our set of candidate tests to incorporate what we call *complex* patterns, namely ordered lists of

Table 12.1: Overview of the algorithm for pattern extraction during tree growing

For each temporal attribute A, and for each class c:

- select an object o of class c from the local learning sample LS;
- compute a piecewise constant model for $A(o,.)$, denoted $\hat{a}(.)$;
- search for the best pattern from $\hat{a}(.)$ and define an optimal test for this pattern;
- if the score of this test is better than the best score so far, retain this test as the current best test.

several elementary patterns; we say that a complex pattern is detected in a signal if all its elementary patterns are detected *in the specified order*. In what follows we will generally leave the qualification of *complex* implicit.

At each step of decision tree induction, among a set of candidate tests one which realizes the best score is selected to split a node. Thus, in the present context we need to define a set of candidate patterns to evaluate at each test node. Since such patterns are useful only if they appear in at least some of the time series present at this node, we propose to extract the set of candidate patterns directly from the sample of signals appearing in the local learning set at the current node. Candidate elementary patterns will thus be chosen from subsignals of the signals appearing in the learning sample.

Nevertheless, for each scenario, there are a priori many possible subsignals (e.g. there are $C^2_{128} = 8128$ elementary subsignals in a time series defined on 128 time points) and they may be possibly corrupted by noise (see the signal of Figure 12.1 for example). So, it is not practically feasible nor desirable to consider all possible sequences of all subsignals of all samples as candidate patterns. The solution adopted here to circumvent these problems is to extract patterns from a piecewise constant approximation derived from the sample signals. This approach will both filter the noise and reduce the size of the search space of candidate patterns. In our work, we build such piecewise constant models with an efficient algorithm inspired from regression tree induction. In practice, it is still too costly to consider every sample signal of the local learning set to define such candidate patterns. Therefore, in the method we propose, we randomly draw only one scenario from each class and extract the best patterns from these scenarios. This should not be too limitative since if there exist patterns typical of some class, they will presumably appear in at least one of these scenarios selected at random.

Eventually, the general form of the algorithm which searches for the best test is given in Table 12.1 and illustrated in Figure 12.2. It computes the best pattern for each temporal attribute and then selects the optimal test among all attributes. Each step of the procedure is described in detail in what follows. First, patterns and tests are formally defined. Then, the algorithm to compute piecewise constant models for time series is presented. Eventually, we detail the heuristics we use to extract patterns from piecewise constant representations.

12.2.2 Elementary and complex patterns and detection tests

Elementary patterns

Like temporal attributes, an *elementary* pattern $p(.)$ is defined as a time series $(p_0, p_1, \dots, p_{P-1})$ (with the same discretization step as temporal attributes in the dataset). We say that this pattern is detected at a certain position in a time series if the distance between the pattern and the time series at this position is lower than a given threshold. More formally, denoting by $a(.) = (a_0, a_1, \dots, a_{N-1})$ a time series, with $N \geq P$, and by d_p the threshold on distance, the pattern

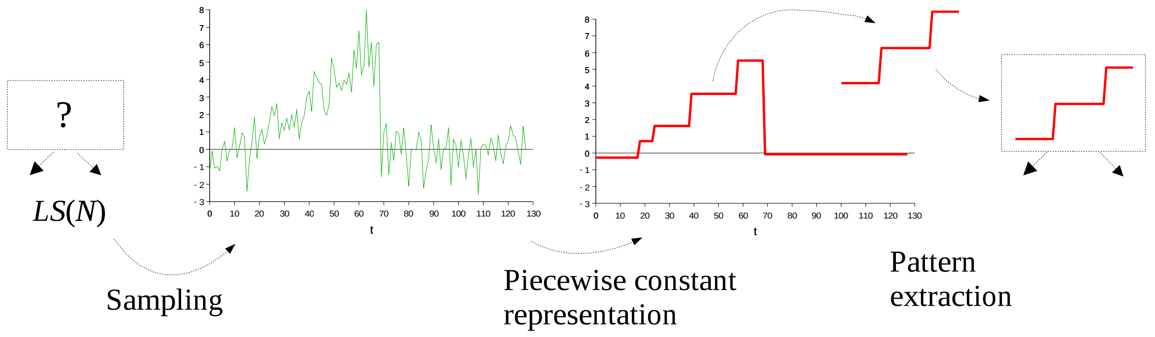


Figure 12.2: Pattern extraction to define decision tree tests

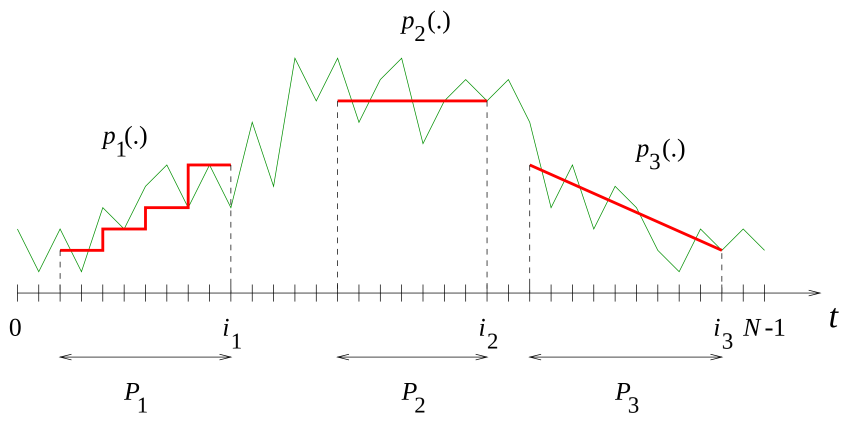


Figure 12.3: One possible configuration of three patterns in a time series

$p(.)$ is detected in the time series $a(.)$ if there exists some index i $(i = P-1, P+1, \ldots, N-1)$ such that:

$$d(p(.), a(.), i) = \sum_{j=i-P+1}^{i} (p_{j-i+P+1} - a_j)^2 < d_p. \tag{12.1}$$

The idea of using a threshold on the euclidian distance to detect a pattern allows some variation (or noise) on the amplitude with respect to the reference pattern. A binary classification rule is constructed from this pattern by means of the following test:

$$\begin{aligned} T(o) = \text{True} \quad &\Leftrightarrow \quad \exists i \in \{P-1, P, \ldots, n(o)-1\} | d(p(.), A(o,.), i) < d_p \quad (12.2)\\ &\Leftrightarrow \quad [\min_{i\in\{P-1,P,\ldots,n(o)-1\}} d(p(.), A(o,.), i)] < d_p \quad (12.3) \end{aligned}$$

where A is a temporal attribute and $n(o)$ is the number of time points for the time series representation of the object o. So, to detect a pattern, a window of size P is slided along the time axis and the distance between the pattern and the signal on this window is computed. If it becomes lower than the threshold, the pattern is detected. Note that distance (12.1) looks like a convolution product between the pattern and the signal at position i.

Complex patterns

The notion of pattern is extended to take into account a succession of elementary patterns. A *complex* pattern is defined as an ordered list of several elementary patterns $\mathcal{P} = (p_1(.), p_2(.), \ldots, p_K(.))$, defined respectively by P_k $(k = 1, \ldots, K)$ time points. Given a distance threshold d_p, a complex pattern is detected in a temporal signal $a(.)$ defined on N time points $(N \geq \sum_{k=1}^{K} P_k)$ if there exist i_1, i_2, $\ldots$,i_K such that:

$$\sum_{k=1}^{K} d(p_k(.), a(.), i_k) < d_p \tag{12.4}$$

where $d(.)$ is defined by 12.1 and i_k $(k = 1, \ldots, K)$ are integers in $\{0, 1, \ldots, N-1\}$ satisfying:

$$\begin{aligned} i_1 &\geq P_1 - 1, && (12.5) \\ i_k &\geq i_{k-1} + P_k, \forall k = 2, \ldots, K, && (12.6) \\ i_K &< N. && (12.7) \end{aligned}$$

The first two inequalities translate the fact that elementary patterns may not overlap; the last one means that they must all be fitted in the signal duration. Figure 12.3 illustrates one possible configuration of i_k values in the case of three patterns.

The derivation of a binary test for a temporal attribute a from eqn. (12.4) is naturally following eqn. (12.2):

$$T(o) = \text{True} \Leftrightarrow [\min_{i_1,\ldots,i_K | C_{\mathcal{P}}(i_1,\ldots,i_K)} \sum_{k=1}^{K} d(p_k(.), A(o,.), i_k)] < d_p, \tag{12.8}$$

where $C_{\mathcal{P}}(i_1, \ldots, i_K)$ corresponds to conditions (12.5),(12.6), and, (12.7).

Candidate test threshold optimization

Candidate tests during tree induction are thus defined by triplets $< A, \mathcal{P}, d_p >$ where A is a temporal attribute, $\mathcal{P}$ is a complex pattern (denoted by its sequence of elementary patterns), and d_p is a distance threshold. Of course, other more complex definitions are possible. For example, we could use instead a different threshold for each subpattern or constrain the difference between the time of occurrence of two successive patterns $i_{k+1} - i_k$. However, we will see in what follows that this simple definition still results in a quite complex induction algorithm.

Notice that the threshold d_p corresponding to a combination $< A, \mathcal{P} >$ plays exactly the same role than thresholds used in standard decision trees to test numerical attributes. Actually, for a given choice of $< A, \mathcal{P} >$, the optimal value of this threshold can be determined with the same exhaustive search procedure (or any other technique) used in standard tree growing. Indeed, for a given sample of objects ls and combination of attribute and pattern $< A, \mathcal{P} >$, it is sufficient to first compute for each object in ls the minimum distance

$$d_{A,\mathcal{P}}(o) = \min_{i_1,\ldots,i_K | C_{\mathcal{P}}(i_1,\ldots,i_K)} \sum_{k=1}^{K} d(p_k(.), A(o,.), i_k), \tag{12.9}$$

and then build an optimal test in the form $[d_{A,\mathcal{P}}(o) < d_p]$ using the standard discretization technique to determine d_p.

Thus, during tree induction, the minimum in (12.9) will be computed for many objects and many candidate patterns and this computation might be very time consuming. For example, by taking into account the constraints (12.5), (12.6), and, (12.7) but assuming that the lengths of patterns are small with respect to N, there are about C_N^K possible values for the vector $(i_1, \ldots, i_K)$. Thus a computation of the minimum in (12.9) by enumeration is practically feasible only for a small number of elementary patterns. Fortunately, there exists an exact Viterbi-like algorithm based on the dynamic programming principle that solves this problem with a complexity $O(N \sum_{k=1}^{K} P_k)$. This algorithm is presented in Appendix D.

Candidate complex patterns

As we will explain in the sequel, we propose to generate for each attribute, a set of candidate patterns in three steps: (i) we randomly select an object for each class; (ii) for each such selected object we generate a piecewise constant approximation of the given attribute; (iii) from the piecewise constant approximation we generate a set of candidate patterns in a combinatorial way while imposing some constraints. The following two subsections explain the two last steps in more detail.

12.2.3 Piecewise constant modeling with regression trees

A general piecewise model for a time series is obtained by segmenting the temporal interval into S segments and representing the signal in these segments by some parametric model of a given form. Here, we consider the simplest piecewise model where the time signal is represented in each segment by a constant. More complex models include for example piecewise linear models where the signal is represented by a linear function of time, $a.t+b$, which parameters, a and b, are different from one segment to another. Given the number of segments S, the goal of piecewise modeling is to find parameters which yield a model as close as possible to the initial signal. Here, the distance between the signal and the model is computed as the squared error.

Let us denote by

$$(a_0, a_1, \ldots, a_{N-1}) \tag{12.10}$$

the time series we want to model (in our case, this will correspond to the time series representation of a temporal attribute for a given object). A piecewise constant model with S segments for this time series is represented for example by a sequence of pairs:

$$((d_0, \hat{a}_0), \ldots, (d_{S-1}, \hat{a}_{S-1})), \sum_{s=0}^{S-1} d_s = N \tag{12.11}$$

where d_s is an integer denoting the length of the s^{th} segment (in time points) and $\hat{a}_s$ is its amplitude. Denoting by i_s the index in the time series at which the s^{th} segment starts:

$$i_s = \sum_{j=0}^{s-1} d_j, s = 1, \ldots, S, \tag{12.12}$$

with $i_0 = 0$ and $i_S = N$, the error of this model at predicting the value of the original time series is given by:

$$Err(a, \hat{a}) = \sum_{s=0}^{S-1} \sum_{j=i_s}^{i_{s+1}-1} (\hat{a}_s - a_j)^2. \tag{12.13}$$

The goal of piecewise constant model induction is to determine from the time series the parameters S, $\hat{a}_s$, and d_s. The induction of this model is generally divided in two separate steps:

- Determine the value of S (i.e. the complexity of the piecewise constant model);
- For a given number of segments S, determine $\hat{a}_s$ and d_s which minimize the error (12.13).

Note that since error (12.13) is null when S is equal to the number of time-points in the series (in which case the piecewise constant model corresponds to the time series itself), the value of S can not be determined as the one which minimizes the error (this is the bias/variance tradeoff applied to this problem). The approaches we adopt to resolve these tasks, in the context of our algorithm, are developed in turn below. Note that in the literature several alternative algorithms have been proposed to build piecewise models for time series data. We postpone their discussion until Section 12.4.

Determination of $\hat{a}_s$ and d_s by regression trees

Given the value of S, the goal of piecewise constant model induction is to select the parameters $\hat{a}_s$ and d_s (or i_s) that minimize (12.13). Once the lengths of the segments are fixed, the estimation of the values $\hat{a}_s$ that minimize (12.13) are given by averaging the signal on each segment:

$$\hat{a}_s = \frac{1}{d_s} \sum_{j=i_s}^{i_{s+1}-1} a_j, s = 0, \ldots, S-1. \tag{12.14}$$

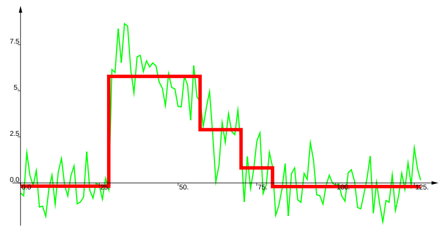


Figure 12.4: the regression tree modeling of a signal with 5 segments.

So the main problem is the determination of the segment length d_k. A dynamic programming algorithm produces an exact (optimal) solution in order $O(SN^2)$ time. In this work, we propose a solution based on regression tree induction which yields a suboptimal solution in order $O(SN)$.

Indeed, using as input attribute the time t and as output attribute the signal values $A(o, t)$ of a given object o, regression tree induction will precisely build a piecewise constant model as the one we search for. To build a tree from a time series like (12.10), every time point is considered as a classical object where the index i represents the value of the only input attribute for this object and a_i corresponds to the output value. The resulting algorithm for the segmentation of a signal in S segments is described in Table 12.2 (pp. 184). Since the goal of learning is to produce a partition of the time interval into a fixed number of segments, a best-first strategy is used for the expansion of the tree[1] and further splitting is stopped when there are already S segments. The discretization by this algorithm of an example signal in 5 segments is reproduced in Figure 12.4.

Besides efficiency, the main advantage of this algorithm is that the segmentation for increasing values of S are nested and hence all models for S going from 1 to N are built in order $O(N^2)$ (vs $O(N^4)$ with the dynamic programming algorithm). We will see below that this facilitates the search for the right value of S.

Determination of the number of segments S

The piecewise constant model is introduced in our extension of decision trees in two goals: reduce the search space for patterns and filter out the noise in time series to enhance interpretability. Hence, two criteria may direct the search for the right number of segments:

- select S in such a way that low error rates are obtained when the resulting patterns are used in the time series classification trees.
- model at best the time series at hand by filtering noise.

In both cases, the number of segments should regulate a bias/variance tradeoff. In the first goal, the more segments we use, the more candidate patterns are considered during node splitting (see Section 12.2.4) and hence bias and variance of the resulting decision trees should be affected by this parameter. On the other hand, as S increases, error (12.13) (and hence, bias) decreases but the model starts matching the noise in the time series (overfitting) and hence the optimal value of S should not be the highest.

In our experiments, we compare two approaches to fix the number of segments. The first one consist in using the same value of S throughout the construction of a decision tree and select the value which minimizes the error rate (estimated by cross-validation or by validation on an independent sample). The second method will consist in determining S without taking into account the future use of the model for pattern extraction but rather focusing on producing good models for the time series under consideration. To select the right complexity in this goal, we can take advantage of the method used to post-prune regression trees. The post-pruning approach we propose in the context of time series is given below.

[1]Contrary to the "most promising" strategy applied in the experiments of Chapter 5, algorithm of Table 12.2 follows a true best first approach.

Table 12.2: Discretization of time series by regression trees

Let us define the following quantities:

$$\text{mean}(i,j) = \frac{1}{j-i}\sum_{l=i}^{j-1} a_i \tag{12.15}$$

$$\text{var}(i,j) = \frac{1}{j-i}\sum_{l=i}^{j-1} (a_l - \text{mean}(i,j))^2 \tag{12.16}$$

$$\text{Score}(i,j,l) = (j-i)\text{var}(i,j) - (j-l)\text{var}(l,j) - (l-i)\text{var}(i,l), i < l < j \tag{12.17}$$

$$\tag{12.18}$$

As for regression trees, Score(i,j,l) is the variance reduction caused by the split of the segment between index i and j at the position l.

Build_PCM($(a_0,\ldots,a_{N-1})$**,**S**):**
(build a piecewise constant model into S segments for the time series $(a_0,\ldots,a_{N-1})$.)

Set $L = \{(\text{mean}(0,N),0,N)\}$, the ordered list of current segments described by their mean value, the start index of this segment and the start index of the next segment. Then proceed as follows:

1. If the length of L is equal to S then go to step 7.
2. For each triplet (a,i,j) in L:
   - Determine Score$(i,j) = \max_{l=i+1,\ldots,j-1}$ Score(i,j,l);
   - Determine $l^*(i,j) = \arg\max_{l=i+1,\ldots,j-1}$ Score(i,j,l);

   NB. These optimal scores and partitions do not need to be recomputed at each iteration for all segments in L. Only the newly added segments actually need to be recomputed.
3. Find $(a^*,i^*,j^*) = \arg\max_{(a,i,j)\in L}$ Score(i,j);
4. Remove (a^*,i^*,j^*) from L;
5. Insert in order in L the two new segments:
   - $(\text{mean}(i^*,l^*(i^*,j^*)),i^*,l^*(i^*,j^*))$,
   - $(\text{mean}(l^*(i^*,j^*),j^*),l^*(i^*,j^*),j^*)$.
6. Go to step 1
7. From the s^{th} element of L, denoted by (a,i,j), define the s^{th} segment as:

$$\hat{a}_s = a$$
$$d_s = j-i$$

8. Return $((d_0,\hat{a}_0),\ldots,(d_S,\hat{a}_S))$;

Table 12.3: Automatic selection of the number of segments by cross-validation

---

Build_PCM_with_pruning$((a_0, \ldots, a_{N-1}))$**:**
(build a piecewise constant model for a time series by selecting its complexity by cross-validation.)

- Split the time series into $a_{\text{even}} = (a_0, a_2, \ldots, a_{N-2})$ and $a_{\text{odd}} = (a_1, a_3, \ldots, a_{N-1})$.
- Compute the sequence of piecewise constant model $\{M_1, \ldots, M_{N/2}\}$, where $M_s =$ Build_PCM(a_{even}, s),
- Select the number of segments s^* which minimizes the error of M_{s^*} at predicting a_{odd}. If the s^{th} model is written $M_s = ((d_0, \hat{a}_0), \ldots, (d_{s-1}, \hat{a}_{s-1}))$, then this error is computed by:

$$Error(s) = \sum_{k=0}^{s-1} \sum_{j=i_k}^{i_{k+1}-1} (\hat{a}_k - a_{2j+1})^2.$$

  and $s^* = \arg\min_{s=1,\ldots,N} Error(s)$.
- Return Build_PCM$((a_0, \ldots, a_{N-1}), s^*)$;

Post-pruning piecewise constant models

In the context of regression trees, post-pruning algorithms aim at selecting the right complexity of the model. To this end, a validation set is used to give an independent estimate of the error and this estimate is used to select the right complexity of the model. Here, the only available data is a series of N time points. One possible way to split this data set into two sets is to subsample the time series with twice the initial discretization step. A time series $(a_0, \ldots, a_{N-1})$ gives thus rise to two[2] time series $(a_0, a_2, \ldots, a_{N-2})$ and $(a_1, a_3, \ldots, a_{N-1})$. The first series is used to learn the model parameters according to the algorithm of Table 12.2 and the second one is used to estimate the error of this model. If the discretization step is sufficiently small with respect to the system dynamics, there will not be an important loss of information by using subsampling. on the other hand, this approach will be useful to determine when the regression tree starts overfitting the data.

The algorithm which automatically learns the number of segments is depicted in Table 12.3. It builds a sequence of piecewise models of increasing complexity. Although that does not appear explicitly in Table 12.3, the whole sequence is obtained by applying only once the algorithm of Table 12.2 with $S = N/2$ and saving at each step intermediate models. In consequence, the complexity of this algorithm is at worst $O(N^2)$.

For example, right part of Figure 12.5 (pp. 186) plots resubstitution error (estimated on a_{even} in Table 12.3) and validation error (estimated on a_{odd}) for the discretization of the signal represented in the right of the same figure with an increasing number of segments. The validation error goes through a minimum for 6 segments. Eventually, the model of 6 segments built from the whole time series is represented in the right part of Figure 12.5.

12.2.4 Search strategy for candidate tests

Tests are defined by triplets $< A, \mathcal{P}, d_p >$ where A is some attribute, $\mathcal{P}$ is a complex pattern, i.e. a sequence of elementary patterns, and d_p is the distance threshold. During node splitting, search is carried out to find a good test in terms of the score. We already noticed that once

[2]the notations assume that N is even.

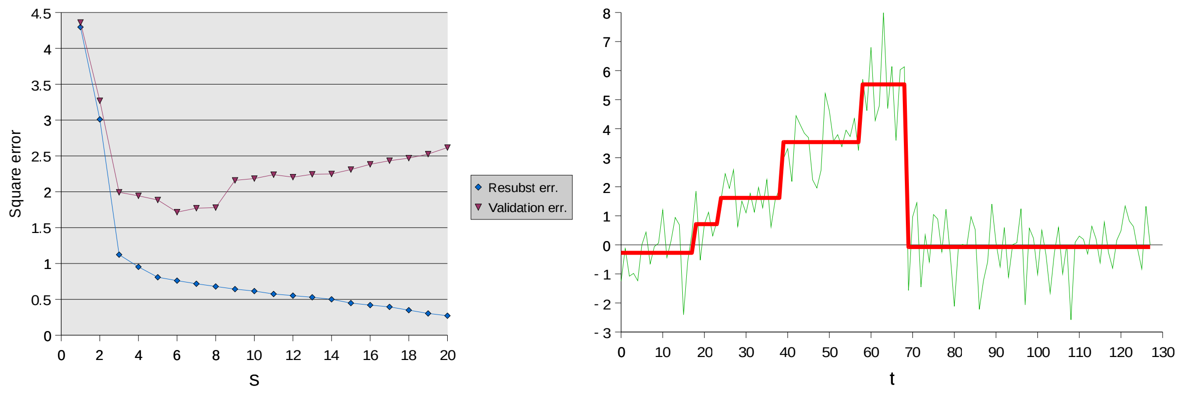


Figure 12.5: Left, resubstitution and validation errors with increasing segment numbers, right, the optimum model with six segments for the time series

we have chosen an attribute A and a pattern $\mathcal{P}$, the value of d_p which realizes the best score may be computed like in the standard tree growing algorithm as the optimum discretization threshold on the numerical attribute:

$$d_{A,\mathcal{P}}(o) = \min_{i_1,\ldots,i_K | C_{\mathcal{P}}(i_1,\ldots,i_K)} \sum_{k=1}^{K} d(p_k(.), A(o,.), i_k). \tag{12.19}$$

So, the main problem is the determination of a set of candidate complex patterns $\mathcal{P}$ and thus, in other words a sequence of elementary patterns $(p_1(.),\ldots,p_K(.))$ for arbitrary K.

As suggested in the introduction of this section, in our approach candidate patterns are extracted from the piecewise representation of some temporal attribute of an object appearing in the local learning sample (selected at random, for each class). From a time series $A(o,.)$ corresponding to an object o, the segmentation algorithm described above produces a piecewise constant model $\hat{a}(.)$ which can be represented by a sequence:

$$\hat{a}(.) = ((d_0, \hat{a}_0), \ldots, (d_{S-1}, \hat{a}_{S-1})). \tag{12.20}$$

Every subsequence of this sequence equivalently represents a piecewise constant signal defined on a smaller time interval. We propose to consider as candidate elementary patterns $p_k(.)$ subsequences $((d_{i_k}, \hat{a}_{i_k}), \ldots, (d_{j_k}, \hat{a}_{j_k}))$, with $0 \leq i_k \leq j_k < S$. To define complex patterns, it seems natural to impose the further constraint that elementary patterns extracted in this way are not overlapping each other in $\hat{a}(.)$ and also that their ordering of appearance in $\hat{a}(.)$ is respected in the definition of the complex pattern, i.e. $j_k < i_{k+1}$, $k = 1, \ldots, K-1$. We further impose that two elementary patterns are separated in $\hat{a}$ by at least one segment, i.e. $j_k < i_{k+1} + 1$, $k = 1, \ldots, K-1$.

Following these lines, we define a complex pattern from the representation (12.20) by a mask of S boolean values:

$$\underline{b} = (b_0, \ldots, b_{S-1}) \tag{12.21}$$

Each boolean variable b_i is true if the corresponding segment is taken into account to define the pattern. A sequence of consecutive true values for b_i delimited by false values defines an elementary pattern $p_k(.)$. For example, the boolean vector (F, T, T, F, T, T, F) defined on a piecewise representation with 7 segments defines a complex pattern composed of two elementary patterns. The first elementary pattern is the subsignal corresponding to the second and the third segment and the second elementary pattern is defined by the fifth and the sixth segments. When applied to the piecewise representation of Figure 12.6, this boolean vector defines a complex pattern which detects the strict succession of the elementary patterns p_1 and p_2 represented in the same figure.

Table 12.4: Pattern extraction from a segmented signal

Pattern_search(LS**,**A**,**$\hat{a}$**):**
(search the best test from the signal $\hat{a}$ for the attribute A and on the subset LS.)

- Denote by S the number of segments in the representation of $\hat{a}(.)$.
- Compute the following sequence of pattern definition masks:

$$\{\underline{b}^0, \underline{b}^1, \ldots, \underline{b}^{S-1}\},$$

  where:

  - $b_i^0 = \mathrm{T}$, $\forall i = 0, \ldots, S-1$,
  - $\underline{b}^{k+1}$, for $k = 1, \ldots, S-1$, is obtained from $\underline{b}^k$ by flipping the previously unflipped true value (there are $S-k$ of them in $\underline{b}^k$) which yields the highest score:

$$\mathrm{Score}(LS, < A, \mathcal{P}(\underline{b}^{k+1}, \hat{a}(.)), d_p^*(\underline{b}^{k+1}, \hat{a}(.)) >),$$

    where $\mathcal{P}(\underline{b}^{k+1}, \hat{a}(.))$ is the complex pattern defined by $\underline{b}^{k+1}$ from $\hat{a}(.)$ and $d_p^*(\underline{b}^{k+1}, \hat{a}(.))$ is defined by:

$$d_p^*(\underline{b}^{k+1}, \hat{a}(.)) = \arg\max_d \mathrm{Score}(LS, < A, \mathcal{P}(\underline{b}^{k+1}, \hat{a}(.)), d >).$$

- Among this sequence, select the vector $\underline{b}^*$ which yields the best score.
- Return the test $< a, \mathcal{P}(\underline{b}^*, \hat{a}), d_p^*(\underline{b}^*, \hat{a}) >$.

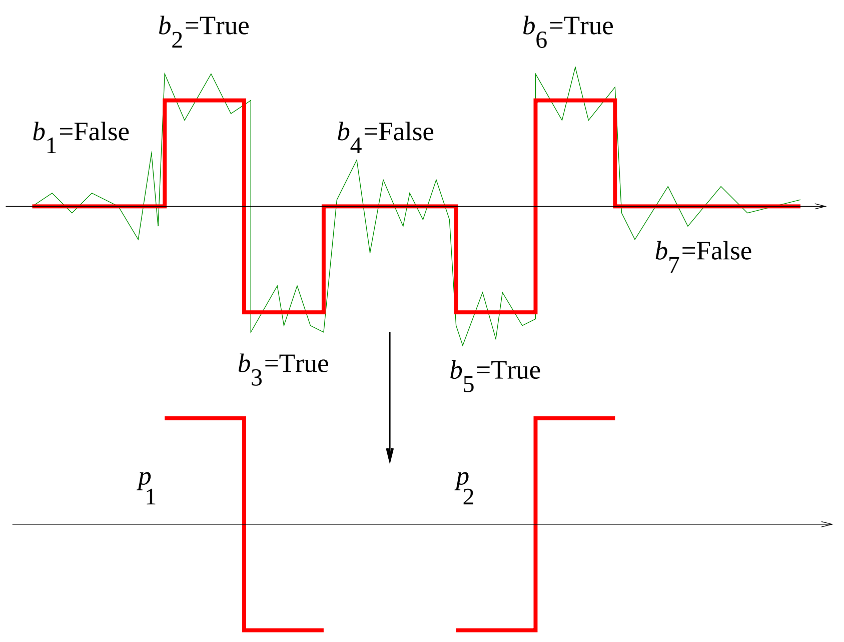


Figure 12.6: Definition of a pattern (composed of two elementary patterns) by a boolean mask $\underline{b}$

So searching for interesting patterns amounts to searching the space of boolean vectors of size S. There are in principle 2^S different complex patterns in a signal decomposed into S pieces. If we want the search algorithm to remain tractable in most cases, we thus need a way to reduce the search complexity. We propose here to use a greedy strategy. The resulting algorithm which is looking for a pattern in a signal $\hat{a}$ is described in Table 12.4. Starting from a vector of only true values[3], we flip at each step the previously unchanged boolean value which leads to the pattern which together with the optimum threshold on (12.19) causes the highest increment (or lowest decrease) in the score. So, the search path in the space of patterns is composed of S steps, starting from a unary pattern composed of all S segments ($b_i = \text{True}, \forall i$) and ending with a unary pattern composed of only one segment among the S ones ($b_i = \text{False}\ \forall i \neq j$). Thus, this heuristic reduces the number of patterns to evaluate from 2^S to $\frac{S(S+1)}{2}$.

To summarize, for a given temporal attribute, the procedure of Table 12.4 should be applied on every signal $A(o,.)$ corresponding to some *master* objects selected from the local subset LS. As already explained, in our experiments only one master object will be drawn from each class (and for each candidate attribute) at each node of the tree. The resulting search algorithm for candidate tests when splitting decision tree nodes is depicted in Table 12.5. It replaces the determination of the optimal test in the standard decision tree induction algorithm of Table 5.1 of Chapter 5 (pp. 79).

12.2.5 Computational complexity

Let us estimate the complexity of the algorithm of Table 12.5 for a learning sample LS of size N. Let N_t be the number of time points used to define scenarios of the datasets and assume that this number is the same for every scenario. The complexity of the segmentation of a time series (construction and pruning) is at most $O(N_t^2)$. Assuming that the resulting piecewise constant model gives S segments, $\frac{S(S+1)}{2}$ different patterns are considered by the algorithm of Table 12.4. For each of them, the minimal distance to the N objects must be computed and the best threshold on distance searched. The complexity of the computation of the minimal distance for a complex pattern in one scenario is at most $O(N_t S)$ for patterns defined on S

[3]We could also start from a vector of only false value.

Table 12.5: Search algorithm for candidate tests during tree growing

For each temporal attribute A, and for each class c:

- select an object o of class c from LS,
- compute either:
  - $\hat{a}(.) = \text{Build\_PCM}(A(o,.), S)$ for a predefined number of segments S, or
  - $\hat{a}(.) = \text{Build\_PCM\_with\_pruning}(A(o,.))$ by using the pruning algorithm;
- compute the test $< A, \mathcal{P}, d_p >= \text{Pattern\_search}(LS, A, \hat{a})$;
- if the score of this test is better than the best score so far, retain the triplet $< A, \mathcal{P}, d_p >$ as the best current test

segments (see Appendix D for the algorithm) and thus this computation for the N scenarios is $O(NN_tS)$. The determination of the best threshold requires to sort the learning sample and to pass once through all objects. This can be done in $O(N \log N)$.

The whole process of segmentation and best pattern search must be repeated one time for each temporal attribute and each class. If we assume that the number of segments given by piecewise modeling is always the same and denoted by S, all in all, the complexity of the procedure of Table 12.5 is:

$$O(c_1 nmN_t^2 + c_2 nm\frac{S(S+1)}{2}(c_3 NN_tS + c_4 N \log N))), \tag{12.22}$$

where n is the number of temporal attributes and m is the number of classes. In practice, the dominant part of this algorithm results from the computation of the minimal distance of the pattern to each scenario of the learning sample. On the other hand, the computation of the piecewise models and the search for the best threshold are negligible.

Globally, this procedure must be repeated at each test node of the tree and hence the complexity of the complete tree induction algorithm depends of course on the complexity of the tree. In practice however, the number of nodes often increases less than linearly with respect to the size of the learning sample and hence, the average complexity of the full algorithm is in practice closer to linear than to quadratic with respect to the size of the learning sample. It is on the other hand linear with respect to the number of candidate attributes and quadratic with respect to the number of sampling steps and with the number of segments used for piecewise constant modeling.

12.2.6 Discussion

In this section, we have proposed a way to extend decision trees to handle the time series classification problem. Tests are extended to detect local shift invariant properties or patterns in time series. These patterns are extracted from a piecewise constant representation of time series of scenarios which appear in the local subsample at decision tree node. The price to pay for the increase of flexibility and interpretability is in terms of computational efficiency, which is mainly governed by part of the dynamic programming algorithm which computes the distance from a complex pattern to a time series. The resulting method is a combination of bottom-up and top-down approaches in machine learning: bottom-up because properties of an object of one class are generalized to classify all objects of the same class and top-down because the tree growing algorithm essentially finds more and more specific tests to classify objects.

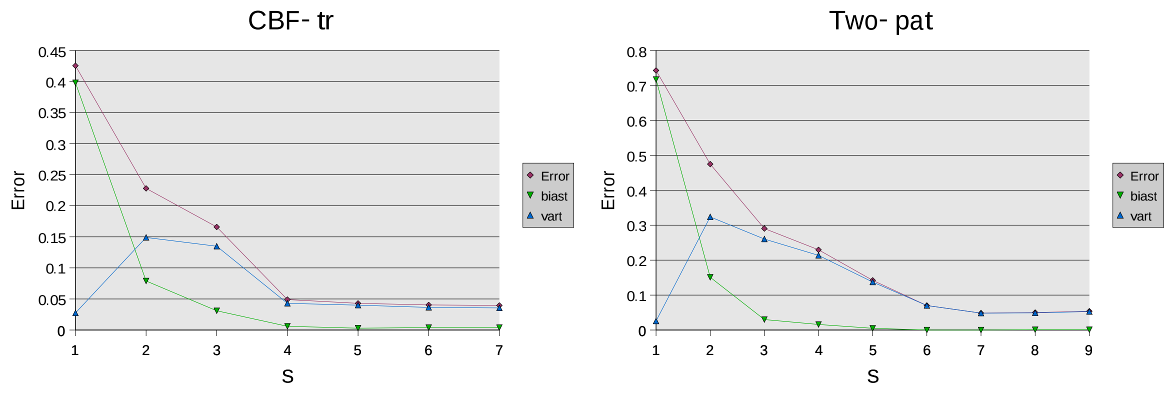


Figure 12.7: Effect of the number of segments on bias and variance

Note that the algorithm is random since it defines patterns from scenarios which are randomly selected in the local sample. However, the hope is that specificity of the chosen scenario will be filtered by the pattern extraction technique and hence this will not result in too much variance of the resulting models.

Finally, note also that this extension of the decision tree induction method is especially adapted to cope with shift invariant properties and our pattern definition is not invariant to changes in the time scale. We will discuss in the last section of this chapter some directions to extend this method so as to handle scale invariant properties and other peculiarities relevant in temporal problems.

12.3 Validation of the method

In this section, the proposed pattern detection based tree induction technique is validated on our test problems. First, we assess the accuracy of the method. Among other things, we study the effect of the way of choosing the number of segments for piecewise constant modeling on bias, variance, and error rates. Then, some indicative results in terms of computation times are given. Eventually, we discuss the interpretability of the models which are produced.

12.3.1 Accuracy

Bias/variance study

One parameter which should regulate some bias/variance tradeoff in this method is the number of segments used to represent signals by a piecewise constant model. Indeed, when S increases the number of patterns which are searched during induction increases. Figure 12.7 plots the evolution of classification bias and variance with the number of segments and, left on the CBF-tr problem and right on the Two-pat problem. For each value of S, 50 fully grown trees are learned from learning samples of size 300 drawn from a pool sample of size 3000 and tested on a test sample of size 1000. During tree induction, time series are segmented by the algorithm of Table 12.2 without pruning and with the same value of S at each tree node.

As expected, on both problems, bias decreases with the number of segments. It is almost null on the CBF-tr problem at 4 segments. From our knowledge of this problem, this is not surprising since this is the minimum number of segments that allows to distinguish between an increasing and a decreasing ramp. On the two-pat problems, the minimum bias is reached for larger values of S. And indeed, from the problem definition, we know that the signals in this problem are best modeled by a piecewise constant model in 7 segments. On both problems however, variance first increases from 1 to 2 segments and then monotonically decreases with the segment number. On the Two-pat problems, we observe a very slight increase after 7 segments.

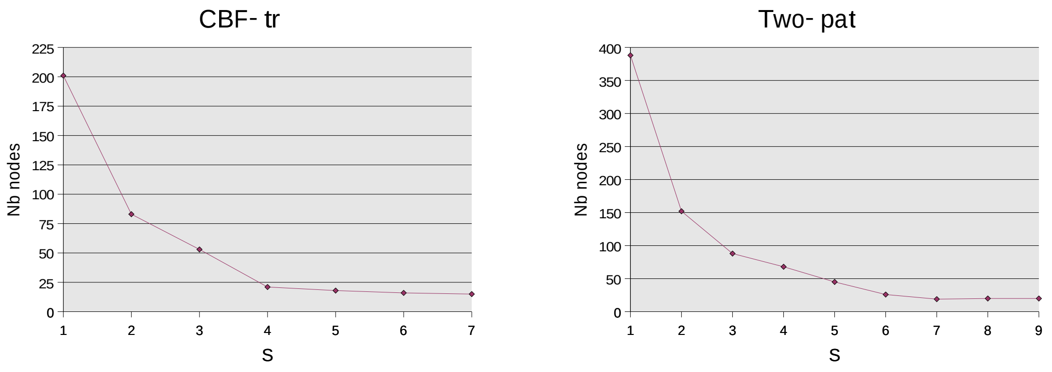


Figure 12.8: Evolution of decision tree complexity with respect to the number of segments

The fact that variance does not increase with the number of segments may be attributed to two effects. Figure 12.8 shows on the two problems the evolution of the tree complexity (the total number of nodes) with respect to the number of segments. When the number of segments increases, the complexity of decision trees decreases very much. It goes from 201 nodes to 15 nodes on the CBF-tr problem and from 388 nodes to 19 nodes on the Two-pat problem. When S is small, we know from the problem definitions that it is impossible to build a decision tree which generalizes well (the bias is non null). However, because of the noise, it is nevertheless possible to discriminate perfectly between the different classes in the learning sample and hence tests at deep levels in the tree overfit very much the data. The small variance and high bias with only one segment is due to the fact that in this case, it is very difficult to discriminate between patterns even on the learning sample (which is confirmed by the high tree complexity). On the other hand, when S is large, a small tree suffices to realize low error on the learning sample and this small tree turns out to generalize better than larger trees obtained with smaller values of S.

Second, even if more patterns are considered when the number of segments increases, they do not necessarily translate into more discriminative power for the test. Indeed, the distance to a complex pattern (12.4) is the sum of many terms. The addition or deletion of a segment in the piecewise representation does not necessarily change the ordering of objects with respect to this distance. In other word, the minimal distance is quite stable with respect to the model (at least for large enough values of S). This is confirmed by the fact that the complexity of the tree remains almost constant when S increases behind the value of S which minimizes the bias (see Figure 12.8). However, these two phenomena are probably related to the fact that the method is well adapted to this kind of problems. On other problems, variance could increase with the number of segments.

Table 12.6 compares three variants of the algorithm on both problems. The first line corresponds to the minimum of error on Figure 12.7. The second line is obtained by using the post-pruning algorithm presented in Table 12.3 in the same condition. The third line is obtained from the second line by pruning the decision trees from a validation sample of size 1000 (according to the algorithm described in Chapter 6). On both problems, the bias is very small. This shows that the model and the search procedure is well adapted to the problems. In consequence, the variance is responsible for almost all the error. The effect of pruning is not important: it reduces slightly the variance and the complexity while leaving the bias unchanged. The complexity of full and pruned trees is small. On CBF-tr, in average, decision trees use 5 tests and on Two-pat, they use 7 tests (the minimum is 2 tests to separate 3 classes and 3 tests to separate 4 classes). On these problems, using post-pruning to select the number of segments for the piecewise representation is beneficial. It reduces both error rates and complexity with respect to the utilization of the same value for all time series.

With respect to the approach of the previous chapter, the error on CBF-tr is slightly greater than with the segment and combine algorithm while it is slightly lower on the Two-pat problem.

Table 12.6: DT with pattern extraction on CBF-tr and Two-pat

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
CBF-tr								
Full DT ($S = 7$)	0.0396	15	0.0040	0.0356	0.0316	0.0264	0.0053	0.0211
Full DT (S learned)	0.0327	14	0.0060	0.0267	0.0260	0.0218	0.0044	0.0174
Pruned DT (S learned)	0.0322	11	0.0060	0.0262	0.0251	0.0208	0.0048	0.0160
Two-pat								
Full DT ($S = 7$)	0.0484	19	0.0000	0.0484	0.0427	0.0242	0.0029	0.0214
Full DT (S learned)	0.0472	19	0.0000	0.0472	0.0423	0.0236	0.0025	0.0212
Pruned DT (S learned)	0.0445	15	0.0000	0.0445	0.0396	0.0216	0.0026	0.0189

Table 12.7: DT with pattern extraction on other problems

	Error	Compl
CC		
Full DT ($S = 6$)	2.67 ± 1.53	22
Full DT (S learned)	3.33 ± 2.11	24
CBF		
Full DT ($S = 6$)	1.00 ± 0.75	15
Full DT (S learned)	1.25 ± 0.56	13
JV		
Full DT ($S = 6$)	19.19	45
Full DT (S learned)	21.62	49
Auslan		
Full DT ($S = 4$)	15.50 ± 6.50	32
Full DT (S learned)	17.5 ± 10.55	29

However, although very close, a comparison of the bias/variance profile of both methods shows that they work differently. Segment and combine is characterized by a low variance while the present method is characterized by a low bias and a relatively high variance. We have seen in previous chapters that a high variance seems to be the price to pay for interpretable models.

Other problems

We further experiment with this method on the other test problems used in the preceding chapter. Table 12.7 reports for each problem the best error obtained with a constant value of S and the error obtained by determining the value of S with the post-pruning algorithm. In each case, trees are fully grown and the average complexity is reported in the table.

On all problems, the results are better without the post-pruning algorithm although both approaches give trees of comparable complexity. The difference is less pronounced on CC and CBF. On JV and Auslan, it may be attributed to the fact that the number of time-points is lower in average and hence subsampling is not appropriate to select the right complexity.

On CC and CBF, the method gives better results than the one obtained with decision trees and low-level features. On CBF, the result are even competitive with the best results obtained with variance reduction techniques (random trees and segment and combine). However, while on Auslan, results are only slightly better than the results obtained by decision trees with low-level attributes, on JV, there are significantly worse.

These rather bad results on the two last problems can be explained. Indeed, on JV, we have seen that the temporal behavior was not important for classification (since models built

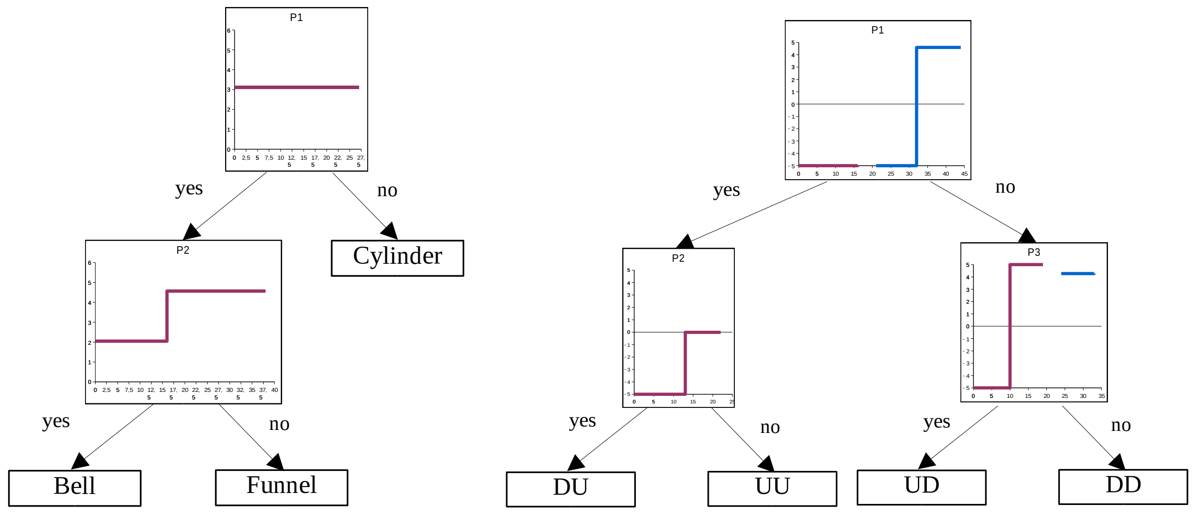


Figure 12.9: Decision trees with pattern extraction, left on CBF-tr, right on Two-pat.

with only two low-level features give very good results) and hence the increase of flexibility of our approach for handling temporal behaviors is useless in this case. Also, both problems are characterized by a large number of classes and many attributes in comparison to the size of the learning sample and we know that decision tree induction especially suffers in such situations because of the recursive partitioning.

12.3.2 Computing times

To give an idea of the computing times of the proposed algorithm, we ran it with the selection of segment numbers by post-pruning on a learning sample of size 500 on the CBF-tr problems. It took $123s$ to build a tree of 17 nodes. For comparison, it takes only $14s$ to build a tree of 89 nodes from 128 low-level attributes and $24s$ to build a set of 25 random trees with the same set of attributes (which however is less accurate on this problem). So, the pattern detection based tree method is quite demanding with respect to other approaches but computation times still remain reasonable, especially since they are increasing only in (approximately) order $O(NlogN)$ with respect to the learning sample size.

12.3.3 Interpretability

Since one of the main reasons for the introduction of a new method is interpretability, it is important to verify that the proposed method indeed produces interpretable models. To this end, we have built a decision tree from 500 random scenarios on the CBF-tr and Two-pat problems, by fixing the value of S with post-pruning. Figure 12.9 reproduces the shallow nodes of both trees. Complex patterns are plotted in each box corresponding to an interior node. When these patterns are composed of several elementary patterns, they are drawn in order using different colors.

On the CBF problem, the first pattern is a signal at mid-level with respect to the maximum amplitude. Since the cylinder class does not go through this mid-level, scenarios of this class are directed to the right successors. Bell and funnel classes are then separated by a pattern which interpretation is obvious from the knowledge of the bell class: an increasing trend from the mid-level to the maximum level. On the Two-pat problem, the first test is composed of two elementary patterns which only appear in this order in the DU and UU classes (U for upward step and D for downward step). The first elementary pattern which appears in both upward and downward step is used by the method to impose that the second elementary pattern appears in second position. Then at the left successor, DU and UU classes are separated by

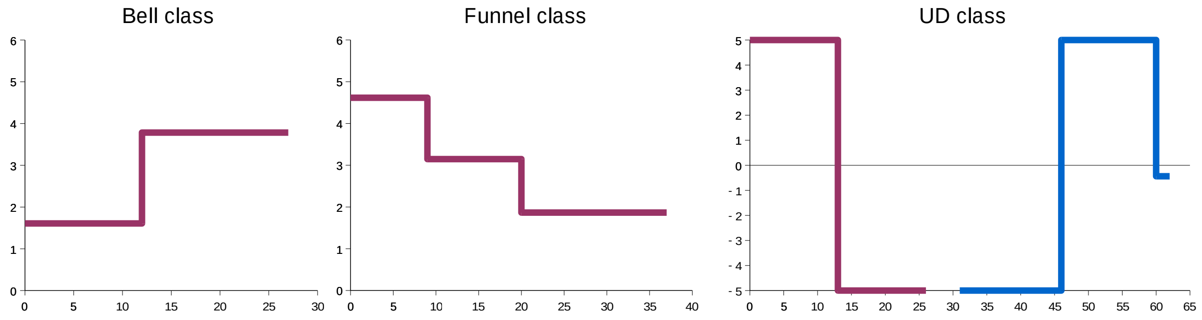


Figure 12.10: Descriptive patterns for the bell and funnel classes on the CBF-tr problem and for the UD class on the Two-pat problem

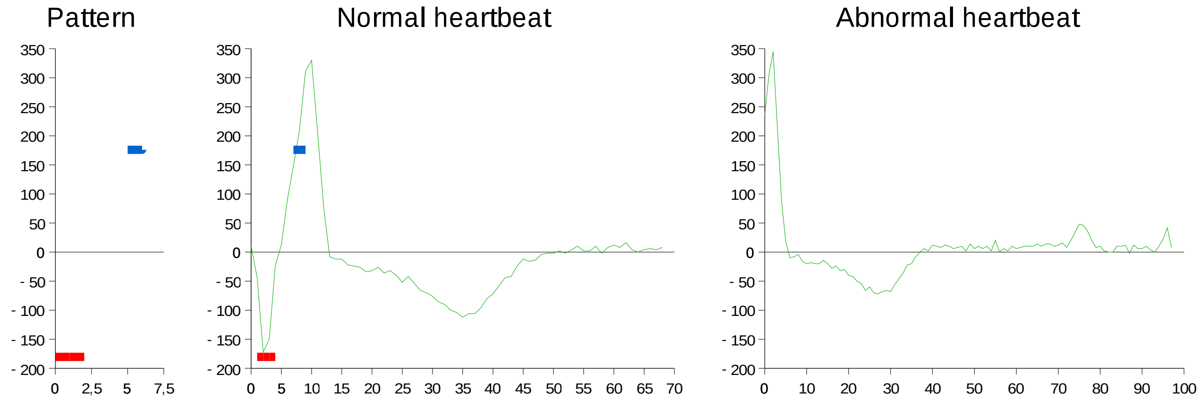


Figure 12.11: From left to right, a pattern characterizing a normal heartbeat, its detection in a normal heartbeat, and, an abnormal heartbeat

a less interpretable pattern which nevertheless only appears when there is a downward step in the signal (which may go from -5 to 0 while the upward step always goes from -5 to 5). UD and DD classes are separated by repeating the "trick" of the top level test. The first elementary pattern is an upward step and the second elementary pattern imposes that the first one appears in first position.

Although interpretable, the patterns produced by these trees are by construction more discriminative than descriptive. They do not explicitely characterize elements of one class. In some cases, it is useful to give descriptive rules for each class. More descriptive patterns may be extracted by our method if we search for patterns which separate one class from all other ones and if we further impose that patterns are extracted only from scenarios of this class (by selecting only scenarios of this class in the procedure of Table 12.5). For example, Figure 12.10 shows typical patterns of the bell and funnel classes on the CBF-tr problem and of the UD class on the Two-pat problem. These patterns were extracted from a scenario of the given class by using the algorithm of Table 12.5 in trying to separate this class from all the other ones in a learning sample of size 500. This time, patterns are very readable and really characterize scenarios of each class.

To illustrate the interest of such methods in a more practical problem, we carried out one further experiment on a dataset of ECG signals which is discussed in [Ols01b]. Each scenario of this dataset records the sequence of measurements obtained by two electrodes during one heartbeat. Each heartbeat is further classified as normal or abnormal. In the latter case, it corresponds to a cardiac pathology known as supra-ventricular premature beat which it is of course important to detect in practice. We run our decision tree algorithm on a sample of 200 cases. At the top node of this tree, a pattern is detected on the second electrode which detects in majority normal heartbeat. This pattern is reproduced in the left part of Figure 12.11 and its occurrence in a normal heartbeat is plotted in the center part of the same figure.

It tells us that normal heartbeats are characterized by a low value during at least 3 time steps followed by a high value during at least 2 time steps. Such a model gives us the information that abnormal heartbeat are characterized by the fact that they do not present this pattern (Figure 12.11 plots one abnormal heartbeat which indeed does not present this properties). This single pattern yields an error rate of 15% on the learning sample. Although prior knowledge of the problem (which we do not have) can either confirm or infirm that such a pattern is really the best one for classification, we believe that such results are interesting in themselves since they are confirmed by visual inspection of the signals appearing in the learning sample.

12.4 Related work

The work presented in this chapter is essentially a combination and an adaptation of several ideas which have been studied separately by others in the literature. We first give some guide on the work on piecewise modeling for time series data. Then, we review pattern matching algorithms for time series. Eventually, we provide pointers to work which also proposes interpretable models for time series classification.

Piecewise models for time series

General (not only constant) piecewise representations for time series have been introduced in many works for several purposes, e.g. for time series classification in [Man97] or to fasten the search for similar time series in large databases in [KP99]. The more general problem of image segmentation is also tackled in the very well-known textbook by Duda and Hart [DH73]. The most used type of piecewise model is the piecewise linear model. Finding the optimal segmentation for polynomial models in S segments in general requires an order $O(N.S^2)$ dynamic programming algorithm (see for example [Man97] or [Ols01b] for a description of the exact algorithm). Several suboptimal algorithms have been devised. A review of these algorithms is given in [KCHP01]. They can be divided into three classes: bottom-up algorithms (which recursively merges consecutive segments), top-down algorithms, and online algorithms (which segments the time series as data arrive). With respect to these approaches, our algorithm is a combination of top-down and bottom-up (at the post-pruning stage).

Also, a main difference of our approach is the criterion proposed to decide on the number of segments to use, which is essentially guided in our case by the bias/variance tradeoff, whereas, in the related literature, the criterion used is generally a “fidelity” criterion, similar to those used in lossy data compression. Indeed, usually, the selection of the optimal number of segments simply relies on a predefined threshold on error. In the top-down approach, this amounts at stopping the development of a node if the local error in this segment is lower than a given threshold. Possible criteria use the MDL principle ([Man97]) or statistical tests ([LSL$^+$00]). A more original approach is adopted in [Ols01b] where the optimal number of segments is determined by the discontinuity point in the approximation of the error curve (the resubstitution error in Figure 12.5) by a piecewise linear model with two segments. Pruning for piecewise modeling was also proposed in [SÚW98], however in the more general context of scatter-plot smoothing by linear hinges models.

Pattern matching and extraction in time series

In our algorithm, patterns are detected in the time series by using the euclidian distance and an exact algorithm based on dynamic programming ideas. The resulting pattern definition is certainly not invariant with respect to several interesting transformation in the temporal domain (such as time scaling) but it was chosen so as to make learning quite efficient. Nevertheless, several works exist on pattern definition and pattern matching in time series. The main motivations of such works are generally the design of a distance measure which is invariant to some transformation in the temporal domain and/or the efficiency of the corresponding pattern detection algorithm for searching in large databases. With these goals, distance measures have been proposed which are based on Fourier transformation [AFS93], dynamic time warping [BC96], or probabilistic models [KS97, GS00]. In all these works, it is however assumed that the patterns are previously defined by the user and to our knowledge, none of these works have been coupled with a pattern extraction algorithm for time series classification purpose.

The problem of pattern extraction in time series is tackled for example in [DLM+98] and [Oat99]. Patterns are defined as subsequences of a fixed number of time points and relevant patterns are found by unsupervised clustering techniques. In [DLM+98], these patterns are used to find association rules (i.e. conditional rules of the form "pattern A is always followed by pattern B"). In [Oat99], the goal is to find distinctive patterns, i.e. patterns which appear frequently in the scenarios of one class but not in scenarios which do not belong to this class. Although the goal of the author is not directly time series classification, it is clear that his technique may be used in this purpose.

Interpretable models for time series classification

Several machine learning approaches have been developed recently to solve the time series classification problem with a focus on interpretable models. Manganaris [Man97], for example, constructs piecewise polynomial models for univariate time series and directly uses the parameters of this representation as attributes for classification with decision trees. Olszewski [Ols01b] uses a very similar approach but with composite piecewise models (with an automatic choice among linear, quadratic, and exponential models in each segment). He also feeds decision trees with the parameters of these models. Although they both use piecewise models for feature extraction, these approaches are very different from our own approach. The piecewise model is used in their work to provide a small set of attributes while it is used to support pattern extraction in our method.

At least three works are based also on pattern extraction. Kadous [Kad99] defines "parametrized event primitives" which are extracted from the time series of the learning sample. The number of such primitive events is reduced by using clustering techniques in their parameter spaces and then characteristic functions of the resulting clusters are used with standard automatic learning algorithms (Naive Bayes or decision trees). Kudo et al. [KTS99] transforms multivariate signals into binary vectors. Each element of these vectors corresponds to one rectangular region of the "value-time" plane and tells if the signal passes through this region. The number of regions is reduced by a discretization of time and attribute. A method of their own, subclass, builds rules from these binary vectors. Gonzales et al. [AR00] extends (boosted) inductive logic programming systems with predicates that are suited for the task of time series classification. The number of possible predicates considered during induction is reduced by discretization of their possible defining parameters.

These three approaches share some common characteristics with our decision tree extension. First, authors are all interested in getting interpretable rules and so use an interpretable classifier. Second, they use some discretization techniques to reduce the search space for patterns (by discretization of the parameter space in [KTS99] and [AR00] and unsupervised clustering in [Kad99]). In the first two methods, patterns are extracted globally before the construction of the classifier while, in the last one, the search for patterns is incorporated in the induction algorithm like in our work.

The main difference with our approach is that primitive patterns are defined by some parametric models which are chosen so as to be useful in most application domains. In our method, patterns are extracted from the the time series of the learning sample and hence they do not have to satisfy any predefined form. Another difference is that patterns in these works are defined on absolute time values (even if some variations about their start time is allowed by the pattern definitions) while our pattern definition explicitly captures shift invariant properties. This may be an advantage over these methods in problems which really present such patterns but also a disadvantage when shift invariant patterns are not relevant for classification.

12.5 Conclusions and future research directions

In this chapter, we have presented a new tool to handle time series in classification problems which produces interpretable results. This tool is based on a piecewise constant modeling of temporal signals by regression trees. Patterns are extracted from these models and combined in decision trees to give interpretable rules. With respect to the use of low-level features, the advantage of this technique in terms of interpretability is undeniable. In terms of accuracy, better results can be obtained by using either random trees or the segment and combine approach. However, in problems where we know that classes are actually characterized by local shift invariant properties (like in CBF-tr or Two-pat), the results given by this algorithm are competitive with these latter methods.

From this preliminary study, two future work directions seem particularly interesting: extensions of the model to handle other temporal peculiarities and combination of this techniques with perturb and combine algorithms.

Extensions of the model

The presented method is only a first attempt to produce interpretable models for time series classification problems and it suffers from several shortcomings essentially in terms of representation capabilities.

First, the representation of time series with piecewise constant models while computationally efficient is somewhat crude and often needs a lot of segments to reach a high precision. It could be advantageously exchanged for a more powerful decomposition, for example using piecewise linear models. Although the computational cost of extracting such piecewise linear models from the attribute values of an object would be higher, we believe that both the interpretability of the resulting rules and the computational complexity of the resulting automatic learning method could be significantly improved in this way. Indeed, the optimum number of segments with a piecewise linear model should be lower than the same number with a piecewise constant model and hence this should reduce the number of candidate patterns to consider during the development of decision tree nodes.

Second, our model was tailored for applications where classes are characterized by local shift invariant properties but it does not allow the detection of shrunk or extended versions of the reference pattern along the time or value axis. This limitation comes from the use of euclidian distance which is sensitive to change in time scale and furthermore can only compare portions of signals of the same size. At the moment, the only way to take into account such scale invariant properties within our method is to introduce several patterns corresponding to this property at different time scales.

One way to circumvent the problem is to detect our patterns by using another distance measure which would be invariant or at least less sensible to scale changes in patterns. One possible distance measure is dynamic time warping (see for example [BC96]) which finds the point-to-point alignment between two time series (possibly of different lengths) which minimizes the distance between them. Another possibility is to use probabilistic pattern matching like in [GS00] or [KS97]. More flexibility could be for example added to our system by putting a probability distribution on the lengths and amplitudes of segments which would naturally take into account time and amplitude deformations of the pattern. Actually, this transforms our piecewise constant model into a particular case of a segmental hidden Markov model which was studied in the literature (see [ODK96] or [AMGD00] for applications). In this approach, a pattern would be viewed as a probabilistic model generating time series and the pattern matching based on euclidian distance would be replaced by the computation of the (log-)likelihood of the signal given that model. We could then take advantage of the existing learning algorithms for probabilistic models to learn the pattern parameters from the whole set of time series of one class (instead of only one randomly drawn from the learning sample).

When considering extensions of the algorithm, great care should be taken for not increasing too much the variance. If we know from prior knowledge that such extensions are not necessary for classification, then they should be avoided.

Perturb and combine algorithm

With decision trees and pattern extraction, the bias is low but the variance is high (because of the increase of the size of the hypothesis space). This suggests that perturb and combine algorithms would give good results in this context. As it selects scenarios at random in the learning sample, our algorithm is actually random. The simplest perturb and combine variant would thus consist in running the algorithm several times on the same data and combine the predictions of the resulting models to reduce variance. However, the resulting method will not be computationally efficient since the search for one model is already quite demanding.

A better approach would either consist in extending the dual perturb and combine algorithm to the present method or to design a random tree growing algorithm in this context, specifically so as to reduce computation times. For example, the bit-vector masks that define complex patterns might be selected at random to develop tree nodes and hence result in a fast random tree growing algorithm. Nevertheless since this approach would result in non interpretable classification models, it should be compared with variance reduction algorithms which use low-level attributes. If both give the same error, then the approach which is the most efficient (for learning or for testing), should be preferred.

Chapter 13

Conclusion and future work

13.1 Conclusion

This thesis has explored two common problems in machine learning, the improvement of an existing learning algorithm, in our case the decision tree induction method, and the treatment of a "new" kind of data, in our case temporal data.

Our walk to the first goal was very systematic. First, we studied in depth the behavior of standard decision tree induction in various conditions in order to highlight its main shortcomings. The main drawback of this method turns out to be its variance, be it the prediction variance which affects accuracy or the parameter variance which affects interpretability. From the general considerations about variance reduction techniques in the first part of this thesis, we had at our disposal two main approaches to reduce this variance: regulate the bias/variance tradeoff during induction or combine several perturbed decision trees to decrease variance. The first kind of techniques led us to the evaluation of the well-known pruning algorithm and to the proposal of discretization threshold stabilization methods. As expected, these two approaches only improve slightly accuracy because they inevitably increase the bias. On the other hand, perturb and combine algorithms are very effective in combination with decision trees. As our study of decision tree variance has shown that the parameter variance is very high, we devised an extremely randomized decision tree based algorithm which compares favorably with bagging and boosting algorithms. Trying to propose a compromise between the interpretability of decision trees and the accuracy of perturb and combine algorithm, we reinvented another type of soft tree model as a particular instance of a general variance reduction technique, the dual perturb and combine algorithm.

The comparison of the resulting algorithms given in the closure of the second part of the thesis shows that the proposed methods, from single trees to bagging of soft trees, span a large spectrum of improvements and hence of solutions according to the main criteria used to assess learning algorithms: interpretability, computational efficiency, and accuracy. Nevertheless, we are convinced that there still remains ample room for improvements and further research. In particular, the most promising approaches are certainly the dual perturb and combine algorithm and randomized trees in general. On the other hand, from our experiments with threshold stabilization methods, we do not expect much improvements from variance reduction techniques within the strict hypothesis space of standard decision trees.

Tackling the treatment of time series data was more difficult. First, we defined the general problem of supervised learning with temporal data and then restrained our ambition to the problem of time series classification. An a priori analysis allowed us to highlight the peculiarities of such problems that could prevent standard learning algorithms to succeed. Then, we chose a small number of test problems which were artificially generated to present such peculiarities. As our goal was to propose general approaches, our first experiments logically consisted in applying existing learning algorithms on the low-level attributes with minimum preprocessing. However, even the best variance reduction techniques were not able to tackle all test problems in a satisfactory way. So, we proposed the segment and combine algorithm which allowed to improve the bias/variance tradeoff by artificially providing more information to the learning algorithm while taking into account the temporal ordering of low-level attributes. If the problem of accuracy on our test problems was partially solved by this variant, methods that use low-level attributes were not able to provide really interpretable rules and hence in a second step, we tried to design an interpretable method. To this end, we decided to extend decision tree induction, motivated by the interpretability and the open-minded structure of this method. Our extension focused on the detection of local shift invariant properties. The resulting method gave good results in terms of accuracy and interpretability on problems presenting shift invariant properties but was powerless on other problems.

The main benefits of our study in the last part are, on the one hand, to show that the problem is indeed difficult and hence that it deserves further attention and, on the other hand, to open up two complementary directions for the search of solutions: first adapt perturb and combine algorithms to the properties of temporal data and second try to devise interpretable temporal models. In both directions, we have made several choices (for example, the use of decision trees) but it is clear that many other approaches could have been adopted which might prove superior to the one selected here. We therefore consider that our work is only a first step towards accurate and interpretable automatic learning methods for time series classification.

All in all, our search for solutions to both problems has shown that the design of automatic learning algorithms is a complex multiobjective optimization problem. Among the main conflicting objectives in this context, we have average (and worst-case) accuracy over a large class of problems, interpretability of models, computational efficiency and scalability. Often interpretability means low accuracy and high accuracy means low computational efficiency and no interpretability. While extra-trees show that it is possible to combine efficiency and accuracy in the same algorithm, interpretability and accuracy on the other hand seem to be the most difficult criteria to combine in one algorithm. While dual P&C keeps the interpretability of decision trees, it is not as accurate as other perturb and combine algorithms. Although the extension of decision trees to detect patterns gives accurate results in some types of problems, in average, it is not as good as the segment and combine approach.

Another important compromise which must be taken into account when designing a learning algorithm is the bias/variance tradeoff which regulates the accuracy. Our experiments have shown that thinking problems in terms of bias and variance is useful to understand the behavior of learning algorithms and helps avoiding a priori bad design choices. Although it is not the only way to study learning algorithms, one of its main advantages is that bias and variance can be effectively evaluated (even if this evaluation is cumbersome in some cases). Furthermore, even if the available measures of bias and variance in classification are not fully sound from the theoretical point of view, we have seen that, in combination with the bias/variance decomposition of the regression error on probability estimates, it allows to effectively explain many things.

Eventually, a last conclusion from our experiments is that decision tree induction definitely deserves its place among the most popular learning algorithms. Even if it suffers from several well-known shortcomings (such as its variance), its open-minded structure and the simplicity and efficiency of its induction algorithm makes it the method of choice for many improvements. This thesis has shown for example that the extra-tree extension is a very promising alternative for handling very large datasets and that the power of the tests may be easily extended to handle different kinds of problems. Even if the resulting extensions are sometimes very far from the standard decision tree algorithm, such improvements would not have been easy with more rigid and black-box type algorithms.

13.2 Future work directions

We collect here some future research directions. First, we present short term developments which concern improvements or extensions of the presented algorithms. Second, we discuss interesting long term research directions in a more general point of view.

13.2.1 Short term investigations

Several possible enhancements of the proposed algorithms have already been mentioned in the course of the thesis. The most important ones are summarized below.

Concerning our method for improving decision trees, the two methods which deserve more attention in the future are certainly the dual perturb and combine algorithm and the extra-tree method.

Concerning the dual P&C algorithm, it would be interesting to investigate how its performances compare with those of bagging and boosting in a broader range of conditions (larger learning samples, larger number of attributes). Although this method can be combined with any learning algorithm, we have studied its effect and implementation only in combination with decision trees. It would be also interesting to investigate the use of this method in combination with other learning algorithms such as neural networks. In this context, a closed-form derivation of the algorithm may further increase its practical interest.

The extra-tree algorithm provides certainly the most promising basis for future research. We would suggest to make a more in depth assessment of its accuracy, in particular, in comparison to techniques such as multilayer perceptrons, and on the basis of broadly accepted datasets, such as for instance those used in the Statlog project [MS94]. The question of computational performance optimization in the context of very large databases also deserves special attention. In particular, among the questions which have not been addressed in the thesis we mention the storage of extra-trees which sizes might become problematic for very large sample sizes and for some specific applications (such as embedded systems).

Concerning the segment and combine approach for time series classification, it would also be interesting to investigate the behavior of this method on a broader range of problems to better circumscribe its limitations. Depending on the results of such a study, other perturbation schemes could be imagined to tackle problems for which it does not work well enough.

The most attractive extension of pattern extraction techniques is certainly the use of probabilistic models for pattern matching which has been mentioned in the conclusion of chapter 12. ¿From our experiments in this chapter, we have also concluded that the tree partitioning was maybe responsible for the bad results on the two problems with small learning sample sizes and high number of classes. So, it would be interesting to investigate other ways to take advantage of the pattern extraction algorithm which would be more robust in such conditions (for example, using a set of overlapping rules instead of the recursive partitioning of decision trees).

Finally, although we have focused in this thesis on the treatment of numerical input attributes and symbolic outputs, from a practical point of view it would be necessary to extend our methods to handle other types of data, such as symbolic attributes or regression problems. Notice that, since most of our algorithms rely on decision trees, their extension to regression problems is often trivial (for example, dual P&C and extra-trees are easy to transpose). The extension to symbolic attributes with dual P&C and extra-trees is also direct, although it would need further validation. On the other hand, the treatment of discrete sequences of events with the segment and combine approach or with our extension of decision trees for pattern detection would require more creativity.

13.2.2 Long term research

We expose here two long term research directions which we find interesting from a broader point of view.

Bias/variance tradeoff and other paradigms

In this thesis, all our improvements have been motivated from a bias/variance point of view. However, the bias/variance decomposition is apparently not able to explain the good behavior of boosting algorithms and furthermore there exist in machine learning other paradigms which have motivated new learning algorithms. For example, recently the machine learning community shows an increasing interest in support vector machines and large margin classifiers which have emerged from Vapnik's work on the statistical learning theory. We have seen that at least qualitatively some connections may be drawn between random tree algorithms and these latter methods (in terms of their ability to overcome the overfitting problem). The Bayesian approach to automatic learning offers another motivation for model averaging techniques. We believe that understanding the relationships between these different paradigms or between the learning algorithms which they have fostered, would be helpful to improve or to design new algorithms and hence certainly deserves further research in the future.

Fully automatic method for complex data types

A very challenging and interesting question which is partially addressed in the last part of this thesis concerns the adaptation of automatic learning methods to handle complex data types such as time series, images, texts, or biological sequences. Indeed, more and more databases containing such data become available for automatic learning and there is still a lot of work to be done before these datasets can be handled in an efficient and fully automatic way. According to our preliminary results with time series, we believe that this problem should be tackled from two complementary points of view: first the development of generic approaches, as far as possible independent of the use of prior knowledge about the problem domain, and second, the development of interpretable methods.

With the goal of providing general approaches without consideration about interpretability questions, we believe that variance reduction techniques studied in the first part of this thesis should be methods of choice. In particular, similarly to the segment and combine algorithm, we could devise perturb and combine methods adapted to the considered type of data. Another solution is to extend decision trees with candidate tests adapted to the type of data (just like our pattern detection tests for time series) and then use randomization to avoid the problem of overfitting which will inevitably appear with the increase of flexibility.

On the other hand, the problem of inducing interpretable models seems more challenging. When extending the decision tree method, we have made the hypothesis that the problem may be characterized by local shift invariant properties. This assumption makes the method even competitive with the segment and combine approach on problems which satisfy this hypothesis but however, does not improve the results over the use of low-level attributes on problems which do not satisfy it. Such hypotheses seem to be necessary to give interpretable and accurate models especially in complex domains but prevent the design of very general solutions. One goal of research in this context would be to evaluate to which extent it is possible to free oneself from such a priori hypotheses.

Appendix A

Bias/variance decompositions of zero-one loss functions

In this appendix, we review the most representative decompositions of the mean error rate into a bias and variance terms which have been proposed in the machine learning literature. After the derivation of each decomposition, we end this review with a short discussion of their pros and cons and motivate our choice for a particular decomposition. Before reading this appendix, we suggest to first review chapter 3 where the notation and relevant concepts are introduced.

Throughout the following presentation we will refer to Figure 3.5 in order to illustrate the special properties of the measures introduced. This figure represents, at a particular point of the input space, in its leftmost part a hypothetical true distribution $P(c|\underline{a})$, and in its center and righmost parts the sampling distributions of the classes predicted at this point by two hypothetical learning algorithms. Let us recall here that $P^i_{LS}(c|\underline{a})$ denotes the distribution of predictions at point $\underline{a}$ by algorithm i, generated when this algorithm is repeatedly applied to infer models from random learning sets of fixed size (say N).

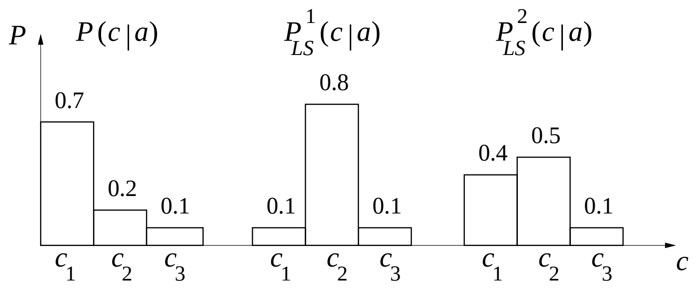


Figure A.1: Left, a classification problem, right, two different learning algorithms

A.1 Bias/Variance decompositions

The local mean error in classification with zero-one loss functions is the probability of misclassification at point $\underline{a}$:

$$E_{LS}\{Err(f_{LS}(\underline{a}))\} \quad = \quad 1 - \sum_c P(c|\underline{a}) \cdot P_{LS}(c|\underline{a}). \tag{A.1}$$

The residual error at point $\underline{a}$, noted $\sigma_C(\underline{a})$, is given by:

$$\sigma_C(\underline{a}) = 1 - P(f_B(\underline{a})|\underline{a}), \tag{A.2}$$

where f_B is the Bayes classifier. By analogy with the decomposition of the square error, we have defined in Chapter 3 the following "natural" bias and variance terms:

$$\text{bias}_C(\underline{a}) \quad = \quad 1(f_B(\underline{a}) \neq f^{MAJ}_{LS}(\underline{a})) \tag{A.3}$$

$$\text{var}_C(\underline{a}) \quad = \quad 1 - P_{LS}(f^{MAJ}_{LS}(\underline{a})|\underline{a}). \tag{A.4}$$

where $\text{bias}_C(\underline{a})$ is the error of the majority vote classifier with respect to the Bayes model at point $\underline{a}$ and $\text{var}_C(\underline{a})$ is defined by symetry with respect to the residual error σ_C.

We have also noted that these three terms do not add to give the mean error rate. Indeed, for the problem illustrated in Figure A.1, the models built by the two algorithms decide class 2, while the Bayes classifier would decide class 1. So, the two algorithms are (equally) biased at point $\underline{a}$, i.e. $\text{bias}_C(\underline{a}) = 1$. On the other hand, residual error at point $\underline{a}$ is equal to $\sigma_C(\underline{a}) = 30\%$, while the average error of algorithm 1 is equal to 76% and that of algorithm 2 is 61%. Intuitively also, the "variance" of algorithm 2 is stronger than that of algorithm 1, since its sampling distribution is closer to uniform. Quantitatively, we have $\text{var}_C^1(\underline{a}) = 0.2$ and $\text{var}_C^2(\underline{a}) = 0.5$. However, since in both cases the average error rates are actually smaller than the error rate of the corresponding majority vote classifier (80% in both cases), we conclude that in this example variance is beneficial (since algorithm 2 is better that algorithm 1). This counter-intuitive behavior (when compared to the bias/variance decomposition of the square regression error) leads to the difficulties already mentioned in chapter 3 and is responsible for the multitude of zero-one loss function decompositions discussed below.

A.1.1 Tibshirani [Tib96] and James and Hastie [JH96]

Tibshirani [Tib96] defines the bias at point $\underline{a}$ as the difference between the probability of the Bayes class and the probability of the majority vote class:

$$\text{bias}_T(\underline{a}) = P(f_B(\underline{a})|\underline{a}) - P(f_{LS}^{MAJ}(\underline{a})|\underline{a}). \tag{A.5}$$

Thus, the misclassification error of the majority vote classifier is the sum of this quantity and residual error:

$$\begin{aligned} \sigma_C(\underline{a}) + \text{bias}_T(\underline{a}) &= 1 - P(f_B(\underline{a})|\underline{a}) + P(f_B(\underline{a})|\underline{a}) - P(f_{LS}^{MAJ}(\underline{a})|\underline{a}) && \text{(A.6)} \\ &= 1 - P(f_{LS}^{MAJ}(\underline{a})|\underline{a}) && \text{(A.7)} \\ &= Err(f_{LS}^{MAJ}(\underline{a})). && \text{(A.8)} \end{aligned}$$

This is exactly the part of the error which would remain if we could completely cancel the variability of the predictions. The variance is then defined as the difference between the misclassification error and the error of the majority vote classifier:

$$\text{var}_T(\underline{a}) = E_{LS}\{Err(f_{LS}(\underline{a}))\} - Err(f_{LS}^{MAJ}(\underline{a})). \tag{A.9}$$

Tibshirani denotes this variance term the aggregation effect. Indeed, this is the variation of error which results from the aggregation of the predictions over all learning sets. Note that this variance term is not necessarily positive. For example, for the second classifier of Figure A.1, $E_{LS}\{Err(f_{LS}(\underline{a}))\}$ is 0.61 and $Err(f_{LS}^{MAJ}(\underline{a}))$ is 0.8, yielding a variance term of -0.19. James and Hastie [JH96] have proposed the same decomposition. To distinguish these bias and variance terms from bias_C and var_C, they call them systematic and variance effect respectively. Note that both Tibshirani and James and Hastie have come to this decomposition from theoretical considerations about possible generalization of the bias/variance decomposition of squared error to other loss functions.

A.1.2 Breiman in [Bre96a]

Breiman has successively proposed two decompositions of the mean error rate. The first one [Bre96a] is based on the idea that only biased points should participate to bias and only unbiased points should participate to variance. Hence, the difference between the actual error and the residual error is attributed to bias if $f_{LS}^{MAJ}(\underline{a}) \neq f_B(\underline{a})$ and to variance otherwise. Since this difference may be written at point $\underline{a}$:

$$E_{LS}\{Err(f_{LS}(\underline{a}))\} - \sigma_C(\underline{a}) = (1 - \sum_c P(c|\underline{a}) \cdot P_{LS}(c|\underline{a})) - (1 - P(f_B(\underline{a})|\underline{a})) \tag{A.10}$$

$$= P(f_B(\underline{a})|\underline{a}) - \sum_c P(c|\underline{a}) \cdot P_{LS}(c|\underline{a}) \tag{A.11}$$

$$= \sum_c (P(f_B(\underline{a})|\underline{a}) - P(c|\underline{a})) \cdot P_{LS}(c|\underline{a}), \tag{A.12}$$

Breiman's bias and variance terms are given by:

$$\text{bias}_{B,1}(\underline{a}) = 1(f_B(\underline{a}) \neq f_{LS}^{MAJ}(\underline{a})) \cdot (P(f_B(\underline{a})|\underline{a}) - \sum_c P(c|\underline{a}) \cdot P_{LS}(c|\underline{a})) \tag{A.13}$$

$$\text{var}_{B,1}(\underline{a}) = 1(f_B(\underline{a}) = f_{LS}^{MAJ}(\underline{a})) \cdot (P(f_B(\underline{a})|\underline{a}) - \sum_c P(c|\underline{a}) \cdot P_{LS}(c|\underline{a})). \tag{A.14}$$

By definition of $f_B(\underline{a})$, we have $P(f_B(\underline{a})|\underline{a}) \geq P(c|\underline{a}), \forall c$ and hence, the two terms are positive.

A.1.3 Breiman in [Bre00]

To define the bias of his second decomposition, Breiman [Bre00] simply isolates the term corresponding to the class $f_{LS}^{MAJ}(\underline{a})$ from the sum A.12:

$$\text{bias}_{B,2}(\underline{a}) = (P(f_B(\underline{a})|\underline{a}) - P(f_{LS}^{MAJ}(\underline{a})|\underline{a})) \cdot P_{LS}(f_{LS}^{MAJ}(\underline{a})|\underline{a}). \tag{A.15}$$

The variance then becomes:

$$\text{var}_{B,2}(\underline{a}) = \sum_{c \neq f_{LS}^{MAJ}(\underline{a})} (P(f_B(\underline{a})|\underline{a}) - P(c|\underline{a})).P_{LS}(c|\underline{a}). \tag{A.16}$$

To distinguish this variance from the regression one, he calls this term the spread. Again, both terms are positive.

A.1.4 Kong and Dietterich [DK95]

Kong and Dietterich [DK95] propose a decomposition of

$$1 - P_{LS}(f_B(\underline{a})|\underline{a}) \tag{A.17}$$

which is the local misclassification error of the learning algorithm with respect to the Bayes classifier or, in other words, the probability that a classifier does not predict the Bayes class at $\underline{a}$. This error coincides with the true misclassification error only when there is no noise. In the presence of noise, it can be either greater or smaller than the true misclassification error (for example, in the case of the second classification algorithm of Figure A.1, (A.17) is 0.9 whereas the misclassification error is 0.76). Their definition of bias matches the definition (A.3):

$$\text{bias}_{KD}(\underline{a}) = 1(f_B(\underline{a}) \neq f_{LS}^{MAJ}(\underline{a})). \tag{A.18}$$

Variance is the difference between (A.17) and bias:

$$\text{var}_{KD}(\underline{a}) = 1 - P_{LS}(f_B(\underline{a})|\underline{a}) - \text{bias}_{KD}(\underline{a}). \tag{A.19}$$

This variance is positive at unbiased points and negative at biased ones.

A.1.5 Domingos [Dom00a, Dom00b]

Domingos in [Dom00a] and [Dom00b] agrees with the definition of bias, variance, and, residual error given in our introduction to chapter 3 and combines them into an expression like:

$$E_{LS}\{Err(f_{LS}(\underline{a}))\} = c_1(\underline{a}).\sigma_C(\underline{a}) + \text{bias}_C(\underline{a}) + c_2(\underline{a}).\text{var}_C(\underline{a}), \tag{A.20}$$

where $c_2(\underline{a})$ is defined by:

$$c_2(\underline{a}) = \begin{cases} 1 & \text{if } f_B(\underline{a}) = f_{LS}^{MAJ}(\underline{a}) \\ -\frac{P_{LS}(f_B(\underline{a})|\underline{a})}{1-P_{LS}(f_{LS}^{MAJ}(\underline{a})|\underline{a})} & \text{if } f_B(\underline{a}) \neq f_{LS}^{MAJ}(\underline{a}). \end{cases} \tag{A.21}$$

So we have:

$$c_2(\underline{a}).\mathrm{var}_C(\underline{a}) = \begin{cases} 1 - P_{LS}(f_{LS}^{MAJ}(\underline{a})|\underline{a}) & \text{if } f_B(\underline{a}) = f_{LS}^{MAJ}(\underline{a}) \\ -P_{LS}(f_B(\underline{a})|\underline{a}) & \text{if } f_B(\underline{a}) \neq f_{LS}^{MAJ}(\underline{a}) \end{cases} \tag{A.22}$$

and hence, by combining (A.4) and (A.22):

$$\mathrm{bias}_C(\underline{a}) + c_2(\underline{a}).\mathrm{var}_C(\underline{a}) = 1 - P_{LS}(f_B(\underline{a})|\underline{a}). \tag{A.23}$$

The decomposition of Domingos is thus similar to the one of Kong and Dietterich. However, he further decomposes the global variance into two terms (according to (A.22)):

$$\begin{aligned} V_u &= E_{\underline{A}}\{1(f_{LS}^{MAJ}(\underline{A}) = f_B(\underline{A})).(1 - P_{LS}(f_{LS}^{MAJ}(\underline{A})|\underline{A})\} && \text{(A.24)} \\ V_b &= -E_{\underline{A}}\{1(f_{LS}^{MAJ}(\underline{A}) \neq f_B(\underline{A})).(-P_{LS}(f_B(\underline{A})|\underline{A})\}. && \text{(A.25)} \end{aligned}$$

V_u is thus the contribution from unbiased points to variance and V_b is the contribution from biased points to variance. The overall variance is thus the difference between V_u and V_b:

$$E_{\underline{A}}\{c_2(\underline{A}).\mathrm{var}_C(\underline{A})\} = V_u - V_b \tag{A.26}$$

A.1.6 Kohavi and Wolpert [KW96]

Kohavi and Wolpert [KW96] propose a very different decomposition which is closer in spirit to the decomposition of the square regression error. Introducing the new (random) numerical models $f_{LS}^i(\underline{a}) = 1(f_{LS}(\underline{a}) = c_i), i = 1, \ldots, m$, and using the variable $Y_C^i = 1(C = c_i)$, the local mean error may be written:

$$\begin{aligned} E_{LS}\{Err(f_{LS}(\underline{a}))\} &= E_{LS}\{E_{C|\underline{a}}\{1(C \neq f_{LS}(\underline{a}))\}\} && \text{(A.27)} \\ &= E_{LS}\{E_{C|\underline{a}}\{\frac{1}{2}\sum_{i=1}^{m} 1(Y_C^i \neq f_{LS}^i(\underline{a}))\}\} && \text{(A.28)} \\ &= E_{LS}\{E_{C|\underline{a}}\{\frac{1}{2}\sum_{i=1}^{m}(Y_C^i - f_{LS}^i(\underline{a}))^2\}\} && \text{(A.29)} \\ &= \frac{1}{2}\sum_{i=1}^{m} E_{LS}\{E_{Y_C^i|\underline{a}}\{(Y_C^i - f_{LS}^i(\underline{a}))^2\}\}. && \text{(A.30)} \end{aligned}$$

Applying the decomposition of square error for each i, we get:

$$\begin{aligned} E_{LS}\{E_{Y_C^i|\underline{a}}\{(Y_C^i - f_{LS}^i(\underline{a}))^2\}\} &= E_{Y_C^i|\underline{a}}\{(Y_C^i - E_{Y_C^i|\underline{a}}\{Y_C^i\})^2\} \\ &\quad +(E_{Y_C^i|\underline{a}}\{Y_C^i\} - E_{LS}\{f_{LS}^i(\underline{a})\})^2 \\ &\quad +E_{LS}\{(f_{LS}^i(\underline{a}) - E_{LS}\{f_{LS}^i(\underline{a})\})^2\} && \text{(A.31)} \\ &= (P(c_i|\underline{a}) - P(c_i|\underline{a})^2) + (P(c_i|\underline{a}) - P_{LS}(c_i|\underline{a}))^2 \\ &\quad +(P_{LS}(c_i|\underline{a}) - P_{LS}(c_i|\underline{a})^2), && \text{(A.32)} \end{aligned}$$

since:

$$E_{Y_C^i|\underline{a}}\{Y_C^i\} = E_{Y_C^i|\underline{a}}\{(Y_C^i)^2\} = P(c_i|\underline{a}), \tag{A.33}$$

and similarly,

$$E_{LS}\{f_{LS}^i(\underline{a})\} = E_{LS}\{(f_{LS}^i(\underline{a}))^2\} = P_{LS}(c_i|\underline{a}). \tag{A.34}$$

Combining (A.32) and (A.30), we obtain Kohavi and Wolpert's decomposition at point $\underline{a}$:

$$E_{LS}\{Err(f_{LS}(\underline{a}))\} = \sigma_{KW}(\underline{a}) + \mathrm{bias}_{KW}(\underline{a}) + \mathrm{var}_{KW}(\underline{a}), \tag{A.35}$$

with

$$\begin{aligned}
\sigma_{KW}(\underline{a}) &= \frac{1}{2}(1 - \sum_c P(c|\underline{a})^2), & \text{(A.36)}\\
\text{bias}_{KW}(\underline{a}) &= \frac{1}{2}\sum_c [P(c|\underline{a}) - P_{LS}(c|\underline{a})]^2, & \text{(A.37)}\\
\text{var}_{KW}(\underline{a}) &= \frac{1}{2}(1 - \sum_c P_{LS}(c|\underline{a})^2). & \text{(A.38)}
\end{aligned}$$

This decomposition results into a sum of three positive terms. Their definition of variance is almost (a factor 2 missing) equal to the gini entropy of the distribution $P_{LS}(c|\underline{a})$ and hence is a finer measure of the uncertainty of the prediction at point $\underline{a}$ than var_C. Note that this decomposition ignores both the Bayes model and the majority vote classifier. A consequence of this is that the bias of the Bayes model is not null.

A.1.7 Heskes [Hes98]

Heskes [Hes98] adopts the "natural" variance term var_C:

$$\text{var}_H = 1 - P_{LS}(f_{LS}^{MAJ}(\underline{a})|\underline{a}), \tag{A.39}$$

and, ignoring the residual error, defines bias as the difference between the average error and his variance:

$$\begin{aligned}
\text{bias}_H &= E_{LS}\{Err(f_{LS}(\underline{a}))\} - \text{var}_H(\underline{a}) & \text{(A.40)}\\
&= (1 - \sum_c P(c|\underline{a}) \cdot P_{LS}(c|\underline{a})) - (1 - P_{LS}(f_{LS}^{MAJ}(\underline{a})|\underline{a})) & \text{(A.41)}\\
&= \sum_c P(c|\underline{a})(P_{LS}(f_{LS}^{MAJ}(\underline{a})|\underline{a}) - P_{LS}(c|\underline{a})). & \text{(A.42)}
\end{aligned}$$

Although the two terms are positive, his bias term can not be splitted into a new positive bias term and the residual error. Indeed, it can happen that his bias is smaller than the residual error. For example, for the second algorithm of Figure A.1 (pp. 203), we have $\sigma_C(\underline{a}) = 0.3$ and $\text{bias}_H(\underline{a}) = 0.61 - (1 - 0.5) = 0.11$.

Actually, Heskes derives his decomposition as a limit case of a more general decomposition of error measure that can be derived from a loglikelihood. However, by taking the limit, he notes that the derived bias (A.40) does not satisfy all the obvious requirements that a bias/variance decomposition should satisfy.

A.2 Discussion

The multitude of decompositions which have been proposed translates well the complexity of the interaction between bias and variance in classification. This makes the choice, both in theoretical and in empirical studies, of a particular bias/variance decomposition difficult. Each decomposition has its pros and cons and has proven to be useful for a study of the bias/variance tradeoff, each one at least in the context of its introduction.

All authors which have participated to the debate agree that bias should be a measure of the discrepancy between the Bayes classifier and the majority vote classifier and that variance should be a measure of the prediction variability. For this reason, all bias terms are null when the prediction of the Bayes classifier is equal to the prediction of the majority vote classifier (except for Kohavi-Wolpert's and Heske's bias) and all variance terms are null when the model prediction is constant throughout all learning sets.

However, strictly following these definitions, we have seen that more variance can be beneficial if bias remains unchanged. Hence, when the variance and bias terms of a particular decomposition both are positive, there is necessarily a "bias effect" included in the variance term or a "variance effect" included in the bias term. For example, Kohavi and Wolpert's bias is dependent on the distribution $P_{LS}(c|\underline{a})$, rather than solely on the majority vote classifier, and thus includes some effect which would normally be assigned to variance. A consequence is that in a non noise setting, the bias is null only when $P_{LS}(f_B(\underline{a})|\underline{a}) = 1$ which also corresponds to a case of null variance. In the case of Breiman's first decomposition, no error is imputed to variance at biased points. However, it seems natural to at least attribute to variance the error caused by the non null probability of c_1 and c_3 in the two cases of Figure A.1. Similarly, the second bias term of Breiman changes when the probability $P_{LS}(c_{LS}(\underline{a})|\underline{a})$ is affected although the majority vote classifier remains the same. This dependence between bias and variance makes the previous decompositions difficult to interpret.

On the other hand, Domingo's and Tibshirani's bias terms depend only on the majority vote classifier and are easy to interpret. Although their variance can be negative, we believe that a negative variance effect better reflects the kind of interaction which exists between the two types of error in classification. If the variance term is negative, a reduction of the prediction variability which does not affect the bias is indeed going to increase the error rate. On the contrary, if the variance term is positive, a reduction of the prediction variability results in a decrease of error.

Domingos' decomposition still suffers from one drawback. It ignores the effect of the noise which is hidden in the term $c_1(\underline{a}) \cdot \sigma_C(\underline{a})$. As this term can be negative, the exact consequence of a reduction of its variance or bias terms on the error may be difficult to predict. When there is no noise on the other hand, it is easy to see that Domingos' and Tibshirani's decompositions are exactly the same. For this reason, we have prefered Tibshirani's decomposition to the other ones. Another advantage of Tibshirani's decomposition is that the evaluation of the sum of bias and irreducible error does not make use of the Bayes classifier and hence can be estimated in real-world problems where $P(c|\underline{a})$ is normally unknown (see Section 3.2.4).

Tibshirani's bias is the additional error with respect to the Bayes classifier which would remain if we could drive variance to zero without affecting the majority vote classifier. Tibshirani's variance can thus be interpreted as the modification of error resulting from a complete reduction of the prediction variability. So it is in no way a measure of the prediction variability. A complete analysis of the bias/variance profile of a learning algorithm would be advantageously completed with a more adequate measure of this variability. Kohavi and Wolpert's variance term is a good candidate for this purpose, as it satisfies the nice properties of entropy measures. However, a diminution of this variance can not be interpreted as a direct diminution of the error rate.

Appendix B

Datasets

In this appendix, we describe the datasets which are used in our experiments. First, we discuss the non temporal classification problems used in Part II and then we give the equations underlying the four artificial temporal problems used in Part III.

B.1 Classification problems

Table B.1 gives a description of the datasets used in the experiments of the second part of this thesis. All datasets are quite large and contain only numerical attributes. In Table B.1, $\#PS$ denotes the size of the pool set from which smaller learning sets of size $\#LS$ are randomly drawn for repeated building of models, $\#VS$ the size of an independent validation set used for example to prune trees or to select the optimal value of the noise level for the dual perturb and combine algorithm, and $\#TS$ is the size of the test set used to estimate errors and bias/variance decompositions.

- **Gaussian.** This synthetic database contains 20000 samples drawn each one randomly from one of two bi-dimensional Gaussian distributions (each one corresponding to one class). The first one is centered at $(0.0, 0.0)$ and has a diagonal covariance matrix, while the second one is centered at $(2.0, 2.0)$ and has a non-diagonal covariance matrix. There is a rather large overlapping between the two Gaussian distributions which results in a rather high Bayes error rate of 11.85% (See figure B.1). Notice that the Bayes classifier is of quadratic shape for the Gaussian database.

- **Waveform.** This is a well-known simulated database used as an illustration in the CART book ([BFOS84]) and by many other researchers to test their algorithms. The version we used contains 5000 cases. There are 3 equally frequent classes corresponding to three different waveforms. Each waveform is described by 20 numerical attributes, which are evenly distributed noisy samples obtained from the particular signal. The error rate of the Bayes rule was estimated to be 14% and the CART decision tree induction method is known to give about 28% on this problem.

Table B.1: Data set summaries

Data set	No. attributes	No classes	$\sigma_C(\%)$	$\#PS$	$\#LS$	$\#VS$	$\#TS$
Gaussian	2	2	11.85	14000	100	2000	4000
Waveform	21	3	14	3000	300	1000	1000
two-norm	20	2	2.3	7000	300	1000	2000
Omib	6	2	0.0	14000	500	2000	4000
Satellite	36	6	-	3435	500	1000	2000
Pendigits	16	10	-	5494	500	2000	3498
Dig44	16	10	-	9000	500	2000	4000

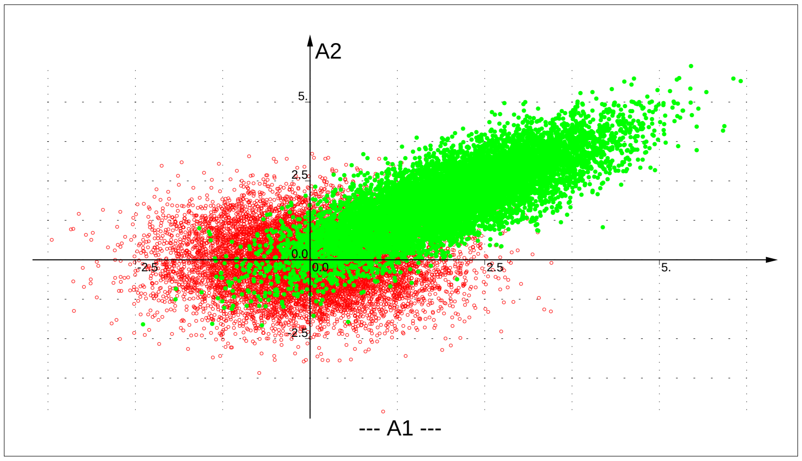


Figure B.1: GAUSSIAN problem

- **Two-norm.** This artificial problem was proposed by Breiman in [Bre96a]. It is a 20 dimensional, 2 class classification example. Each class is drawn from a multivariate normal distribution with unit covariance matrix. Class 1 has mean $(a, a, ..a)$ while Class 2 has mean $(-a, -a, .. -a)$, where $a = \frac{2}{\sqrt{(20)}}$. Breiman reports the Bayes error rate as 2.3%. The Bayes classifier is linear in this particular case.

- **Omib.** This database comes from simulations of a small electric power system [Weh98]. The state of the system is described by six numerical attributes and the goal classification tells us if this state is stable or unstable. All 6 attributes act on the output variable in a non-linear but deterministic way (there is no residual error). The database contains 20000 random system conditions.

- **Satellite.** Each object of this dataset corresponds to a small 3x3 grid of pixels corresponding to different portions of a satellite image of a particular region of the earth. Each of these grids corresponds to one of six possible types of soil. The goal of learning is to predict the type of soil. There are 36 attributes reflecting energy levels in different frequency bands obtained for each pixel in the 3x3 neighborhoods. This datasets was used in [MS94] to compare different learning algorithms. Note that the results obtained in our experiments are not comparable with those in ref. [MS94] since we use much smaller learning sets.

- **Pendigits.** This dataset corresponds to a handwriting digit recognition problem. Each object corresponds to one of the ten digits produced by some writer on a sensitive tablet. Digits are described by 16 attributes which are obtained by resampling spatially the temporal signals generated by the writer.

- **Dig44.** This problem is also a digit recognition problem. The dataset consists of examples of the digits 0 to 9 gathered from postcodes on letters in Germany. Examples were digitized onto images with 16x16 pixels and then 16 attributes were extracted by averaging over 4x4 neighborhoods in the original images. Like satellite, this dataset is described in [MS94]. The results obtained in our experiments are again not comparable with those in ref. [MS94] since we use much smaller learning sets.

All these datasets are available at the UCI repository of machine learning [BM98], except for Gaussian and Omib.

B.2 Temporal classification problems

We give in this section, the exact equations that define the four artificial time-series classification problems used in the third part of the thesis.

B.2.1 CBF problem

Equations are taken from [Sai94]. Each object is described by one temporal attribute defined on the integer time axis $0, 1, \ldots, 127$ by:

$$A(o,t) = \begin{cases} (6+\eta).\chi_{[a,b]}(t) + \epsilon(t) & \text{if } C(o) = c, \\ (6+\eta).\chi_{[a,b]}(t).(t-a)/(b-a) + \epsilon(t) & \text{if } C(o) = b, \\ (6+\eta).\chi_{[a,b]}(t).(b-t)/(b-a) + \epsilon(t) & \text{if } C(o) = f, \end{cases}$$

where

$$\chi_{[a,b]}(t) = \begin{cases} 1 & \text{if } a \leq t \leq b \\ 0 & \text{otherwise} \end{cases}$$

In the original problem, η and $\epsilon(t)$ are drawn from a standard normal distribution $N(0,1)$, a is an integer drawn uniformly from $[15,31]$ and $b-a$ is an integer drawn uniformly from $[32,96]$.

B.2.2 Cbf-tr

We have designed this dataset specifically for this research. With respect to the previous problem, the only difference is that a is drawn from $[0,64]$ and $b-a$ is drawn from $[32,64]$.

B.2.3 Two-pat

This dataset has also be designed by us for our experiments. In this problem, each scenario is described by a single temporal input attribute A which is defined on the integer time axis $0, 1, \ldots, 127$ by the following function:

$$A(o,t) = \begin{cases} \epsilon(t) & \text{if } 0 \leq t < t_1 \\ s_1(t-t_1, l_1) & \text{if } t_1 \leq t < t_1 + l_1 \\ \epsilon(t) & \text{if } t_1 + l_1 \leq t < t_2 \\ s_2(t-t_2, l_2) & \text{if } t_2 \leq t < t_2 + l_2 \\ \epsilon(t) & \text{if } t_2 + l_2 \leq t < 128 \end{cases} \tag{B.1}$$

where the time functions s_1 and s_2 are defining the patterns of respective lengths l_1 and l_2. $\epsilon(t)$ is some noise surrounding the patterns, drawn from a standard normal distribution $N(0,1)$. l_1 and l_2 are integers uniformly drawn in $[16,32]$ and t_1 and t_2 are integers randomly drawn in $[0,127]$ such that the two patterns are not overlapping each other and are fully comprised in the interval $[0,127]$ (i.e., mathematically, such that $t_1 + l_1 < t_2$ and $t_2 + l_2 < 128$).

As possible instances for s_1 and s_2, we consider simple upward and downward steps defined by the following time function where l is the length of the pattern:

$$us(t,l) = \begin{cases} -5 & \text{if } 0 \leq t < \frac{l}{2} \\ 5 & \text{if } \frac{l}{2} \leq t < l \end{cases} \tag{B.2}$$

$$ds(t,l) = \begin{cases} 5 & \text{if } 0 \leq t < \frac{l}{2} \\ -5 & \text{if } \frac{l}{2} \leq t < l \end{cases} \tag{B.3}$$

s_1 and s_2 are independently chosen as being either us or ds (each one with probability 0.5). According to this random choice, the classification of a scenario in one of four classes is defined as follows:

$$c(o) = \begin{cases} uu & \text{if } s_1 = s_2 = us \\ ud & \text{if } s_1 = us \text{ and } s_2 = ds \\ du & \text{if } s_1 = ds \text{ and } s_2 = us \\ dd & \text{if } s_1 = s_2 = ds \end{cases} \tag{B.4}$$

The four classes are thus equiprobable.

B.2.4 CC

This problem is obtained from [AM99]. There are six classes and the objects are described by one temporal attribute on the discrete time interval $[1, 60]$. Denoting by $r(t)$ a random real number in $[-3, 3]$ and taking $m = 30$ and $s = 2$, each of the six classes is defined as follows:

- Normal: $a(o, t) = m + sr(t)$.
- Cyclic: $a(o, t) = m + sr(t) + a\sin(2\pi t/T)$, with a and T randomly selected in $[10, 15]$.
- Increasing: $a(o, t) = m + sr(t) + gt$, g a random number in $[0.2, 0.5]$.
- Decreasing: $a(o, t) = m + sr(t) - gt$.
- Upward: $a(o, t) = m + sr(t) + x.1(t < t_0)$, with x drawn in $[7.5, 20]$ and t_0 drawn in $[n/3, 2n/3]$.
- Downward: $a(o, t) = m + sr(t) - x.1(t < t_o)$.

The data used in our experiments is taken from the UCI KDD Archive [BM98]. It contains 100 examples per class.

Appendix C

Experimental results of part II

This appendix contains all the experimental results analyzed and summarized in Chapter 6 to Chapter 8 of the second part of this thesis. Each one of the four sections of this appendix corresponds to one of the four chapters of Part II. The results are provided only for reference when reading the main text. Hence, we do not recall here the experimental protocol used to generate them. The precise meaning of the values gathered in the tables is given in the corresponding part of the main text of the thesis. To facilitate the search for a particular result, the caption of each table provides the section of the chapter which discusses these results. For information, the generation of these tables has necessitated the construction of more than 100,000 decision trees with our software.

C.1 Results of Chapter 5

Table C.1: Prediction variance of fully grown trees (see Section 5.4.1)

	Mean error	compl	bias_T	$\text{var}_T(\%)$	$\text{var}_{KW}(\%)$	$\text{var}_{\hat{P}}(\%)$
Gaussian (#ls=100)	0.1725	30	0.1192	0.0533	0.0876	0.0876
Waveform (#ls=300)	0.2970	74	0.1630	0.1340	0.1737	0.1158
Two-norm (#ls=300)	0.2112	50	0.0380	0.1732	0.1473	0.1473
Omib (#ls=500)	0.1202	74	0.0487	0.0714	0.0775	0.0775
Satellite (#ls=500)	0.2087	114	0.1185	0.0902	0.1185	0.0395
Pendigits (#ls=500)	0.1878	98	0.0803	0.1075	0.1190	0.0238
Dig44 (#ls=500)	0.3052	181	0.1160	0.1892	0.1979	0.0396

Table C.2: Prediction variance with learning sample size (see Section 5.4.2)

	Mean error	compl	bias_T	$\text{var}_T(\%)$	$\text{var}_{KW}(\%)$	$\text{var}_{\hat{P}}(\%)$
GAUSSIAN (N=100)						
N	0.1725	30	0.1192	0.0533	0.0876	0.0876
$2 \cdot N$	0.1729	58	0.1160	0.0569	0.0874	0.0874
$4 \cdot N$	0.1670	111	0.1173	0.0498	0.0818	0.0818
WAVEFORM (N=300)						
N	0.2970	74	0.1630	0.1340	0.1737	0.1158
$2 \cdot N$	0.2767	140	0.1540	0.1227	0.1581	0.1054
$4 \cdot N$	0.2680	269	0.1690	0.0990	0.1473	0.0982
TWO-NORM (N=300)						
N	0.2112	50	0.0380	0.1732	0.1473	0.1473
$2 \cdot N$	0.1949	95	0.0340	0.1608	0.1357	0.1357
$4 \cdot N$	0.1823	180	0.0360	0.1463	0.1267	0.1267
OMIB (N=500)						
N	0.1202	74	0.0487	0.0714	0.0775	0.0775
$2 \cdot N$	0.1017	128	0.0342	0.0675	0.0665	0.0665
$4 \cdot N$	0.0893	218	0.0320	0.0573	0.0586	0.0586
SATELLITE (N=500)						
N	0.2087	114	0.1185	0.0902	0.1185	0.0395
$2 \cdot N$	0.1862	208	0.1095	0.0767	0.1045	0.0348
$4 \cdot N$	0.1708	375	0.1100	0.0608	0.0945	0.0315
PENDIGITS (N=500)						
N	0.1878	98	0.0803	0.1075	0.1190	0.0238
$2 \cdot N$	0.1522	149	0.0726	0.0796	0.0962	0.0192
$4 \cdot N$	0.1168	223	0.0537	0.0630	0.0724	0.0145
DIG44 (N=500)						
N	0.3052	181	0.1160	0.1892	0.1979	0.0396
$2 \cdot N$	0.2586	307	0.0953	0.1633	0.1661	0.0332
$4 \cdot N$	0.2227	517	0.0880	0.1347	0.1411	0.0282

Table C.3: Prediction variance with tree complexity (see Section 5.4.3)

	Mean error	compl	bias_T	$\text{var}_T(\%)$	$\text{var}_{KW}(\%)$	$\text{var}_{\hat{P}}(\%)$
GAUSSIAN (#ls=100)						
Stump	0.1883	3	0.1497	0.0386	0.0829	0.0427
$0.25 \cdot \overline{C(\mathcal{T})}$	0.1544	9	0.1213	0.0332	0.0646	0.0356
$0.5 \cdot \overline{C(\mathcal{T})}$	0.1569	15	0.1175	0.0394	0.0711	0.0497
Normal	0.1725	30	0.1192	0.0533	0.0876	0.0876
WAVEFORM (#ls=300)						
Stump	0.4461	3	0.3930	0.0531	0.1749	0.0246
$0.25 \cdot \overline{C(\mathcal{T})}$	0.3053	19	0.1830	0.1223	0.1723	0.0583
$0.5 \cdot \overline{C(\mathcal{T})}$	0.2941	37	0.1650	0.1291	0.1690	0.0794
Normal	0.2970	74	0.1630	0.1340	0.1737	0.1158
TWO-NORM (#ls=300)						
Stump	0.3372	3	0.1041	0.2331	0.2069	0.0471
$0.25 \cdot \overline{C(\mathcal{T})}$	0.2616	13	0.0571	0.2045	0.1747	0.1005
$0.5 \cdot \overline{C(\mathcal{T})}$	0.2335	25	0.0420	0.1915	0.1602	0.1202
Normal	0.2112	50	0.0380	0.1732	0.1473	0.1473
OMIB (#ls=500)						
Stump	0.2294	3	0.2157	0.0136	0.0518	0.0139
$0.25 \cdot \overline{C(\mathcal{T})}$	0.1505	19	0.0825	0.0680	0.0891	0.0473
$0.5 \cdot \overline{C(\mathcal{T})}$	0.1305	37	0.0582	0.0723	0.0823	0.0576
Normal	0.1202	74	0.0487	0.0714	0.0775	0.0775
SATELLITE (#ls=500)						
Stump	0.5899	3	0.5820	0.0078	0.1693	0.0071
$0.25 \cdot \overline{C(\mathcal{T})}$	0.2147	29	0.1525	0.0621	0.1074	0.0180
$0.5 \cdot \overline{C(\mathcal{T})}$	0.2054	57	0.1345	0.0709	0.1087	0.0246
Normal	0.2087	114	0.1185	0.0902	0.1185	0.0395
PENDIGITS (#ls=500)						
Stump	0.8029	3	0.7933	0.0096	0.3508	0.0016
$0.25 \cdot \overline{C(\mathcal{T})}$	0.2951	25	0.2213	0.0738	0.1419	0.0160
$0.5 \cdot \overline{C(\mathcal{T})}$	0.2135	49	0.1032	0.1103	0.1251	0.0189
Normal	0.1878	98	0.0803	0.1075	0.1190	0.0238
DIG44 (#ls=500)						
Stump	0.7843	3	0.7795	0.0048	0.3575	0.0022
$0.25 \cdot \overline{C(\mathcal{T})}$	0.3253	47	0.1768	0.1486	0.1911	0.0203
$0.5 \cdot \overline{C(\mathcal{T})}$	0.3121	91	0.1355	0.1766	0.1967	0.0271
Normal	0.3052	181	0.1160	0.1892	0.1979	0.0396

Table C.4: Source of variance in decision tree induction, fully grown trees (see 5.5)

	Mean error	compl	bias_T	$\text{var}_T(\%)$	$\text{var}_{KW}(\%)$	$\text{var}_{\hat{P}}(\%)$
GAUSSIAN (#ls=300)						
Normal	0.1678	84	0.1195	0.0483(100)	0.0829(100)	0.0829(100)
Att. fixed	0.1672	90	0.1187	0.0484(100)	0.0820(99)	0.0820(99)
All fixed	0.1305	53	0.1180	0.0125(26)	0.0377(45)	0.0131(16)
WAVEFORM (#ls=300)						
Normal	0.2950	74	0.1570	0.1380(100)	0.1739(100)	0.1160(100)
Att. fixed	0.2847	114	0.1920	0.0927(67)	0.1542(89)	0.1028(89)
All fixed	0.2667	103	0.2590	0.0077(6)	0.0737(42)	0.0267(23)
TWO-NORM (#ls=300)						
Normal	0.2091	52	0.0325	0.1766(100)	0.1476(100)	0.1476(100)
Att. fixed	0.1975	86	0.0620	0.1355(77)	0.1324(90)	0.1324(90)
All fixed	0.1467	103	0.0935	0.0532(30)	0.0690(47)	0.0426(29)
OMIB (#ls=500)						
Normal	0.1194	74	0.0480	0.0714(100)	0.0772(100)	0.0772(100)
Att. fixed	0.1071	88	0.0445	0.0626(88)	0.0688(89)	0.0688(89)
All fixed	0.0932	137	0.0768	0.0165(23)	0.0380(49)	0.0304(39)
SATELLITE (#ls=500)						
Normal	0.2072	116	0.1235	0.0837(100)	0.1176(100)	0.0392(100)
Att. fixed	0.1906	164	0.1200	0.0706(84)	0.1040(88)	0.0346(88)
All fixed	0.1441	110	0.1245	0.0196(23)	0.0515(44)	0.0100(25)
PENDIGITS (#ls=500)						
Normal	0.1860	96	0.0718	0.1142(100)	0.1199(100)	0.0240(100)
Att. fixed	0.1708	124	0.0975	0.0733(64)	0.0988(82)	0.0197(82)
All fixed	0.1305	113	0.1126	0.0179(16)	0.0429(36)	0.0080(33)
DIG44 (#ls=500)						
Normal	0.3093	178	0.1142	0.1951(100)	0.1989(100)	0.0398(100)
Att. fixed	0.2676	236	0.1297	0.1379(71)	0.1633(82)	0.0324(81)
All fixed	0.2071	203	0.1698	0.0374(19)	0.0754(38)	0.0121(30)

Table C.5: Source of variance in decision tree induction, ten test nodes (see Section 5.5)

	Mean error	compl	bias_T	$\text{var}_T(\%)$	$\text{var}_{KW}(\%)$	$\text{var}_{\hat{P}}(\%)$
GAUSSIAN (#ls=300)						
Normal	0.1454	21	0.1140	0.0314(100)	0.0594(100)	0.0290(100)
Att. fixed	0.1386	21	0.1135	0.0251(80)	0.0508(85)	0.0249(86)
All fixed	0.1258	21	0.1220	0.0038(12)	0.0247(42)	0.0054(19)
WAVEFORM (#ls=300)						
Normal	0.3029	21	0.1790	0.1239(100)	0.1733(100)	0.0603(100)
Att. fixed	0.2958	21	0.2100	0.0858(69)	0.1483(86)	0.0414(69)
All fixed	0.2916	21	0.2870	0.0046(4)	0.0386(22)	0.0048(8)
TWO-NORM (#ls=300)						
Normal	0.2328	21	0.0390	0.1938(100)	0.1619(100)	0.1125(100)
Att. fixed	0.2333	21	0.0910	0.1423(73)	0.1489(92)	0.0886(79)
All fixed	0.2283	21	0.2165	0.0118(6)	0.0653(40)	0.0084(7)
OMIB (#ls=500)						
Normal	0.1508	21	0.0763	0.0746(100)	0.0909(100)	0.0489(100)
Att. fixed	0.1469	21	0.0822	0.0646(87)	0.0871(96)	0.0464(95)
All fixed	0.1330	21	0.1268	0.0063(8)	0.0302(33)	0.0032(7)
SATELLITE (#ls=500)						
Normal	0.2202	21	0.1640	0.0562(100)	0.1049(100)	0.0171(100)
Att. fixed	0.2108	21	0.1790	0.0318(57)	0.0818(78)	0.0110(64)
All fixed	0.1968	21	0.1865	0.0103(18)	0.0493(47)	0.0013(7)
PENDIGITS (#ls=500)						
Normal	0.3374	21	0.2844	0.0530(100)	0.1488(100)	0.0146(100)
Att. fixed	0.3517	21	0.3270	0.0246(46)	0.1039(70)	0.0083(57)
All fixed	0.3419	21	0.3308	0.0112(21)	0.0455(31)	0.0008(6)
DIG44 (#ls=500)						
Normal	0.4391	21	0.3103	0.1289(100)	0.2236(100)	0.0157(100)
Att. fixed	0.4202	21	0.3530	0.0672(52)	0.1454(65)	0.0098(62)
All fixed	0.3511	21	0.3500	0.0011(1)	0.0014(1)	0.0012(8)

Table C.6: Discretization threshold variance with various score measures (see Section 5.6.1)

method	$N = 50$		$N = 500$		$N = 2000$	
	σ_{Th} (%)	Thres. bias	σ_{Th} (%)	Thres. bias	σ_{Th} (%)	Thres. bias
GAUSSIAN ($\sigma_A = 1.4619$, $A_{th}^{\infty} = 1.2414219$)						
Score$_W$	0.5855(40)	-0.2102	0.3205(22)	0.0090	0.2146(15)	-0.0688
Score$_{\text{info}}$	0.5494(38)	-0.2433	0.2948(20)	-0.0670	0.1934(13)	-0.1210
Score$_Q$	0.7268(50)	-0.2426	0.3715(25)	0.0478	0.2783(19)	0.0530
Score$_{\text{med}}$	0.3140(21)	-0.2913	0.0919(6)	-0.3272	0.0471(3)	-0.3211
Score$_{KS}$	0.3402(23)	-0.2827	0.1898(13)	-0.2232	0.1285(9)	-0.2587
Score$_{Err}$	0.3622(25)	-0.3434	0.2028(14)	-0.2412	0.1308(9)	-0.2622
WAVEFORM ($\sigma_A = 2.01557$, $A_{th}^{\infty} = 2.565$)						
Score$_W$	0.7329(36)	-0.0654	0.4271(21)	-0.0828	0.1232(6)	-0.0371
Score$_{\text{info}}$	0.7026(35)	-0.0676	0.3657(18)	-0.0931	0.1015(5)	-0.0221
Score$_Q$	1.3028(65)	-0.3266	0.5919(29)	-0.1578	0.2022(10)	-0.0360
Score$_{\text{med}}$	0.3679(18)	-0.1040	0.1367(7)	-0.0385	0.0522(3)	-0.0575
Score$_{KS}$	0.5300(26)	-0.2614	0.2394(12)	-0.2137	0.1301(6)	-0.1177
Score$_{Err}$	0.5634(28)	-0.4506	0.2444(12)	-0.1643	0.1548(8)	-0.0927
TWO-NORM ($\sigma_A = 1.0938$, $A_{th}^{\infty} = 0.391884$)						
Score$_W$	0.9183(84)	-0.4430	0.5661(52)	-0.4376	0.3458(32)	-0.5424
Score$_{\text{info}}$	0.7740(71)	-0.4833	0.4226(39)	-0.4275	0.2289(21)	-0.4750
Score$_Q$	1.3266(121)	-0.2969	1.7452(160)	-0.6224	1.6638(152)	-1.4315
Score$_{\text{med}}$	0.1870(17)	-0.4400	0.0577(5)	-0.3783	0.0238(2)	-0.3857
Score$_{KS}$	0.4232(39)	-0.4683	0.2431(22)	-0.4190	0.1383(13)	-0.4593
Score$_{Err}$	0.5276(48)	-0.5557	0.2591(24)	-0.4433	0.1418(13)	-0.4700
OMIB ($\sigma_A = 171.1327$, $A_{th}^{\infty} = 1060.695$)						
Score$_W$	96.5026(56)	6.2134	47.3234(28)	-7.0138	35.8138(21)	-3.7539
Score$_{\text{info}}$	87.3155(51)	-5.1421	43.1451(25)	-18.7450	26.7255(16)	-20.9630
Score$_Q$	116.8300(68)	76.4987	93.8516(55)	78.9720	48.1274(28)	31.5532
Score$_{\text{med}}$	41.6863(24)	-52.6901	12.9687(8)	-60.6760	6.9955(4)	-60.3774
Score$_{KS}$	63.5767(37)	-3.0804	26.3415(15)	-16.1272	17.1937(10)	-18.3568
Score$_{Err}$	61.5731(36)	55.5413	27.4454(16)	56.1267	13.6702(8)	61.8505
SATELLITE ($\sigma_A = 13.5366$, $A_{th}^{\infty} = 77.0$)						
Score$_W$	6.1666(46)	-0.7400	1.6386(12)	0.4550	0.9236(7)	0.5950
Score$_{\text{info}}$	5.0225(37)	-1.5450	2.0371(15)	-0.4800	1.1349(8)	-0.1100
Score$_Q$	13.7581(102)	-6.8950	14.4146(106)	-6.9550	8.9967(66)	-0.1200
Score$_{\text{med}}$	2.4338(18)	-8.5650	0.6031(4)	-9.3450	0.3782(3)	-9.1850
Score$_{KS}$	13.6245(101)	-7.5500	2.8994(21)	0.9750	0.6254(5)	1.3800
Score$_{Err}$	6.3308(47)	-3.1600	1.5438(11)	0.1550	0.4630(3)	0.0300
PENDIGITS ($\sigma_A = 35.7765$, $A_{th}^{\infty} = 24.5$)						
Score$_W$	12.2439(34)	-3.5650	5.4438(15)	-1.2950	2.8582(8)	-2.9700
Score$_{\text{info}}$	11.7152(33)	-4.4550	4.5207(13)	-2.2250	2.8892(8)	-3.3500
Score$_Q$	21.6334(60)	8.1350	14.6218(41)	23.2850	14.4483(40)	24.6900
Score$_{\text{med}}$	8.0132(22)	-13.7020	1.9878(6)	-15.5350	0.7895(2)	-16.3150
Score$_{KS}$	26.2262(73)	32.0300	20.9179(58)	15.6000	11.5584(32)	19.6800
Score$_{Err}$	14.5203(41)	-12.6950	11.3976(32)	-12.5350	8.4447(24)	-16.2600
DIG44 ($\sigma_A = 71.1541$, $A_{th}^{\infty} = 103.149994$)						
Score$_W$	36.6957(52)	-14.7060	15.7411(22)	-6.1865	7.3613(10)	-5.2945
Score$_{\text{info}}$	35.9359(51)	-18.5750	13.4792(19)	-8.7540	8.1819(11)	-7.4915
Score$_Q$	55.5525(78)	39.9125	47.8297(67)	58.0010	32.7944(46)	28.6040
Score$_{\text{med}}$	18.3232(26)	-34.4560	6.0664(9)	-33.8595	2.4686(3)	-34.4245
Score$_{KS}$	66.1367(93)	14.0375	13.7154(19)	32.2580	6.8888(10)	29.0315
Score$_{Err}$	40.4877(57)	-35.5190	27.7125(39)	-16.1415	12.2102(17)	-0.6535

Table C.7: Global effect of score measures on decision tree accuracy (see Section 5.6.3)

	Mean error	compl	bias_T	var_T	var_{KW}	$\text{var}_{\hat{P}}$
GAUSSIAN						
Score_W	0.1771	29	0.1175	0.0596	0.0910	0.0910
$\text{Score}_{\text{info}}$	0.1757	28	0.1172	0.0584	0.0891	0.0891
Score_Q	0.1779	29	0.1165	0.0614	0.0909	0.0909
$\text{Score}_{\text{med}}$	0.1685	48	0.1200	0.0485	0.0810	0.0765
Score_{KS}	0.1714	28	0.1180	0.0534	0.0858	0.0858
Score_{Err}	0.1769	107	0.1190	0.0579	0.0904	0.0904
WAVEFORM						
Score_W	0.2985	76	0.1530	0.1455	0.1752	0.1168
$\text{Score}_{\text{info}}$	0.2972	73	0.1750	0.1222	0.1710	0.1140
Score_Q	0.3122	111	0.1790	0.1332	0.1808	0.1206
$\text{Score}_{\text{med}}$	0.3188	156	0.1690	0.1498	0.1875	0.1250
Score_{KS}	0.3004	79	0.1650	0.1354	0.1742	0.1161
Score_{Err}	0.3179	259	0.1570	0.1609	0.1890	0.1260
TWO-NORM						
Score_W	0.2152	50	0.0330	0.1822	0.1503	0.1503
$\text{Score}_{\text{info}}$	0.2118	47	0.0310	0.1808	0.1484	0.1484
Score_Q	0.2023	79	0.0641	0.1382	0.1339	0.1339
$\text{Score}_{\text{med}}$	0.2154	109	0.0526	0.1629	0.1453	0.1453
Score_{KS}	0.2102	50	0.0340	0.1761	0.1462	0.1462
Score_{Err}	0.2263	239	0.0395	0.1868	0.1558	0.1558
OMIB						
Score_W	0.1196	74	0.0432	0.0763	0.0774	0.0774
$\text{Score}_{\text{info}}$	0.1121	69	0.0445	0.0676	0.0720	0.0720
Score_Q	0.1346	92	0.0537	0.0808	0.0863	0.0863
$\text{Score}_{\text{med}}$	0.1206	169	0.0537	0.0669	0.0750	0.0737
Score_{KS}	0.1127	74	0.0532	0.0595	0.0707	0.0707
Score_{Err}	0.2077	711	0.1148	0.0929	0.1166	0.1166
SATELLITE						
Score_W	0.2089	117	0.1230	0.0859	0.1181	0.0394
$\text{Score}_{\text{info}}$	0.2060	107	0.1200	0.0860	0.1179	0.0393
Score_Q	0.2139	146	0.1315	0.0824	0.1173	0.0391
$\text{Score}_{\text{med}}$	0.2241	189	0.1250	0.0991	0.1309	0.0436
Score_{KS}	0.2145	117	0.1245	0.0901	0.1239	0.0413
Score_{Err}	0.2289	205	0.1380	0.0909	0.1298	0.0433
PENDIGITS						
Score_W	0.1831	95	0.0803	0.1028	0.1162	0.0232
$\text{Score}_{\text{info}}$	0.1851	96	0.0838	0.1013	0.1172	0.0234
Score_Q	0.1802	98	0.0789	0.1013	0.1110	0.0222
$\text{Score}_{\text{med}}$	0.2185	215	0.0929	0.1256	0.1413	0.0283
Score_{KS}	0.1933	114	0.0700	0.1232	0.1268	0.0254
Score_{Err}	0.2009	147	0.0812	0.1197	0.1306	0.0261
DIG44						
Score_W	0.2964	180	0.1048	0.1916	0.1931	0.0386
$\text{Score}_{\text{info}}$	0.3000	180	0.1098	0.1903	0.1950	0.0390
Score_Q	0.3232	209	0.1158	0.2074	0.2103	0.0421
$\text{Score}_{\text{med}}$	0.3545	367	0.1260	0.2284	0.2307	0.0461
Score_{KS}	0.3499	227	0.1128	0.2371	0.2301	0.0460
Score_{Err}	0.3320	261	0.1065	0.2255	0.2207	0.0441

C.2 Results of Chapter 6

Table C.8: Effect of pruning on bias and variance (see Section 6.2.2)

	Mean error	compl	bias_T	$\text{var}_T(\%)$	$\text{var}_{KW}(\%)$	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}(\%)$
GAUSSIAN ($VS = 2000$)								
Full tree	0.1757	31	0.1182	0.0574	0.0909	0.1757	0.0848	0.0909
Pruned tree	0.1372	9	0.1157	0.0215	0.0477	0.1199	0.0862	0.0337
WAVEFORM ($VS = 1000$)								
Full tree	0.2987	76	0.1530	0.1457	0.1752	0.1991	0.0823	0.1168
Pruned tree	0.2873	39	0.1670	0.1203	0.1613	0.1652	0.0857	0.0795
TWO-NORM ($VS = 1000$)								
Full tree	0.2159	51	0.0350	0.1809	0.1507	0.2159	0.0652	0.1507
Pruned tree	0.2143	46	0.0360	0.1783	0.1496	0.2099	0.0654	0.1445
OMIB ($VS = 2000$)								
Full tree	0.1213	75	0.0420	0.0793	0.0789	0.1213	0.0424	0.0789
Pruned tree	0.1223	54	0.0530	0.0693	0.0771	0.1161	0.0463	0.0698
SATELLITE ($VS = 1000$)								
Full tree	0.2078	117	0.1240	0.0838	0.1181	0.0693	0.0299	0.0394
Pruned tree	0.1986	46	0.1400	0.0586	0.0992	0.0561	0.0334	0.0227
PENDIGITS ($VS = 2000$)								
Full tree	0.1903	97	0.0812	0.1091	0.1196	0.0381	0.0141	0.0239
Pruned tree	0.1895	88	0.0818	0.1077	0.1181	0.0369	0.0144	0.0226
DIG44 ($VS = 2000$)								
Full tree	0.3030	181	0.1230	0.1800	0.1951	0.0606	0.0216	0.0390
Pruned tree	0.2965	112	0.1203	0.1762	0.1875	0.0532	0.0229	0.0302

Table C.9: Discretization threshold variance with variance reduction techniques (see Section 6.3.2)

method	$N = 50$			$N = 500$			$N = 2000$		
	$\sqrt{Err.}$	bias	σ_{Th}	$\sqrt{Err.}$	bias	σ_{Th}	$\sqrt{Err.}$	bias	σ_{Th}
GAUSSIAN ($\sigma_A = 1.4619$, $A_{th}^{\infty} = 1.2414219$)									
Classic	0.619	-0.138	0.604	0.305	-0.099	0.289	0.226	-0.034	0.224
Smoothing	0.493 (0.4)	-0.177	0.460	0.207 (0.3)	-0.104	0.179	0.140 (0.2)	-0.099	0.098
Aggregation	0.319 (0.8)	-0.253	0.195	0.194 (0.6)	-0.175	0.084	0.136 (0.2)	-0.100	0.092
Averaging	0.329 (3.5)	-0.224	0.241	0.174 (3.5)	-0.132	0.113	0.131 (2.5)	-0.101	0.083
Bootstrap	0.403	-0.181	0.360	0.205	-0.093	0.183	0.152	-0.080	0.129
WAVEFORM ($\sigma_A = 2.01557$, $A_{th}^{\infty} = 2.565$)									
Classic	0.687	-0.084	0.682	0.358	-0.115	0.339	0.162	-0.058	0.151
Smoothing	0.521 (0.4)	-0.089	0.514	0.131 (0.6)	-0.021	0.129	0.066 (0.4)	-0.021	0.062
Aggregation	0.298 (0.8)	0.061	0.291	0.114 (0.8)	0.029	0.110	0.063 (0.4)	-0.015	0.061
Averaging	0.414 (2.0)	-0.086	0.405	0.146 (3.5)	0.013	0.145	0.076 (3.0)	0.010	0.075
Bootstrap	0.439	-0.179	0.401	0.288	-0.128	0.258	0.143	-0.077	0.121
TWO-NORM ($\sigma_A = 1.0938$, $A_{th}^{\infty} = 0.391884$)									
Classic	0.858	-0.075	0.855	0.534	0.055	0.532	0.406	0.087	0.396
Smoothing	0.858 (0.0)	-0.075	0.855	0.376 (0.4)	0.141	0.349	0.171 (0.7)	0.148	0.086
Aggregation	0.234 (0.9)	0.157	0.174	0.171 (0.9)	0.155	0.072	0.147 (0.8)	0.130	0.070
Averaging	0.303 (3.0)	0.101	0.286	0.240 (3.5)	0.147	0.190	0.154 (3.5)	0.083	0.130
Bootstrap	0.413	0.030	0.411	0.330	0.068	0.323	0.267	0.088	0.252
OMIB ($\sigma_A = 171.1327$, $A_{th}^{\infty} = 1060.695$)									
Classic	108.2	23.1	105.7	57.8	-13.0	56.3	35.8	1.6	35.7
Smoothing	108.2 (0.0)	23.1	105.7	35.6 (0.5)	-23.6	26.7	17.8 (0.3)	-7.3	16.2
Aggregation	43.5(0.6)	-23.4	36.7	32.4 (0.5)	-22.8	23.0	17.2 (0.3)	-6.2	16.0
Averaging	56.1 (2.5)	-44.3	34.4	33.4 (2.3)	-23.3	24.0	17.2 (2.0)	-7.3	15.6
Bootstrap	63.5	29.2	56.3	40.0	-6.7	39.4	23.3	0.7	23.3
SATELLITE ($\sigma_A = 13.5366$, $A_{th}^{\infty} = 77.0$)									
Classic	6.53	-1.14	6.43	2.05	0.21	2.04	1.07	0.54	0.92
Smoothing	5.69 (0.2)	-2.45	5.14	0.97 (0.2)	-0.33	0.91	0.60 (0.2)	-0.24	0.55
Aggregation	5.22 (0.1)	-2.01	4.82	0.93 (0.2)	-0.29	0.89	0.67 (0.2)	0.04	0.67
Averaging	6.53(0.0)	-1.14	6.43	1.42 (1.5)	-0.85	1.14	0.80 (0.5)	-0.07	0.80
Bootstrap	5.29	-2.47	4.68	1.39	0.18	1.38	0.77	0.42	0.65
PENDIGITS ($\sigma_A = 35.7765$, $A_{th}^{\infty} = 24.5$)									
Classic	15.23	-0.84	15.21	6.87	-1.81	6.63	3.84	-2.43	2.98
Smoothing	11.47 (0.3)	-2.89	11.10	4.06 (0.2)	-1.75	3.67	1.52 (0.3)	-0.47	1.45
Aggregation	6.38 (0.7)	3.31	5.45	4.30 (0.3)	-0.05	4.30	1.81 (0.3)	-0.13	1.81
Averaging	9.69 (2.0)	1.11	9.63	4.94 (1.0)	-2.04	4.50	3.13 (2.0)	-2.48	1.91
Bootstrap	10.62	-1.85	10.45	5.14	-1.96	4.75	3.28	-2.72	1.83
DIG44 ($\sigma_A = 71.1541$, $A_{th}^{\infty} = 103.149994$)									
Classic	36.86	-14.49	33.89	15.66	-6.18	14.39	10.08	-6.29	7.88
Smoothing	34.03 (0.3)	-15.83	30.12	12.66 (0.5)	-9.81	8.00	8.35 (0.1)	-6.13	5.67
Aggregation	21.32 (0.9)	-18.13	11.22	11.99 (0.5)	-9.77	6.96	8.54 (0.1)	-6.69	5.30
Averaging	19.47 (3.5)	-12.34	15.06	10.32 (3.5)	-7.90	6.64	9.18 (1.0)	-7.60	5.16
Bootstrap	27.01	-14.56	22.75	13.17	-6.46	11.47	8.11	-6.20	5.23

Table C.10: Threshold variance reduction techniques, fixed attribute choice, fully grown trees (see Section 6.3.3)

	Mean error	compl	bias_T	$\text{var}_T(\%)$	$\text{var}_{KW}(\%)$	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}(\%)$
GAUSSIAN								
Classic	0.1608	25	0.1180	0.0428	0.0760	0.1456	0.0841	0.0615
Smoothing	0.1521(0.5)	38	0.1197	0.0323	0.0624	0.1321	0.0859	0.0462
Aggregation	0.1578(0.5)	48	0.1195	0.0383	0.0725	0.1507	0.0840	0.0667
Averaging	0.1568(2.0)	41	0.1180	0.0388	0.0718	0.1467	0.0840	0.0627
Local boot.	0.1618	41	0.1197	0.0420	0.0761	0.1551	0.0851	0.0700
Global boot.	0.1350	17	0.1158	0.0192	0.0442	0.1115	0.0852	0.0263
WAVEFORM								
Classic	0.2864	89	0.1970	0.0894	0.1506	0.1654	0.0903	0.0751
Smoothing	0.2849(0.1)	101	0.1970	0.0879	0.1520	0.1628	0.0899	0.0729
Aggregation	0.2909(0.1)	123	0.2010	0.0899	0.1524	0.1695	0.0924	0.0771
Averaging	0.2883(1.0)	112	0.1990	0.0893	0.1480	0.1629	0.0929	0.0700
Local boot.	0.2872	149	0.2100	0.0772	0.1473	0.1664	0.0920	0.0744
Global boot.	0.2788	73	0.2190	0.0598	0.1274	0.1403	0.1000	0.0402
TWO-NORM								
Classic	0.2236	73	0.0985	0.1250	0.1406	0.1956	0.0846	0.1110
Smoothing	0.2258(0.2)	89	0.1041	0.1218	0.1394	0.1912	0.0908	0.1004
Aggregation	0.2324(0.3)	129	0.1221	0.1103	0.1359	0.2004	0.0986	0.1017
Averaging	0.2315(1.0)	98	0.1061	0.1254	0.1438	0.1983	0.0906	0.1077
Local boot.	0.2340	136	0.1206	0.1135	0.1394	0.2072	0.0963	0.1109
Global boot.	0.2349	59	0.1436	0.0913	0.1298	0.1776	0.1146	0.0630
OMIB								
Classic	0.1089	78	0.0398	0.0692	0.0709	0.1044	0.0387	0.0657
Smoothing	0.1094(0.1)	89	0.0450	0.0644	0.0699	0.1008	0.0406	0.0602
Aggregation	0.1013(0.6)	153	0.0535	0.0478	0.0593	0.0925	0.0422	0.0503
Averaging	0.1039(1.0)	107	0.0365	0.0674	0.0673	0.0981	0.0372	0.0609
Local boot.	0.1109	144	0.0473	0.0637	0.0701	0.1053	0.0419	0.0634
Global boot.	0.1183	61	0.0670	0.0513	0.0686	0.0945	0.0549	0.0397
SATELLITE								
Classic	0.1845	130	0.1400	0.0445	0.0870	0.0560	0.0317	0.0244
Smoothing	0.1848(0.1)	136	0.1390	0.0458	0.0880	0.0557	0.0317	0.0240
Aggregation	0.1873(0.1)	145	0.1450	0.0423	0.0908	0.0568	0.0317	0.0251
Averaging	0.1894(1.0)	151	0.1405	0.0489	0.0915	0.0572	0.0321	0.0251
Local boot.	0.1873	173	0.1410	0.0463	0.0899	0.0573	0.0318	0.0255
Global boot.	0.1905	105	0.1550	0.0355	0.0855	0.0509	0.0346	0.0163
PENDIGITS								
Classic	0.1650	116	0.1029	0.0621	0.0929	0.0316	0.0142	0.0173
Smoothing	0.1605(0.1)	130	0.0986	0.0619	0.0926	0.0305	0.0135	0.0169
Aggregation	0.1523(0.4)	157	0.0915	0.0608	0.0825	0.0285	0.0140	0.0145
Averaging	0.1561(2.0)	147	0.0875	0.0686	0.0901	0.0293	0.0131	0.0162
Local boot.	0.1578	139	0.0963	0.0615	0.0875	0.0304	0.0141	0.0163
Global boot.	0.1716	102	0.1215	0.0501	0.0858	0.0302	0.0170	0.0132
DIG44								
Classic	0.2680	226	0.1328	0.1352	0.1616	0.0516	0.0213	0.0303
Smoothing	0.2685(0.1)	249	0.1232	0.1453	0.1642	0.0518	0.0209	0.0308
Aggregation	0.2689(0.1)	268	0.1343	0.1347	0.1615	0.0515	0.0215	0.0300
Averaging	0.2654(1.0)	269	0.1295	0.1359	0.1599	0.0511	0.0212	0.0299
Local boot.	0.2696	294	0.1348	0.1349	0.1594	0.0514	0.0220	0.0294
Global boot.	0.2556	183	0.1555	0.1001	0.1397	0.0439	0.0247	0.0192

Table C.11: Threshold variance reduction techniques, fixed attribute choice, limited complexity (see Section 6.3.3)

	Mean error	compl	bias_T	$\text{var}_T(\%)$	$\text{var}_{KW}(\%)$	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}(\%)$
GAUSSIAN (Nb. tests=2)								
Classic	0.1948	5	0.1655	0.0293	0.0744	0.1450	0.1192	0.0257
Smoothing	0.1804(0.5)	5	0.1648	0.0157	0.0491	0.1416	0.1222	0.0194
Aggregation	0.1745(0.6)	5	0.1653	0.0093	0.0376	0.1357	0.1228	0.0129
Averaging	0.1699(3.5)	5	0.1635	0.0064	0.0268	0.1373	0.1241	0.0132
Local boot.	0.1817	5	0.1645	0.0172	0.0549	0.1391	0.1209	0.0182
Global boot.	0.1852	5	0.1628	0.0224	0.0601	0.1400	0.1199	0.0200
WAVEFORM (Nb. tests=12)								
Classic	0.3022	23	0.2480	0.0542	0.1319	0.1454	0.1106	0.0348
Smoothing	0.3030(0.2)	24	0.2410	0.0620	0.1311	0.1400	0.1136	0.0264
Aggregation	0.2993(0.4)	25	0.2680	0.0313	0.1039	0.1359	0.1177	0.0182
Averaging	0.2960(1.0)	24	0.2500	0.0460	0.1215	0.1401	0.1126	0.0274
Local boot.	0.2929	25	0.2570	0.0359	0.1135	0.1368	0.1137	0.0231
Global boot.	0.2899	23	0.2440	0.0459	0.1106	0.1365	0.1145	0.0220
TWO-NORM (Nb. tests=11)								
Classic	0.2887	19	0.1746	0.1141	0.1554	0.1951	0.1393	0.0558
Smoothing	0.2844(0.4)	21	0.2376	0.0468	0.1297	0.1890	0.1524	0.0366
Aggregation	0.2861(0.3)	21	0.2446	0.0414	0.1253	0.1850	0.1548	0.0302
Averaging	0.2822(1.5)	21	0.2096	0.0726	0.1323	0.1863	0.1521	0.0342
Local boot.	0.2820	23	0.2191	0.0628	0.1330	0.1846	0.1475	0.0372
Global boot.	0.2772	19	0.2071	0.0701	0.1329	0.1842	0.1483	0.0360
OMIB (Nb. tests=14)								
Classic	0.1455	25	0.0765	0.0690	0.0843	0.1125	0.0663	0.0463
Smoothing	0.1437(0.4)	26	0.1045	0.0392	0.0727	0.1065	0.0770	0.0294
Aggregation	0.1339(0.5)	27	0.1080	0.0259	0.0558	0.0977	0.0770	0.0206
Averaging	0.1351(2.5)	25	0.1128	0.0224	0.0532	0.0980	0.0791	0.0188
Local boot.	0.1342	29	0.0940	0.0402	0.0669	0.1023	0.0717	0.0307
Global boot.	0.1359	25	0.0877	0.0481	0.0701	0.1028	0.0722	0.0305
SATELLITE (Nb. tests=13)								
Classic	0.2063	26	0.1795	0.0268	0.0758	0.0501	0.0411	0.0090
Smoothing	0.2073(0.1)	27	0.1815	0.0258	0.0730	0.0497	0.0417	0.0080
Aggregation	0.2083(0.1)	27	0.1895	0.0188	0.0737	0.0498	0.0417	0.0080
Averaging	0.2127(1.0)	27	0.2005	0.0122	0.0725	0.0514	0.0428	0.0086
Local boot.	0.2052	27	0.1955	0.0097	0.0716	0.0490	0.0417	0.0073
Global boot.	0.2106	26	0.1940	0.0166	0.0752	0.0496	0.0420	0.0076
PENDIGITS (Nb. tests=21)								
Classic	0.2309	42	0.1893	0.0417	0.0843	0.0400	0.0281	0.0119
Smoothing	0.2170(0.2)	43	0.1930	0.0241	0.0697	0.0372	0.0280	0.0092
Aggregation	0.2060(0.3)	43	0.1855	0.0204	0.0573	0.0351	0.0278	0.0072
Averaging	0.2173(2.0)	43	0.1904	0.0269	0.0651	0.0365	0.0282	0.0082
Local boot.	0.2199	43	0.1941	0.0258	0.0671	0.0380	0.0291	0.0090
Global boot.	0.2228	43	0.1998	0.0230	0.0681	0.0382	0.0294	0.0088
DIG44 (Nb. tests=25)								
Classic	0.3111	51	0.2350	0.0761	0.1354	0.0475	0.0349	0.0126
Smoothing	0.3061(0.5)	50	0.2610	0.0451	0.1015	0.0471	0.0400	0.0071
Aggregation	0.3083(0.1)	51	0.2487	0.0595	0.1203	0.0464	0.0368	0.0096
Averaging	0.3079(1.0)	51	0.2490	0.0589	0.1201	0.0459	0.0363	0.0096
Local boot.	0.2971	51	0.2512	0.0459	0.1107	0.0454	0.0368	0.0086
Global boot.	0.2894	51	0.2392	0.0502	0.1080	0.0450	0.0366	0.0084

Table C.12: Threshold variance reduction techniques, free attribute choice, fully grown trees (see Section 6.3.3)

	Mean error	compl	bias_T	$\text{var}_T(\%)$	$\text{var}_{KW}(\%)$	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}(\%)$
GAUSSIAN								
Classic	0.1736	30	0.1182	0.0554	0.0885	0.1736	0.0851	0.0885
Smoothing	0.1742(0.6)	71	0.1177	0.0564	0.0893	0.1742	0.0849	0.0893
Aggregation	0.1736(0.5)	50	0.1187	0.0549	0.0888	0.1718	0.0848	0.0869
Averaging	0.1713(1.5)	41	0.1165	0.0548	0.0859	0.1705	0.0854	0.0851
Local boot.	0.1723	41	0.1155	0.0568	0.0870	0.1720	0.0853	0.0867
Global boot.	0.1462	18	0.1163	0.0299	0.0582	0.1190	0.0854	0.0336
WAVEFORM								
Classic	0.2959	73	0.1600	0.1359	0.1727	0.1973	0.0822	0.1151
Smoothing	0.3003(0.1)	84	0.1600	0.1403	0.1749	0.2002	0.0836	0.1166
Aggregation	0.3050(0.1)	87	0.1610	0.1440	0.1778	0.2033	0.0848	0.1186
Averaging	0.2988(1.0)	83	0.1610	0.1378	0.1753	0.1992	0.0823	0.1169
Local boot.	0.2991	91	0.1620	0.1371	0.1756	0.1994	0.0823	0.1171
Global boot.	0.2938	65	0.1740	0.1198	0.1678	0.1575	0.0893	0.0682
TWO-NORM								
Classic	0.2161	51	0.0345	0.1816	0.1510	0.2161	0.0651	0.1510
Smoothing	0.2096(0.4)	73	0.0340	0.1756	0.1472	0.2096	0.0624	0.1472
Aggregation	0.2095(0.6)	80	0.0370	0.1724	0.1449	0.2095	0.0645	0.1449
Averaging	0.2101(3.0)	64	0.0385	0.1716	0.1462	0.2101	0.0639	0.1462
Local boot.	0.2187	64	0.0320	0.1867	0.1523	0.2187	0.0664	0.1523
Global boot.	0.2329	46	0.0380	0.1948	0.1599	0.1868	0.0835	0.1032
OMIB								
Classic	0.1213	75	0.0447	0.0765	0.0786	0.1213	0.0427	0.0786
Smoothing	0.1160(0.2)	98	0.0423	0.0737	0.0762	0.1160	0.0398	0.0762
Aggregation	0.1157(0.7)	139	0.0490	0.0667	0.0736	0.1157	0.0421	0.0736
Averaging	0.1129(1.5)	99	0.0408	0.0722	0.0736	0.1129	0.0393	0.0736
Local boot.	0.1258	109	0.0510	0.0748	0.0814	0.1258	0.0444	0.0814
Global boot.	0.1289	59	0.0705	0.0584	0.0772	0.1032	0.0585	0.0448
SATELLITE								
Classic	0.2119	120	0.1245	0.0874	0.1192	0.0706	0.0309	0.0397
Smoothing	0.2107(0.5)	155	0.1175	0.0932	0.1222	0.0702	0.0295	0.0407
Aggregation	0.2109(0.2)	132	0.1190	0.0919	0.1223	0.0703	0.0295	0.0408
Averaging	0.2099(3.0)	129	0.1140	0.0959	0.1230	0.0700	0.0290	0.0410
Local boot.	0.2116	125	0.1230	0.0886	0.1208	0.0705	0.0303	0.0403
Global boot.	0.2079	102	0.1420	0.0659	0.1092	0.0581	0.0332	0.0250
PENDIGITS								
Classic	0.1877	98	0.0843	0.1034	0.1186	0.0375	0.0138	0.0237
Smoothing	0.1807(0.2)	115	0.0778	0.1029	0.1152	0.0361	0.0131	0.0230
Aggregation	0.1770(0.2)	116	0.0729	0.1041	0.1137	0.0354	0.0127	0.0227
Averaging	0.1746(2.5)	113	0.0609	0.1137	0.1149	0.0349	0.0119	0.0230
Local boot.	0.1790	108	0.0758	0.1033	0.1140	0.0358	0.0130	0.0228
Global boot.	0.1951	90	0.0952	0.0999	0.1184	0.0351	0.0157	0.0193
DIG44								
Classic	0.3016	180	0.1123	0.1893	0.1953	0.0603	0.0213	0.0391
Smoothing	0.3006(0.1)	196	0.1098	0.1909	0.1957	0.0601	0.0210	0.0391
Aggregation	0.3077(0.2)	222	0.1153	0.1924	0.1992	0.0615	0.0217	0.0398
Averaging	0.2999(1.5)	203	0.1128	0.1871	0.1948	0.0600	0.0210	0.0390
Local boot.	0.3095	207	0.1153	0.1943	0.2005	0.0619	0.0218	0.0401
Global boot.	0.2997	158	0.1290	0.1707	0.1880	0.0511	0.0238	0.0272

Table C.13: Threshold variance reduction techniques, free attribute choice and pruned trees (see Section 6.3.3)

	Mean error	compl	bias_T	$\text{var}_T(\%)$	$\text{var}_{KW}(\%)$	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}(\%)$
GAUSSIAN								
Classic	0.1397	7	0.1207	0.0189	0.0494	0.1182	0.0861	0.0321
Smoothing	0.1366(0.7)	19	0.1263	0.0104	0.0396	0.1194	0.0886	0.0308
Aggregation	0.1379(0.3)	12	0.1250	0.0129	0.0437	0.1180	0.0876	0.0302
Averaging	0.1341(2.0)	12	0.1200	0.0141	0.0433	0.1175	0.0864	0.0311
Local boot.	0.1364	9	0.1210	0.0154	0.0434	0.1188	0.0888	0.0301
Global boot.	0.1375	7	0.1190	0.0185	0.0468	0.1159	0.0882	0.0277
WAVEFORM								
Classic	0.2890	38	0.1700	0.1190	0.1635	0.1660	0.0858	0.0802
Smoothing	0.2886(0.2)	44	0.1750	0.1136	0.1636	0.1640	0.0872	0.0768
Aggregation	0.2889(0.6)	50	0.1810	0.1079	0.1621	0.1611	0.0882	0.0729
Averaging	0.2889(1.0)	45	0.1580	0.1309	0.1660	0.1662	0.0853	0.0810
Local boot.	0.2883	42	0.1720	0.1163	0.1635	0.1616	0.0870	0.0746
Global boot.	0.2906	41	0.1710	0.1196	0.1635	0.1517	0.0919	0.0599
TWO-NORM								
Classic	0.2162	46	0.0340	0.1822	0.1510	0.2118	0.0658	0.1460
Smoothing	0.2090(0.4)	65	0.0330	0.1760	0.1470	0.2035	0.0636	0.1399
Aggregation	0.2064(0.6)	62	0.0360	0.1704	0.1430	0.1989	0.0654	0.1335
Averaging	0.2095(3.0)	58	0.0390	0.1705	0.1457	0.2050	0.0650	0.1400
Local boot.	0.2189	54	0.0325	0.1864	0.1522	0.2105	0.0678	0.1427
Global boot.	0.2334	37	0.0395	0.1939	0.1600	0.1878	0.0861	0.1017
OMIB								
Classic	0.1189	57	0.0485	0.0704	0.0763	0.1132	0.0441	0.0691
Smoothing	0.1146(0.1)	74	0.0450	0.0696	0.0741	0.1108	0.0415	0.0693
Aggregation	0.1089(0.7)	89	0.0558	0.0532	0.0662	0.1012	0.0443	0.0569
Averaging	0.1107(1.5)	80	0.0428	0.0680	0.0710	0.1061	0.0408	0.0654
Local boot.	0.1234	77	0.0553	0.0682	0.0781	0.1149	0.0473	0.0675
Global boot.	0.1291	44	0.0727	0.0564	0.0768	0.1033	0.0606	0.0426
SATELLITE								
Classic	0.2008	52	0.1450	0.0558	0.1012	0.0577	0.0334	0.0243
Smoothing	0.1978(0.1)	53	0.1420	0.0558	0.0989	0.0568	0.0331	0.0237
Aggregation	0.1933(0.5)	54	0.1395	0.0538	0.0972	0.0537	0.0322	0.0215
Averaging	0.1987(2.5)	56	0.1475	0.0512	0.1002	0.0561	0.0326	0.0236
Local boot.	0.1991	53	0.1455	0.0537	0.0990	0.0571	0.0337	0.0234
Global boot.	0.2036	54	0.1620	0.0416	0.0974	0.0548	0.0352	0.0196
PENDIGITS								
Classic	0.1870	86	0.0886	0.0984	0.1162	0.0363	0.0143	0.0220
Smoothing	0.1812(0.2)	97	0.0903	0.0908	0.1120	0.0347	0.0139	0.0207
Aggregation	0.1773(0.4)	114	0.0675	0.1098	0.1142	0.0346	0.0127	0.0219
Averaging	0.1746(3.0)	101	0.0652	0.1095	0.1136	0.0337	0.0124	0.0213
Local boot.	0.1793	91	0.0838	0.0955	0.1117	0.0345	0.0138	0.0207
Global boot.	0.1966	81	0.0978	0.0988	0.1171	0.0347	0.0161	0.0186
DIG44								
Classic	0.2947	107	0.1253	0.1694	0.1858	0.0523	0.0229	0.0293
Smoothing	0.2933(0.1)	123	0.1263	0.1670	0.1854	0.0525	0.0226	0.0299
Aggregation	0.2975(0.2)	132	0.1315	0.1659	0.1866	0.0525	0.0232	0.0293
Averaging	0.2919(1.5)	121	0.1245	0.1673	0.1837	0.0509	0.0227	0.0283
Local boot.	0.2974	113	0.1353	0.1621	0.1865	0.0512	0.0236	0.0276
Global boot.	0.2942	112	0.1333	0.1609	0.1811	0.0484	0.0248	0.0236

Table C.14: Threshold variance reduction techniques, free structure, small trees (see Section 6.3.3)

	Mean error	compl	bias$_T$	var$_T$(%)	var$_{KW}$(%)	$Err_{\hat{P}}$	bias$^2_{\hat{P}}$	var$_{\hat{P}}$(%)
GAUSSIAN								
Classic	0.1638	5	0.1318	0.0320	0.0639	0.1286	0.0928	0.0359
Smoothing	0.1588(0.1)	5	0.1285	0.0303	0.0582	0.1263	0.0930	0.0333
Aggregation	0.1565(0.1)	5	0.1293	0.0273	0.0557	0.1252	0.0928	0.0323
Averaging	0.1513(1.5)	5	0.1255	0.0258	0.0551	0.1234	0.0913	0.0321
Local boot.	0.1533	5	0.1318	0.0215	0.0494	0.1245	0.0955	0.0290
Global boot.	0.1536	5	0.1298	0.0238	0.0508	0.1247	0.0935	0.0312
WAVEFORM								
Classic	0.2962	24	0.1700	0.1262	0.1672	0.1550	0.0909	0.0641
Smoothing	0.3003(0.1)	24	0.1800	0.1203	0.1688	0.1522	0.0937	0.0585
Aggregation	0.3005(0.6)	24	0.1900	0.1105	0.1675	0.1456	0.0971	0.0484
Averaging	0.3005(1.0)	24	0.1670	0.1335	0.1705	0.1513	0.0932	0.0581
Local boot.	0.2944	24	0.1840	0.1104	0.1655	0.1495	0.0935	0.0559
Global boot.	0.2953	24	0.1810	0.1143	0.1630	0.1469	0.0962	0.0507
TWO-NORM								
Classic	0.2394	23	0.0395	0.1999	0.1640	0.2028	0.0835	0.1193
Smoothing	0.2395(0.1)	23	0.0385	0.2010	0.1649	0.1944	0.0853	0.1091
Aggregation	0.2403(0.5)	23	0.0435	0.1968	0.1630	0.1867	0.0899	0.0968
Averaging	0.2406(1.0)	23	0.0375	0.2031	0.1658	0.1951	0.0862	0.1089
Local boot.	0.2465	23	0.0370	0.2095	0.1681	0.1955	0.0895	0.1060
Global boot.	0.2515	23	0.0485	0.2030	0.1701	0.1869	0.0985	0.0885
OMIB								
Classic	0.1371	29	0.0635	0.0736	0.0852	0.1111	0.0565	0.0546
Smoothing	0.1355(0.1)	29	0.0653	0.0703	0.0821	0.1071	0.0591	0.0480
Aggregation	0.1295(0.5)	29	0.0747	0.0547	0.0728	0.0978	0.0626	0.0351
Averaging	0.1302(2.0)	29	0.0720	0.0582	0.0740	0.0993	0.0626	0.0367
Local boot.	0.1408	29	0.0798	0.0610	0.0827	0.1077	0.0662	0.0415
Global boot.	0.1385	28	0.0803	0.0582	0.0797	0.1049	0.0685	0.0364
SATELLITE								
Classic	0.2141	26	0.1635	0.0506	0.1028	0.0553	0.0380	0.0173
Smoothing	0.2192(0.5)	26	0.1700	0.0492	0.1029	0.0550	0.0389	0.0162
Aggregation	0.2127(0.7)	26	0.1610	0.0517	0.1001	0.0535	0.0377	0.0158
Averaging	0.2189(2.5)	26	0.1715	0.0474	0.0999	0.0546	0.0386	0.0160
Local boot.	0.2140	25	0.1745	0.0395	0.0996	0.0547	0.0386	0.0161
Global boot.	0.2159	26	0.1740	0.0420	0.1006	0.0543	0.0390	0.0153
PENDIGITS								
Classic	0.2219	43	0.1155	0.1064	0.1241	0.0380	0.0201	0.0179
Smoothing	0.2179(0.1)	43	0.1169	0.1010	0.1231	0.0367	0.0199	0.0168
Aggregation	0.2063(0.3)	43	0.1158	0.0905	0.1114	0.0345	0.0196	0.0149
Averaging	0.2129(2.5)	43	0.1138	0.0991	0.1198	0.0351	0.0194	0.0157
Local boot.	0.2132	43	0.1181	0.0952	0.1174	0.0362	0.0200	0.0162
Global boot.	0.2239	43	0.1266	0.0973	0.1229	0.0372	0.0213	0.0158
DIG44								
Classic	0.3262	51	0.1700	0.1562	0.1945	0.0506	0.0291	0.0214
Smoothing	0.3274(0.2)	51	0.1798	0.1476	0.1936	0.0494	0.0304	0.0190
Aggregation	0.3309(0.1)	51	0.1803	0.1506	0.1958	0.0504	0.0302	0.0201
Averaging	0.3251(1.5)	51	0.1783	0.1468	0.1889	0.0494	0.0306	0.0188
Local boot.	0.3277	51	0.1755	0.1521	0.1936	0.0498	0.0302	0.0196
Global boot.	0.3231	51	0.1775	0.1455	0.1885	0.0487	0.0308	0.0179

Table C.15: Attribute choice variance reduction, fully grown trees (see Section 6.3.4)

	Mean error	compl	bias_T	$\text{var}_T(\%)$	$\text{var}_{KW}(\%)$	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}(\%)$
GAUSSIAN								
KVALSETH	0.1736	30	0.1182	0.0554	0.0885	0.1736	0.0851	0.0885
LOCAL-BAGGING	0.1750	41	0.1215	0.0535	0.0895	0.1732	0.0854	0.0877
GLOBAL-BAGGING	0.1466	16	0.1165	0.0301	0.0577	0.1177	0.0862	0.0316
WAVEFORM								
KVALSETH	0.2959	73	0.1600	0.1359	0.1727	0.1973	0.0822	0.1151
LOCAL-BAGGING	0.3159	124	0.1680	0.1479	0.1865	0.2095	0.0862	0.1233
GLOBAL-BAGGING	0.3015	45	0.1830	0.1185	0.1678	0.1470	0.0961	0.0509
TWO-NORM								
KVALSETH	0.2161	51	0.0345	0.1816	0.1510	0.2161	0.0651	0.1510
LOCAL-BAGGING	0.2235	88	0.0450	0.1784	0.1537	0.2229	0.0699	0.1531
GLOBAL-BAGGING	0.2371	39	0.0330	0.2041	0.1636	0.1878	0.0930	0.0948
OMIB								
KVALSETH	0.1213	75	0.0447	0.0765	0.0786	0.1213	0.0427	0.0786
LOCAL-BAGGING	0.1362	137	0.0612	0.0749	0.0868	0.1355	0.0495	0.0860
GLOBAL-BAGGING	0.1434	35	0.0872	0.0562	0.0806	0.1091	0.0696	0.0395
SATELLITE								
KVALSETH	0.2119	120	0.1245	0.0874	0.1192	0.0706	0.0309	0.0397
LOCAL-BAGGING	0.2184	165	0.1275	0.0909	0.1236	0.0726	0.0316	0.0410
GLOBAL-BAGGING	0.2191	60	0.1735	0.0456	0.1019	0.0560	0.0377	0.0183
PENDIGITS								
KVALSETH	0.1877	98	0.0843	0.1034	0.1186	0.0375	0.0138	0.0237
LOCAL-BAGGING	0.1955	137	0.0872	0.1083	0.1230	0.0390	0.0145	0.0246
GLOBAL-BAGGING	0.2195	77	0.1086	0.1109	0.1305	0.0361	0.0184	0.0177
DIG44								
KVALSETH	0.3016	180	0.1123	0.1893	0.1953	0.0603	0.0213	0.0391
LOCAL-BAGGING	0.3149	257	0.1228	0.1921	0.2019	0.0629	0.0226	0.0404
GLOBAL-BAGGING	0.3243	132	0.1570	0.1673	0.1981	0.0507	0.0275	0.0232

C.3 Results of Chapter 7

Table C.16: Perturb and combine algorithms and variants (see Sections 7.2.1, 7.4.2, and 7.5)

	Mean error	compl	bias$_T$	var$_T$(%)	var$_{KW}$(%)	$Err_{\hat{P}}$	bias$^2_{\hat{P}}$	var$_{\hat{P}}$(%)
GAUSSIAN ($T = 25$, $\lambda = 0.52 \pm 0.38$)								
Full tree	0.1757	31	0.1182	0.0574	0.0909	0.1757	0.0848	0.0909
Bagging	0.1507	521	0.1180	0.0327	0.0645	0.1117	0.0852	0.0265
Random single tree	0.1845	67	0.1243	0.0603	0.0972	0.1822	0.0872	0.0949
Random averaging	0.1463	1651	0.1187	0.0275	0.0607	0.1091	0.0841	0.0250
Boosting	0.1539	75	0.1170	0.0369	0.0673	0.1196	0.0927	0.0269
WAVEFORM ($T = 25$, $\lambda = 0.73 \pm 0.21$)								
Full tree	0.2987	76	0.1530	0.1457	0.1752	0.1991	0.0823	0.1168
Bagging	0.2030	1349	0.1540	0.0490	0.0938	0.0965	0.0824	0.0141
Random single tree	0.3379	263	0.1530	0.1849	0.2045	0.2253	0.0889	0.1364
Random averaging	0.1845	6707	0.1410	0.0435	0.0824	0.0957	0.0869	0.0088
Boosting	0.1960	832	0.1430	0.0530	0.0947	0.0984	0.0886	0.0098
TWO-NORM ($T = 25$, $\lambda = 1.03 \pm 0.28$)								
Full tree	0.2159	51	0.0350	0.1809	0.1507	0.2159	0.0652	0.1507
Bagging	0.0857	912	0.0260	0.0597	0.0587	0.0827	0.0670	0.0155
Random single tree	0.2077	173	0.0365	0.1712	0.1465	0.2077	0.0612	0.1465
Random averaging	0.0566	4341	0.0275	0.0291	0.0347	0.0693	0.0600	0.0091
Boosting	0.0634	548	0.0250	0.0384	0.0415	0.0828	0.0734	0.0094
OMIB ($T = 25$, $\lambda = 0.69 \pm 0.23$)								
Full tree	0.1213	75	0.0420	0.0793	0.0789	0.1213	0.0424	0.0789
Bagging	0.0842	1386	0.0475	0.0367	0.0483	0.0611	0.0466	0.0144
Random single tree	0.1299	221	0.0415	0.0884	0.0854	0.1299	0.0445	0.0853
Random averaging	0.0676	5450	0.0370	0.0306	0.0381	0.0525	0.0434	0.0090
Boosting	0.0696	926	0.0290	0.0406	0.0437	0.0627	0.0547	0.0079
SATELLITE ($T = 25$, $\lambda = 0.24 \pm 0.07$)								
Full tree	0.2078	117	0.1240	0.0838	0.1181	0.0693	0.0299	0.0394
Bagging	0.1517	2048	0.1280	0.0237	0.0595	0.0366	0.0315	0.0051
Random single tree	0.2490	374	0.1150	0.1340	0.1520	0.0830	0.0323	0.0507
Random averaging	0.1361	9371	0.1155	0.0206	0.0462	0.0348	0.0316	0.0032
Boosting	0.1388	1483	0.1145	0.0243	0.0609	0.0372	0.0327	0.0045
PENDIGITS ($T = 25$, $\lambda = 0.40 \pm 0.10$)								
Full tree	0.1903	97	0.0812	0.1091	0.1196	0.0381	0.0141	0.0239
Bagging	0.1260	1931	0.0895	0.0366	0.0613	0.0195	0.0155	0.0040
Random single tree	0.2914	438	0.0743	0.2171	0.2023	0.0583	0.0178	0.0405
Random averaging	0.0912	10828	0.0675	0.0237	0.0430	0.0191	0.0166	0.0025
Boosting	0.0900	1472	0.0600	0.0300	0.0448	0.0179	0.0151	0.0027
DIG44 ($T = 25$, $\lambda = 0.75 \pm 0.19$)								
Full tree	0.3030	181	0.1230	0.1800	0.1951	0.0606	0.0216	0.0390
Bagging	0.1802	3269	0.1150	0.0652	0.0965	0.0284	0.0232	0.0053
Random single tree	0.4702	643	0.1028	0.3675	0.3105	0.0940	0.0319	0.0621
Random averaging	0.1468	16085	0.0845	0.0623	0.0833	0.0339	0.0306	0.0033
Boosting	0.1405	2119	0.0817	0.0587	0.0776	0.0288	0.0251	0.0037

C.4 Results of Chapter 8

Table C.17: Dual P&C and variants (see Sections 8.3, 8.4.1, and 8.4.2)

	Mean error	compl	bias_T	$\text{var}_T(\%)$	$\text{var}_{KW}(\%)$	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}(\%)$
GAUSSIAN ($T = 25$, $\lambda = 0.52 \pm 0.38$)								
Full tree	0.1757	31	0.1182	0.0574	0.0909	0.1757	0.0848	0.0909
Dual P&C	0.1371	31	0.1165	0.0206	0.0432	0.1165	0.0994	0.0171
Struct. fixed bagging	0.1640	537	0.1182	0.0458	0.0765	0.1204	0.0868	0.0336
Bagging + dual P&C	0.1303	103	0.1160	0.0143	0.0308	0.1101	0.0995	0.0105
WAVEFORM ($T = 25$, $\lambda = 0.73 \pm 0.21$)								
Full tree	0.2987	76	0.1530	0.1457	0.1752	0.1991	0.0823	0.1168
Dual P&C	0.2451	76	0.1540	0.0911	0.1380	0.1207	0.1033	0.0173
Struct. fixed bagging	0.2666	1562	0.1540	0.1126	0.1494	0.1264	0.0824	0.0440
Bagging + dual P&C	0.1721	1352	0.1370	0.0351	0.0714	0.1155	0.1116	0.0039
TWO-NORM ($T = 25$, $\lambda = 1.03 \pm 0.28$)								
Full tree	0.2159	51	0.0350	0.1809	0.1507	0.2159	0.0652	0.1507
Dual P&C	0.1372	51	0.0345	0.1027	0.0970	0.1283	0.1084	0.0199
Struct. fixed bagging	0.1747	2085	0.0280	0.1467	0.1248	0.1205	0.0655	0.0550
Bagging + dual P&C	0.0492	925	0.0245	0.0247	0.0301	0.1213	0.1170	0.0044
OMIB ($T = 25$, $\lambda = 0.69 \pm 0.23$)								
Full tree	0.1213	75	0.0420	0.0793	0.0789	0.1213	0.0424	0.0789
Dual P&C	0.0752	75	0.0268	0.0484	0.0491	0.0810	0.0706	0.0104
Struct. fixed bagging	0.0881	2861	0.0448	0.0433	0.0534	0.0629	0.0448	0.0180
Bagging + dual P&C	0.0492	1404	0.0278	0.0215	0.0283	0.0846	0.0811	0.0035
SATELLITE ($T = 25$, $\lambda = 0.24 \pm 0.07$)								
Full tree	0.2078	117	0.1240	0.0838	0.1181	0.0693	0.0299	0.0394
Dual P&C	0.1913	117	0.1290	0.0623	0.1010	0.0468	0.0346	0.0121
Struct. fixed bagging	0.1816	2391	0.1280	0.0536	0.0922	0.0441	0.0305	0.0136
Bagging + dual P&C	0.1482	2031	0.1300	0.0182	0.0516	0.0390	0.0356	0.0035
PENDIGITS ($T = 25$, $\lambda = 0.40 \pm 0.10$)								
Full tree	0.1903	97	0.0812	0.1091	0.1196	0.0381	0.0141	0.0239
Dual P&C	0.1715	97	0.0789	0.0926	0.1046	0.0304	0.0229	0.0075
Struct. fixed bagging	0.1687	2044	0.0812	0.0876	0.1038	0.0254	0.0144	0.0110
Bagging + dual P&C	0.1124	1904	0.0843	0.0281	0.0494	0.0257	0.0234	0.0023
DIG44 ($T = 25$, $\lambda = 0.75 \pm 0.19$)								
Full tree	0.3030	181	0.1230	0.1800	0.1951	0.0606	0.0216	0.0390
Dual P&C	0.2511	181	0.1117	0.1394	0.1591	0.0475	0.0411	0.0065
Struct. fixed bagging	0.2592	3696	0.1308	0.1285	0.1589	0.0365	0.0227	0.0139
Bagging + dual P&C	0.1433	3266	0.1030	0.0403	0.0658	0.0426	0.0409	0.0017

Appendix D

Pattern detection algorithm

In this appendix, we describe the algorithm used to match a complex pattern with a time-series. A complex pattern is composed of a sequence of subsignals of fixed shape. The matching consists of determining the optimal positions of the subsignals in such way that they do not overlap and that the Euclidean distance between the complex pattern and the original time-series is minimized. This is essentially a combinatorial problem, but the algorithm described below is efficient.

D.1 Problem statement

In this section, we view a (complex) pattern as a compact representation of a set of time-series and the pattern matching problem as the computation of the distance between a particular time-series and this set. Although these ideas could be applied to any kind of signals, we particularize the presentation to the case of discrete time, finite duration, and real-valued signals (i.e. elements of $I\!R^N$).

We define a complex pattern $\mathcal{P}$ as an ordered sequence of K elementary patterns

$$\mathcal{P} \triangleq (p_1(.), p_2(.), \ldots, p_K(.)),$$

where each elementary pattern $p_k(.)$ $(k = 1, \ldots, K)$ is defined as a time-series of P_k values, $(p_{k,0}, \ldots, p_{k,P_k-1})$. Such a complex pattern is essentially a compact representation of a set of candidate signals. More precisely, given a total length N such that

$$N \geq \sum_{k=1}^{K} P_k,$$

we identify the complex pattern with the following set of signals[1]

$$S_{\mathcal{P}}^N \triangleq \left\{ s(\cdot) \in I\!R^N | s(\cdot) = \left([*]^{l_0} p_1(\cdot) [*]^{l_1} p_2(\cdot) \cdots [*]^{l_{K-1}} p_K(\cdot) [*]^{l_K} \right) \right\}, \tag{D.1}$$

where $[*]^{l_k}$ denotes a (dummy) sequence of length l_k of arbitrary real numbers ($l_k \geq 0$). Since the total length of any element of $S_{\mathcal{P}}^N$ is equal to N we have the following additional constraint among the lengths of the dummy signals $[*]^{l_k}$:

$$\sum_{k=0}^{K} l_k = N - P_{\mathcal{P}}, \tag{D.2}$$

where $P_{\mathcal{P}} = \sum_{k=1}^{K} P_k$ denotes the total length of the elementary patterns defining $\mathcal{P}$.

[1]In what follows, we will identify elements of $I\!R^N$, sequences of real numbers of length N, and temporal signals defined on the bounded integer time axis $0, \ldots, N-1$; moreover, we will (non-orthodoxly) mix sequence, vector, and signal notation where we find it appropriate.

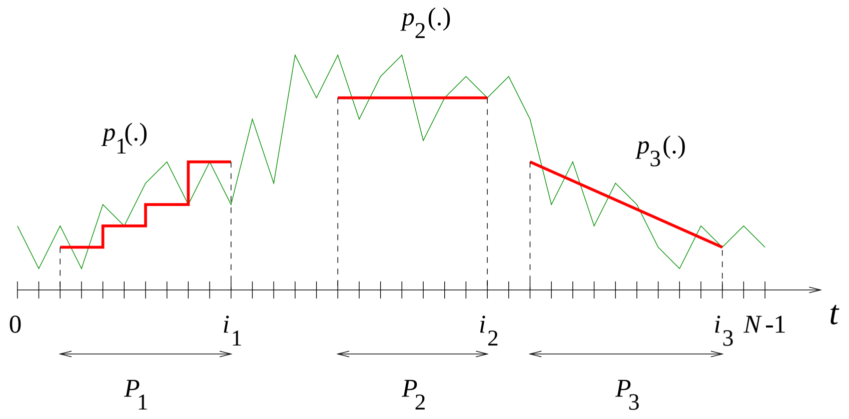


Figure D.1: One possible configuration of three patterns in a time series

Given two time series $a(\cdot) = (a_0, \ldots, a_{N-1})$ and $s(\cdot) = (s_0, \ldots, s_{N-1})$ defined on N, their Euclidean distance is defined by

$$d(a(\cdot), s(\cdot)) \triangleq \sum_{i=0}^{N-1} (a(i) - s(i))^2. \tag{D.3}$$

Given a pattern $\mathcal{P}$ and a time series $a(\cdot) = (a_0, \ldots, a_{N-1})$, the Euclidean distance between these two objects is thus defined by

$$D(a(\cdot), \mathcal{P}) \triangleq \inf_{s(\cdot) \in S_{\mathcal{P}}^N} \{d(a(\cdot), s(\cdot))\}. \tag{D.4}$$

The objective of the pattern matching algorithm described in the next section is essentially to compute this distance efficiently.

Notice that, once the lengths l_k of the dummy portions in (D.1) are fixed, the minimization of the Euclidean distance is obvious. Indeed, for fixed values of l_k the element of $S_{\mathcal{P}}^N$ which minimizes (D.4) is simply obtained by setting the values in each dummy portion of $s(\cdot)$ to the values of $a(\cdot)$ in the same position. Consequently, the lower bound of (D.4) can be determined by searching the optimal lengths l_k of the $k+1$ dummy signals under the constraints

$$\sum_{k=0}^{K} l_k = N - P_{\mathcal{P}} \quad \text{and} \quad l_k \geq 0. \tag{D.5}$$

This is essentially a combinatorial problem. Notice that the number of combinations of $k+1$ integer values satisfying the constraints (D.5) is lower bounded by $\mathcal{C}_{N-P_{\mathcal{P}}}^k$ and upper bounded by $(N - P_{\mathcal{P}})^k$. In many practical situations (long-time series, large number of short patterns), this number is far too high to allow for a complete enumeration. The next section describes an iterative algorithm to solve efficiently this problem.

D.2 Optimal matching algorithm

In order to formulate our algorithm, let us consider a complex pattern $\mathcal{P}$ as defined in the preceding section, and define, for a particular signal of $s(\cdot) \in S_{\mathcal{P}}^N$, by i_k the end-position of the k-th elementary pattern in $s(\cdot)$. This is illustrated in Figure D.1 in the case of a complex pattern composed of three elementary patterns. Clearly, the above considered set of sequences $(l_0, l_1, \ldots, l_K)$ characterizing the signals $S_{\mathcal{P}}^N$ is in one-to-one relationship with the sequences of end-positions $(i_1, \ldots, i_K)$ satisfying the following constraints:

$$i_1 \geq P_1 - 1, \tag{D.6}$$
$$i_k \geq i_{k-1} + P_k, \forall k = 2, \ldots, K, \tag{D.7}$$
$$i_K < N. \tag{D.8}$$

Thus, in terms of the i_k values, the determination $D(a(\cdot), \mathcal{P})$ is equivalent to computing the following quantity:

$$\min_{i_1,\ldots,i_K | C_{\mathcal{P}}(i_1,\ldots,i_K;N)} \sum_{k=1}^{K} d(p_k(.), a(.), i_k) \tag{D.9}$$

where $c_P(i_1, i_2, \ldots, i_K; N)$ corresponds to conditions (D.7), and the distance $d(p_k(.), a(.), i)$ (for $k = 1, \ldots, K$ and $i = P_k - 1, \ldots, N-1$) is given by:

$$d(p_k(.), a(.), i) = \sum_{j=i-P_k+1}^{i} (p_{k,j-i+P_k+1} - a_j)^2. \tag{D.10}$$

Notice that in the distance computation (D.9) the parts of $s(\cdot)$ and $a(\cdot)$ corresponding to the dummy portions have been dropped since they can match perfectly for each choice of locations and thus lead to a zero contribution in the minimum distance.

D.2.1 Backwards recursion

The algorithmic solution of the above problem is based on a recursion relation holding for the function $g(i,k)$ defined by

$$g(i,k) = \min_{i_1,\ldots,i_k | C_{\mathcal{P}}(i_1,\ldots,i_k;i+1)} \sum_{j=1}^{k} d(p_j(.), a(.), i_j), \tag{D.11}$$

for all values of $k = 1, \ldots, K$ and $i = \sum_{j=1}^{k} P_j - 1, \ldots, N-1$. Notice that $g(i,k)$ is by definition the minimum distance between the time-series $a(\cdot)$ truncated to the first $i+1$ time instants and the complex pattern truncated to the first k elementary patterns. Thus, obviously,

$$D(a(\cdot), \mathcal{P}) = g(N-1, K). \tag{D.12}$$

Notice also that the values $g(i,k)$ are by definition infinite for those values of i, k for which there is no admissible way to match the sub-signal and the sub-pattern, namely when $i+1 < \sum_{j=1}^{k} P_j$.

The function $g(\cdot,\cdot)$ satisfies the following recursive relationship

$$g(i,k) = \min\{d(p_k(.), a(.), i) + g(i - P_k, k-1), g(i-1, k)\}. \tag{D.13}$$

Indeed, to optimally match the first k patterns in the first $i+1$ values of $a(\cdot)$,

- either the k^{th} elementary pattern ends at position i: in this case the positions of the $k-1$ first patterns must be optimal over the $i - P_k + 1$ first instants, and this indeed corresponds to the value $g(i - P_k, k-1)$;

- or the k^{th} elementary pattern ends at a strictly lower position in the time-series than i: in this case the minimum distance is provided by $g(i-1, k)$.

D.2.2 Forward sequential algorithm

This recursive formulation of $g(i,k)$ is the basis for the forward sequential algorithm of Table D.1, which properly initializes the table representing the values of $g(i,k)$ and, for each value of k, computes only those values of $g(i,k)$ required to determine $g(N-1,K)$. Since it is also interesting to find the values of i_k, $k = 1, \ldots, K$, which realize the minimum in (D.9), the algorithm stores this information in another table denoted $p(i,k)$. This latter table is then used a posteriori to retrieve the positions i_k which realize the minimum, according to the algorithm of Table D.2.

Table D.1: Forward sequential pattern matching algorithm

```
Set $\alpha$=$-1$
Set $\beta$=$N - P_{\mathcal{P}} - 1$
For $i$=$\alpha$ to $\beta$
    Set $g(i, 0)$=$0$
End for
For $k$=$1$ to $K$
    Set $\alpha$=$\alpha + P_k$                 ; The smallest candidate value of $i_k$.
    Set $\beta$=$\beta + P_k$                   ; The largest candidate value of $i_k$.
    Set $g(\alpha - 1, k) = +\infty$
    For $i$=$\alpha$ to $\beta$
        Set $d$=$d(p_k(.), a(.), i)$
        If $d + g(i - P_k, k - 1) < g(i - 1, k)$ then
            Set $p(i, k)$=$i$
            Set $g(i, k)$=$d + g(i - P_k, k - 1)$
        Else
            Set $p(i, k)$=$p(i - 1, k)$
            Set $g(i, k)$=$g(i - 1, k)$
    End for
End for
Return $g(N - 1, K)$
```

Table D.2: Algorithm which computes the positions corresponding to the minimal distance

```
Set $i_K = p(N - 1, K)$
For $k$=$K - 1$ to $1$
    Set $i_k = p(i_{k+1} - P_{k+1}, k)$
```

D.2.3 Computational complexity

The algorithm computes $K(N+1-P_{\mathcal{P}})$ values of $g(i,k)$ and requires P_k operations at each step k to compute the distances $d(p_k, a(\cdot), i)$ for each one of the $N+1-P_{\mathcal{P}}$ candidate values of i. Hence, the number of operations of this algorithm is in general $O((N-P_{\mathcal{P}})P_{\mathcal{P}})$.

In the particular case where the elementary patterns $p_k(.)$ are constant over the P_k time points, the distances $d(p_k, a(\cdot), i)$ for growing values of i may be computed in constant time using the following update formula:

$$d(p(.), a(.), i) = d(p(.), a(.), i-1) + (p - a_i)^2 - (p - a_{i-P+1})^2, i = P, \dots, N-1$$

with

$$d(p(.), a(.), P-1) = \sum_{j=0}^{P-1} (p - a_j)^2.$$

Hence, in this particular case, the computation of the distance for all values of i requires order N operations. Following the same idea, when the elementary pattern is represented by a piecewise constant signal in s segments, these distances may be computed in $O(N.s)$.

Finally, when the complex patterns are extracted from piecewise constant signals represented with S segments (and hence contain at most S elementary patterns, which all together contain at most S segments), the computation of the minimal distance using to the algorithm of Table D.1 requires at most $O(N.S)$ operations, whatever the number of elementary patterns chosen and whatever the lengths P_k of these patterns.

Bibliography

[AB01] Y. Amit and G. Blanchard. Multiple randomized classifiers: Mrcl. Technical report, Dept. of Statistics, University of Chicago, 2001.

[AD97] Y. Amit and Geman D. Shape quantization and recognition with randomized trees. *Neural computation*, 9:1545–1588, 1997.

[AFS93] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Proceedings of the fourth International Conference on Foundations of Data Organization and Algorithms*, pages 69–84, Chicago, October 1993. Springer-Verlag.

[Alp93] E. Alpaydin. Multiple networks for function learning. In *Proceedings of theInternational Conference on Neural networks*, pages 9–14, California, USA, 1993.

[AM99] R. J. Alcock and Y. Manolopoulos. Time-series similarity queries employing a feature-based approach. In *Proc. of the 7th Hellenic Conference on Informatics*, Ioannina, Greece, 1999.

[AMGD00] T. Artières, J-M. Marchand, P. Gallinari, and B. Dorizzi. Stroke level modelling of on line handwriting through multi-modal segmental models. In *International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, 2000.

[AP95] K.M. Ali and M.J. Pazzani. On the link between error correlation and error reduction in decision tree ensembles. Technical report, Department of Information and Computer Science, University of California, Irvine, 1995.

[AP96] K.M. Ali and M.J. Pazzani. Error reduction through learning multiple descriptions. *Machine Learning*, 24(3):173–206, 1996.

[AR00] C.J. Alonso Gonzalez and J. Rodriguez Diez. Time series classification by boosting interval based literals. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, 11:2–11, 2000.

[Bar94] A.R. Barron. Approximation and estimation bounds for artificial neural networks. *Machine learning*, 14:115–133, 1994.

[BAS$^+$98] M.C. Burl, L. Asker, P. Smyth, U.M. Fayyad, P Perona, L. Crumpler, and J. Aubele. Learning to recognize volcanoes on venus. *Machine Learning*, 30(2-3):165–194, 1998.

[Bay99] S. D. Bay. The UCI KDD archive, 1999. http://kdd.ics.uci.edu.

[BB98] P. Baldi and S. Brunak. *Bioinformatics, the machine learning approach.* MIT Press, 1998.

[BC96] D. J. Berndt and J. Clifford. Finding patterns in time series: A dynamic programming approach. In *Advances in Knowledge discovery and data mining.* AAAI Press/MIT Press, 1996.

[Ben99] Y. Bengio. Markovian models for sequential data. *Neural computing surveys*, 2:129–162, 1999.

[BFOS84] L. Breiman, J.H. Friedman, R.A. Olsen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth International (California), 1984.

[Bis95] C.M. Bishop. *Neural Networks for Pattern Recognition*. Oxford: Oxford University Press, 1995.

[BJ76] G.E.P. Box and F.M. Jenkins. *Time series analysis: forecasting and control*. Holden-Day, 2nd edition, 1976.

[BK99] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms : Bagging, boosting, and variants. *Machine Learning*, 36:105–139, 1999.

[BM98] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[Bre96a] L. Breiman. Arcing classifiers. Technical report, University of California, Department of Statistics, 1996.

[Bre96b] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[Bre98] L. Breiman. Arcing classifiers. *Annals of statistics*, 26:801–849, 1998.

[Bre99] L. Breiman. Prediction games and arcing algorithms. *Neural computations*, 11(7):1493–1517, 1999.

[Bre00] L. Breiman. Randomizing outputs to increase prediction accuracy. *Machine Learning*, 40(3):229–242, September 2000.

[Bre01] L. Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[Bun92] W. Buntine. Learning classification trees. *Statistics and Computing*, 2:63–73, 1992.

[Bur98] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Knowledge Discovery and Data Mining*, 2(2):121–167, 1998.

[BW99] X. Boyen and L. Wehenkel. Automatic induction of fuzzy decision trees and its application to power system security assessment. *Fuzzy sets and systems*, 102(1):3–19, February 1999.

[CC87] C. Carter and J. Catlett. Assessing credit card applications using machine learning. *IEEE Expert*, Fall:71–79, 1987.

[CC00] P. Cunningham and J. Carney. Diversity versus quality in classification ensembles based on feature selection. In *Proc. of the 11th European Conference on Machine Learning (ECML-2000), Barcelona*, pages 109–116, May 2000.

[CDLS99] R.G. Cowell, A.P. Dawid, S.L. Lauritzen, and D.J. Spiegelhalter. *Probabilistic networks and expert systems*. Spinger-Verlag, Berlin-Heidelberg-New York, 1999.

[CM98] V. Cherkassky and F. Mulier. *Learning from data: Concepts, Theory, and Methods*. Wiley-Interscience, 1998.

[Coh96] W.W. Cohen. Learning to classify English text with ILP methods. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 124–143. IOS Press, 1996.

[CT91] M.T. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

[DC96] H. Drucker and C. Cortes. Boosting decision trees. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 479–485. The MIT Press, 1996.

[DH73] R.O. Duda and P.E. Hart. *Pattern classification and scene analysis.* Wiley, New York, 1973.

[DHS00] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification.* John Wiley & Sons, 2nd edition, 2000.

[Die00a] T.G. Dietterich. The divide-and-conquer manifesto. In *Algorithmic Learning Theory*, pages 13–26, 2000.

[Die00b] T.G. Dietterich. Ensemble methods in machine learning. In *Proc. of the first International Workshop on Multiple Classifier Systems*, 2000.

[Die00c] T.G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, August 2000.

[DK95] T.G. Dietterich and E.B. Kong. Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Department of Computer Science, Oregon State University, 1995.

[DKN01] J. Dahmen, D. Keysers, , and Hermann Ney. Combined classification of handwritten digits using the "virtual test sample method". In J. Kittler and F. Roli, editors, *Proceedings of the second international workshop on Multiple Classifier Systems, MCS 2001, Cambrige, UK*, pages 109–118, July 2001.

[DKS95] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous attributes. In *Proc. of the Twelth International Conference on Machine Learning (ICML-95)*, pages 194–202. Morgan Kaufmann, 1995.

[DLM$^+$98] G. Das, K Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. In *Proc. of Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, 1998.

[DM91] R.L. De Mantaras. A distance-based attributes selection measures for decision tree induction. *Machine Learning*, 6:81–92, 1991.

[Dom97a] P. Domingos. Knowledge acquisition from examples via multiple models. In *Proc. of the 14th Int. Conf. on Machine Learning*, pages 148–156, Nashville, 1997.

[Dom97b] P. Domingos. Why does bagging work? a bayesian account and its implications. In D. Heckerman, H. Mannila, D. Pregibon, and R. Uthurusamy, editors, *Proceedings of the third international conference on Knowledge Discovery and Data Mining*, pages 155–158. AAAI Press, 1997.

[Dom00a] P. Domingos. A unified bias-variance decomposition and its applications. In *Proc. 17th International Conf. on Machine Learning*, pages 231–238. Morgan Kaufmann, San Francisco, CA, 2000.

[Dom00b] P. Domingos. A unified bias-variance decomposition for zero-one and squared loss. In *AAAI/IAAI*, pages 564–569, 2000.

[DP97] P. Domingos and M. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29:103–130, 1997.

[ET93] B. Efron and R.J. Tibshirani. *An introduction to the boostrap.* Chapman & Hall, 1993.

[FGG97] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian networks classifiers. *Machine learning*, 29:131–163, 1997.

[FH00a] M.S. Fairhurst and M.S. Hoque. Analysis of a moving window classifier for off-line character recognition. In *Proceedings of Int. workshop on frontiers in handwriting recognition*, pages 595–600, Amsterdam, 2000.

[FH00b] J.H. Friedman and P. Hall. On bagging and nonlinear estimation. Technical report, Statistics Department, Standford University, January 2000.

[FHT00] J.H. Friedman, T. Hastie, and R.J. Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 38(2):337–374, April 2000.

[FPSS96] U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: an overview. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in knowledge discovery and data mining.* AAAI Press, Menlon Park, Calif., 1996.

[Fri77] J.H. Friedman. A recursive partitioning decision rule for nonparametric classifier. *IEEE Transactions on Computers*, C-26:404–408, 1977.

[Fri91] J.H. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19(1), 1991.

[Fri96] J.H. Friedman. Local learning based on recursive covering. Technical report, Department of Statistics, Standford University, August 1996.

[Fri97] J.H. Friedman. On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1:55–77, 1997.

[FS95] Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the second European Conference on Computational Learning Theory*, pages 23–27, 1995.

[FS96] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 148–156, 1996.

[FW99] E. Frank and I. H. Witten. Making better use of global discretization. In *Proc. of the Sixteenth International Conference on Machine Learning (ICML-99)*, pages 115–123. Morgan Kaufmann, 1999.

[Gal98] P. Gallinari. Predictive models for sequence modelling, application to speech and character recognition. In C. Lee Giles and M. Gori, editors, *Adaptive processing of sequences and data structures.* Springer, 1998.

[GBD92] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemna. *Neural computation*, 4:1–58, 1992.

[Geu00] P. Geurts. Some enhancements of decision tree bagging. In *Proc. of the 4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD-2000)*, pages 136–147, Lyon, France, September 2000.

[Geu01a] P. Geurts. Dual perturb and combine algorithm. In *Proc. of the Eighth International Workshop on Artificial Intelligence and Statistics*, pages 196–201, Key-West, Florida, January 2001.

[Geu01b] P. Geurts. Pattern extraction for time-series classification. In *Proc. of the 5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD-2000)*, Freiburg, September 2001.

[Geu01c] P. Geurts. towards soft tree induction. *Engineering i Intelligent systems*, 2001.

[GS00] X. Ge and P. Smyth. Deformable markov model templates for time-series pattern matching. In *Proc. of the 6th Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 81–90, Boston, MA, August 2000.

[GW98] P. Geurts and L. Wehenkel. Early prediction of electric power system blackouts by temporal machine learning. In *Proc. of ICML-AAAI'98 Workshop on "AI Approaches to Times-series Analysis"*, Madison (Wisconsin), 1998.

[GW00] P. Geurts and L. Wehenkel. Investigation and reduction of discretization variance in decision tree induction. In *Proc. of the 11th European Conference on Machine Learning (ECML-2000), Barcelona*, pages 162–170, May 2000.

[Han00] J.V. Hansen. *Combining predictors: Meta machine learning methods and bias/variance & ambiguity decompositions.* PhD thesis, Dept. of computer science, University of Aarhus, 2000.

[Has97] S. Hashem. Optimal linear combinations of neural networks. *Neural Networks*, 10(4):599–614, August 1997.

[Has00] E. Haselsteiner. *Neural based methods for time series classification.* PhD thesis, Technischen Universitat Graz, 2000.

[Hes98] T. Heskes. Bias/variance decompositions for likelihood-based estimators. *Neural Computation*, 10(6):1425–1433, 1998.

[HK92] L. Holmström and P. Koistinen. Using additive noise in back-propagation training. *IEEE Transaction on Neural Networks*, 3:24–38, 1992.

[Ho95] T.K. Ho. Random decision forests. In *Proc. of the 3rd International Conference on Document Analysis and Recognition*, pages 278–282, Montreal, Canada, August 1995.

[Ho98] T.K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[HR76] L. Hyafil and R.L. Rivest. Contructing optimal binary decision tree is NP-complete. *Information Processing Letters*, 5:15–17, 1976.

[HTF01] T. Hastie, R.J. Tibshirani, and J.H. Friedman. *The elements of statistical learning: data mining, inference and prediction.* Springer, 2001.

[Hun62] E.B. Hunt. *Concept Learning.* New York: Wiley, 1962.

[IL92] W. Iba and P. Langley. Induction of one-level decision trees. In *Proceedings of the ninth International Conference on Machine Learning*, pages 233–240. Morgan Kaufman, 1992.

[JH96] G. James and T. Hastie. Generalizations of the bias/variance decomposition for the prediction error. Technical report, Dept. of Statistics, Standford University, February 1996.

[JJNH91] R. Jacobs, M.I. Jordan, S. Nowlan, and G. Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[Jor94] M.I. Jordan. A statistical approach to decision tree modeling. In *Proc. of the Seventh Annual ACM Conference on Computational Learning Theory. New York.* ACM Press, 1994.

[Kad98] M.W. Kadous. A general architecture for supervised classification of multivariate time series. Technical report, Unversity of New South Wales, Departement of Artificial Intelligence, School of Computer Science and Engineering, September 1998.

[Kad99] M.W. Kadous. Learning comprehensible descriptions of multivariate time series. In *Proceedings of the Sixteenth International Conference on Machine Learning, ICML'99*, pages 454–463, Bled, Slovenia, 1999.

[KC90] S.W. Kwok and C. Carter. Multiple decision trees. In R.D. Schachter, T.S. Levitt, L.N. Kanal, and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence 4*, pages 327–335. Elsevier Science, 1990.

[KCC01] M. Kayaalp, G. Cooper, and G. Clermont. Predicting with variables constructed from univariate temporal sequences. In *Proceedings of AISTATS2001, int. workshop on AI and Statistics*, Key West, Florida, January 2001.

[KCHP01] E. Keogh, S. Chu, D. Hart, and M.J. Pazzani. An online algorithm for segmenting time series. In *Proceedings of IEEE International Conference on Data Mining*, pages 289–296, 2001.

[KJ97] R. Kohavi and G. John. Wrappers for feature subset selection. *Artificial Intelligence journal, special issue on relevance*, 97(1-2):273–324, 1997.

[KK97] R. Kohavi and C. Kunz. Option decision trees with majority votes. In Doug Fisher, editor, *Machine Learning: Proceedings of the Fourteenth International Conference.* Morgan Kaufmann Publishers, Inc., 1997.

[KP99] E. Keogh and M.J. Pazzani. An indexing scheme for fast similarity search in large time series databases. In *Proceedings of the eleventh International Conference on Scientific and Statistical Database Management*, pages 56–67, July 1999.

[KS97] E. Keogh and P. Smyth. A probabilistic approach to fast pattern matching in time series databases. In *Proceedings of the third International Conference on Knowledge Discovery and Data Mining*, 1997.

[KTS99] M. Kudo, J. Toyama, and M. Shimbo. Multidimensional curve classification using passing-through regions. *Pattern Recognition Letters*, 20(11-13):1103–1111, 1999.

[KV95] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation and active learning. In D. S. Touretzky, G. Tesauro, and T. K. Leen, editors, *Advances in Neural Information Processing Systems*, pages 231–238. MIT Press, 1995.

[Kvå87] T.O. Kvålseth. Entropy and correlation : Some comments. *IEEE Trans. on Systems, Man and Cybernetics*, SMC-17(3):517–519, 1987.

[KW96] R. Kohavi and D.H. Wolpert. Bias plus variance decomposition for zero-one loss functions. In *Proc. of the 13th Int. Conf. on Machine Learning*, 1996.

[LLS00] T.S. Lim, W.Y. Loh, and Y.S. Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithm. *Machine learning*, 40(3):203–228, 2000.

[LS98] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, 1998.

[LSL$^+$00] V. Lavrenko, M. Schmill, D. Lawrie, P. Ogilvie, D. Jensen, and J. Allan. Mining of concurrent text and time series. In *Proceedings of the sixth International Conference on Knowledge Discovery and Data Mining*, pages 37–44, 2000.

[Mac92] D.J.C. MacKay. *Bayesian methods for adaptive models.* PhD thesis, California Institute of Technology, Pasadena, California, 1992.

[Man97] S. Manganaris. *Supervised classification with temporal data.* PhD thesis, Vanderbilt University, 1997.

[MCSL01] V. Medina-Chico, A. Suarez, and J.F. Lutsko. Backpropagation in decision trees for regression. In Luc De Raedt and P. Flach, editors, *Proc. of ECML 2001, 12th european conference on machine learning*, pages 348–359. Springer-Verlag, 2001.

[Min89a] J. Mingers. An empirical comparision of pruning methods for decision tree induction. *Machine learning*, 4(2):227–243, 1989.

[Min89b] J. Mingers. An empirical comparison of selection measures for decision tree induction. *Machine learning*, 3(4):319–342, 1989.

[MS94] D. Michie and D.J. Spiegelhalter, editors. *Machine learning, neural and statistical classification.* Ellis Horwood, 1994.

[Muñ96] A. Muños. *Aplicación de técnicas de redes neuronales artificiales al diagnóstico de procesos industriales.* PhD thesis, Univeridad Pontificia Comillas, September 1996.

[Nea96] R. M. Neal. *bayesian learning for neural networks.* Springer-Verlag, New York, 1996.

[Oat99] T. Oates. Identifying distinctive subsequences in multivariate time series by clustering. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, pages 322–326, 1999.

[OD95] J.J. Oliver and D.L. Dowe. On pruning and averaging decision trees. In *Proc. of the 12th International Conference on Machine Learning*, pages 430–437, 340 Pine Street, 6th Floor San Francisco, California 94104 U.S.A., 1995. Morgan Kaufmann.

[ODK96] M. Ostendorf, V. Digalakis, and O. Kimball. From hmms to segment models: a unified view of stochastic modeling for speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 4:360–378, 1996.

[Ols01a] R.T. Olszewski. *Generalized feature extraction for structural pattern recognition in time series data.* PhD thesis, School of Computer Science, Carnegie Mellon University, February 2001.

[Ols01b] R.T. Olszewski. *Generalized feature extraction for structural pattern recognition in time-series data.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 2001.

[OM99] D. Opitz and R. Maclin. Popular ensemble methods: an empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.

[OW02] C. Olaru and L. Wehenkel. A complete fuzzy decision tree technique. *Fuzzy sets and systems (submitted)*, 2002.

[Pea88] J. Pearl. *Probabilistic reasoning in intelligent systems.* Morgan Kaufmann, Los Altos, California, 1988.

[Qui86] J.R. Quinlan. *C4.5: Programs for machine learning.* Morgan Kaufmann (San Mateo), 1986.

[Qui96] J.R. Quinlan. Bagging, boosting, and c4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 725–730. AAAI Press, 1996.

[Rab89] R. L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *IEEE Proceedings*, 77(2):257–285, February 1989.

[Ris78] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[Rou80] E.M. Rounds. A combined nonparametric approach to feature selection and binary decision tree design. *Pattern recognition*, 12:313–317, 1980.

[Sai94] N. Saito. *Local feature extraction and its application using a library of bases.* PhD thesis, Department of Mathematics, Yale University, 1994.

[Sch78] G. Schwartz. Estimating the dimension of a model. *Annals of statistics*, 6, 1978.

[Sch90] R.E. Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[Sen99] A. Senior. Recognizing faces in broadcast video. In *Proceedings of the IEEE International Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-time systems*, pages 105–110, Kerkyra, Greece, 1999.

[Set95] I.K. Sethi. Neural implementation of tree classifiers. *IEEE Transactions on systems, man and cybernetics*, 25(8):1243–1249, August 1995.

[SFBL97] R.E. Schapire, Y. Freund, P. Bartlett, and W.S. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. In *Proceedings of the 14th International Conference on Machine Learning*, pages 322–330. Morgan Kaufmann, 1997.

[Shu82] R.H. Shumway. Discriminant analysis for time series. In P.R. Krishnaiah, editor, *Handbook of Statistics 2: Classification, Pattern Recognition, and Reduction of Dimensionality.* North-Holland, 1982.

[SK96] P. Sollich and A. Krogh. Learning with ensembles: How over-fitting can be useful. In M. E. Hasselmo, D. S. Touretzky, and M. C. Mozer, editors, *Advances in Neural Information Processing Systems*, pages 190–196. MIT Press, 1996.

[SLCM98] L. Seabra Lopes and L.M. Camarinha-Matos. Feature transformation strategies for a robot learning problem. In H. Liu and H. Motoda, editors, *Feature extraction, construction, and selection: a data mining perspective.* Kluwer Academic Publishers, 1998.

[SÚW98] E.F. Sánchez-Úbeda and L. Wehenkel. The hinges model: a one-dimensional continuous piecewise polynomial model. In *Proceedings of IPMU'98, International Conference on Information Processing and Management of Uncertainty in Knowledge based Systems*, Paris, 1998.

[Tib96] R.J. Tibshirani. Bias, variance and prediction error for classification rules. Technical report, University of Toronto, November 1996.

[Vap95] V.N. Vapnik. *The nature of statistical learning theory.* Springer Verlag, 1995.

[Web00] G. Webb. Multiboosting: A technique for combining boosting and wagging. *Machine Learning*, 40(2), 159-196 2000.

[Weh93] L. Wehenkel. Decision tree pruning using an additive information quality measure. In B. Bouchon-Meunier, L. Valverde, and R.R. Yager, editors, *Uncertainty in Intelligent Systems*, pages 397–411. Elsevier, North Holland, 1993.

[Weh96] L. Wehenkel. On uncertainty measures used for decision tree induction. In *Proceedings of Info. Proc. and Manag. of Uncertainty*, pages 413–418, 1996.

[Weh97] L. Wehenkel. Discretization of continuous attributes for supervised learning : Variance evaluation and variance reduction. In *Proceedings of The Int. Fuzzy Systems Assoc. World Congress (IFSA'97)*, pages 381–388, 1997.

[Weh98] L. Wehenkel. *Automatic learning techniques in power systems.* Kluwer Academic, Boston, 1998.

[WG94] A.S. Weigend and N.A. Gershanfeld. *Time series prediction: forecasting the future and understanding the past.* Addison-Wesley Publishing, Santa Fe Institute, 1994.

[Wol92] D.H. Wolpert. Stacked generalization. *Neural networks*, 5:241–259, 1992.

[Wol97] D.H. Wolpert. On bias plus variance. *Neural computation*, pages 1211–1243, 1997.

[WVCRP89] L. Wehenkel, Th. Van Cutsem, and M. Ribbens-Pavella. Inductive inference applied to on-line transient stability assessment of electric power systems. *Automatica*, 25(3):445–451, 1989.

[ZC01] G. Zenobi and P. Cunningham. Using diversity in preparing ensembles of classifiers based on different feature subsets to minimize generalization error. In L. De Raedt and P. Flach, editors, *Proceedings of the 12th European Conference on Machine Learning*, Freiburg, Germany, September 2001.

[ZR00] D.A. Zighed and R. Rakotomalala. *Graphes d'Induction : Apprentissage et Data Mining.* Editions Hermes, 2000.

[ZRF99] D.A. Zighed, R. Rakotomalala, and F. Feschet. Is it worth to achieve optimality in a discretization process ? *Information and Science*, 1999.

Author Index