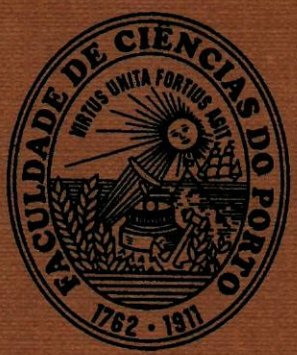
João Manuel Portela da Gama

Combining Classification Algorithms



Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
1999

João Manuel Portela da Gama

Combining Classification Algorithms



Tese submetida à Faculdade de Ciências da Universidade do Porto para obtenção do grau de Doutor em Ciência de Computadores

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
1999

Para os meus Pais,
o Luis, a Rita e a Júlia

Acknowledgements

This work would not have been realized without the help of many people. It is not possible to enumerate even an approximately overview of all people that influenced it. Nevertheless I would like to express my gratitude to some people in particular.

First of all, I want to thank Pavel Brazdil, my superviser, that has been an unlimited source of encouragement, support, enthusiasm and patience. He introduced me to Machine Learning, guided me, gave me freedom to pursue my interests, provided me with the advices and resources I needed.

My colleagues in LIACC, in particular those in NIAAD (the Machine Learning group), provided a good working atmosphere and estimulating discussions. I am specially grateful to my colleagues in the office and those who solved some intriguing problems when configuring computers and writing in English.

Thanks to Faculty of Economics U. Porto, MLNet, StatLog project, PRAXIS XXI, ECO project, and METAL project for the financial support to attend conferences, a great source of information, discussion, and friends. A special thank to the anonymous reviews of the papers I have submited. Most of their recomendations have influenced my work.

Thanks also to my family, and friends for their support, care, and patience. A particular thanks to my kids, for the very good relaxing moments they provide.

Abstract

The ability of a chosen classification algorithm to induce a good generalization depends on how appropriate its representation language used to express generalizations of the examples is for the given task. Since different learning algorithms employ different knowledge representations and search heuristics, different search spaces are explored and diverse results are obtained. The problem of finding the appropriate model for a given task is an active research area. In this dissertation, instead of looking for methods that fit the data using a single representation language, we present a family of algorithms, under the generic name of *Cascade Generalization*, whose search spaces contains models that use different representation languages.

The basic idea of the method is to use the learning algorithms in sequence. At each iteration a two step process occurs. In the first step, a model is built using a base classifier. In the second step, the instance space is extended by the insertion of new attributes, generated by the base model. The constructive step generates terms in the representational language of the base classifier. If the high level classifier chooses one of these terms, its representational power has been extended. The *bias* restrictions of the high level classifier is relaxed by incorporating terms of the representational language of the base classifiers. This is the basic idea behind *Ltree* and the *Cascade Generalization* architecture.

The method is presented in two parts, following somewhat different perspectives. In the first part, it is presented as a method for building multivariate trees. Here we present system *Ltree*, a multivariate decision tree. *Ltree* uses as constructive operator a *linear discriminant* function. It was the precursor of the *Cascade* architecture. In the second part, we present a general framework for combining classifiers. The method *Cascade Generalization* is an extension of the method presented in the first part. The base classifiers are not restricted to *discriminant functions*, but are generalized to other types of classifiers. We define the conditions that the base classifier must satisfy so that it can be used in this framework, and define criteria for selecting the suitable

types of low and high level classifiers. We present two different variants of Cascade Generalization. The first one employs loosely coupled classifiers whilst the second one uses tight coupling. In the first schema base classifier(s) pre-process data for another stage. This framework can be used to combine most of the existing classifiers without changes, or with rather small changes. The method only requires that the original data be extended by the insertion of the probability class distribution that must be generated by the base classifier. In the second schema, we use constructive induction locally. That is, two or more classifiers are coupled locally. Although in this dissertation we have used only *Local Cascade Generalization* in conjunction with decision trees, the method could be easily extended to other *divide-and-conquer* systems, like *decision lists.*

We empirically show that the method improves the accuracy of the base classifiers and competes well against other methods that combine classifiers, namely the state of the art method - Boosting.

Resumo

A capacidade de um algoritmo de aprendizagem induzir,para um determinado problema, uma boa generalização depende da linguagem de representação usada para generalizar os exemplos. Como diferentes algoritmos usam diferentes linguagens de representação e estratégias de procura, são explorados diferentes espaços e são obtidos diferentes resultados. O problema de encontrar a representação mais adequada para o problema em causa, é uma área de investigação bastante activa. Nesta dissertação, em vez de procurar métodos que fazem o ajuste aos dados usando uma única linguagem de representação, apresentamos uma família de algoritmos, sob a designação genérica de *Generalização em Cascata*, onde o espaço de procura contem modelos que utilizam diferentes linguagens de representação.

A ideia básica do método consiste em utilizar os algoritmos de aprendizagem em sequência. Em cada iteração ocorre um processo com dois passos. No primeiro passo, um classificador constrói um modelo. No segundo passo, o espaço definido pelos atributos é extendido pela inserção de novos atributos gerados utilizando este modelo. Este processo de construção de novos atributos constrói atributos na linguagem de representação do classificador usado para construir o modelo. Se posteriormente na sequência, um classificador utiliza um destes novos atributos para construir o seu modelo, a sua capacidade de representação foi extendida. Desta forma as restrições da linguagem de representação dos classificadores utilizados a mais alto nível na sequência, são relaxadas pela incorporação de termos da linguagem de representação dos classificadores de base. Esta é a metodologia base subjacente ao sistema *Ltree* e à arquitectura da *Generalização em Cascata.*

O método é apresentado segundo duas perspectivas. Numa primeira parte, é apresentado como uma estratégia para construir árvores de decisão multivariadas. É apresentado o sistema *Ltree* que utiliza como operador para a construção de atributos um discriminante linear. Foi o percursor da arquitectura de *Generalização em Cascata.*

Na segunda parte, apresentamos um esquema genérico para combinar classificadores. O método de *Generalização em Cascata* é uma extensão da metodologia apresentada na primeira parte. Os classificadores de base não são restrictos a funções discriminantes mas generalizados a outros classificadores, podendo ser combinados vários classificadores. Nesta segunda parte definimos as condições que um classificador deve satisfazer para poder ser utilizado neste enquadramento. Definimos também, os critérios para seleccionar o tipo mais adequado de classificadores a utilizar na sequência.

Apresentamos duas variantes para a arquitectura de *Generalização em Cascata*. No primeiro esquema, os classificadores de base são usados como pré-processadores dos dados para a etapa seguinte. Este método pode ser usado para combinar a maioria dos classificadores existentes. O método apenas requer a extensão dos dados iniciais com a probabilidade de distribuição de classes geradas pelo classificador de base. No segundo esquema, os classificadores de base são integrados localmente de uma forma mais aprofundada. Apesar de nesta tese só termos utilizado *Generalização em Cascata Local* no contexto de árvores de decisão, o método poderá ser extendido para outros algoritmos que utilizam uma estratégia de dividir-e-conquistar, tais como *listas de decisão*.

Na avaliação empírica efectuada, foi verificado que o método melhora a capacidade de generalização dos classificadores de base e é competitivo com outros métodos para combinar classificadores, nomeadamente *o estado da arte - Boosting*.

Résumé

La capacité d'un algorithme d'apprentissage à induire, pour un problème déter-miné, une bonne généralisation dépend du langage de représentation utilisé pour généraliser les exemples. Étant donné que différents algorithmes emploient différents langages de représentation et stratégies de recherche, de différents espaces sont exploités et de différents résultats sont obtenus. Le problème de trouver la représentation la plus adéquate au problème en question est une aire de recherche assez active. Dans cette dissertation, au lieu de chercher des méthodes qui font l'ajustement aux données en utilisant un seul langage de représentation, nous présentons une famille d'algorithmes, sous la désignation générique de Généralisation en Cascade, oú l'espace de recherche contient des modèles qui utilisent de différents langages de représentation. L'idée base de la méthode consiste à utiliser les algorithmes d'apprentissage en séquence. À chaque itération un processus arrive avec deux pas. Dans un premier pas, un classificateur construit un modèle. Dans le deuxième pas, l'espace définit par les attributs est élargit par l'insertion de nouveaux attributs créés en utilisant ce modèle. Ce procès permet la construction de nouveaux attributs dans le langage de représentation du classificateur utilisé dans la construction du modèle. Si après dans la suite, un classificateur fait usage d'un de ces nouveaux attributs pour construire son modèle, sa capacité de représentation a été élargie. De cette manière les restrictions de langage de représentation des classificateurs utilisés au plus haut niveau de la séquence, sont libérés par l'incorporation de termes du langage de représentation des classificateurs de base. Voilà la méthodologie base sous-jacente au système *Ltree* et à l'architecture de la Généralisation en Cascade.

La méthode est présentée selon deux perspectives. Dans une première partie, on la présente comme une stratégie pour la construction d'arbres de décision multivariés. On présente le système *Ltree* qu'utilise un discriminateur linéaire comme opérateur pour la construction d'attributs. Il fut le percurseur de l'architecture de la Généralisation en Cascade. Dans la deuxième partie, nous présentons un schéma général pour combiner

des classificateurs. La méthode de Généralisation en Cascade est une extension de la méthodologie présentée dans la première partie. Les classificateurs de base ne sont pas restricts à des fonctions discriminatoires mais généralisés à d'autres classificateurs. Dans cette deuxième partie, nous définissons les conditions qu'un classificateur doit satisfaire pour être utilisé dans ce cadre. Nous définissons aussi les critères pour sélectionner le genre de classificateurs le plus adéquat pour l'utilisation dans la séquence. Nous présentons deux possibilités pour l'architecture de la Généralisation en Cascade. Dans le premier schéma, les classificateurs de base sont utilisés comme preprocesseurs de données pour l'étape suivante. Cette méthode peut être utilisée pour combiner la plupart des classificateurs existants. La méthode n'exige que l'extension des premières données avec la probabilité de distribution de classes gérées par le classificateur de base. Dans le deuxième schéma, les classificateurs de base sont intégrés d'une façon plus approfondie. Bien que dans cette thèse nous n'avons utilisé la Généralisation en Cascade local que dans le contexte d'arbres de décision, la méthode pourra être étendue à d'autres algorithmes qu'utilisent une stratégie de diviser-et-conquérir, tel que dans les *listes de décision.* Au cours de l'évaluation empirique efectuée, on a vérifié que la méthode améliore la capacité de généralisation des classificateurs de base et elle est compétitive par rapport à d'autres méthodes de combiner des classificateurs, surtout l'état de l'art - *Boosting.*

Contents

Acknowledgements **5**

Abstract **7**

Resumo **9**

Résumé **11**

List of Tables **23**

List of Figures **26**

1 Introduction **27**

1.1 Introduction 28

1.2 The Learning Bias 30

1.3 Extending the Representational Power 32

1.3.1 Theory Revision 32

1.3.2 Constructive Induction 33

1.4 The Limits of Machine Learning 34

1.5 Overview of the Dissertation 36

2 Methods for Inductive Learning **37**

2.1 Introduction 38
2.2 Supervised Learning Algorithms 38
2.2.1 Decision Trees 38
2.2.1.1 Splitting rules 40
2.2.1.2 Pruning strategies 42
2.2.1.3 Missing values 46
2.2.1.4 Characteristics of Decision Trees 47
2.2.2 Extensions 49
2.2.2.1 Multivariate trees 49
2.2.2.2 Option trees 55
2.2.2.3 Incremental decision trees 56
2.2.2.4 Decision rules 57
2.2.3 Discriminant Functions 58
2.2.3.1 Naive Bayes classifier 58
2.2.3.2 Linear discriminant classifier 60
2.2.3.3 Quadratic discriminant classifier 63
2.2.3.4 Logistic discriminant classifier 63
2.2.4 Comparison Between the Different Models 66
2.3 Summary 67

3 Methods for Evaluation Learning Algorithms 69

3.1 Introduction 70
3.2 Evaluation of Learning Algorithms 71
3.2.1 Sources of Variability of the Error Rate 71
3.2.2 Estimating the Error Rate 72
3.2.3 Test of Hypotheses 74

3.2.4 Ranking of Algorithms 76
3.2.4.1 How far from the best performance? 76
3.2.5 Additional Comments 77
3.2.6 Learning Time 77
3.3 Bias-Variance Decomposition of the Error Rate 78
3.3.1 The Bias-Variance Decomposition 79
3.3.2 Measuring the Bias-Variance Components 81
3.4 Model Complexity 82
3.4.1 Minimum Description Length 83
3.4.2 Codification Schema 83
3.5 Evaluating Base Algorithms 84
3.6 Summary 88

4 Discriminant Trees **89**

4.1 Introduction 90
4.2 An Illustrative Example 90
4.2.1 Motivation 90
4.2.2 Constructive Induction 91
4.2.3 Our Approach 93
4.3 Generating New Attributes in a Pre-processing Step: C4.5Oblique . . . 96
4.4 Linear Tree 96
4.4.1 An Illustrative Example: The Iris Data 96
4.4.2 New Attributes 98
4.4.3 Splitting Criteria 99
4.4.4 Stopping Criteria 99
4.4.5 Smoothing 100

4.4.6 Pruning . . . 101
4.4.7 Missing Values . . . 102
4.4.8 Classifying New Examples . . . 102
4.5 Empirical Evaluation . . . 102
4.5.1 Evaluation on Artificial Data . . . 103
4.5.2 Evaluation on UCI and StatLog Datasets . . . 105
4.5.3 Why does *Ltree* Work? . . . 108
4.5.4 Evaluation of Ltree Variants . . . 109
4.5.4.1 The Influence of Smoothing . . . 109
4.5.4.2 The Effect of Downward Propagation of the New Attributes . . . 111
4.5.4.3 The Effect of Using Different Attribute Constructors . 112
4.5.5 Bias-Variance Analysis . . . 113
4.5.6 How Far from the Best Algorithm? . . . 113
4.5.7 Model Complexity - Minimum Description Length . . . 114
4.5.8 Learning Time . . . 114
4.6 Conclusions . . . 115

5 Multiple Models **119**

5.1 Multiple Models . . . 120
5.2 Combining Classifiers Predictions . . . 122
5.2.1 Voting versus Non-voting Methods . . . 122
5.2.1.1 Voting Methods . . . 122
5.2.1.2 Non Voting Methods . . . 123
5.2.2 Dynamic versus Static Methods . . . 123
5.2.2.1 Model Applicability Induction . . . 123

5.2.2.2 Composite Learner 124
5.2.2.3 SCANN - Correspondence Analysis followed by a Nearest Neighbor 125
5.3 Generating Different Models 125
5.3.1 Combining Homogeneous Classifiers 125
5.3.1.1 Bagging 126
5.3.1.2 Boosting 128
5.3.1.3 Option Trees 130
5.3.1.4 Error-correcting output codes 130
5.3.2 Combining Heterogeneous Classifiers 131
5.3.2.1 Stacked Generalization 131
5.3.2.2 Meta Learning 133
5.4 Hybrid Systems 133
5.5 Discussion 135
5.6 Summary 136

6 Cascade Generalization **137**

6.1 Introduction 138
6.2 Cascade Generalization 138
6.3 An Illustrative Example 140
6.4 Discussion 142
6.5 Empirical Evaluation 143
6.5.1 Comparison with Stacked Generalization 145
6.5.1.1 The Type of $level_1$ Attributes in Cascade Generalization 146
6.5.2 Rank of Algorithms 147
6.6 Why does Cascade Generalization Work? 148

6.6.1 Error Correlation 148
6.6.2 The Bias-Variance Decomposition 149
6.6.3 The effect on Between-Classes Distance 151
6.6.4 A Detailed Analysis on Satimage Dataset 152
6.6.4.1 Error rates 152
6.6.4.2 Bias-Variance analysis 153
6.6.4.3 Between-Classes Distances 154
6.7 Conclusions 155

7 Local Cascade Generalization **157**

7.1 Introduction 158
7.2 Local Cascade Generalization 158
7.3 An Illustrative Example 160
7.4 Relation to Other Work on Multivariate Trees 162
7.4.1 Relation to Linear Tree 162
7.5 Empirical Evaluation 162
7.5.1 Analysis of the Error Rates 162
7.5.1.1 Comparison of Local and Global Cascade Combinations 163
7.5.1.2 Comparison of Local Cascade Generalization with Boosting and Stacked Generalization 164
7.5.1.3 A Step Ahead - Boosting a Global Cascade Algorithm 165
7.5.2 Analysis of the Number of Leaves 165
7.5.3 Learning Times 165
7.6 Concluding Remarks 166

8 Conclusions **169**

8.1 Concluding Remarks 170

8.2 Limitations and Future Work 172

A Datasets used **175**

A.1 Datasets Description 176

References **182**

List of Tables

3.1 Error rates of base classifiers 85

3.2 Summary of the results of base algorithms 86

3.3 Average Distance to the Best Algorithm 86

3.4 Average of Ranks 86

3.5 Relative learning times for the base algorithms 87

3.6 Bias and Variance contribution to error rate 87

4.1 Three examples from the Iris dataset 97

4.2 Error rates and Nr. of leaves on 10-Fold Cross-Validation 103

4.3 Error rates of C4.5 versus Ltree and C45Oblique 105

4.4 Summary of Results (error-rate) 106

4.5 Summary of Results (Nr. of Leaves) 106

4.6 Error rates of Multivariate Trees 107

4.7 Summary of Multivariate Trees Results (error rate) 108

4.8 Summary of Multivariate Trees Results (Nr. of Leaves) 108

4.9 Error rates of Ltree variants 110

4.10 Summary of Ltree Variants (error-rate) 110

4.11 Summary of Ltree Variants (Nr. of Leaves) 111

4.12 Summary of Bias-Variance decomposition 113

4.13 Averages of Distances to the Best Algorithm 114

4.14 Minimum Description Length 115

4.15 Average of relative running times 116

5.1 Example of Output Representations 131

6.1 Results of Cascade Generalization (error rates) 144

6.2 Results of Cascade Generalization (error rates) 145

6.3 Summary of results of Cascade Generalization (error rates) 146

6.4 Comparison of Cascade Generalization versus Stacked Generalization . 147

6.5 Summary of comparison against Stacked Generalization (error rates) . 147

6.6 The type of $level_1$ attributes. Summary of Results. 148

6.7 Average error rates of classifiers 148

6.8 Average distance to the best algorithm 148

6.9 Error Correlation between base classifiers 149

6.10 Bias Variance decomposition of error rate 150

6.11 Average Between-classes and Within-class distances. 153

6.12 Error rates and mean nr. of leaves of base and composite models 153

6.13 Bias Variance decomposition of error rate 153

6.14 Between-classes Distances 155

6.15 Within-class Distances 155

6.16 Average Distances 155

7.1 Results of (a)Local Cascade Generalization (b)Boosting and Stacked (c) Boosting a Cascade algorithm 163

7.2 Summary of results comparing Global versus Local Cascade Generalization 164

7.3 Summary of results (Error Rates) of Local Cascade Generalization, Boosting and Stacked Generalization 164

7.4 Summary of results (Nr. of Leaves) of Local Cascade Generalization . . 165

7.5 Comparison of relative learning times of local Cascade Classifiers 166

A.1 Datasets Characteristics 176

List of Figures

2.1 Algorithm for growing a Decision Tree 39

2.2 Decision surfaces generated by linear, logistic and quadratic discriminants on a two-class problem defined by two attributes. 65

3.1 Bias and Variance components of the error of C4.5, Discrim and naive Bayes. 87

4.1 The diagonal problem. The left figure shows the original data. The right part shows the decision surface determined by a decision tree. . . 90

4.2 The extended instance space. 94

4.3 Decision tree for iris data 97

4.4 Smooth propagation of class probabilities 101

4.5 Learning Curves for LS10 data (a) Nr. leaves and Error rate for C4.5 (b) Error rates for Multivariate Trees (c) Nr. leaves for Multivariate Trees 104

4.6 Learning curves for XOR data (a) Data set (b) Error Rates (c) Nr. leaves105

4.7 Bias Variance decomposition of the error rate. For each dataset the decomposition is shown for C4.5, Ltree, Qtree, and LgTree in this order 113

4.8 The learning time of multivariate algorithms for each dataset. 116

5.1 (a) Error rate versus nr. of classifiers in an ensemble. (b) Probability that exactly n of 24 classifiers will make an error. 120

5.2 Model Applicability Induction: the algorithm to generate referees. . . . 124

5.3 The Bagging Algorithm 126

5.4 The left figure represents the individual decision surfaces of three classifiers. The right figure presents, in bold, the decision surface obtained by uniform vote of the three classifiers. 127

5.5 The ADABOOST.M1 Algorithm 129

5.6 Stacked Generalization architecture. 132

5.7 MCS rule base: an example. 134

6.1 Tree generated by C4.5 ∇Bayes 141

6.2 Bias-Variance decomposition of the error rate for C4.5, Bayes and C4.5∇Bayes for different datasets. 150

6.3 Between and within cluster distances. 152

6.4 (a) C4.5 versus Bayes and (b) C4.5 versus C4.5∇Bayes 154

6.5 (a) C4.5 versus Discrim and (b) C4.5 versus C4.5∇Discrim 154

7.1 Local Cascade Algorithm based on a Decision Tree 159

7.2 Tree generated by a CGTree using Discrim∇Bayes as constructive operator 161

7.3 Learning Curves and Learning Times on Satimage Dataset. 166

7.4 Learning Curves and Learning Times on Satimage Dataset. 167

Chapter 1

Introduction

In this chapter we introduce the problem tackled in this dissertation. It is situated within the area of supervised Machine Learning. *The basic concepts used throughout this dissertation are defined and the problem of the learning bias is presented. The basic notions of* Constructive Induction *as a method to overcome the limitations of inductive bias are discussed. The problem that "there is no universal bias" introduces limitations on the generalization ability of any algorithm. This is the underlying problem that we address in this dissertation. The final section of the chapter presents an overview of the dissertation.*

1.1 Introduction

Informally speaking, one of the tasks of machine learning is to generate a *model* from a given set of labelled examples. This *model* can be used to classify unlabelled, previously unseen examples. The generated model can be evaluated in terms of accuracy, comprehensibility, compactness, learning speed, classification speed, etc. This task has been denoted as *pattern recognition, discrimination* or *supervised learning* in contrast to the task of *unsupervised learning* or *clustering* where the *labels* are deduced from the data.

Several areas of human activity can involve *supervised machine learning*: predicting the use of land, based on satellite images; assigning credit to individuals on the basis of financial information; sorting letters on the basis of machine-readable postcodes; preliminary diagnosis of a patient's disease; etc. Several problems in industry, commerce, and science are *decision problems* and require the analysis of complex and extensive data. Most of those problems can be analyzed from a *machine learning* perspective.

Throughout this dissertation we use the following definitions.

- An *example*, sometimes referred as an *instance*, is an ordered set of *variables*.
- Each variable, sometimes referred to as *attribute* or *feature*, takes values from a pre-defined set of values that are problem dependent. The pre-defined set of values is denoted as the *domain* of the variable. Typical real-world machine learning problems, are heterogeneous problems, *i.e.* on the same problem we can have attributes with very different domains. In this study we differentiate between the following two types of domains:

  1. *Nominal domains*
     These are domains for which we can enumerate all possible variable values and there is no order relation between them. For example the set of colors: {white, black, green, red} represents a nominal domain.
  2. *Ordered domains*
     With ordered domains different variable values are ordered. Often it is possible to enumerate all possible values that the variable can take (e.g. the age of a patient). Sometimes it is not possible to enumerate all the possible values (e.g. the temperature of the body). This type of domains is also referred as *continuous*.

One of the attributes of an example is a special variable, referred as the *target* attribute. The *target* describes the phenomenon of interest, that is, the phenomenon we would like to make predictions about. An example for which the target is known is referred to a *labelled example.* An *unlabelled example*, sometimes referred as *query example*, is an example for which the target attribute is unknown.

When the target attribute is a continuous variable the supervised learning problem is usually referred as *regression* [DH73, BJ77, BFOS84]. Regression methods explore the *order relation* in the target variable. In this dissertation, the target variable it is assumed to take values in a predefined set of nominal values. It is also referred to *class* or *label.* The supervised learning problem is usually referred as a *classification* problem.

- A *dataset* is a set of examples. We distinguish between the *training set* that is a set of labelled examples, and the *test set* that is a set of unlabelled examples.

- A *model* (or *classifier* in the case of classification problems) is a mapping between the given variables and the target variable. It is used to predict the target variable for unlabelled examples. Usually the target variable is referred to as the *dependent* variable, and all the other variables are referred to as *independent* variables.

- A *learner* is any procedure used to build a model. In the context of this work we restrict the learners to computer programs that built the *model* from a *training set* without interaction with the users or other form of knowledge about the problem.

The underlying assumption of the learning task is that there is an unknown *function* that performs the mapping between the independent variables and the dependent variable. In general it is not possible to discover the *true function* from a finite set of examples. Also in real-world problems, the available data contains *noise* both in the attribute values and class values (i.e. examples may be misclassified). The goal of the learner is to obtain an approximation to the unknown function.

The problem of supervised learning can also be formulated within *Inductive Logic Programming* (or ILP for short) framework [Mit97, Jor98]. In this setting the generated model is expressed using first-order Horn clauses. Those models can be seen as PROLOG programs which are induced from the examples. In this dissertation, we restrict the models to the propositional logic.

1.2 The Learning Bias

Formally the learning problem can be formulated as follows: A phenomenon is described by an unknown function that maps a set of independent variables into a dependent variable:

$$f : \mathcal{X} \mapsto \mathcal{Y}$$

Given a training set $\mathcal{L}$ with M examples, defined as:

$$\mathcal{L} = \{(\vec{x}_m, y_m), m = 1, ..., M\}$$

and a learner $\Im$, we obtain a model that is an estimation for function f:

$$\hat{f} = \Im(\mathcal{L}) \tag{1.1}$$

Several research communities studied this problem. The statistics community [Fuk90, DH73], sometimes using the term *pattern recognition*, has developed different paradigms, such as *discriminant analysis* [MST94, DG84], *naive Bayes* or *nearest neighbor* methods usually focusing on numerical relations between variables. The Machine Learning community has developed methods oriented for nominal variables, without assuming any data distribution. The models in the form of *Decision trees* [Qui93] and *Rules learners* [CN89] try to capture logical relations between variables. The neural network community has developed various forms of feed-forward and recurrent networks [Rip96].

Each approach uses different representation schema and explores different search strategies. A representation, denoted by $\mathcal{R}$, is the decision structure of certain type (*i.e.* a decision tree, a set of discriminant equations, a table of conditional probabilities, etc.) used to generalize the examples. The representation used by a learning algorithm restricts the set of hypotheses considered by the algorithm. Some authors call it the *restricted hypotheses space bias* [Mit90a].

A *search strategy*, denoted by $\mathcal{S}$, is the set of methods and heuristics used to explore the search space defined by the set of possible representations. Associated with each search strategy is the *evaluation* component. This is the set of methods and heuristics used to guide the search, by preferring one hypothesis over others, or by ranking the set of possible hypotheses. Both the search strategy and the evaluation component

contain preferences on the possible set of hypotheses. It is also known in the machine learning literature as *preference bias* because it imposes a certain preference order on the elements of the hypotheses space. A commonly used heuristic is *to prefer general hypotheses over specialized ones.* For example, if we consider decision trees, this corresponds to the preference of small trees to larger ones[1]. This kind of preferences that minimize the syntactic complexity of the hypotheses representation, has roots in the *Occam's razor* [BEHM90], that says:

> *The simpler of two competing hypotheses should always be preferred.*

One argument in favor to simplicity is that there are fewer simple hypotheses than complex ones (based on combinatorial arguments). As such if both fit the data we should prefer the simpler hypothesis because is less likely that it is a statistical coincidence [Mit97]. Domingos [Dom98b] presents a thorough discussion about the interpretation of *Occam's razor*, in the context of machine learning. The author argues that a correct interpretation is:

> *Given two models with the same generalization error, the simpler one should be preferred because simplicity is desirable in itself.*

He concludes: "if a model with low training-set error is found within a sufficiently small set of models, it is likely to also have low generalization error". That is, fully adequate model evaluation is only possible if the search process by which models are obtained is also taken into account [Dom98c].

Some of the commonly used models, for example decision trees, can obtain an arbitrarily good fit to the data by considering very complex hypotheses. Some technique must be used to balance the complexity of the hypothesis against the number of training examples misclassified by the hypothesis. One of such techniques is the *Minimum Description Length* [Ris83] which states:

> *The best hypothesis is the one that minimizes the total length of the hypothesis plus the description of the exceptions to the hypothesis.*

The intuition is that *ad hoc* hypothesis consists of a list of examples, that make them no shorter than the examples. In contrast, good hypotheses reduce many examples to a single, general rule.

[1] In nested trees smaller trees correspond to more general ones.

Both *representation* ($\mathcal{R}$) and *search* ($\mathcal{S}$) define a unique *learning bias* for each algorithm [Mer98]. To make these more evident in the definition of the model, equation 1.1 can be rewritten as:

$$\hat{f} = \Im_{\{\mathcal{R},\mathcal{S}\}}(\mathcal{L}) \tag{1.2}$$

The main topic of this dissertation is about the interdependence of representation and learning. In the next section we present some machine learning techniques used to extend the representational ability of systems.

1.3 Extending the Representational Power

The importance of representation is well known throughout the artificial intelligence community. Representation affects strongly the comprehensibility of knowledge and the learnability of concepts [MBB94]. A change of representation is desirable for several reasons. If performance losses are observed it may be appropriate to revise the representation. If a concept cannot be determined new terms in the vocabulary should be used to extend the hypotheses space. Changing representation can also produce cleaner and more comprehensible knowledge.

Two techniques that allow a system to change its representation language are briefly presented in the following subsections.

1.3.1 Theory Revision

Theory revision employs inductive learning techniques to revise the existing theory, by taking into account new examples and background knowledge to produce a revised theory. Usually we require that the revised theory is in some sense "better" (e.g. more accurate) than the existing one. There are two challenges that theory revision systems face. First, the representation language which is appropriate for the initial theory may be inappropriate for the improved theory. While the original representation may concisely express the initial theory, the revised theory expressed using the same representation may be bulky, cumbersome, and difficult to obtain. Second, systems that produce only small, local changes to a theory have limited chance of accomplishing complex structural alterations that may be required [DR95].

In this dissertation we focus on a special technique that allows systems to change its representation, known as *constructive induction* [Mic90].

1.3.2 Constructive Induction

Learning can be a hard task if the attribute space is inappropriate to describe the target concept, given the bias of an algorithm. To overcome this problem some researchers propose the use of constructive learning. Matheus and Rendell [MR89] define *Constructive Induction* as:

> *the application of a set of constructive operators to a set of existing features resulting in the construction of one or more new features intended for use in describing the target concept.*

Performance of an induction algorithm, in terms of predictive accuracy, is poor if the task-supplied attributes are not appropriate for describing target theories.

We selected two illustrative examples of this situation from the machine learning literature: the first one is the *KRK* problem, and the second one is the *Iris* dataset. In the *KRK* problem, the goal is to classify as *legal* or *illegal* the chess board position formed by randomly placing the three pieces *White king*, *White rook*, and *Black king*. Task attributes consist of the column and the row of each piece. Michie and Bain applied a state-of-the-art decision-tree learner with 69% of accuracy, not differing significantly from the result achieved by the default rule. By applying a constructive operator *difference* to all of the pairs of attributes, the accuracy of the decision-learner grows to 97%.

In the well known *Iris* dataset, the *product* of features *Petal Length* and *Petal Width* produces a single feature that can be interpreted as area. This new single feature can be employed in a decision rule that produces only four errors in the whole dataset. In both problems, the target concept is hard to learn, by the induction algorithms that use only the task-supplied attributes. The verified increase in performance occurs because the introduction of new attributes allows the description, in a decision-tree language, of useful new concepts for describing the target concept. In the KRK problem the useful concepts are *same column* and *same row*, whilst in the Iris problem the useful concept is *Area*. These new concepts are hard to express in the representation language of decision trees, using only the original attributes. The introduction of the new attributes produces an extension of the representational power and relaxes the language bias of *decision tree* learning algorithms.

Wnek and Michalski [WM94] classify existing Constructive Induction systems into four categories:

1. *Data-Driven constructive induction systems*, that analyze and explore the input data, particularly the interrelationships among descriptors used in the examples, and on that basis suggest changes in the representation space.

2. *Hypothesis-Driven Constructive Induction Systems*, that incrementally transform the representation space by analyzing inductive hypothesis generated in one iteration and then use detected patterns as attributes for the next iteration.

3. *Knowledge-Driven Constructive Induction Systems*, that apply expert-provided domain knowledge to construct and/or evaluate new representation spaces.

4. *Multistrategy Constructive Induction Systems* that combine different approaches and methods for constructing new representation spaces.

Constructive induction systems usually consist of two components - one for constructing new attributes, and the other for generating theories. After the construction, new attributes are treated exactly in the same way as the original ones. For a non-trivial learning problem the number of possible constructive operators is usually very large. It is not feasible to search through all the possible combinations. Usually systems use only some of the possible operators with the objective of reducing the search space of new attributes.

Although the models generated using this methodology present considerable advantages when compared to other approaches, they have the general limitations that all learning algorithms are subject to. These restrictions, which are related to the generalization ability, are discussed in the next section.

1.4 The Limits of Machine Learning

It is well known that no algorithm can be the best choice in all possible domains. Each algorithm contains an explicit or implicit bias [Mit90b] that leads it to prefer certain generalizations over others. It will be successful only insofar as this bias matches the characteristics of the application domain. Later theoretical results [Sch95] show that performance over the set of *all* possible domains is subject to the so called *"conservation law"*: If an algorithm is better than another in some domains, then there are necessarily other domains in which this relationship is reversed. The average accuracy of a given algorithm over all domains is constant, independent of the algorithm. There is a simple example that illustrates well the *conservation law*:

> *Consider two algorithms, say $\mathcal{A}$ and $\mathcal{B}$, and a two class problem. Given a training set and test set, both algorithms generate a model from the training set and classify the examples on the test set. Suppose that algorithm $\mathcal{A}$ performs better than algorithm $\mathcal{B}$. Consider a second test set similar to the previous one except that the class label is inverted for all the examples. That is, an example that is classified as + in the first test set is classified as − in the second test set. If both algorithms are consistent, that is, if they always predict the same class for the same example when learning from the same training set, then on the second test set algorithm $\mathcal{B}$ performs better than $\mathcal{A}$.*

The underlying assumption behind the *conservation law* is that all unknown functions f are equally likely. The only information the learning algorithm has is the training examples. Under the assumption that all $f : \mathcal{X} \mapsto \mathcal{Y}$ are equally likely, the training set does not provide any information about the label of new points x different from the cases in the training set.

In machine learning the underlying assumption is that some unknown functions are more probable than others. The goal of machine learning algorithms is to find regularities, similarities, trends, and redundancies in the data. When this is possible, the description of the data is more compact than the original data itself. The model constructed from the given data sample can predict properties of future observations from the same data source.

The ability of a classifier to meaningfully respond to novel, previously unseen instances, or *generalize*, is perhaps its most important property. In general however, the generalization is not unique. Different classifiers may provide different generalizations. The problem of *Model Selection* is one the most studied in statistics [DG84] and machine learning [Mit97]. It appears in several forms, for example when trying to decide which attribute to select for test at a node of a decision tree, or what kind of topology to use for a neural net. In a more restrictive perspective of the model selection problem, it involves selection of the most appropriate classifier for a given domain.

In this dissertation we take another perspective. Selecting a single classifier, albeit the "best" one, is not necessarily the ideal thing to do. Discarding the results of less successful classifiers may waste potentially valuable information. This observation leads to the concept of "combining classifiers" or "ensembles of classifiers" [Die97] where a decision takes into account the predictions of several classifiers. A related approach, known as *Multi-strategy* learning [MT94] attempts to tackle this problem by

combining multiple methods within one algorithm. Although no multi-generalization scheme is guaranteed to yield a classifier with minimal generalization error [Wol92] it can achieve excellent results. The necessity of a system to manipulate different generalizations poses complex problems however. Citing Michalski [MT94]:

> *Multistrategy learning represents an effort to unify the entire field of machine learning and is likely to become a central research theme for the field in the future.*

1.5 Overview of the Dissertation

The main topic of this dissertation is how to extend the representational power of classifiers by relaxing their bias. Our method exploits constructive induction. The difficult problem here is how to choose the appropriate constructive operators for the problem in question. Another problem is that more powerful representations do not necessarily lead to better results. Relaxing the bias and introducing thus more flexibility can lead to increased instability (variance) which needs to be controlled. Our methodology is presented in two parts. In the first part we explore constructive induction with *multivariate trees* and use *discriminant analysis* as the constructive operator. In the second part this methodology is extended. We present a general method for combining classifiers.

Chapter 2 introduces the relevant methods of supervised learning used in this dissertation. Here we focus mainly on decision trees and discriminant functions. Chapter 3 defines methods for empirical evaluation of learning algorithms. Chapter 4 introduces a new method for generating multivariate trees, by combining a univariate tree with a discriminant function by means of constructive induction. This chapter concludes the first part of the thesis.

The second part starts with chapter 5 which presents the general perspective concerning how to generate and combine multiple models. Chapter 6 presents "Cascade Generalization", a new method for combining heterogeneous models. Chapter 7 extends this work, by applying Cascade Generalization locally, which enables to merge different classifiers in the course of learning. It can be seen as a method that permits to generate multistrategy learners.

Chapter 2

Methods for Inductive Learning

In this chapter we present several approaches to propositional supervised inductive learning. Decision trees, naive Bayes and discriminant functions are described in detail. We discuss the strengths and limitations of each method. Current trends concerning some recent developments in these area are also presented.

2.1 Introduction

The problem of supervised machine learning is a research area of increasing importance. Nowadays there exist innumerable set of methods, approaches and systems that increase all the time. For example in StatLog project [MST94], twenty one classification algorithms have been evaluated. In this chapter we analyze two different approaches to the learning problem. The first one includes a family of non-parametric methods, that is, methods that do not make any assumption about the data. Here it is represented by *decision trees.* The second approach includes the family of parametric methods that make strong assumptions about the data and includes *naive Bayes* and *discriminant functions.*

The origins of the two approaches can be traced back to different scientific areas, and are based on different perspectives. In this chapter we present the basic ideas behind the two families of methods, their strengths and limitations and the current developments in the respective areas. Those families of methods are usually seen as different directions to the supervised learning problem. As we will see later in the following chapters they can be used in a complementary way.

2.2 Supervised Learning Algorithms

2.2.1 Decision Trees

A decision tree uses a divide-and-conquer strategy. It attacks a complex problem by dividing it into simpler problems and recursively applying the same strategy to the sub-problems. Solutions of sub-problems can be combined to yield a solution of the complex problem. This is the basic idea behind the well known decision tree based algorithms, such as, ID3 [Qui79], ASSISTANT [CKB87], CART [BFOS84], C4.5 [Qui93]. Recently several statistical packages, S_{plus}, *Statistica*, and *SPSS* [Mat98], have incorporated functions that implement binary decision trees for classification and regression problems.

The power of this approach comes from the ability to split the instance-space into subspaces and each subspace is fitted with different models.

Formally a decision tree is a direct acyclic graph in which each node is either a *decision node* with two or more successors or a *leaf node.* A *leaf node* is labelled with a *class.*

Input: A data set D
Output: A decision tree: $Tree$

Function GrowTree(D)
[1] **If** stopping criteria(D) = TRUE
[2] **return** a $Leaf$
Else
[3] Choose the attribute that maximizes splitting criterion on D
[4] **For each** partition of examples based on chosen attribute values
generate a subtree $Tree_i$ = GrowTree(D_i)
[5] **return** $Tree$ containing a decision node based on chosen attribute,
and descendants $Tree_i$
End If
End

Figure 2.1: Algorithm for growing a Decision Tree

A *decision node* has some *test* based on attribute values.

The hypothesis space of ID3 and its descendants are within the DNF formalism. Classifiers generated by those systems encode a DNF for each class. For each DNF, the conditions along a branch represent conjuncts and the individual branches can be seen as disjuncts. Each branch forms a rule with a conditional part and a conclusion. The conditional part is a conjunction of conditions. Conditions are tests that involve a particular attribute, operator (e.g. $=$, $\geq$, etc.) and a value from the domain of that attribute. These kinds of tests correspond, in the input space, to a hyper-plane that is orthogonal to the axes of the tested attribute and parallel to all other axis. The regions produced by these classifiers are all hyper-rectangles. Each leaf corresponds to a region. The regions are mutually exclusive and exhaustive (*i.e.* cover all the instance space).

The algorithm that builds a decision tree from data is quite simple. The main steps of the algorithm are described in figure 2.1.

It is known that the problem of building a minimal decision tree (in terms of number of nodes), consistent with a set of data, is a *NP hard* problem [Riv87]. Usually algorithms exploit heuristics that locally perform a one-step lookahead search. Once a decision is taken it never is reconsidered. This hill-climbing search without backtracking is

susceptible to the usual risks of converging to locally optimal solutions that are not globally optimal. On the other hand, this strategy allows to built decision trees in time *linear* to the number of examples.

In the next subsections will briefly describe the most important aspects of the algorithm. We focus on the two most successful and representative decision tree systems *CART* and *C4.5*.

2.2.1.1 Splitting rules

A splitting rule is guided by a measure of the "goodness of split", that indicates how well a given attribute discriminates the classes. It is used to select the attribute that maximizes this measure (see step 3 in figure 2.1). A splitting rule typically works as a one-step lookahead heuristic. For each possible test, the system hypothetically considers the subsets of data obtained. The system chooses the test that maximizes (or minimizes) some heuristic function over the subsets. A natural approach is to label each split subsets by their most frequent class and choose the split, which has fewest errors. There are several problems with this approach (see e.g. Breiman *et al.* [BFOS84]). One of them is that it simply has not worked well empirically.

Several methods have been described in the literature. Most of them agree in the extreme points, i.e. that a split which maintains the classes proportions in every subset has no utility, and a split in which each subset contains only examples of one class has maximum utility. Intermediate cases may be ranked differently by the different measures. Martin [Mar97] groups the measures into the following categories of the so called "merit functions":

1. Measures of the difference between the parent and the split subsets on some function of the class proportions (such as entropy). These measures emphasize the *purity* of the subsets. CART terms these *impurity* functions.

2. Measures of the difference between the split subsets on some function of the class proportions (typically a distance or an angle). These measures emphasize the *disparity* of the subsets.

3. Statistical measures of independence (typically a χ^2 test) between the class proportions and the split subsets. These measures emphasize the weight of the evidence, the *reliability* of class predictions based on subset membership.

The rest of this section explains the splitting rules used in *C4.5* and *CART*.

C4.5 Regarding a decision tree as an information source that conveys a message concerning the classification of an object and letting p and n denote objects of different classes, the expected information content of the message is

$$I(p,n) = -\frac{p}{p+n}log(\frac{p}{p+n}) - \frac{n}{p+n}log(\frac{n}{p+n}) \tag{2.1}$$

where the probability of each possible message is computed from the training set. If attribute A is selected, which has k different values, the resulting tree has an expected information content of

$$E(A,p,n) = \sum_{i=1}^{k} \frac{p_i + n_i}{p+n} I(p_i, n_i) \tag{2.2}$$

where p_i and n_i are the number of instances from each class in the subtree associated with the partition i based on the values of the attribute A. The information gain achieved is

$$gain(A,p,n) = I(p,n) - E(A,p,n) \tag{2.3}$$

The corresponding heuristic selects the attribute that results in the maximum gain for that step.

A test on a nominal attribute will divide the data into as many subsets as there are values of the attribute. A test on a continuous attribute will divide the data into two subsets: *attribute value* $>$ *cut point* and *attribute value* $\leq$ *cut point*. To obtain the *cut point* the values of the continuous attribute are first sorted. The mid point of two consecutive values is a possible cut point and is evaluated by the merit function. The possible cut point that maximizes the merit function is chosen.

This heuristic was shown to favor attributes with a large number of different values [Qui93, Mán91] and a normalization procedure was proposed. The *gain ratio* [Qui93] heuristic suggests that an attribute should be selected if it delivers good information gain, even when its actual information content is low. The *gain ratio* is defined as

$$gain\ ratio(A,p,n) = \frac{gain(A,p,n)}{IV(A,p,n)} \tag{2.4}$$

where $IV(A,p,n) = -\sum_{i=1}^{k} \frac{p_i+n_i}{p+n} log(\frac{p_i+n_i}{p+n})$ is the potential information obtained by splitting the data using the values of attribute A.

CART Breiman *et al.* [BFOS84] proposed the *GINI* function to measure the impurity as

$$i(t) = 1 - \sum_i p_i^2 \tag{2.5}$$

where p_i is the probability for each class. When an attribute is examined, the weighted average impurity of the implied descendant nodes is subtracted from $i(t)$ and the attribute that results in the largest impurity decrease is selected.

Brodley [Bro95a] presents an interesting method for automatic selection of split criterion during tree growing. She concludes:

> *Given the objective of maximizing predictive accuracy, test nodes near the root of the tree should be chosen using a measure based on information theory, whereas test nodes closer to the leaves of the pruned tree should be chosen to maximize classification accuracy on the training set.*

Mingers [Min89b] performs an empirical comparison between several splitting criteria. He concludes:

> *The results show that the choice of measure affects the size of a tree but not its accuracy which remains the same even when attributes are selected randomly.*

Later, several studies [BN92, EMS97] question the design methodology used by Mingers. In the cited studies, the conclusion is that random splitting leads to increased error, but there is no overall better splitting criteria.

2.2.1.2 Pruning strategies

Pruning is considered to be the most important part of the tree building process, at least in noisy domains. Noisy data gives rise to two problems. The first is that the induced tree classifies new objects in an unreliable way. Statistics computed at deeper nodes of a tree have low level of significance due to the small number of examples that fall at these nodes. Deeper nodes reflect the training set too much (overfitting) and increase the error due to the variance of the classifier. The second is that the induced tree tends to be large and thus hard to understand. Tree pruning, that is replacing deeper nodes by leaves, helps to minimize these problems.

Pruning a decision tree will almost certainly cause it to misclassify some examples from the training set. The advantage of pruning becomes apparent when classifying new examples not used in the process of growing the tree. Pruning leads to lower generalization error in general. The pruning methods can be divided into two main groups. First, methods that stop building the tree when some criteria is satisfied, referred as *pre-pruning*, and second, methods that built a full tree and prune it back, referred as *post-pruning*. All the methods make a *trade-off* between the size of the tree and an estimate of the error rate. As for the estimates of error rate we can differentiate between methods that estimate it from the *training set*, usually known as *resubstitution error*, and methods that use a separate *pruning set*, not used when growing the tree, to estimate the error rate.

Pre-pruning Pre-pruning relies on stopping rules that prevent the growth of those branches that do not seem to improve the predictive accuracy of the tree. It has the advantage that time is not wasted growing a structure that is not used in the final tree. In the algorithm of figure 2.1 the stopping condition is tested in step 1. Esposito *et al.* [EMS97] present several commonly used stopping rules:

1. All observations reaching a node belong to the same class.
2. All observations reaching a node have the same feature vector (but not necessarily belong to the same class).
3. The number of observations in a node is less than a certain threshold.
4. There is no rejection for statistical tests on the independence between a feature x_j and the class attribute.
5. The merit attributed to all possible tests which partition the set of observations in the node is too low.

Rules 1 and 2 are universally accepted. The other rules have been criticized because threshold tests quite often terminate the splitting procedure prematurely. It may be that all the attributes seem individually meaningless but some combinations of them may be highly discriminant [EMS93b]. Breiman *et al.* [BFOS84] also point out that such stopping rules are not easy to get right: too high threshold can terminate the splitting before its benefits become evident, while too low thresholds results in large trees and poor generalization error.

For this reason we do not explore more pre-pruning, although in the literature there are too many termination criteria by setting a non-trivial stop criteria.

Post-pruning This is the most common method of pruning decision trees. It has been exhaustively described in literature [BFOS84, Qui93]. A full tree is generated, overfiting the data, and then pruned back. Quinlan [Qui88] points out that *"Growing and pruning trees is slower but more reliable."*. The key decision in post-pruning is whether to prune a subtree or not. One of the simplest methods [Bra84] is based on two measures: the *static error* and the *backed-up error*. *Static error* is the number of misclassifications considering that all the examples that fall at this node are classified using the majority class of the classes distribution at this node. *Backed-up error* is the sum of misclassifications of all subtrees of the current node. If *backed-up error* is greater or equal than *static error*, then the node is replaced by a leaf with the majority class of the node.

Minimal cost complexity pruning of Breiman *et al.* [BFOS84] is based on two parameters for a tree T – the misclassification error, $R(T)$, and the size of the tree, $|T|$ measured in terms of leaves. The *cost-complexity* measure for the tree is

$$R_\alpha(T) = R(T) + \alpha|T|$$

where α is a parameter that weighs the relative importance of the tree size to the error rate. For each value of α, the objective is to find that subtree $T_\alpha \leq T$ which minimizes $R_\alpha(T)$. If α is small, the penalty for having a large number of terminal nodes is small and T_α will be large. As α increases the minimized subtree will have fewer terminal nodes. Although α runs through a continuum of values, the number of subtrees of T is finite. Thus the pruning process produces a finite sequence of nested subtrees $T_1, T_2, T_3, ...$ with progressively fewer terminal nodes. The pruned tree selected is the one that minimizes $R_\alpha(T)$ in the sequence of subtrees. The CART book [BFOS84] presents an efficient algorithm for implementing this pruning process.

The *reduced error* pruning [Qui88] examines every internal node in the decision tree and replaces it by the best possible leaf, if the number of misclassifications does not increase. The nodes are examined successively using a bottom-up strategy. The method uses a separate pruning set to estimate the error rate. It is guaranteed to deliver a small and accurate tree with respect to the pruning set. It is prone, however, to pruning branches that correspond to rare cases [Min89a].

The *pessimistic* pruning [Qui93] estimates for each subtree an error bound based on the proportion of the training examples misclassified at the leaves. It uses the same training set to grow and prune a tree. The *apparent* error rate, that is the error rate on the training set, is *optimistically* biased and does not provide the best criterion for choosing the best pruned tree. For this reason Quinlan introduces the continuity correction based on the binomial distribution of errors. A leaf replaces a subtree when the error rate for the subtree is not *significantly lower* than the error of the leaf. As pessimistic pruning uses a top-down strategy and does not require a separate pruning set, its main advantage is speed.

The *error-based* pruning is the method used in C4.5 [Qui88]. It uses the information of the training set for building and simplifying the tree using a bottom-up post-order traversal strategy. A novelty in this strategy is the ability of *grafting* a branch onto the place of the parent tree, in addition to pruning nodes. Taking the set of examples covered by a leaf as a statistical sample, it is possible to estimate a confidence interval $[L_{CF}, U_{CF}]$ relative to the probability of misclassification of the leaf. The upper limit of the confidence interval, U_{CF}, is of particular interest for the worst-case analysis. Under the assumption that errors in the training set are binomially distributed with probability p in n trials, it is possible to compute the value of U_{CF} as the value for which a binomially distributed variable X shows $\#errors$ successes in $\#examples$ trials with probability CF, a user provided confidence level. Having found the upper limit, the error estimates for leaves and subtrees are computed assuming that they will be used to classify a set of unseen cases of the same size as the training set. Thus, the predicted error rate for a leaf will be $\#examples \times U_{CF}$. The sum of the predicted error rates of all leaves in a branch is considered to be an estimate of the error rate of the branch itself. Thus, by comparing the predicted error rate for a given node, as if it was replaced by a leaf, with the subtree rooted at this node we can decide whether it is convenient to prune or to keep the node. Moreover, comparing with the predict error of the sub-branch rooted at a child of this node, we can decide to graft the sub-branch.

There are two strong assumptions underlying this pruning method. It is hard to accept that the examples of the training set covered by a node represent a *statistical sample*, since the tree has been built to fit the data as well as possible. According to Esposito *et al.* [EMS97] the assumption that errors in the sample have a *binomial* distribution is even more questionable.

Mingers [Min89a] performs an empirical comparison of five pruning methods for decision tree induction. He concludes that *"The results show that two methods– cost*

complexity and reduced error–perform well, while the other two may cause problems. They also show that there is no significant interaction between the method used to generate a tree and pruning methods." Esposito *et al.* [EMS93b] claim and justify that the experimental methodology used by Mingers is fair. As with the splitting rules, pruning is a domain where no approach exists that is best over-all. Pruning is a *bias* towards simplicity. If the problem domain admits simple solutions then pruning is a sensible option [Sch93].

2.2.1.3 Missing values

When using a tree as a classifier, the example to be classified passes down through the tree. At each node a test based on the values of the attributes is performed. If the value of the tested attribute is not known (often in real data some attribute values are unknown or undetermined) the procedure can not determine the path to follow. Since a decision tree constitutes a hierarchy of tests, the *unknown value problem* has special relevance in this type of classifiers.

Several solutions have been proposed in the literature. Quinlan [Qui86] has examined some of them. The most common are:

- A simple strategy consists of replacing the unknown value with the most common value for the attribute found in the training set.
- Another strategy consists of considering the unknown value as another possible value of the attribute [KSD97]. When building the tree, each decision node can contain a branch for the case where the tested attribute takes an unknown value.
- C4.5 [Qui93] uses a more complex strategy. It assigns a probability to each of the possible values of the attribute. The probabilities are estimated on the basis of the observed frequencies of the possible values for the attribute in the examples at the current node. These fractional values are used to compute the *information gain* and can be further subdivided at subsequent branches of the tree if another missing attribute value is to be tested. When classifying a query example, C4.5 passes the example through all branches where the unknown attribute value was detected. Each branch outputs a vote for a class. The overall output is computed as the majority class of all branch outputs.
- CART [BFOS84] uses a more sophisticated strategy known as *surrogate splits*. Instead of storing, at each node, only the attribute that minimizes the impurity

function, CART stores a ranking of attributes that produce a similar split. When classifying an example, if the value of the best attribute is unknown, CART will look for the value of the attribute with a known value following the order given by the ranking.

2.2.1.4 Characteristics of Decision Trees

Decision trees have a number of advantages. They are one of the most commonly used algorithms, on both in real world applications and in academic research. Some of the positive points referred to in the literature are:

1. *Flexibility*
   Decision trees do not assume any distribution for the data. They are non-parametric methods. The instance space is split into subspaces and each subspace is fitted with different models. A decision tree provides an exhaustive coverage of the instance space. It can approximate the *optimal Bayes error* of any function if there are enough data.

2. *Robustness*
   Univariate trees are invariant under all (strictly) monotone transformations of the individual input variables. For example using x_j, $log\ x_j$, or e^{x_j} as the jth input variable yields the same structural result. There is no need to consider input variable transformations. As a consequence of this invariance, sensitivity to long tail distributions and outliers is also reduced [Fri99].

3. *Robustness (2)*
   The internal feature selection produces trees that tend to be quite robust against the addition of irrelevant input variables.

4. *Interpretability*
   Global and complex decisions can be approximated by a series of simpler and local decisions. All decisions are based on the values of the attributes used to describe the problem. Both aspects contribute to the popularity of decision trees.

5. *Speed*
   Decision tree is a greedy algorithm that grows top-down using a divide-and-conquer strategy without backtracking. The *time complexity* is linear with the number of examples.

In the machine learning literature, the most commonly referred to problems are:

1. *Fragmentation*
   It causes partitioning of the data into smaller subsets. Supposing that data splits equally on every split, then a decision tree cannot test more than $O(log\ n)$ attributes. This can be undesirable, for example, on tasks with many relevant attributes.

2. *Replication*
   The term refers to the duplication of a sequence of tests in different branches of a decision tree, leading to a inconcise representation that also tends to have low predictive accuracy. For example, to represent the concept $(A \wedge B) \vee (C \wedge D)$ the subtree, representing either $(A \wedge B) or (C \wedge D)$ must be duplicated. Pagallo *et al.* [PH90] argue that *replication* is inherent to the decision tree representation rather than to the concept itself. Rendel and Cho [RC90] identify a number of concept characteristics that affect the accuracy and conciseness of machine learning systems in general and decision trees in particular. Some of the more important characteristics are *concept dispersion* in the instance space, relative *size* of the concept, and *embedded parity* problems.

3. *Missing values*
   A decision tree is a hierarchy of tests. If the value of an attribute is unknown this causes problems in deciding which branch to follow. Algorithms must employ special mechanisms to handle missing values. Friedman maintains that *"About half the code in CART and 80% of the programming effort went into missing values!"* [FKY96].

4. *Continuous attributes*
   The bottleneck of the algorithm is the presence of *continuous* attributes. In this case a *sort* operation is required for each *continuous* attribute at each decision node. Some authors [Cat91] estimate that the sort operation takes around 70% of the time needed to induce a decision tree on large datasets with many continuous attributes. Due to this observation, some researchers [Cat91, FI93, GTS98] have examined the possibility of discretization of continuous attributes.

5. *Instability*
   Several researchers, namely Breiman [BFOS84, Bre96a] and Kohavi [KK97] have pointed out that small variations in the training set can produce large variations in the final tree. At each node the split-merit criterion ranks the attributes,

and the best attribute is chosen to split the data. If two or more attributes are ranked similarly, small variations of the training data can alter the ranking. All the subtrees below this node change. The recursive partitioning strategy implies that as each split is made, the data is split based on the test attribute. After some splits, there is usually very little data on which to base decisions. There is a strong tendency for inferences made near the leaves to be less reliable than those made near the root.

In the next section we present several extensions that have been proposed in the machine learning community to deal with the referred problems.

2.2.2 Extensions

2.2.2.1 Multivariate trees

The standard algorithm for building decision trees uses at each node a test based on one attribute and one of its possible values. The resulting trees are called *univariate trees.* In this section we examine methods that can use tests based on linear combination of attributes, that is, decisions are based on the values of a set of variables. This kind of trees are denoted as *oblique decision trees* or *multivariate trees.* There is a trade-off to consider though. Simple, univariate tests may result in large trees that are difficult to understand. Multivariate tests are more difficult to understand, but usually produce simpler, more compact trees.

In this section we only review the work concerned with building linear combinations of ordered attributes. The work of Pagallo and Haussler [PH90] and Zheng [Zhe96] referred to earlier (section 1.3.2) is oriented towards building logical combinations of boolean attributes. Also, higher degree polynomials could be considered here. Nevertheless most authors prefer to consider only linear combinations mainly due to their simplicity.

Breiman *et al.* [BFOS84] suggest the use of linear combination of attributes instead of using a single attribute. They affirm that:

> *In some data classes are naturally separated by hyper-planes not perpendicular to the coordinate axes. These problems are difficult for the unmodified tree structure procedure and result in large trees as the algorithm attempts to approximate the hyper-planes by multidimensional rectangular regions.*

A multivariate test can be represented as a *linear threshold unit* for two class problems or as a *linear machine* for the general case of q classes [DH73, Lan96]. A linear threshold unit (LTU) $\mathbf{W}$ is a vector of $n+1$ coefficients also known as weights. Given an instance Y, consisting of a constant 1 and n attribute-values, and a LTU W, the decision rule is: *if* $W^TY \geq 0$ *then* Y *belongs to class* $+$, *otherwise class* $-$. A linear machine (LM) is a set of R LTU's that are used collectively to assign an instance to one of the R classes. Given an instance Y, consisting of a constant 1 and n attribute-values, and a LM W, the decision rule is: Y *belongs to class* i *such that* W_i^TY *is maximized.*

In this dissertation we refer to two pertinent problems of *multivariate trees*:

1. Methods to find the coefficients of a multivariate test.
2. Methods to simplify the linear combinations.

Finding the coefficients of a multivariate test. Most optimization problems can be seen as a problem of finding the free variables (parameters) in a model given a set of data. This problem has been exhaustively studied in the *neural network* community. Ripley [Rip96] and Mitchell [Mit97] give overviews of this work. In this section we review the work done in the machine learning community.

- *The Recursive Least Squares Method - (RLS)*
  minimizes the mean-squared error $\sum_i (y_i - \hat{y}_i)^2$ over the training data. To find the coefficients of the linear function the RLS algorithm incrementally updates the coefficients using the error between the estimated value of the dependent variable $\hat{y}_i$ and the true value y_i. After instance k has been observed the weight vector $\mathbf{W}$ is updated as follows:

  $$W_k = W_{k-1} - K_k(X_k^T W_{k-1} - y_k)$$

  where X_k is the instance vector, y_k is the value of the dependent variable, and $K_k = P_k X_k$ is the weight vector assigned to the update. P represents the error-covariance matrix, a weighting matrix of size $n * n$, which over time decreases and smoothes the errors made by the linear combination. After each instance k is observed, P is updated as follows:

  $$P_k = P_{k-1} - P_{k-1}X_k[1 + X_k^T P_{k-1} X_k]^{-1} X_k^T P_{k-1}$$

As more instances are observed, each individual instance has less effect on **W**. This is reflected in the decreasing values of the entries of the matrix P.

- *The Absolute Error Correction Rule*
  is the most common method to determine the coefficients for a linear machine [DH73]. This is an incremental algorithm that iteratively cycles through all the instances. In each iteration each example is classified using the actual set of weights. If the example is misclassified the weights are updated. Supposing that the example belongs to class i and is classified as j with $i \neq j$, then the weights W_i and W_j must be corrected. The correction is accomplished by $W_i \leftarrow W_i + kY$ and $W_j \leftarrow W_j - kY$, where the correction

$$k = \frac{(W_j - W_i)^T Y}{2Y^T Y} + \epsilon \tag{2.6}$$

  causes the updated linear machine to classify the instance correctly (ϵ is a small constant). If the instances are linearly separable, then cycling through the instances allows the linear machine to partition the instances into separate convex regions. If the instances are not linearly separable, then the error corrections will not cease, and the classification accuracy of the linear machine is unpredictable. To deal with this situation two variants often referred in the literature are:

  1. *Pocket Algorithm*
     maximizes the number of correct classifications on the training data. It stores in **P** the best weight vector **W**, as measured by the longest run of consecutive correct classifications, called the *pocket count*. A LTU based on the pocket vector **P** is optimal in the sense that no other weight vector visited so far is likely to be more accurate classifier. This is the method to obtain linear combinations used in the system *PT2* [UB90], an incremental multivariate tree.

  2. *Thermal Training*
     converges to a set of coefficients by paying decreasing attention to large errors. The correction is accomplished by $W_i \leftarrow W_i + cY$ and $W_j \leftarrow W_j - cY$, where the correction factor is $c = \frac{\beta^2}{\beta + k}$ where β is annealed during training and k is given by equation 2.6. The training algorithm repeatedly presents examples at each node until the linear machine converges. This is the method used on *Linear Machine Decision Tree (LMDT)* of Brodley and Utgoff [UB91, BU95]. Because convergence cannot be guaranteed, *LMDT* uses heuristics to determine when the node has stabilized.

Brodley and Utgoff [BU95] have reviewed and evaluated several methods for learning the coefficients of tests involving linear combinations. Their study includes the methods referred here. From their empirical evaluation they have concluded that, considering the datasets in the study, the *Recursive Least Squares* achieves the best accuracy on an independent test set and produces the smallest trees when compared to the other methods.

- *Splits based on linear combinations in CART*
  CART searches explicitly for a set of coefficients that minimize the split-merit criterion defined by the user. CART only considers *ordered* attributes, not *nominal* attributes, when searching for a linear attribute combination. Only instances for which no values are missing are used for training. Centering each value of each feature at its median and then dividing by its range normalizes each instance. The algorithm incorporated in CART cycles through the attributes at each step, searching for an improved split using a linear combination. The algorithm takes a set of coefficients $W = (w_1, ..., w_n)$ such that $||W||^2 = \sum_{i=1}^{n} w_i^2 = 1$ and searches for the best split of the form $v = \sum_{i=1}^{n} w_i x_i \leq c$. The search algorithm cycles through the features, $x_1, ..., x_n$ and at each step searches for an improved split using a linear combination. Suppose that at the beginning of the L^{th} cycle the current linear combination split is $v \leq c$. For fixed γ, CART searches for the best split of the form $v - \delta(x_1 + \gamma) \leq c$ such that

$$\delta \geq \frac{v-c}{x_1+\gamma}, x_1 + \gamma \geq 0 \text{ and } \delta \leq \frac{v-c}{x_1+\gamma}, x_1 + \gamma \leq 0$$

  The search for δ is carried out for $\gamma = -0.25, 0.0, +0.25$. The resulting three splits are compared using the chosen split-merit criterion, and the γ and δ corresponding to the best split are used to update v, producing

$$v' = \sum_{i=1}^{n} w_i' x_i$$
$$\text{where } w_1' = w_1 - \delta,\ w_i' = w_i, i > 1 \text{ and } c' = c + \delta\gamma.$$

  This search is repeated for each $x_i, i = 2, ..., n$, resulting in an update split $v_L \leq c_L$. The aim of the final step of the cycle is to find the best c_L. The system searches explicitly for the split that minimizes the impurity of the resulting partition. The result of this search is used to start the next cycle. The search for the coefficients continues until the reduction in the impurity, as measured by the split-merit criterion, is less than some small threshold ε. After the final linear combination has been determined, it is converted to a split test on the original non-normalized features.

Similar to *CART*, *OC1* [MKS94], uses a hill-climbing search algorithm. It begins with the best axes-parallel split, it randomly perturbs each of the coefficients until there is no improvement in the impurity of the hyper-plane. As in *CART*, *OC1* adjusts the coefficients of the hyper-plane individually finding a local optimal value for each coefficient at a time. The innovation of OC1 is a randomization step used to get out of local minima.

Other methods, which are not so well known, have been described in the literature. The FACT system [LV88] uses a recursive partition of the input space using linear discriminant functions. The number of descendents of each node is equal to the number of classes. FACT system has results similar to CART, but achieves this with lower running times. This system has been recently improved, resulting in system QUEST [LS97]. QUEST uses the *Cost complexity pruning* of CART, instead of *pre-pruning* of FACT. It generates binary trees. In domains with more than two classes, a 2-means clustering algorithm is applied. The classes are grouped into two super-classes before the application of discriminant analysis.

Also *linear programming* has been used to compute the coefficients of each multivariate test. Mangasarian [MSW90, BM94] described one of such systems with promising results. George John [Joh97] used a function optimization technique based on steepest descendent. It is used to minimize a variant of the entropy criterion, based on weighted sets, and denoted by *soft entropy*.

Simplifying the coefficients of a multivariate test. Almost all the methods that build linear combinations use *all the ordered variables*. However some of them may contribute very little to the effectiveness of the split, and so, they can be removed. The advantage of using a small number of variables is that interpretability will increase.

As for the methods for simplifying the linear equations, the literature is abundant. The most common methods are [DG84, BU95, LM98]:

1. *Sequential Backward Elimination (SBE)*, begins with all the features, each time removing the feature that contributes least to the performance of the linear combination.

2. *Sequential Forward Selection (SFS)*, begins with an empty set, each time adding the feature that contributes most to the performance of the linear combination.

3. *Heuristic Sequential Search (HSS)* automatically chooses between SBE and SFS. HSS first evaluates a linear combination based on the complete set of features, and compares the result with the best result based on one feature. If the latter is better than the former, the system performs a SFS, otherwise a SBS is chosen.

Some authors claim that is preferable to begin with a complete but informed solution than to start from a empty but uninformed solution. This is, for example, the option taken in *CART* that performs a *Sequential Backward Elimination.* It starts with all the features in the linear combination while trying to remove the feature that will cause the smallest decrease of the split-merit criterion. It continues removing features until a stopping criterion is met.

Brodley and Utgoff [BU95] and Liu and Motoda [LM98] presented a more detailed discussion on the theme.

Linear combinations of attributes in a preprocessing phase. Reconstruction of the input space by means of new attributes, defined as combinations of original ones, appears as a *preprocessing* step in Yip and Webb [YW94]. Their system *CAF* incorporates new attributes based on canonical discriminant analysis. They show that such techniques can improve the performance of machine learning algorithms.

Ittner and Schlosser [IS96] introduce, in a preprocessing step, all the pairwise products and squares of the numerical primitive attributes. The inductive step is done by *OC1.* The constructive preprocessing step allows *OC1* to build non linear decision surfaces. There are two weak points in this approach. The first is that the number of new attributes grows exponentially with the number of primitive attributes. The second is that the type of functions used (products) is arbitrary.

A comparison between multivariate decision trees. With respect to the number of hyper planes, *OC1* uses a similar strategy as *CART.* It searches, at each node, for only one hyper plane, regardless the number of classes. The search is guided by the aim to minimize the impurity function, that is also the attribute selection criterion. *CART* like *OC1* generates binary trees. In *LMDT* and *FACT*, the number of hyper planes is equal to the number of classes, but the trees generated by those systems are not binary. The number of descendants of each decision node is equal to the number of classes that fall at this node.

All those systems normalize the feature vector. This is done to minimize the influence

of the variance of each variable to the magnitude of the corresponding weight used in the linear combinations. For example, *LMDT* uses the standard normal form, that is, with the mean of zero and standard deviation of one. With respect to *missing values*, *OC1* replace each one, before learning, by the mean of the corresponding attribute values, while *CART* and *LMDT* ignore the instances with missing values when training the *linear machines*. With respect to *pruning*, *OC1*, *CART*, and *LMDT* use an *error complexity pruning* using a separate pruning set. The exception is system *FACT* that uses pre-pruning. *CART*, *OC1* and *LMDT* use a gradient descent approach for determining the hyper-planes, while *FACT* uses an analytical process. This aspect is relevant for learning time and explains the good performance of *FACT* system.

Further on, in section 4.5 we provide results of an empirical evaluation of these systems.

2.2.2.2 Option trees

Usually decision trees incorporate a single test at each node and follow a single path corresponding to test outcomes until a leaf is reached and a prediction is made. *Option decision trees* [Bun90, KK97] were introduced by Buntine as a generalization of decision trees. Option trees can include *option nodes*, which replace a single decision by a set of decisions. An option node is like an *or* node in and-or trees. Instead of selecting the 'best' attribute, all the promising attributes are selected. For each selected attribute a decision tree is constructed. Note that an option tree can have three types of nodes: Nodes with only one test attribute - *decision nodes*; nodes with disjunctions of test attributes - *option nodes*, and leaf nodes.

Classification of an instance x using an option tree is a recursive procedure:

- For a leaf node, return the class label predicted by the leaf.
- For a decision node the example follows the unique child that matches the test outcome for instance x at the node.
- For an option node the instance follows all the subtrees linked to the test attributes. The predictions of the disjunctive test are aggregated by a voting schema.

The main disadvantage of *option trees* is that they require much more memory and disk space than ordinary trees. Kohavi *et al.* [KK97] claim that it is possible to achieve

significant reduction of the error rates (in comparison with regular trees) using *option trees* restricted to two levels of option nodes at the top.

2.2.2.3 Incremental decision trees

The process of building a decision tree described in previous sections could be regarded as a *batch* process. Given a particular set of examples the greedy tree construction process generates the decision tree. This is a very fast process but has one drawback. If new data becomes available one need to construct a new tree from scratch. This is why researchers have studied the possibility of building the decisions structures incrementally. In this framework, given an example and a decision structure, the system updates the decision structure to assimilate the example.

This ability has philosophical implications, because it is one of paradigms of the human learning. For example, Michalski's [Mic90] definition of learning is *"self-constructing or self-modifying representations of what is being experienced for possible future use."*.

Several authors [Utg89, KM96, UBC97] have developed incremental versions of decision tree algorithms. They all share the idea of maintaining at each node statistics about the attribute-values of the examples that reached that node, without retaining these past examples. The difficult problem of incremental induction of decision trees, is the need of *tree revision* when incorporating new examples. Suppose that at a decision node a test on attribute A_i has been chosen. After a few more examples have been analyzed, the system can conclude, on the basis of the statistics at this node, that another attribute should have been chosen instead. The distribution of the examples that pass down each branch can change and all the sub-tree further down this node need to be reconsidered. Utgoff *et al.* [UBC97] present two operators that efficiently perform a tree revision:

- *Pull-up* operator
  that swaps the attributes, from the leaves to the upper decision nodes, in consecutive levels of the decision tree.
- *Recursive tree transposition* operator
  that revises the descendent nodes in order to ensure that the desirable test is chosen.

In the same article the authors present a more detailed discussion concerning the incremental decision tree induction.

2.2.2.4 Decision rules

Large decision trees are difficult to understand because each node appears in a specific context established by the outcomes of tests at the antecedent nodes. The work of Rivest [Riv87] presents a new representation, *decision lists*, that generalizes decision trees. The advantage of this representation is *modularity* and consequently *interpretability.*

Generating Decision Lists Several algorithms appear in literature for building decision lists [Riv87, CN89, Coh95, Dom96, WI98]. In this dissertation we restrict to those approaches that build decision lists from decision trees, as it is done in [Qui93].

In any decision tree when a case reaches a leaf, the conditions that must be satisfied appear along the path from the root to the leaf. So, any tree can be easily transformed into a collection of rules. Each rule corresponds to the path from the root to a leaf. There are as many rules as leaves.

The process described generates a set of rules with the same complexity as the decision tree. However, it can be shown that the antecedents of individual rules may contain irrelevant conditions. C4.5rules [Qui93] uses an optimization procedure to simplify conditions. The optimization is done in two phases. First, each rule is generalized by deleting conditions that do not seem to be helpful in discriminating the classes. A greedy search method is used. At each step the rule is evaluated as if one condition was dropped. The condition that produces the lowest increase of the pessimistic estimate of the error rate of the rule is eliminated. The pessimistic error rate is estimated similarly to the process discussed in the pruning phase of C4.5. After generalizing individual rules some of them may become identical to others, and so the duplicates are removed. Also, some of them may have no conditional part. They are also removed. In the second phase, all rules are grouped by the predicted class. For each class, the set of rules is simplified by removing rules whose removal does not diminish the accuracy of the complete set. Empirical studies [Qui93] have shown that the set of rules is both simpler and more accurate than the initial tree.

Frank and Witten [FW98] present a method for generating rules from decision trees without using global optimization. The basic idea is to generate a decision tree, select the best rule, remove the examples covered by the rule and iteratively induce further rules for the remaining instances.

2.2.3 Discriminant Functions

Earlier work in the area of pattern recognition literature [DH73] has been concerned with classification. Most of the work is formulated in a probabilistic setting. Let $P(C_i|\vec{x})$ denote the probability that example $\vec{x}$ belongs to class i. The zero-one loss is minimized if and only if, $\vec{x}$ is assigned to the class C_k for which $P(C_k|\vec{x})$ is maximum [DH73]. Formally, the class attached to example $\vec{x}$ is given by the expression:

$$argmax_i \; P(C_i|\vec{x}) \tag{2.7}$$

Any function that computes the conditional probabilities $P(C_i|\vec{x})$ is denoted by *discriminant function* [DH73]. Given an example $\vec{x}$, the Bayes theorem provides a method to compute $P(C_i|\vec{x})$:

$$P(C_i|\vec{x}) = \frac{P(C_i)P(\vec{x}|C_i)}{P(\vec{x})} \tag{2.8}$$

$P(\vec{x})$ can be ignored, since it is the same for all the classes, and does not affect the relative values of their probabilities. Although this rule is optimal, its applicability is reduced due to the large number of conditional probabilities required to compute $P(\vec{x}|C_i)$. To overcome this problem several assumptions are usually made.

Depending on the assumptions made we get different discriminant functions leading to different classifiers. In this work we study two sets of discriminant functions, leading to two different types of classifiers: *naive Bayes* classifier and the *discriminant* classifiers. As for the second type, here we analyze the *linear discriminant*, the *quadratic discriminant*, and the *logistic discriminant*.

2.2.3.1 Naive Bayes classifier

Assuming that the attributes are independent given the class, $P(\vec{x}|C_i)$ can be decomposed into the product $P(x_1|C_i) * ... * P(x_n|C_i)$. Then, the probability that an example belongs to class i is proportional to:

$$P(C_i|\vec{x}) = P(C_i) \prod_j P(x_j|C_i) \tag{2.9}$$

The classifier obtained by using the discriminant function 2.9 together with the decision rule 2.7 is known as the *naive Bayes Classifier*. The term *naive* comes from the assumption that the attributes are independent given the class.

Implementation details. All the required probabilities are computed from the training data. To compute the prior probability of observing class i, $P(C_i)$, a counter is required for each class. To compute the conditional probability of observing a particular attribute-value given that the example belongs to class i, $P(x_j|C_i)$, we need to distinguish between nominal attributes, and continuous ones. In the case of nominal attributes, the set of possible values is an enumerable set. To compute the conditional probability we only need to maintain a counter for each attribute-value and for each class. In the case of continuous attributes, the number of possible values is infinite. There are two possibilities. We can assume a particular distribution for the values of the attribute. Usually the normal distribution is assumed. As an alternative we can discretize each attribute in a pre-processing phase. The former has been observed to be inferior to the latter on many datasets [DKS95, DP97]. Several methods for discretization appear in the literature. Domingos [DP97] computes the number of intervals as $k = min(10;\ nr.\ of\ different\ values)$ equal width intervals. Once each particular attribute has been discretized, a counter for each class and for each interval is used to compute the corresponding conditional probability.

All the probabilities required by equation 2.9 can be computed from the training set in one step. The process of building the probabilistic description of the dataset is very fast. Another interesting aspect of the algorithm is that it is easy to implement in an incremental fashion because only counters are used and these are easy to update.

Analysis of the algorithm. Domingos and Pazzani [DP97] have shown that naive Bayes classifier has a surprisingly good performance in a wide variety of domains, including many where there are clear dependencies between attributes. They argue that the naive Bayes classifier is still optimal when the independence assumption is violated, as long as the ranks of the conditional probabilities of classes for a given example are correct. Some authors [Kon91, Lan93a] maintain that this classifier is robust to noise and to irrelevant attributes. Also they refer that the learned theories are easy to understand by domain experts, mostly due to the fact that the naive Bayes summarizes the variability of the dataset by a single probabilistic description, and assumes that these are sufficient to distinguish between classes.

Improvements. A few techniques have been developed to improve the performance of the naive Bayes classifier. Some of them are:

1. The semi-naive Bayes classifier [Kon91]. It attempted to join pairs of attributes,

making a cross-product attribute, based on statistical tests for independence. The experimental results were disappointing.

2. The recursive naive Bayes [Lan93b]. This algorithm recursively constructs a hierarchy of probabilistic concept descriptions. The author concludes that *"the results are mixed and not conclusive, but they are encouraging enough to recommend closer examination."*.

3. The naive Bayes tree [Koh96]. It is a hybrid algorithm. It generates a regular univariate decision tree, but the leaves contain a naive Bayes classifier built from the examples that fall at this node. The approach retains the interpretability of naive Bayes and decision trees, while resulting in classifiers that frequently outperform both constituents, especially in large datasets.

4. The *Flexible Bayes* of George John [Joh97] that uses, for continuous attributes, a kernel density estimation (instead of the single Gaussian assumption) but retains the independence assumption. The estimated density is averaged over a large set of kernels:

$$P(\vec{x}|C_i) = \frac{1}{nh} \sum_i K(\frac{x-\mu_i}{h})$$

where h is the bandwidth parameter and K the kernel shape $K = g(x, 0, 1)$. Empirical tests on UCI datasets, have shown that flexible Bayes achieves significantly higher accuracy than simple Bayes in many domains.

2.2.3.2 Linear discriminant classifier

A linear discriminant function is a linear composition of the given attributes, where the sum of squared differences between class means is maximal relative to the internal class variance. To minimize the number of misclassifications, the Bayes criterion assigns an example to the most probable class. Equations 2.7 and 2.8 can be rewritten as:

$$argmax_i \; P(C_i)P(\vec{x}|C_i) \tag{2.10}$$

Assuming that the attribute vectors for examples of class C_i are independent and follow a certain probability density function f_i, a new point $\vec{x}$ is then assigned to that class for which the probability density function $f_i(x)$ is greatest. One common assumption is that the distributions are *normal* with different means but the same

covariant matrix for all the classes [DG84, MST94]. The probability density function of the normal distribution is:

$$N(\mu, \Sigma) = \frac{1}{\sqrt{2\pi|\Sigma|}} exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)) \tag{2.11}$$

where μ is a vector denoting the (theoretical) mean for a class and Σ is the (theoretical) covariance matrix. The covariance matrix can be thought as a set of parameters for the distribution [MST94]. This means that the points for each class are distributed in a cluster centered at μ_i of ellipsoid shape described by Σ [MST94]. In this case the boundary separating two classes is a hyper-plane and it passes through the mid-point of the two centers.

Suppose the prior probability of class C_i is $P(C_i)$ and that $f_i(x)$ is the probability density function of $\vec{x}$ relative to class C_i using the normal density function given in equation 2.11. The joint probability of observing class i and example x is

$$P(C_i) * f_i(x) \tag{2.12}$$

The logarithm of this probability is:

$$log(P(C_i)) - \frac{1}{2}(x-\mu_i)^T\Sigma^{-1}(x-\mu_i)$$

to within an additive constant. After some algebraic manipulations we obtain:

$$log(P(C_i)) - \frac{1}{2}x^T\Sigma^{-1}x + \frac{1}{2}x^T\Sigma^{-1}\mu_i + \frac{1}{2}\mu_i^T\Sigma^{-1}x - \frac{1}{2}\mu_i^T\Sigma^{-1}\mu_i$$

The term $x^T\Sigma^{-1}x$ does not depend on the class and can be ignored. Taking into account the symmetry of Σ we obtain [DG84, PF96, MST94]:

$$log(P(C_i)) - \frac{1}{2}\mu_i^T\Sigma^{-1}\mu_i + x^T\Sigma^{-1}\mu_i \tag{2.13}$$

To find the *a posteriori* class probabilities for a *query point*, the exponential is taken of the discriminant and the resulting quantities normalized to sum to unity. Thus the *a posteriori* class probabilities $P(C_i|\vec{x})$ are given by:

$$P(C_i|\vec{x}) = exp[log(P(C_i)) - \frac{1}{2}\mu_i^T\Sigma^{-1}\mu_i + x^T\Sigma^{-1}\mu_i] \tag{2.14}$$

Implementation details. In practice the population parameters μ_i and Σ are unknown and have to be estimated from the sample. The discriminant function is estimated using the sample estimates $\overline{x}_i$ and the pooled covariance matrix S_{pooled}.

The linear discriminant builds hyper-planes of the form

$$H_i = \alpha_i + x^T \beta_i \tag{2.15}$$

where

$$\beta_i = S_{pooled}^{-1} \bar{x}_i \quad and \quad \alpha_i = log(P(C_i)) - \frac{1}{2} \bar{x}_i^T S_{pooled}^{-1} \bar{x}_i \tag{2.16}$$

A common problem appears when the S_{pooled} matrix is singular or near singular. This can happen for example, if one attribute is constant in one class or along all the dataset. One solution to the singular matrix, is the use of *Singular Value Decomposition* (SVD) [PTVF92] to compute S^{-1}. SVD is numerically stable and is a tool for detecting sources of collinearity. This last aspect can be used as a method for reducing the features of each linear combination.

The number of hyper-planes can be reduced by one, normalizing the coefficients α and β of each hyper-plane by subtracting the corresponding coefficients of the last hyper-plane, as suggested in [MST94].

Analysis of the algorithm. Linear discriminant is a one step procedure, since it does not make a recursive partitioning of the input space. The advantage of this kind of system is its ability to generate decision surfaces with arbitrary slopes. Also it works often well even if multivariate normality is not satisfied [Rip96]. The linear discriminant does not necessarily minimize the classification error. In particular there is no guarantee that the hyper-planes separate the classes in the separable case, but a useful solution is obtained in both the separable and the non-separable case [SW97].

Linear discriminant is a parametric approach. It is assumed that classes can be separated by hyper-planes, and the problem is to determine the coefficients of the hyper-plane. Although parametric approaches are often viewed as forcing an arbitrary imposition of certain model assumptions to the data, discriminant functions are stable with respect to small variations on the training set [Bre96a]. This is due to the small number of free parameters: $(\#Classes - 1) \times (\#Attributes + 1)$.

2.2.3.3 Quadratic discriminant classifier

The quadratic discriminant is similar to the linear discriminant but does not assume equal covariance matrices for each class. In this case, the logarithm of equation 2.12 is rewritten as:

$$log(P(C_i)) - \frac{1}{2}log(|\Sigma_i|) - \frac{1}{2}(x - \mu_i)^T\Sigma_i^{-1}(x - \mu_i) \tag{2.17}$$

providing the quadratic discriminant for class C_i.

Given a query example, the value of the quadratic discriminant of each class is calculated for this example and the class with largest discriminant is chosen. To find the *a posteriori* class probabilities explicitly, the exponential is taken of the discriminant and the resulting quantities normalized to sum to unity. Thus the *a posteriori* class probabilities $P(C_i|\vec{x})$ are given by:

$$P(C_i|\vec{x}) = exp[log(P(C_i)) - \frac{1}{2}log(|\Sigma_i|) - \frac{1}{2}(x - \mu_i)^T\Sigma_i^{-1}(x - \mu_i)] \tag{2.18}$$

Implementation details. As in the previous case the population parameters μ_i and Σ_i are unknown and have to be estimated from the sample. The discriminant function is estimated using the sample estimates $\overline{x}_i$ and the covariance matrices S_i. As in the case of linear discriminant we use the Singular Value Decomposition [PTVF92] technique to compute S_i^{-1}.

Analysis of the algorithm. In [MST94] the authors state that quadratic discriminant is robust with respect to small departures from normality. The general form of the decision boundaries is not linear. Not only the attributes are weighed, but also their second powers and crossproducts, giving terms like $2a_1^2 + 7a_1a_2 + ...$ [WK91]. Geometrically, this represents a separating boundary that is a quadratic hyper-surface, rather than a hyper-plane. The quadratic discriminant does not assume equal covariance matrices as linear discriminant. However the number of parameters to be estimated becomes $\#Classes \times \#Attributes \times (\#Attributes + 1)/2$, that can be problematic especially for small or moderate sized datasets.

2.2.3.4 Logistic discriminant classifier

The *Logistic discriminant classifier* operates by determining a position of a hyper-plane to separate the classes as well as possible. However, the criterion for a good separation

is different from the one followed in the case of the linear discriminant classifier. The latter optimizes a quadratic cost function, whereas the former maximizes the conditional likelihood. The method is only partially parametric, as the probability density functions for the classes are not modeled, but rather the ratios between them are. More precisely, the logarithms of the ratios of the probability density functions for the classes, referred as *log odds*, are modeled as linear functions of the attributes. For a q class problem, one class is taken as reference class, and so there are $q-1$ sets of parameters for each class relative to the reference class.

Given an example $\vec{x}$ and a set of $q-1$ vectors β of coefficients, the conditional class probability for the reference class C_q and for class C_i $(i \neq q)$, take the form:

$$P(C_q|x) = \frac{1}{\sum_{j=1,...,q-1} exp(\beta_j \vec{x})} \tag{2.19}$$

$$P(C_i|x) = \frac{exp(\beta_i \vec{x})}{\sum_{j=1,...,q-1} exp(\beta_j \vec{x})} \tag{2.20}$$

Given independent samples from the q classes, the conditional likelihood for the set of vectors β is defined to be

$$L(\beta_1, ..., \beta_{q-1}) = \prod_{\vec{x} \in C_1} P(C_1|\vec{x}) \prod_{\vec{x} \in C_2} P(C_2|\vec{x}) ... \prod_{\vec{x} \in C_q} P(C_q|\vec{x}) \tag{2.21}$$

The parameter estimates are the values that maximize this likelihood [MST94].

Given a *query example* the $q-1$ vectors β of coefficients are evaluated for this example. The method assigns this example to the class for which the *log odd* is the greatest, or to the reference class if all the log odds are negative.

Implementation details. The optimization problem, whose objective is to find the parameter estimates that maximize the likelihood, is solved using an iterative procedure. The linear discriminant provides the initial set of parameters. They are refined using a gradient descent. The Newton-Raphson iterative procedure [PTVF92] is used because the derivatives of the likelihood are easy to calculate.

Analysis of the algorithm. The logistic discriminant and linear discriminant are very similar. The greatest differences between the two happen, as could be expected, when the attributes have very non-normal distributions with very dissimilar covariances [MST94]. In each iteration logistic discriminant requires to compute the

Hessian, that is a symmetric square matrix with $(\#Classes - 1) \times (\#Attributes + 1)$ rows. Each term requires a summation over all the examples. Thus there are $\#Classes^2 \#Attributes^2 N/2$ operations at each iteration. For large datasets this can be a very time consuming process.

Figure 2.2 illustrates the different decision boundaries drawn by linear, logistic and quadratic discriminants in a two class problem.

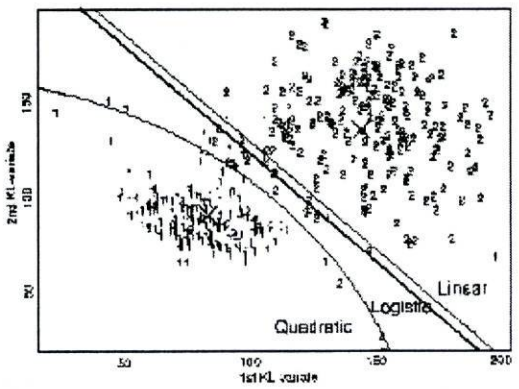


Figure 2.2: Decision surfaces generated by linear, logistic and quadratic discriminants on a two-class problem defined by two attributes.

In one of earlier sections (2.2.2.1) we have referred other methods that can be used to obtain the coefficients of linear combinations that also use an iterative gradient descent approach. There are two important points that differentiate those methods and the method used in logistic discriminant:

1. In logistic discriminant the update of the coefficients is done only after all the instances have been processed. The *Absolute Error Correction Rule* and the *Recursive Least Squares* update the coefficients just after one misclassified example has been found.

2. The *Absolute Error Correction Rule* minimizes the number of misclassifications. In *logistic discriminant*, one maximizes the likelihood of *a posteriori* probabilities. This means that although an instance is correctly classified, because the *a posteriori* probability of the observed class is the greatest, it can produce an update of the coefficients because this probability is not 1.

2.2.4 Comparison Between the Different Models

The main difference between the decision trees and the models generated by discriminant functions is that they use *different representation languages.* A decision tree is able to fit different regions of the instance space with different functions, and so it can approximate any function, if there are enough examples. A decision tree grows more or less linearly with the number of examples, and so does the number of degrees of freedom (decisions taken at each node). On the other hand the number of degrees of freedom of any discriminant function depends only on the number of attributes, the number of values of each attribute, and the number of classes. It does not depend on the number of examples.

Breiman [Bre96a] and Kohavi and Wolpert [KW96] have examined the behavior of the error of decision trees, linear discriminant, and naive Bayes from the point of view of *bias-variance* analysis. They show that while naive Bayes and linear discriminant are algorithms with *low variance* but *high bias*, decision trees have shown *high variance* but *low bias.* This aspect is developed further in the next chapter.

Decision trees and discriminant functions have different behavior when classifying one example. In the former the example follows a path from the root to a leaf. This path represents the common properties to all the examples that fall at this leaf. It represents a conjunction of conditions that is a *sufficient condition* that characterizes the class attached to the leaf. On the other hand the value of the discriminant function is computed for each class. The class attributed to the example maximizes this value.

Another important difference between the classifiers is related to how many attributes they use to classify a *query point.* Both naive Bayes and linear discriminant use all attributes to classify a new example, unless they incorporate special pre-processing procedures to eliminate irrelevant attributes. *Decision trees* often use only a subset of the available attributes. Quinlan [Qui94] distinguishes between parallel and sequential classification problems. In the former all (or almost all) attributes are relevant for classification, whereas in the latter case the class of an example depends only on a few attributes and the relevance of some attributes strongly depends on the values of other attributes. Quinlan conjectures, and shows some evidence, that a decision tree algorithm is more appropriate for sequential problems and neural nets or discriminant functions for parallel tasks.

2.3 Summary

In this chapter we have presented the most representative methods of inductive learning that we have used throughout this dissertation. We have analyzed *decision trees*, *naive Bayes*, *linear*, *quadratic*, and *logistic* discriminant functions. The positive and negative aspects of these methods have been analyzed. The current trends concerning new developments in the area have been presented. In the next chapter we discuss a related and very important point in machine learning – methods for evaluation of learning algorithms.

Chapter 3

Methods for Evaluation Learning Algorithms

Learning algorithms can be evaluated along several dimensions. In this dissertation we consider the error-rate, learning times, and theory compression. The methodology for estimating the generalization error are discussed together with statistical tests of significance. The bias-variance decomposition is presented. The final part of this chapter is dedicated to evaluate a set of classifiers used along this dissertation.

3.1 Introduction

A very important point in machine learning is the design of empirical evaluation of learning algorithms. In this chapter we review the common practice in the learning community and present the methodology of empirical evaluation followed in this dissertation.

Learning algorithms are difficult to evaluate through analytical reasoning [KL90, LB91]. The difficulty is related to the usage of various heuristics, the behavior of which is difficult to predict. Thus information concerning the *complexity* of an algorithm is rarely provided. The usual method to evaluate a learning system is by *experimentation* [MST94] with the goal of highlighting some important relations between its behavior and certain conditions that affect it.

The models generated by a learning algorithm can be analyzed along different dimensions. In the machine learning literature the parameters considered frequently include error rate, comprehensibility, compactness, learning speed, classification speed, etc.

The most relevant dimension is the *error rate*. It is an estimator of the difference between $\hat{f}$ and the unknown f. The error rate is computed as the ratio between the number of misclassifications and the total number of examples. We need to distinguish between two notions of error rate. One, denoted as *resubstitution error*, estimates the error rate from the training set. The other, the *generalization error rate*, estimates the error from an independent test set. What we usually wish to know is the *generalization error rate* because this is the error that can be expect when applying the model to future examples. Later in this chapter, we provide a detailed discussion of this issue.

Another dimension that we consider here is related to the size of the model or its degree of compactness. As discussed in a previous chapter some methodologies for model selection are based on these metrics.

Yet another dimension that we consider here is *learning time*, that is the time needed by the algorithm to built the classifier. The application of a learning algorithm to real problems strongly depends on its ability to generate the model fast. Another parameter that we could consider is *classification time* or *testing time*, that is the time needed to classify a query example. All the algorithms that we use through this dissertation are relatively fast and so we have decided to exclude it from our experimental comparisons.

This section describes the methodology used in the design of the experiments oriented

towards the evaluation of the classification algorithms.

3.2 Evaluation of Learning Algorithms

In this section we discuss three different issues:

- Given an algorithm and a dataset, how to estimate the error rate of the algorithm on this dataset?
- Given two algorithms and a dataset, how to answer the question: *Do both algorithms generalize similarly on this dataset?*
- Given a set of algorithms and a set of datasets, how can we obtain a *ranking* of algorithms that reflects the generalization ability of the algorithms over those datasets?

All these questions are related, and we can not answer one question without solving the previous ones. Also, before trying to answer any of these questions we must understand the causes of variability of the error rate of an algorithm on a given dataset. This is why we consider this issue first.

3.2.1 Sources of Variability of the Error Rate

Any desiderata concerning the estimation of the error rate of a given learning algorithm on a given dataset, must take into account the sources of variability of the error rate. These include:

1. *Internal randomness on the learning algorithm.*
   Some algorithms, for example some multivariate trees, have internal data structures that are initialized in a random manner. Two different runs of the algorithm on exactly the same conditions can produce different results.
2. *Variability of the test set.*
   The generalization error is estimated on the test set. If there is sufficient amount of data, the variability of the test set does not affect much the variability of the estimate of the error rate. For small datasets, this assumes particular importance and can lead to wrong conclusions.

3. *Variability of the training set.*
   The model produced by some algorithms, namely decision trees, is unstable with respect to small variations of the training set. This means that small changes in the training examples can lead to very different models.

4. *Parameter setting.*
   With most of the available algorithms, the user can control the value of some parameters, such as, the level of pruning. Different values of the parameters can lead to different models and in consequence different estimations of the error rate.

The first source of variability does not affect the algorithms considered in this thesis, namely decision trees, naive Bayes, and Discriminants. This is due to the fact that there is no randomness incorporated in the learning algorithms. All the other three sources of variability are relevant. Here we consider the changes in the training set (item 3) as the most important source of variability because it implies the variability of the learned models. With respect to the last point (item 4), we have always used the default parameters of the algorithms. This approach affect all the algorithms. Had we tried to tune parameters of the algorithms, then our different expertise with each method would of advantage for some algorithms, but disadvantage for others.

3.2.2 Estimating the Error Rate

In theory, to evaluate a given classifier one should construct a training and test sets by drawing randomly a very large independent set of cases from the same population. Then, the classifier generated from the training set is used to classify each of the examples in the test set. If the number of cases is indeed large the proportion of misclassified examples gives an *unbiased* estimate of the error rate. In practice, however, it is difficult to obtain such large train and test sets. The available data, which is usually of limited size, is used both to build the classifier and to estimate the error rate. One possibility is to use so called *resubstitution* estimate. This estimate uses the training set to build the classifier and also to estimate the error rate. This is a *biased* estimator [BFOS84, MST94] that leads to unreliable conclusions about the generalization ability of algorithms. For example, the *1-Nearest Neighbor* rule or an unpruned decision tree will have, using this estimator, the *very optimistic* error rate near 0, independently of the dataset. This is why in practice the methods of evaluation nearly always split the available data into two samples: the training set and the test

set. Several different methods exist for obtaining the samples.

1. *Bootstrap*
   It is a process of sampling with replacement[1] from the available data. If n examples are available, a sample of size n is drawn with replacement to form the training set. The test set contains the set of examples that do not appear in the training set. The learning algorithm generates a model from the training set. The model is used to classify the examples of the test set. This whole process is repeated several times (usually 10), each time using a different bootstrap sample. The estimate of the error rate on the given dataset is the average of the error rates in each bootstrap sample.

2. *n-fold Cross-Validation*
   In cross-validation the examples are randomly placed into one of the n partitions. A sample is formed by setting aside one of n partitions as the test set, and the remaining partitions make up the training set. The learning algorithm generates a model from the training set. The model is used to classify the examples of the test set. This whole process is repeated for n training/test splits. The estimate of the error rate on the given dataset is the average of the error rate in each fold. A variant of this method denoted by *stratified cross-validation*, obtains partitions of the data that maintain the distribution of the classes in the original data.

3. *Monte-Carlo Cross-Validation*
   Monte-Carlo cross-validation is a special case of *n-fold Cross-Validation*. A percentage of the available examples (typically 2/3) is randomly placed in the training set and the remaining examples are placed in the test set. After learning takes place on the training set, generalization performance is estimated on the test set. This whole process is repeated for many training/test splits, usually between 30 and 50. The estimate of the error rate on the given dataset is the average of the error rates in each run.

Cross-validation methods are commonly used for comparing the generalization performance of two algorithms. The objective of evaluating two algorithms, say A and B on a given dataset is to answer questions such as: *"Does algorithm A generalize better than algorithm B on this dataset?"* or *"Do algorithm A and algorithm B generalize similarly on this dataset?"*. The method that we have chosen is 10-fold stratified cross-validation. Our decision is in agreement with the argument [Koh95a]:

[1] That is, one example could be chosen more than once.

> *Our results indicate that for real-world datasets similar to ours, the best method to use for model selection is ten-fold stratified cross-validation, even if computation power allows using more folds.*

Cross-validation allows us to estimate the error rate of an algorithm on a given dataset. The problem with cross-validation is the variance of the estimator [Koh95b]. To minimize this problem we repeat 10 times the 10-fold stratified cross-validation, each time using a different permutation of the dataset. This means that we run 100 times the learning algorithm. The estimate of the error rate is the mean of the 10 estimations using 10-fold cross-validation.

The estimated error rate using the methods described is used as a measure of the generalization performance of an algorithm. This estimate represents a mean of a random variable with an unknown distribution. To have confidence in any conclusion we need methods to determine when two means are significantly different.

3.2.3 Test of Hypotheses

Statistical literature concerning hypotheses testing offers a large set of tests. It is usual to distinguish between tests on matched pairs of data or on unmatched data. For our problem *paired tests* are more appropriate because: *"The objective in paired comparisons tests is to eliminate a maximum number of sources of extraneous variation by making the pairs similar with respect to as many variables as possible."* [Dan95]. To apply paired tests, we must ensure that the two learning algorithms learn (test) on the same partitions of the data. In this way we ensure that the differences in the error rate are only due to the differences in the generated models.

On another dimension it is usual to distinguish between *parametric* tests and *non parametric* tests. The former assumes some distribution for the samples, such as *normal* or *binomial* distributions. The latter are *distribution free* tests.

If our aim is to determine whether two means are different with given confidence level, statistical literature offers a large set of tests [BJ77, Dan95, Die96] for paired comparisons. Here we analyze:

- *The sign test*
  The sign test only takes into account the signal of the differences. It assumes a binomial distribution. Is the crudest and most insensitive test.

- *The Student paired t-tests*
  This is a parametric test that assumes normal distribution for the samples. It is commonly used by *Machine Learning* researchers.

- *The Wilcoxon matched-pairs signed ranks test*
  This test takes into account the magnitude of the differences. The null hypothesis is that the difference between the values of each pair has *median* equal to zero. That is the two pairs have identical distributions. Some authors maintain a position that this test is preferable to the *Student paired t-tests.*

All these methods require that the user determines the required confidence level. Parametric tests require, in addition, assumptions about the distribution of the random variables in the study.

Recent studies [Sal97, Die96] have shown that some of commonly used significance tests (namely *paired t-tests*) have a high probability of detecting a difference in generalization performance when no difference exists (Type I error). This is mainly due to the nature of the sampling process in the experimental design. Because the number of examples is limited, in all sampling methods described earlier identical cases may appear in different training samples. For example, with the standard 10-fold cross-validation procedure, we obtain 10 training sets and 10 test sets. With respect to the test sets, there is no overlap, that is all the examples that appear in one of the test sets never appear in another test set. With respect to the training sets, 80% of the examples that appear in one training set appear also in another training set. This fact violates the assumptions of many statistical tests concerning significant differences between means [Die96], that is, the independence of each estimator.

Another relevant issue is that when comparing multiple algorithms on multiple datasets, the confidence level must be adjusted [CJ97]. If one performs n independent statistical tests, each with α_c probability of a type I error, then the probability of at least one type I error in all n tests is

$$\alpha_e = 1 - (1 - \alpha_c)^n \tag{3.1}$$

This adjustment of the confidence level is called the *Bonferroni* adjustment [GC92, CJ97]. In our experiments we minimize this effect by setting the confidence level α_c to a high value, usually 99%.

3.2.4 Ranking of Algorithms

In this subsection we examine methods to answer the third question presented earlier in this chapter, and that is, how we can obtain a ranking of the available algorithms.

One possible approach could involve calculating the average error rate for all datasets and algorithms, and then sorting these values, and ranking the algorithms accordingly. This is not very satisfactory solution, however, mainly because error rates are not comparable across datasets. An error rate of say 5% could be an excellent result in one dataset, but a very poor one in another dataset. This solution does not take into account the difficulty of the classification task.

A more satisfactory approach consists of recording the ranks of all algorithms for all datasets and calculating average ranks. Rank 1 is assigned to the most accurate algorithm, rank 2 to the second best, and so on.

Another criterion that we have adopted, takes into account problem difficulty. It is explained in the following section. It is our adaptation of a similar method used in the StatLog project [BH94].

3.2.4.1 How far from the best performance?

This method proceeds as follow[2]. First, for each dataset, we identify the algorithm with the lowest error rate. Let E_{Low} be its error rate. We designate *error margin* as the standard deviation from a Bernoulli distribution [DG84] for the error rate of this algorithm:

$$Error\ Margin = z_C \sqrt{\frac{E_{low} * (1 - E_{low})}{N}} \tag{3.2}$$

where N represents the number of examples in the test set. The constant z_C is used to calculate the $C\%$ confidence interval. For example a 95% confidence interval is obtained setting $z_C = 1.96$ [Mit97]. The distance of the error rate of an algorithm to the best algorithm on each dataset in terms of the error margin is:

$$Distance_i = \frac{E_{alg_i} - E_{low}}{Error\ Margin} \tag{3.3}$$

The error margin satisfies the following properties:

[2]The method described here is a variant of the method used in the StatLog project [MST94].

- The error margin decreases with E_{low}. This implies that the distance increases when E_{low} decreases.
- The error margin decreases (increases) when the number of examples increases (decreases). This implies that the distance increases when the number of examples increases.
- A practical difficulty appears when E_{low} is zero. In this case the *Error Margin* is zero, and we can not compute $Distance_i$. This problem is solved by estimating error rates using Laplace correction [Mit97][3].

When performing multiple comparisons we average the distances for all datasets and use the averages to rank the different algorithms.

This method gives us an indication about the algorithms performance, taking into account problem difficulty. It must be seen as an heuristic, that gives a general indication about learners' performance.

3.2.5 Additional Comments

The methodology that we follow here is to calculate several global measures based on error rates. These include the arithmetic and geometric mean of error rates across datasets, which should obviously be taken as indicative. The significance of the differences of the error rates are evaluated using statistical tests. This includes signal test and paired t-tests relative to pairs of algorithms per dataset. Moreover, for each set of comparisons, we present the ranking of the algorithms using the approach described in the previous section.

When a test is used to evaluate the significance of difference in prediction error, the results should be interpreted with caution. In particular, a statistical difference should be read as *very probable to hold for the given data* and not as *very probable to hold for future data.*

3.2.6 Learning Time

Learning time is another dimension used for comparing classifiers. Here comparisons may be difficult to make as results may strongly depend on implementation details

[3]Previously we have solved this problem by defining $Distance_i = E_{alg_i}$ when $E_{Low} = 0$

as well on the underlying hardware. However at least the order of magnitude of time complexity is a useful indicator.

The process that we follow in this dissertation consists of choosing the faster algorithm as reference and report the time of the other algorithms relative to the time of the reference algorithm.

3.3 Bias-Variance Decomposition of the Error Rate

A very interesting analysis of error rates of learning algorithms is provided by the so called *Bias-Variance* decomposition of the error. The basic idea behind this theoretical framework is that a learning algorithm makes two kind of errors: systematic errors, that are due to the representation language used by the learning algorithm, and errors due to the dependence of the generated model from the training set. In this section we present this general framework. We use it later when analyzing the experimental results of the learning algorithms considered in this dissertation.

The origins of bias-variance analysis can be related to *quadratic regression.* It is a powerful tool from sampling theory of statistics for analyzing supervised learning scenarios that have a *quadratic loss function.* Given a fixed target and training set size, the conventional formulation of the decomposition breaks the expected error into the sum of three non-negative quantities:

- *Intrinsic target noise*
  This quantity is a lower bound on the expected error of any learning algorithm. It is the expected error of the Bayes-optimal classifier.

- *Squared "bias"*
  This quantity measures how well the average prediction of the learning algorithm (over all possible training sets of the given training set size) matches the target.

- *Variance*
  This quantity measures how much the prediction of the learning algorithm "bounces around" for different sets of the given size.

The bias-variance decomposition provides several useful insights. The most relevant is known as *bias-variance trade-off.* When modifying some aspect of a particular learning algorithm, it may have opposite effect on the bias and the variance. Usually, as one

increases the number of degrees of freedom of the algorithm, the bias decreases but the variance increases. The optimal number of degrees of freedom optimizes this trade-off between bias and variance.

For classification problems the quadratic loss function is inappropriate because class labels are not numeric. Several proposals for decomposing classification error into bias and variance have been suggested, including Breiman [Bre96a, Bre98], Kong and Dietterich [KD95a, KD95b], and Kohavi and Wolpert [KW96]. There is no consensus concerning the decomposition. Here we follow the decomposition suggested in [KW96].

3.3.1 The Bias-Variance Decomposition

Let **X** and **Y** be the input and output spaces respectively with cardinalities $|X|$ and $|Y|$ and elements **x** and **y** respectively. The "target" **F** represents a conditional probability distribution $P(Y_F = y_F|x)$ where Y_F is a random variable. If the target is a noise-free function from X to Y, for any fixed x we have $P(Y_F = y_F|x) = 1$ for one value of y_F, and 0 for all the others. The "hypothesis" **H** generated by a learning algorithm represents a similar distribution $P(Y_H = y_H|x)$ where Y_H is a random variable. As an example if the hypothesis is a single-valued function from X to Y, as it is for many classifiers (e.g. decision trees), then $P(Y_H = y_H|x) = 1$ for one value of y_H and 0 for all the others. In what follows we use $P(y_H)$ instead of $P(Y_H = y_H)$.

The *zero-one loss function* is a mapping from $\ell : Y \times X \rightarrow 0, 1$. $\ell(y_F, y_H) = 0$ iff $y_F = y_H$. The *cost*, C, is a random variable defined as the loss over the random variables Y_F and Y_H. The expected cost for a single example and target function is:

$$E(C) = 1 - \sum_{y \in Y} P(Y_H = Y_F = y) \tag{3.4}$$

The equation 3.4 can be rewritten as follows:

$$E(C) = \sum_{y \in Y} -P(Y_H = Y_F = y) + \sum_{y \in Y} P(Y_H = y)P(Y_F = y) +$$

$$\sum_{y \in Y} [-P(Y_H = y)P(Y_F = y) + \frac{1}{2}P(Y_F = y)^2 + \frac{1}{2}P(Y_H = y)^2] +$$

$$[\frac{1}{2} - \frac{1}{2}\sum_{y \in Y} P(Y_H = y)^2] + [\frac{1}{2} - \frac{1}{2}\sum_{y \in Y} P(Y_F = y)^2]$$

Rearranging the terms, we have

$$E(C) = \sum_{y \in Y} [P(Y_H = y)P(Y_F = y) - P(Y_H = Y_F = y)] +$$
$$\frac{1}{2} \sum_{y \in Y} (P(Y_F = y) - P(Y_H = y))^2 +$$
$$\frac{1}{2}(1 - \sum_{y \in Y} P(Y_H = y)^2) +$$
$$\frac{1}{2}(1 - \sum_{y \in Y} P(Y_F = y)^2)$$

Assuming that Y_F and Y_H are conditionally independent given f and x, then $P(Y_H = Y_F = y)$ is equal to $P(Y_H = y)P(Y_F = y)$ and so the first term in the previous equation is zero.

We want to estimate the expected cost for a fixed target and averaging over training sets $\mathbf{T}$ of size m. One way to evaluate this quantity is to write it as

$$\sum_x P(x)E(C|f, m, x)$$

where $x \in T$.

So

$$E(C) = \sum_x P(x)(\sigma_x^2 + bias_x^2 + variance_x) \tag{3.5}$$

where

$$bias_x^2 = \frac{1}{2} \sum_{y \in Y} (P(Y_F = y) - P(Y_H = y))^2 \tag{3.6}$$

$$variance_x = \frac{1}{2}(1 - \sum_{y \in Y} P(Y_H = y)^2) \tag{3.7}$$

$$\sigma_x^2 = \frac{1}{2}(1 - \sum_{y \in Y} P(Y_F = y)^2) \tag{3.8}$$

These definitions of $bias^2$, *variance*, and *noise* (σ) have the following properties:

1. The $bias^2$ component measures the squared difference between the average output of the target and the average output of the algorithm. It is a real-valued non-negative quantity and equals zero only if $P(Y_F = y|x) = P(Y_H = y|x)$ for all x and y. These properties are shared by the $bias^2$ for quadratic loss.

2. The *variance* term measures the "variability" (over Y_H) of $P(Y_H|x)$. It is a real-valued non-negative quantity and equals zero for an algorithm that always makes the same guess regardless of the training set (e.g. the Bayes optimal classifier). As the algorithm becomes more sensitive to changes in the training set, the variance increases. Moreover, given a distribution over training sets, the variance only measures the sensitivity of the learning algorithm to changes in the training set and is independent of the underlying target.

3. The noise is independent of the learning algorithm.

3.3.2 Measuring the Bias-Variance Components

For a given algorithm and a given dataset, we estimate the $bias^2$ and the *variance* as follows.

1. We randomly split the dataset into two parts, **T** and **E**. **T** is used as if it were the "world" from which we sample the training sets, while **E** is used to evaluate the terms in the decomposition.

2. We generate **N** training sets from T, each generated using uniform random sampling without replacement.

3. We run the learning algorithm on each of the training sets and estimate the terms of the *variance* equation 3.7 and *bias* equation 3.6 using the generated classifier for each example x in the evaluation set E. All the terms are estimated using frequency counts.

4. The noise component σ^2, that is the error of the optimal Bayes, is very difficult to estimate in practice. Using a frequency count estimator the estimate of σ^2 would be zero if all the instances are unique (regardless of the true σ^2). In our experiments it is aggregated to the bias term and so we assume that $\sigma^2 = 0$ as it is usually done.

Informally speaking, this analysis attempts to capture the following quantities: The bias term measures the *persistent error* of the learning algorithm. This error cannot be eliminated by increasing the number of independently trained classifiers. The variance term measures the error that is due to *fluctuation* when generating a single classifier. Averaging over many classifiers one can reduce this component and in that way reduce the expected error.

It should be noted that the sum of the bias component plus the variance component may slightly differ from the error rate measured using cross-validation. One reason is that training sets of different sizes are used to estimate these quantities.

3.4 Model Complexity

The *Occam's razor* principle, that was discussed in section 1.2, states:

> *Given two models with the same generalization error, the simpler one should be preferred because simplicity is desirable in itself.*

This principle could be used for selection of the *"best"* algorithm for a given dataset [Qui95, Dom98b]. The main problem is how to measure *simplicity*. Given two models using the same representation language, *simplicity* could be interpreted in terms of the syntactic descriptions of the models. For example, given two univariate decision trees the *number of nodes* could be used as a measure of the complexity of the models. But this measure is not suitable to compare, for example, a univariate tree and a multivariate tree because it does not take into account the differences in complexity of each node.

In this dissertation we are concerned mainly with partitioning algorithms, such as decision trees and their extensions. A criteria that we use to compare the complexity of models between those kind of systems is the *number of leaves*. Measuring the number of leaves corresponds to measuring the number of different regions into which the instance space is partitioned by the decision tree (univariate or multivariate). This can give insights about *concept fragmentation* and consequently how well the model fits the data.

This criteria only allows us to compare decision trees. If we want to compare models that use different representations languages we need more powerful techniques. One of such techniques is the *Minimum Description Length*. It encodes a model as a string of bits. It provides a unified encoding, useful when comparing the syntactic complexity of different models.

3.4.1 Minimum Description Length

The *Minimum Description Length* is related to ideas of data compression and information transmission. The goal is to transmit the data over a communication channel. Transmitting the data *as is* is not economical: non-random data contains structure and redundancies, and therefore must be amenable to compression. A good model of the data should capture its structure and yield good compression. The optimal model is the one that minimizes the length of the total message required to describe the data. This includes both the length required to specify the model itself and the data given the model. According to Shannon's theory of communication, the length of the message required to communicate an event that has probability p is proportional to $-log\ p$. Thus the most probable message should use the shortest description.

Any theory or hypothesis about data can be used to encode the data as a binary string. In the MDL setting, we have a *sender* and a *receiver*. The *sender* knows the full training set and the model. The *receiver* only needs to know the attribute values[4]. The *sender* must transmit the necessary information that allows the *receiver* to classify all the examples of the training set. As such, he must send the theory generated by the algorithm. Also he must send information identifying the examples misclassified by the model, and moreover, the correct classification. This provides a trade-off between simplicity of the model and the ability to explain the data. There is no guarantee that any codification schema is optimal. Vapnik [Vap95] refers that most of codifications use *smart* tricks. We use a basic coding schema. Although this may not be an optimal schema, for comparisons purposes it is sufficient. The codification schema follows the proposal of Brodley *et al.* [BU92] and Wallace *et al.* [WP93]. It is described in the next subsection.

3.4.2 Codification Schema

The codification schema for univariate decision trees such as *C4.5* we have used is as follows. Each node in a tree can be a leaf or an internal node. Internal nodes can contain a test on a discrete attribute or a test on a continuous attribute. To distinguish these type of nodes we need $log_2(3)$ bits. To codify a leaf, we only need to codify the class attached to this leaf. The number of bits required is $log_2(\#classes)$. To codify a test on a *discrete* attribute, we need to codify the attribute, that requires

[4]Note that if we need to code the data, this is common to all models. As such the code of the data is constant regardless of which model we use.

$log_2(\#Att)$ bits and to identify each descendent node. Each descendent node requires $log_2(VAtt_i)$ bits, where $VAtt_i$ is the number of values for Att_i. The code of a *test on a discrete* attribute requires $log_2(\#Att) + VAtt_i * log_2(VAtt_i)$ bits. To codify a *test on a continuous* attribute, we need to identify the attribute and the *cut-point.* The cut-point is a real number and is codified as in Rissanen [Ris83].

Any multivariate tree is coded similarly. We codify the coefficients that define a hyper plane, as a vector of real numbers. The coefficients of eliminated attributes are coded as 0[5].

To codify a misclassified example, we need to identify the example and provide the correct class. To identify the example we need $log_2(\#Examples)$ bits[6]. As such, to codify the errors we need: $\#errors * log_2(\#Examples) + \#errors * log_2(\#Classes)$ bits.

The codification schema for a *discriminant function* is similar to the method used to codify multivariate tests.

3.5 Evaluating Base Algorithms

Throughout this dissertation we use, as base algorithms a decision tree, a *naive Bayes*, and a set of discriminants: the linear discriminant, referred as *Discrim*, the quadratic discriminant, referred as *Quadisc* and a logistic discriminant, referred as *LogDisc*. As representative of decision trees we have used *C4.5* (release 8) that is generally available. In some experiments we have used *C5* [7], a recent improvement over *C4.5*. Our implementation of the *naive Bayes* classifier follows the standard method. All conditional probabilities are computed using *Laplace* estimators. Continuous attributes are discretized using the heuristic proposed in [KS96]. Our implementation of the discriminant functions follows closely the FORTRAN implementations used in StatLog project that are in the public domain. There are some relevant differences, though. All matrix inversions have been done using a *Singular Value Decomposition (SVD)* instead of the *Cholesky* decomposition. The SVD decomposition is also used to reduce the number of used attributes in the linear combinations, a feature that

[5] We could use one bit to identify whether or not the attribute is used, and codify only the coefficients of used attributes as real numbers.

[6] A strategy based on *offsets* will require fewer bits that is, on average, $log_2(\#errors/\#Examples)$ bits.

[7] RuleQuest Research

was not available in the original FORTRAN version. Another difference is that our implementation deals with missing values, by replacing them by the mean value of the attribute.

The table 3.1 presents the average error rates of those algorithms on 24 datasets from the UCI repository of Machine Learning [BKM99], using ten different runs of stratified 10-cross-validation.

Table 3.1: Error rates of base classifiers

Dataset	*C4.5*	*Discrim*	*Quadisc*	*LogDisc*	*Bayes*
Australian	14.71 ±0.6	14.06 ±0.1	22.41 ±0.6	**13.43** ±0.4	14.48 ±0.4
Balance	22.10 ±0.7	13.35 ±0.3	**8.51** ±0.1	10.52 ±0.5	8.57 ±0.3
Banding	23.98 ±1.8	23.20 ±1.4	**19.82** ±1.8	21.13 ±0.8	23.24 ±1.2
Breast(W)	5.46 ±0.5	4.13 ±0.1	4.87 ±0.1	3.36 ±0.1	**2.65** ±0.1
Cleveland	21.87 ±1.9	16.07 ±0.5	17.66 ±0.8	16.50 ±0.6	**16.06** ±0.7
Credit	14.28 ±0.6	14.23 ±0.1	27.01 ±0.3	**14.08** ±0.3	14.53 ±0.3
Diabetes	26.14 ±0.8	22.71 ±0.2	25.80 ±0.4	**22.37** ±0.3	23.87 ±0.5
German	28.63 ±0.7	23.03 ±0.5	28.22 ±0.5	**22.98** ±0.5	24.39 ±0.4
Glass	**31.96** ±2.6	36.65 ±0.8	41.78 ±1.1	36.26 ±1.4	37.43 ±1.5
Heart	22.85 ±2.0	16.37 ±0.4	17.07 ±0.7	16.44 ±0.6	**15.63** ±0.8
Hepatitis	20.42 ±1.6	21.60 ±2.2	**17.10** ±1.5	17.99 ±1.8	17.31 ±1.0
Ionosphere	10.47 ±1.2	13.38 ±0.8	**10.39** ±0.3	12.24 ±1.0	10.64 ±0.6
Iris	4.80 ±0.9	**2.00** ±0.0	2.73 ±0.4	2.93 ±1.0	4.27 ±0.6
Monks-1	**3.74** ±2.0	33.33 ±0.0	18.75 ±0.7	33.33 ±0.0	25.01 ±0.0
Monks-2	32.86 ±0.0	34.21 ±0.3	**26.52** ±0.9	34.28 ±0.2	34.19 ±0.6
Monks-3	**0.00** ±0.0	22.80 ±0.3	7.80 ±0.4	24.18 ±0.5	2.77 ±0.0
Mushroom	**0.00** ±0.0	6.86 ±0.0	0.64 ±0.0	3.23 ±0.0	3.85 ±0.0
Satimage	**13.65** ±0.4	16.01 ±0.1	14.35 ±0.1	14.20 ±0.2	19.05 ±0.1
Segment	**3.29** ±0.2	8.41 ±0.1	12.33 ±0.2	8.13 ±0.1	10.20 ±0.1
Sonar	25.95 ±2.2	24.95 ±1.2	**23.87** ±2.3	28.03 ±2.1	26.02 ±0.9
Vehicle	27.10 ±1.0	22.16 ±0.1	**14.77** ±0.7	19.99 ±0.6	38.73 ±0.6
Votes	**3.65** ±0.4	5.43 ±0.2	6.45 ±0.4	5.85 ±0.5	9.74 ±0.2
Waveform	24.66 ±0.4	14.94 ±0.2	15.97 ±0.2	**13.71** ±0.1	18.72 ±0.2
Wine	6.93 ±0.6	1.13 ±0.5	**0.84** ±0.3	1.13 ±0.5	2.37 ±0.6

Table 3.2 presents a summary of the results. We present pairwise comparisons between *C4.5* and all the other algorithms. We have chosen *C4.5* as reference because it is well known in the machine learning community. In the pairwise comparisons each row shows a particular statistical measure used, and each column represents the pair of algorithms in comparison. Each entry contains 2 numbers. The first one refers to the first algorithm and the second one refers to the second algorithm.

These results provide an evidence, once more, that no single algorithm is better overall.

Table 3.2: Summary of the results of base algorithms

	C4.5	*Discrim*	*Quadisc*	*LogDisc*	*Bayes*
Arithmetic mean (error rate)	16.23	17.13	16.07	16.51	16.82
Geometric mean (error rate)	11.28	13.28	11.51	12.57	12.79
Mean nr. Leaves	42				

	C4.5 / Bayes	*C4.5 / Discrim*	*C4.5 / Quadisc*	*C4.5 / LogDisc*
Nr. of wins	12 / 12	10 / 14	9 / 15	10 / 14
Nr. of signif. wins(t-test)	9 / 9	9 / 10	9 / 11	8 / 12
Nr. of signif. wins(signal)	9 / 8	8 / 10	8 / 9	7 / 10

We have also compared the different classification algorithms in terms of the normalized distance measure, described in section 3.2.4.1. The means of these distances are presented in table 3.3.

Table 3.3: Average Distance to the Best Algorithm

LogDisc	*C4.5*	*Discrim*	*Quadisc*	*naive Bayes*
3.0	3.4	3.6	3.7	4.7

The average of ranks for each dataset is presented in table 3.4.

Table 3.4: Average of Ranks

LogDisc	*Quadisc*	*Discrim*	*naive Bayes*	*C4.5*
2.7	2.8	3.0	3.25	3.29

Let us examine now the corresponding learning times. In all the experiments reported here, the fastest algorithm was *naive Bayes*, the slowest was *Logistic discriminant*. Table 3.5 presents the mean of relative learning time of all the algorithms, relative to the time of *naive Bayes*.

We have calculated also the bias and the variance components of the error. This was done for all datasets. Then we computed the overall means. The results are presented in table 3.6. As it was previously noted, the *variance* component has strong influence on the error of decision trees, while with the discriminant functions the main component is *bias*[8]. The reduced number of free parameters in the discriminant

[8]Note that in our experiments the bias component includes also the intrinsic noise, or as it is sometimes referred to, error of optimal Bayes.

Table 3.5: Relative learning times for the base algorithms

Bayes	*Discrim*	*Quadisc*	*C4.5*	*LogDisc*
1	1.03	1.12	2.26	22

Table 3.6: Bias and Variance contribution to error rate

	Variance	*Bias*
C4.5	32%	68%
Discrim	8%	92%
Bayes	11%	89%

functions, makes the algorithm quite stable, but limits its ability to fit the data. As such, the systematic errors increase although the dependence on the training set is reduced. On the other hand, decision trees do not have a limit on the number of free parameters, and so they can fit better the data. Increasing the number of free parameters reduces the systematic errors but increases the variance.

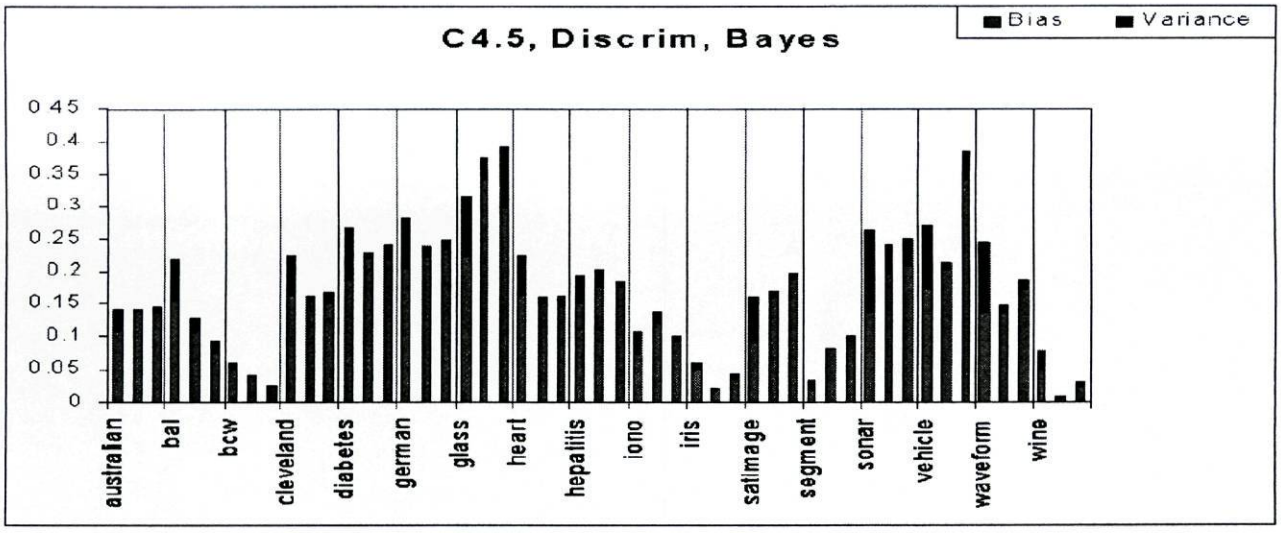


Figure 3.1: Bias and Variance components of the error of C4.5, Discrim and naive Bayes.

With respect to *model complexity*, we have only compared *C4.5* against *Discrim* using *MDL*. On these datasets, to codify a decision tree generated by *C4.5*, we needed, on average 612 bits, while to codify the coefficients of a linear discriminant we required only 363 bits. Using *paired t-tests*, the differences are significant in all the datasets. *C4.5* leads to smaller codes in 10 datasets, while *Discrim* leads to smaller codes in 14 datasets.

3.6 Summary

In this chapter we have presented the main methods for evaluating learning algorithms. We have discussed the sources of variability of the error rate. We have shown how to estimate the error rate of a given algorithm on a given dataset, and carry out statistical tests with a given level of confidence to perform comparisons on multiple datasets. An interesting analysis of the error rate of an algorithm on a given dataset is provided by the *bias-variance* decomposition. We have described the *Minimum Description Length* that can be used to compare the syntactic complexity of models. In the final section we have presented experimental results concerning the base algorithms used throughout this dissertation on a diverse set of datasets from the UCI repository.

Chapter 4

Discriminant Trees

In this chapter we present a method for building multivariate decision trees for propositional supervised learning. The method consists of combining a decision tree with a discriminant function by means of constructive induction. At each decision node a new instance space is defined by the insertion of new attributes. They are obtained by projecting the examples that fall at this node over the hyper-planes given by the discriminant function. This new instance space is propagated down the tree. Tests based on those new attributes are oblique *with respect to the original input space. We have implemented three systems using three different discriminant functions as constructive operators:* Ltree *that uses a linear discriminant function,* Qtree *that uses a quadratic discriminant function, and* LgTree *that uses a logistic discriminant function. All systems are able to define decision surfaces both orthogonal and oblique to the axes defined by the attributes of the input space. We have carried out experiments on eighteen benchmark datasets and compared our system with other well known decision tree systems (orthogonal and oblique) like* C4.5, OC1, LMDT, *and* CART. *On these datasets our system has advantages in what concerns accuracy at statistically significant confidence levels.*

4.1 Introduction

One of the appealing extensions of the standard algorithm for constructing decision trees is the ability to use multivariate tests. Several researchers have explored this idea, but only with a relative success. In this chapter we present a new architecture for constructing multivariate trees. We have evaluated the proposed methodology on several datasets from the UCI repository, and compared against the state of the art in decision tree induction. We show that our system has consistent advantages in what concerns accuracy when compared against trees with univariate tests. When compared against other multivariate trees it exhibits consistent advantages both in terms of accuracy and learning times.

4.2 An Illustrative Example

4.2.1 Motivation

Consider an artificial two class problem defined by two numerical attributes, shown in Fig. 4.1. The dark shaded region contains points that belong to class +. The light shaded region contains points that belong to class −. If we use a standard decision tree

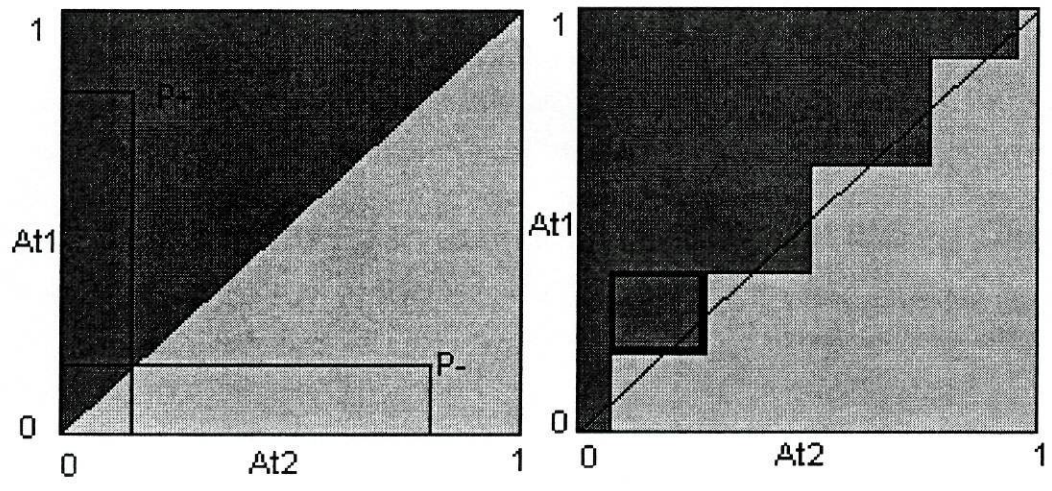


Figure 4.1: The diagonal problem. The left figure shows the original data. The right part shows the decision surface determined by a decision tree.

algorithm C4.5, we obtain a decision tree with a large number of nodes (in our case 65). Obviously this tree is much more complex than expected. By analyzing paths in

the tree we find tests on the same attribute. One of such paths, from the root to a leaf, is:

> **If** $at2 \leq 0.398$ **AND** $at1 \leq 0.281$ **AND** $at2 > 0.108$ **AND** $at1 > 0.184$ **AND** $at2 \leq 0.267$ **AND** $at2 > 0.189$ **AND** $at1 \leq 0.237$ **AND** $at2 \leq 0.218$ **AND** $at1 > 0.2$ **THEN CLASS** $+$

We note that some tests are redundant. If we eliminate the redundant tests we obtain **If** $0.2 < at1 \leq 0.237$ **AND** $0.189 < at2 \leq 0.267$ **THEN CLASS** $+$. The branch defines in effect a rectangular region in which all the points, in the training set, are classified by $+$. Similarly other paths in the tree correspond to rectangular regions in which all (or almost all) the points are of the same class. This means that the tree, as a whole, is making an approximation to an oblique region by means of a staircase-like structure, as the one shown in Figure 4.1b. This example illustrates the well known problem that a univariate test using attribute at_i can only split the instance-space with a boundary that is orthogonal to at_i axis [BFOS84]. This restricts the space of regions in the instance space that can be represented succinctly, and results in a large tree with lower predictive accuracy.

Building a decision tree using continuous attributes has specific problems when compared to nominal attributes. Quinlan [Qui96b] introduces a penalty on the entropy score of continuous attributes based on the number of possible cut-points. Also Merckt [Van93] raises the question of the adequacy of classical entropy-based criteria for finding correct disjunctive descriptions on continuous spaces, and proposes a metric based on cluster analysis.

Both approaches could be useful but do not address to main problem: the problem of representation. The standard algorithm for univariate decision trees splits the instance space using hyper-rectangles. Inter-relations between attributes are difficult to express using this representation language. Several researchers [PH90, MR89, Mat91, BFOS84, HKS93, BU95, MKS94, LV88, Joh96, LS97] have addressed this problem, incorporating in the algorithm the ability of building (implicit or explicit) new attributes. This ability is denoted as *constructive induction.*

4.2.2 Constructive Induction

Performance of selective induction algorithms, in terms of predictive accuracy, is usually poor if the task-supplied attributes are not appropriate for describing the target concepts.

Systems that employ constructive induction consist of two components: one for constructing new attributes, and the other for generating models. The new attributes can be treated in the same way as the original ones. For a non-trivial learning problem the number of possible constructive operators, such as logical operators and mathematical operators, and the number of possible operands, is usually very large. It is not feasible to search through all the possible combinations. Usually the tendency is to select a set of potentially useful operators and reduce thus the search space of new attributes. Conjunction, disjunction, and negation are commonly used as constructive operators, whereby they produce Boolean attributes.

Another type of constructive operator is *M-of-N* including the variants *at-least M-of-N*, *at-most M-of-N*, and *exactly M-of-N*. A *M-of-N* operator generates Boolean attributes. Given an integer M and a set of N conditions based on existing attributes, an *at-least M-of-N* attribute is true if at least M of the N conditions are true. Zheng [Zhe96] proposes the *X-of-N* constructor that returns the number of true conditions. It generates ordered discrete values. One of the quite well known constructive induction system is the *FRINGE* family of algorithms [PH90]. These algorithms iteratively build a decision tree based on the existing attributes (initially only primitive attributes), and then construct new attributes by using conjunctions and/or disjunctions of two conditions from the tree paths. The new attributes are added to the set of existing attributes and the process is repeated. The conditions used for generating new attributes are chosen from fixed positions in the paths, either near the root and/or near the fringe of a tree. The insertion of the new attributes produces an extension of the representational power and relaxes the language bias of decision tree learning algorithms.

In Statistics and Machine learning, several methods exist that generate new attributes in the form of linear combinations of the original continuous ones. *CART* of Breiman *et al.* [BFOS84] can generate split tests in the form of linear combinations $\sum_{i=1}^{n} w_i x_i$, where $x_1, ..., x_n$ are primitive attributes (ordered), and $w_1, ..., w_n$ are coefficients such that $\sum_{i=1}^{n} w_i^2 = 1$. *CART* performs a heuristic hill climbing search to find a vector of coefficients and a backward attribute elimination so as to minimize the impurity of the partition defined by the corresponding test.

The *Oblique Classifier 1 - OC1* of Murthy *et al.* [MKS94] is an extension of CART. Murphy *et al.* refer to this type of trees as *oblique decision trees*. The main contribution of OC1 is its randomized coefficient perturbation algorithm. Similarly to *CART*, heuristic hill climbing search is used to find (local) optimum coefficients for linear combinations. To escape local optimum, after the hill climbing search halts,

two randomization mechanisms are applied. One is that a random vector is added to the coefficients of the linear combination that resulted from the search. If this result in a better partition, the hill climbing search is carried out again. Otherwise, the second mechanism is used. That is, the hill climbing search starts again with another randomly selected starting point.

The *Linear Machine Decision Tree - LMDT* of Brodley and Utgoff [BU95] generates multivariate trees by training *linear machines.* At each node of the tree building process, *LMDT* uses the absolute error correction rule [DH73] to train a linear machine in the form of linear discriminant functions $g_i(\vec{x}) = \vec{w}^T\vec{x}$. When a linear machine is close to its final set of boundaries, *LMDT* deletes the attribute that contributes least to discriminating the set of instances at that node. This decision is based on a dispersion measure. *LMDT* then continues training the linear machine. The linear machine is used as a new nominal test at a particular decision node. Each node may have its own linear machine.

The *Quick, Unbiased and Efficient Statistical Tree - Quest* of Loh *et al.* [LV88, LS97] builds binary trees using discriminant analysis. At each node a *2-means clustering* is used to create two superclasses. The system carries out a linear discriminant analysis on the two superclasses. The linear discriminant is chosen as test for this node. *Quest* uses the error complexity pruning mechanism of *CART.* It is incorporated in SPSS software (AnswerTree)[1] and StatSoft (Statistica) [Sta99][2].

Another important difference to note is that *CART*, *OC1*, and *Quest* implicitly build new *Boolean* attributes whilst *LMDT* implicitly builds *nominal* attributes. *Cart*, *OC1*, and *Quest* generate binary trees, whilst in *LMDT* the number of descendents at each node is equal to the number of classes of the examples that fall at this node.

4.2.3 Our Approach

Consider again our small artificial problem defined in figure 4.1. Running a linear discriminant procedure, we get one discriminant:

$$H = 0.0 + 11.0 * at1 - 11.0 * at2$$

Line H is the hyper-plane generated by the linear discriminant. A new attribute ($at3$) is generated by projecting the examples on the normal to this hyper-plane. For

[1] `http://www.spss.com`

[2] `http://www.statsoft.com`

example, the point P+ with coordinates (0.7,0.1) becomes in this new space P+ with coordinates (0.7,0.1,6.6). The new instance space is shown in Fig. 4.2. In the new

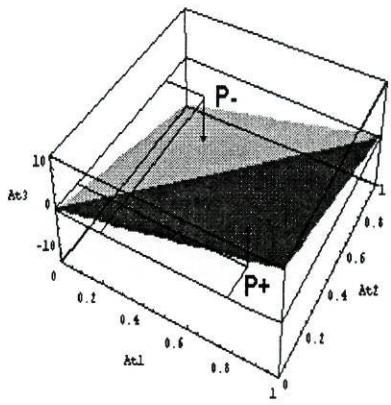


Figure 4.2: The extended instance space.

instance space a hyper-rectangle orthogonal to the new axis is able to split classes using only one test. Running C4.5 on this new dataset, we obtain the following tree:

If $at3 > 0$ **THEN CLASS** +
If $at3 \leq 0$ **THEN CLASS** −

This is the smallest possible tree that discriminates the classes. The test is orthogonal to the axis defined by attribute $at3$ and parallel to the axes defined by $at2$ and $at1$. Rewriting the rule ''**If** `at3 > 0` **THEN** `CLASS +`'' in terms of the original space (that is in terms of $at1$ and $at2$) we get: ''**If** `11 * at1 > 11 * at2 - 0` **THEN** `CLASS +`''. This rule defines a relation between attributes, thus is oblique to the $at1$ and $at2$ axes. This simple example illustrates one fundamental point. Constructive induction based on combinations of attributes extends the representational power and overcomes limitations of the language bias of decision tree learning algorithms.

Any constructive induction system, consists of two components. One that generates new attributes by applying a set of constructive operators and the other one that builds the model.

In this chapter we focus on problems with continuous attributes and therefore use three appropriate discriminant functions as constructive operators. We use the linear and the logistic discriminant that build new attributes in the form of linear combinations

of the previous ones. Besides, we use also the quadratic discriminant that builds new attributes that are quadratic combinations of the previous ones.

In our case the *model generation* component is a univariate decision tree. In each iteration of the tree building process the constructive operator builds new attributes. If one of the new attributes is chosen by the merit function of the univariate decision tree (gain ratio), this corresponds to introducing a multivariate test, as we have seen earlier.

There are three new aspects in our contribution to the state of the art on multivariate decision trees.

- An attribute is chosen as a test attribute at each node, if it is suggested by the univariate tree criteria. This strategy differs from other systems. For example in *LMDT* and *Quest*, the best linear combination is always chosen as multivariate test.
- All the new attributes are continuous. As such they are considered at deeper nodes on the tree. The new attributes built at lower nodes may contain terms based on attributes built at previous nodes. This architecture allows building very complex decision surfaces, for example using a *quadratic discriminant* as constructive operator.
- The number of attributes is variable along the tree, including for two nodes at the same level. The number of attributes that are built at each node depends on the classes distribution at this node.

The rest of this chapter describes the method in more detail and presents an evaluation of the three systems that implement this methodology:

- *Ltree* - Linear Tree,
  that uses as constructive operator a linear discriminant function.
- *Qtree* - Quadratic Tree,
  that uses as constructive operator a quadratic discriminant function.
- *Lgtree* - Logistic Tree,
  that uses as constructive operator a logistic discriminant function.

We focus our analysis on *Ltree*. The other systems, *Qtree* and *LgTree* are similar to *Ltree*. Only the attribute constructor is different.

In the following sections we explain in detail, the proposed methodology. First we explore the application of the methodology as a pre-processing step. Later we study how the process can be integrated within the process of tree growing.

4.3 Generating New Attributes in a Pre-processing Step: C4.5Oblique

We have implemented a two step algorithm that explores this idea as a *pre-processing* step. Given a dataset, a linear discriminant builds the hyper-planes. New attributes are created by projecting all the examples on the normal to the hyper-planes. The transformed dataset is then passed to C4.5. We refer to this algorithm as *C4.5Oblique.* It was included in our experimental study. We have verified that it can achieve quite good results (see section 4.5.2 for details).

4.4 Linear Tree

Ltree explores dynamically the constructive step of *C4.5Oblique.* At each decision node *Ltree* builds new attributes based on linear combinations of the previous ones. If the best orthogonal axis involves one of the new attributes, the decision surface is oblique with respect to the original axes. The motivation is that for each sub-space local interdependencies between attributes can be captured.

Ltree is a top down inductive decision tree system. The main algorithm is similar to many other programs from the TDIDT family, except for the construction of new attributes at each decision node. Before going into details of the process of tree construction we provide an illustrative example using the Iris dataset.

4.4.1 An Illustrative Example: The Iris Data

To illustrate the basic ideas behind *Ltree*, we use the well-known Iris dataset. The tree generated by *Ltree* is shown in Fig. 4.3. The figure shows the class distributions and the probability class distributions at each leaf. This is a tree with 5 nodes. It misclassifies 4% of the examples on the test data[3]. For the same data, C4.5 generates

[3]Using leave-one-out cross-validation.

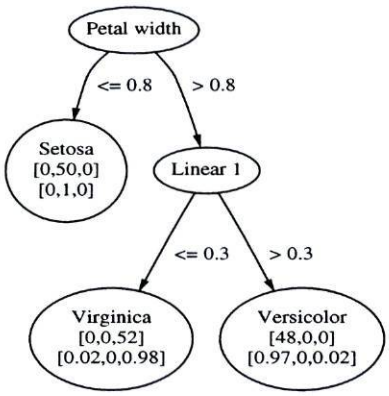


Figure 4.3: Decision tree for iris data

a tree with 11 nodes with 4.7% of errors.

In this example, *Ltree* proceeds as follows. At the root of the tree there are 4 attributes and 3 classes. For each class, there are 50 examples. Three training examples described in terms of the original attributes are shown in Table 4.1.

Table 4.1: Three examples from the Iris dataset

$Sepal_{len}$	$Sepal_{wid}$	$Petal_{len}$	$Petal_{wid}$	**Class**
5.1	2.5	3.0	1.1	versicolor
6.5	3.0	5.5	1.8	virginia
4.8	3.1	1.6	0.2	setosa

Two hyper-planes are constructed. They are based on linear combinations of the four original attributes. The linear combinations generated are:

$Linear\_1 = -31.5 - 3.2 * Sepal_{len} - 3.5 * Sepal_{wid} + 7.5 * Petal_{len} + 14.7 * Petal_{wid}$

$Linear\_2 = -13.3 + 7.7 * Sepal_{len} + 16.6 * Sepal_{wid} - 21.5 * Petal_{len} - 24.3 * Petal_{wid}$

All the examples are projected on the normal to the two hyper-planes. This is done using equation 2.14 and the resulting quantities normalized to sum to unity. The instance space is reconstructed by the insertion of two new attributes (*Ltree* calls them *Linear_1* and *Linear_2*). The same three examples, after the insertion of the two new attributes, become:

$Sepal_{len}$	$Sepal_{wid}$	$Petal_{len}$	$Petal_{wid}$	$Linear_1$	$Linear_2$	**Class**
5.1	2.5	3.0	1.1	0.995	0.005	versicolor
6.5	3.0	5.5	1.8	1.000	0.000	virginica
4.8	3.1	1.6	0.2	0.000	1.000	setosa

Using the *gain ratio* as criterion, the attribute that splits best the data is chosen – the attribute *Petal width*. Since the tested attribute is continuous, the original dataset is divided into two distinct subsets. In the first one, all examples belong to the same class and hence *Ltree* returns a leaf and the splitting process terminates. The second subset, is described using 6 attributes and the examples belong to two classes. One new hyper-plane is generated, and the examples are projected on the normal to the hyper-plane. The equation of this hyper-plane contains terms based on both the original attributes and the new attributes built at the root of the tree. The linear combination generated is:
$Linear\_3 = -16.6 - 3.5 * Sepal_{len} - 5.5 * Sepal_{wid} + 6.9 * Petal_{len} + 12.4 * Petal_{wid} + 0.03 * Linear\_1 - 0.03 * Linear\_2$

One new attribute is built and so the new instance space is described using 7 attributes and the examples belong to two classes. Now, the attribute that best splits the data

$Sepal_{len}$	$Sepal_{wid}$	$Petal_{len}$	$Petal_{wid}$	$Linear_1$	$Linear_2$	$Linear_3$	**Class**
5.1	2.5	3.0	1.1	0.995	0.005	0.0	versicolor
6.5	3.0	5.2	2.0	1.000	0.000	1.0	virginica

is the *Linear_1* that was created at the root. This attribute splits the data into two subsets each containing examples of only one class. The process of generating the tree terminates.

4.4.2 New Attributes

At each decision point, *Ltree* computes new attributes dynamically. New attributes are linear combinations of all ordered attributes of the examples that fall at this decision node.

To build new attributes the system considers at each decision point, a varying number of classes, taking into account classes distribution at this node. Only well populated classes are considered, however. These are classes for which a pre-defined minimal number of examples exists at this node. This done so as not to underfit the concept. Following Breiman et al. [BFOS84] we consider the class i if $\#examples_i > 3 \times \#attributes$. Suppose that in a decision node *Ltree* considers q_{node} classes. The linear discriminant procedure generates $q_{node} - 1$ hyper-planes, that take the form of equation 2.15 and 2.16 shown earlier. The equation of each hyper-plane is given by:

$$H_i = \alpha_i + \sum_j \beta_{ij} * x_j$$

where

$$\alpha_i = log(P(C_i)) - \frac{1}{2}\mu_i^T S^{-1}\mu_i \text{ and } \beta_i = S^{-1}\mu_i$$

Ltree builds $q_{node} - 1$ new attributes. All the examples of this node are extended with the new attributes. Each new attribute is given by the projection of the examples on the normal to the *hyper-plane* H_i. The projection is computed as the dot product of the example vector $\mathbf{x}$ and the coefficients of the *hyper-plane* H_i. Projecting the example on the normal to a hyper-plane corresponds to computing the distance from the example to the hyper-plane. This is done using the equation 2.14 shown earlier, followed by a normalization procedure to sum to unity.

Because class distribution varies along the tree, the number of new attributes is variable. Two different nodes (also at the same level) may have different number of attributes. As the tree grows and the classes are discriminated, the number of new attributes decreases. All new attributes are propagated down the tree. This is one of the most innovative features of *Ltree*.

4.4.3 Splitting Criteria

It is known that building the smallest tree consistent with a given training dataset is a NP complete problem, and so we must use heuristics to guide the search. A splitting rule typically works as a one-step look-ahead heuristic. For each possible test, the system hypothetically considers the subsets of data obtained. It chooses the test that maximizes (or minimize) some heuristic function over the subsets. By default *Ltree* uses the *gain ratio* [Qui93] as the splitting criteria. A test on a *nominal* attribute will divide the data into as many subsets as the number of values of the attribute. A test on a *continuous* attribute will divide the data into two subsets: $attribute\_value > cutpoint$ and $attribute\_value \leq cutpoint$. To determine the *cut point*, we follow a process similar to C4.5 [Qui93].

4.4.4 Stopping Criteria

One possible stopping criteria for decision trees is to stop building the tree when all the examples that fall at a decision node are from the same class. In noisy domains a more relaxed criteria is needed. *Ltree* stops growing the tree whenever the percentage of the examples from the majority class is greater than a user defined parameter (by

default 98%). This is a form of pre-pruning that seems to work well on most of the datasets used in our experimental study. If there are no examples at a decision node, *Ltree* returns a leaf with the class distribution of the predecessor of this node.

4.4.5 Smoothing

In spite of the positive aspects of divide and conquer algorithms, one should be concerned about the statistical consequences of dividing the input space. Dividing the data can improve the bias, because it allows fine fitting to the data, but in general it increases the variance. One way to minimize the effects of splitting the data is by using soft thresholds. *Ltree* uses a *smoothing* process, following Buntine [Bun90], that usually improves performance of tree based classifiers. When classifying a new example, the example traverses the tree from the root to a leaf. The class attached to the example is generated by taking into account not only the class distribution at the leaf, but all class distributions of the nodes on the path. That is, all nodes along the path contribute to the final classification. Instead of computing class distribution for all paths in the tree at classification time, as it is done in [Qui92], *Ltree* computes a class distribution for all nodes when growing the tree. This is done recursively, taking into account class distributions at the current node and at its predecessor. At each node n, *Ltree* combines both class distributions using the *recursive Bayesian update* formula [Pea88]:

$$P(C_i|e_n, e_{n+1}) = P(C_i|e_n)\frac{P(e_{n+1}|e_n, C_i)}{P(e_{n+1}|e_n)} \tag{4.1}$$

Here $P(e_n)$ is the probability that a given example e falls in node n, that can be seen as a shorthand for $P(e \in E_n)$, where e represents the given example and E_n the set of examples in node n. Similarly $P(e_{n+1}|e_n)$ is the probability that an example that falls in node n falls in node $n+1$, and $P(e_{n+1}|e_n, C_i)$ is the probability that an example from class C_i is passed from node n to node $n+1$. Figure 4.4 illustrates how class probabilities are computed.

This recursive formulation for updating beliefs allows us to compute efficiently the required class distributions. Classification done using smoothed class distributions is more robust than when computed from the examples that fall at the leaf [Bun90]. We have evaluated the effect of smoothing in *Ltree*. The results (shown in section 4.5.4) indicate that when smoothing is used, the method leads to more compact trees, although the benefits on *error rate* are not significant. Both pruning the tree and processing missing values explore the smoothed class distributions.

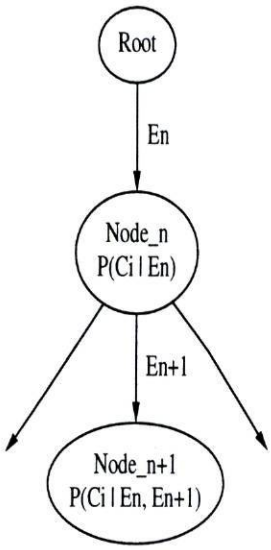


Figure 4.4: Smooth propagation of class probabilities

4.4.6 Pruning

Pruning is considered to be the most crucial part of the tree building process at least in noisy domains. Statistical measures computed at deeper nodes of a tree may have low level of significance. This is due to the fact that usually a small number of examples fall in these nodes. Deeper nodes reflect too much the training set (over-fitting) and increase the error due to the variance of the classifier.

Several methods for pruning decision trees have been presented in literature [BFOS84, EMS93a, Qui93]. The process that we use exploits class probability distributions computed by *Ltree* at each node. In C4.5, pruning is a process that usually leads to increased error rate on the training data. In our case this is not necessarily true. The class that is assigned to an example takes into account the path from the root to the leaf, and could be different from the majority class of the examples that fall in the leaf. At each node *Ltree* considers the static error and the backed-up error. Static error is the number of misclassifications considering that all the examples that fall at this node are classified using the majority class taken from the smoothed class distribution in this node. backed-up error is the sum of misclassifications of all subtrees of the current node. If backed-up error is greater or equal than static error then a terminal leaf node is created.

4.4.7 Missing Values

When using a tree as a classifier, the example to be classified passes down through the tree. At each node a test is performed, and if the value of the tested attribute is not known (often in real data some attribute values are unknown or undetermined) the normal procedure cannot choose the path to follow. Since a decision tree constitutes a hierarchy of tests, the *unknown problem* has special relevance on this type of classifiers. *Ltree* passes the example down all branches of the node where the unknown attribute value was detected, following [Bun90]. Each branch outputs a class distribution. The output is a combination of the different class distributions that sum to 1.

At learning time, if an example has a missing value on the test attribute, a fractional example is passed down all the descendent branches and then the usual procedure is followed. The part of the example that is passed down to each particular branch is proportional to $1/N$ where N represents the number of branches at this node.

4.4.8 Classifying New Examples

To be applied as a predictor *Ltree* stores, at each node, the discriminant function generated. When classifying a new example, the example traverses the tree in the usual way, but at each decision node it is extended by the insertion of the attributes generated by projecting the example on the normal to the hyper-planes generated by the discriminant function.

4.5 Empirical Evaluation

We have performed an extensive empirical evaluation of *Ltree* both on artificial data and real data. The performance of our system was compared with other well known decision tree algorithms. We have used one univariate tree, *C4.5* and four multivariate trees *LMDT* [UB91], *Quest* [LV88, LS97], and *OC1* [MKS94] that are available via the *Internet*[4], and *CART*[5] [BFOS84]. Following the methodology described in the previous chapter, performance of the algorithms is evaluated by measuring the *error rate*, *learning time*, and the *number of leaves* of the generated tree. The latter provides a measure of the number of different regions into which the instance space is partitioned.

[4] `http://www.gmd.de/ml-archive`

[5] California Statistical Software, Inc.

To compare model complexity between univariate and multivariate trees we have used the *Minimum Description Length.*

4.5.1 Evaluation on Artificial Data

The main interest of using artificial data, is the possibility of controlled experiments. We have used two artificial datasets. Both are two class problems. The examples were generated randomly and classified using a known rule.

The first dataset is the *LS10*, defined by Murthy *et al.* [MKS94]. The data is linearly separable with a 10-dimensional hyper-plane defined by the equation $x_1+x_2+x_3+x_4+x_5 > x_6+x_7+x_8+x_9+x_{10}$. The attributes are uniformly distributed in the range [0,1]. The second dataset consists of 1000 data points, defined by two continuous attributes uniformly distributed in the range [0,1] (see Figure 4.6). There are two classes. Each class defines two convex regions, forming a kind of *XOR* problem. We use this simple dataset as a test for the ability of multivariate decision trees to deal with problems where the classes are not linearly separable.

The overall results, using 10-fold cross-validation, in terms of *error rates* and *number of leaves* on both datasets are presented in table 4.2. *Ltree* achieves lower error rate than all the other algorithms considered on the *XOR* problem and it is the third best on *LS10* problem.

Table 4.2: Error rates and Nr. of leaves on 10-Fold Cross-Validation

Dataset	*C4.5*	*CART*	*OC1*	*LMDT*	*Quest*	*Ltree*
Ls10						
Error rate	22.35	2.2	3.7	0.35	1.03	1.3
Nr.Leaves	137.6	2	6.3	2	4.3	2.5
XOR						
Error rate	4.0	3.1	3.4	9.5	4.0	2.7
Nr.Leaves	19.6	13.5	13.2	36	34.9	15.7

We have also performed a set of experiments with learning curves. The experimental methodology was as follows. We have used a 10-fold stratified cross-validation and in each run, the algorithm was trained 10 times, using an increasing number of examples from the training set. This corresponds to generating 10 * 10 classifiers. The process began with 10% of the training set examples, and each time the number of training examples was increased of 10%. On the 10th run, the algorithm used all the training

examples for learning. Each classifier was used to classify all the examples in the test set. After completing this we computed the means of error rates and number of leaves of the runs where the algorithm used the same number of training examples.

Fig. 4.5 shows the results on *LS10* dataset. It shows the dependence of the error rate and the number of leaves on the number of training examples.

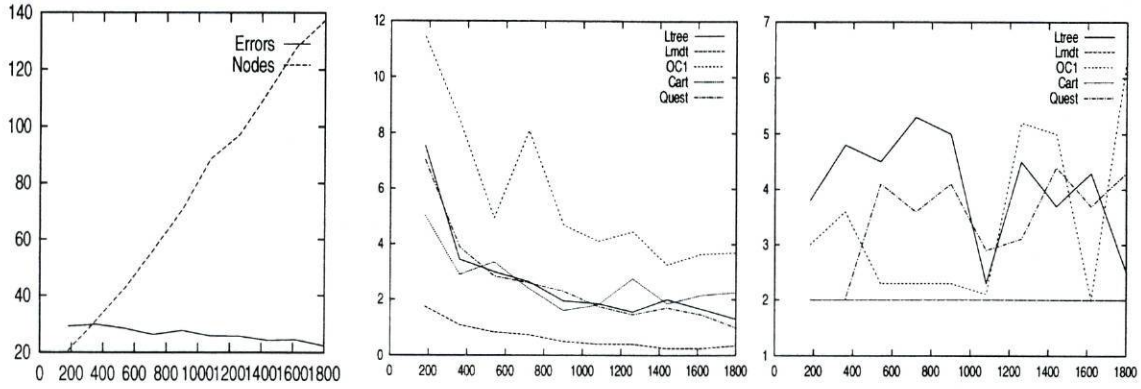


Figure 4.5: Learning Curves for LS10 data (a) Nr. leaves and Error rate for C4.5 (b) Error rates for Multivariate Trees (c) Nr. leaves for Multivariate Trees

This artificial problem is clearly biased for *multivariate* algorithms. The curve for *C4.5* illustrates the typical behavior when the bias of the algorithm is not appropriate for the problem: the tree grows linearly with the number of examples and there are no significant benefits in the performance. *CART*, and *LMDT* in all of the 10 runs generated a tree with 2 leaves. That is, both algorithms partition the instance space into two regions independently of how many examples have been used in training. *LMDT* performs quite well. The *error rate* of *LMDT* trained on 200 examples is similar to the error rate of some other algorithms (*CART*, *Ltree*) trained on 1800 examples. The performance of *Ltree* is similar to *CART* and *Quest*, while *OC1* performs significantly worse than *CART*.

Results on the *XOR* problem are presented in Figure 4.6. In this problem *Ltree* appears to be best overall, although *CART* and *OC1* have a comparable performance, both in terms of *error rates* and *number of leaves*. The exception is *LMDT* that performs significantly worse than *C4.5* here. The *Quest* system performs similarly to *C4.5*.

The results obtained on these artificial problems confirm that the method used by *Ltree* to find linear combinations performs quite well independently of whether the data is (or is not) linearly separable.

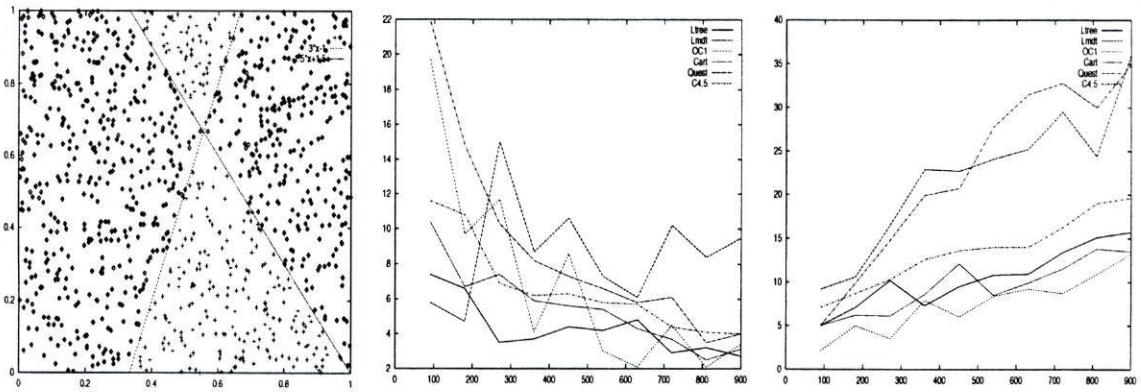


Figure 4.6: Learning curves for XOR data (a) Data set (b) Error Rates (c) Nr. leaves

4.5.2 Evaluation on UCI and StatLog Datasets

Table 4.3: Error rates of C4.5 versus Ltree and C45Oblique

Dataset	*C4.5*		*Ltree*		*C45Oblique*
Australian	14.71±0.6	+	**13.85±0.6**		14.28±0.6
Balance	22.10±0.7	+	8.10±0.6	+	**6.72±1.1**
Breast(W)	5.46±0.5	+	**3.29±0.4**		4.64±0.2
Cleveland	21.87±1.9	+	**16.54±0.8**	+	18.39±1.1
Diabetes	26.14±0.8		24.90±0.5	+	**24.61±1.0**
German	28.63±0.7	+	25.15±0.6	+	**24.65±1.0**
Glass	31.96±2.6		**31.83±2.1**		35.59±2.4
Heart	22.85±2.0	+	**17.00±1.2**	+	18.04±1.4
Hepatitis	20.42±1.6		**19.15±1.9**		**19.15±1.8**
Ionosphere	**10.47±1.2**		11.08±0.7		10.95±1.1
Iris	4.80±0.9	+	**3.00±0.4**	+	3.33±0.6
Satimage	13.65±0.4	+	**11.48±0.2**	+	12.28±0.2
Segment	3.29±0.2		3.49±0.2		**3.06±0.2**
Sonar	25.95±2.2		25.58±2.2		**25.08±2.8**
Vehicle	27.10±1.0	+	**21.59±1.3**	+	22.68±1.1
Votes	3.65±0.4		**3.52±0.5**	−	4.76±0.4
Waveform	24.66±0.4	+	16.80±0.3	+	**16.74±0.4**
Wine	6.93±0.6	+	**2.17±0.8**	+	3.51±0.5

To evaluate our algorithm we performed a $10 * 10$ stratified Cross-Validation (CV) on eighteen datasets some of which come from the *StatLog repository*[6] and others from the *UCI repository*[7] [BKM99]. Datasets were permuted once before each of the ten

[6] `http://www.ncc.up.pt/liacc/ML/statlog/index.html`

[7] `http://www.ics.uci.edu/AI/ML/Machine-Learning.html`

Table 4.4: Summary of Results (error-rate)

	C4.5	*Ltree*	*C45Oblique*
Arithmetic Mean	17.48	14.36	14.91
Geometric Mean	14.00	10.76	11.55
Average Rank	2.7	1.5	1.8
	C4.5 / Ltree	*C4.5 / C45Oblique*	*Ltree / C45Oblique*
Nr. of wins	2 / 16	3 / 15	10 / 8
Nr. of signif. wins(t-test)	0 / 11	1 / 10	4 / 1
Nr. of signif. wins(signal)	0 / 10	1 / 9	4 / 1
Wilcoxon Test	0.0	0.0	0.11

Table 4.5: Summary of Results (Nr. of Leaves)

	C4.5	*Ltree*	*C45Oblique*
Arithmetic Mean	48.56	18.93	31.39
Geometric Mean	25.32	8.91	13.97
Average Rank	2.9	1.2	1.9
	C4.5 / Ltree	*C4.5 / C45Oblique*	*Ltree / C45Oblique*
Nr. of wins	1 / 17	1 / 17	14 / 1
Nr. of signif. wins(t-test)	0 / 17	1 / 17	11 / 1
Nr. of signif. wins(signal)	0 / 17	1 / 15	11 / 1

CV runs. All algorithms were used with default settings. In each iteration of CV all algorithms were trained on the same train partition of the data. Classifiers were also tested on the same test partition of the data.

In the first set of experiments we have compared the univariate decision tree *C4.5* with the multivariate decision trees *Ltree* and *C45Oblique*. To have statistical confidence on the differences we have carried out *paired t-tests* with a confidence level of 99.9%. The null hypothesis is that *C4.5* has equal performance to each one of the other algorithms. The results of the first set of experiments are presented in table 4.3. We observe that *Ltree* performs significantly better than *C4.5* on 11 datasets. Also *C45Oblique* performs significantly better than *C4.5* on 10 datasets and worse on 1 dataset. The summary of these results is shown in Table 4.4 and 4.5. These results clearly illustrate the benefits of using the linear combinations of attributes.

In the second set of experiments we have compared *Ltree* against other decision trees that use multivariate tests at the internal nodes. Minor syntactic modifications of the data were necessary to be able to run *OC1*, *Quest*, and *CART*. All nominal attributes were codified as integers. CART was used with the option of *linear combinations*

active. Also it uses an internal consistence check procedure for the parameters setting. We have used, as default, the following parameters: *Split* = *gini*, *Linear size* = 20, *param* = .2 *and linmax* = 200. These settings work for most of the datasets, although on some of them we needed to increase the *linmax* option, as suggested by *CART*. *Quest* system was used with 10-fold internal stratified cross-validation. It does not allow linear combinations when data contains missing values. Those were codified as -1. *LMDT* was used reserving 10% of the training data for pruning.

Table 4.6 presents the mean and standard deviation of error rates (in percentages). Results were compared using *paired t-tests*. The null hypothesis is that the performance of *Ltree* is equal to each of the other algorithms. A $+$ or $-$ sign means that there was a significant difference at 99.9% confidence level. A $+(-)$ sign attached to a particular algorithm means that *Ltree* was worse (better) than that algorithm at this confidence level. Our system has a good performance overall in terms of error rate. On these datasets, it never performs significantly worse than *LMDT*, *CART*, and *OC1*. It performs significantly better on 11 datasets when comparing to *CART*, *OC1*, and *LMDT*. The most competitive is *Quest*. In comparison against *Quest*, *Ltree* performs significantly better on 5 datasets and worse on 3. Table 4.7 presents a comparative summary of the results in terms of error rate.

Table 4.6: Error rates of Multivariate Trees

Dataset	*Ltree*	*CART*	*OC1*	*LMDT*	*Quest*
Australian	**13.85±0.6**	14.67±0.3	15.60±1.0	16.16±1.4	14.29±0.4
Balance	8.10±0.6	9.01±0.7	**7.49±0.9**	10.73±1.0	8.37±0.1
Breast(W)	**3.29±0.4**	6.79±0.5	4.97±0.6	3.95±0.3	3.38±0.2
Cleveland	16.54±0.8	19.77±1.3	22.42±1.6	20.29±1.6	**15.97±0.3**
Diabetes	24.90±0.5	25.71±0.8	27.76±1.1	27.15±1.6	**22.67±0.4**
German	25.15±0.6	25.73±1.1	26.96±1.5	28.43±0.8	**23.08±0.4**
Glass	**31.83±2.1**	32.93±2.3	34.61±3.8	41.34±4.4	39.35±2.8
Heart	17.00±1.2	25.00±1.3	22.55±2.4	21.18±1.6	**15.96±1.0**
Hepatitis	19.15±1.9	21.62±1.1	20.74±3.4	19.85±2.7	**18.32±1.4**
Ionosphere	11.08±0.7	**10.77±0.7**	11.02±1.4	13.14±2.2	12.13±1.1
Iris	**3.00±0.4**	6.13±0.8	3.73±1.2	4.27±0.5	3.00±0.5
Satimage	**11.48±0.2**	13.63±0.7	13.02±0.3	14.33±0.4	14.21±0.2
Segment	**3.49±0.2**	3.59±0.3	4.83±0.4	4.18±0.4	4.73±0.2
Sonar	25.58±2.2	28.35±1.5	29.01±2.6	26.67±2.7	**25.06±1.1**
Vehicle	**21.59±1.3**	28.12±1.0	31.65±1.1	22.15±1.4	21.91±0.7
Votes	**3.52±0.5**	4.65±0.2	4.97±0.5	5.64±0.4	5.31±0.5
Waveform	16.80±0.3	18.40±0.8	19.18±0.7	17.09±0.6	**15.67±0.4**
Wine	**2.17±0.8**	7.87±2.2	9.87±1.0	4.40±1.3	4.49±1.0

Table 4.7: Summary of Multivariate Trees Results (error rate)

	Ltree	*CART*	*OC1*	*LMDT*	*Quest*
Arithmetic Mean	14.36	16.82	17.24	16.71	14.88
Geometric Mean	10.76	13.88	13.85	13.24	11.79
Average Rank	1.56	3.44	3.83	3.89	2.28

	Ltree / CART	*Ltree / OC1*	*Ltree / LMDT*	*Ltree / Quest*
Nr. of wins	17 / 1	16 / 2	18 / 0	11 / 7
Nr. of signif. wins(t-test)	11 / 0	11 / 0	11 / 0	5 / 3
Nr. of signif. wins(signal)	10 / 0	10 / 0	11 / 0	5 / 3
Wilcoxon Tests	0	0	0	0.52

Table 4.8 presents a summary of the number of leaves with multivariate trees. From this point of view, *Quest* and *LMDT* are the algorithms that produce smaller trees. In both algorithms the output of a multivariate test is a kind of nominal attribute. As such it is used once. This is a justification for the small tree sizes obtained with these systems. In *Ltree* each new attribute is *continuous* and can be used more than once. It obtains better fitting to the data by growing larger trees.

Table 4.8: Summary of Multivariate Trees Results (Nr. of Leaves)

	Ltree	*Cart*	*OC1*	*LMDT*	*Quest*
Arithmetic Mean	18.9	17.92	14.06	8.39	5.23
Geometric Mean	9.10	4.80	8.26	5.71	3.7
Average Rank	3.8	2.7	4.1	2.8	1.7

	Ltree / CART	*Ltree / OC1*	*Ltree / LMDT*	*Ltree / Quest*
Nr. of wins	5 / 13	9 / 9	5 / 12	1 / 15
Nr. of signif. wins(t-test)	3 / 12	3 / 7	1 / 10	1 / 14
Nr. of signif. wins(signal)	3 / 12	3 / 7	2 / 10	1 / 14

4.5.3 Why does *Ltree* Work?

The authors of CART discuss the use of linear combination of attributes in tree learning. They express a view that *"Although linear combination splitting has strong intuitive appeal, it does not seem to achieve this promise in practice...."* [LV88]. This was confirmed in our experiments when comparing systems such as *LMDT* and *OC1* with *C4.5*. The use of linear combination of attributes increases the number of degrees of freedom to obtain better fit to the data, but also increases the variance of the classifier.

In *Ltree* each new attribute is the definition in extension of the linear discriminant function and is therefore continuous. In other words, the new attribute is obtained by applying the linear discriminant function to each example. In typical decision tree systems, the original continuous attributes can be used more than once in each branch of the generated tree. There are three innovations in *Ltree*.

- The linear combinations are explicitly mapped to new attributes. Any new attribute generated during induction is treated similarly as the original ones. No other system do that. All the other systems examined here, implicitly map the linear combinations to attributes. They are only used at the node where they were constructed.
- The new attributes are continuous, and the evaluation of the goodness of a split is accordingly. In *LMDT* the linear combinations are evaluated as nominal attributes that take the values of the class attribute. In *OC1*, *Quest*, and *CART* the linear combinations are treated as Boolean attributes.
- New attributes are propagated down through the tree. This implies that new attributes generated at one decision node can be chosen as test attribute in subsequent nodes.

The linear discriminant function that we use, is not always the best function for representing the linear combinations, since it assumes the equality of the covariance matrices. If this assumption does not hold, the method cannot find the right linear combination, even in the case where the data is linearly separable. This was verified, for instance on the *LS10* data. Although this is a drawback, the linear discriminant function is fast, and always provides a good approximation to a linear combination.

4.5.4 Evaluation of Ltree Variants

In this section we evaluate several variants of *Ltree*. We examine the influence of smoothing, the effect of downward propagation of the new attributes, and the use of different attribute constructors.

4.5.4.1 The Influence of Smoothing

In the first experiment we evaluate the effect of smoothing which is shown in table 4.9. The column *Ltree WS* presents the results of running *Ltree* without smoothing.

Table 4.9: Error rates of Ltree variants

Dataset	*Ltree*	*Ltree WS*		*Mtree*		*Qtree*		*LgTree*
Australian	13.85±0.6	13.90±0.6	−	16.90±0.7		14.24±0.6		**13.53±0.5**
Balance	8.10±0.6	8.10±0.6	−	11.56±0.9	+	**6.44±0.7**		7.93±0.7
Breast(W)	**3.29±0.4**	3.29±0.4		3.29±0.4	−	4.13±0.3		3.56±0.3
Cleveland	16.54±0.8	16.87±0.9		16.77±1.2	−	19.85±0.9		**16.10±0.6**
Diabetes	24.90±0.5	24.94±0.5		24.51±0.9	−	27.57±1.3		**23.95±0.6**
German	25.15±0.6	25.19±0.6		25.23±0.7	−	26.98±1.0	+	**23.91±0.5**
Glass	**31.83±2.1**	32.45±2.0		32.57±2.2		32.05±1.7		32.44±2.1
Heart	17.00±1.2	16.93±1.2		16.85±1.3	−	19.74±1.3		**16.00±0.9**
Hepatitis	19.15±1.9	19.24±2.0		18.87±1.7		**17.52±1.1**		18.64±2.3
Ionosphere	11.08±0.7	10.91±0.6		11.46±0.6	+	**8.34±0.5**		11.40±1.1
Iris	3.00±0.4	3.00±0.4		3.00±0.4		**2.07±0.6**		3.00±0.7
Satimage	**11.48±0.2**	11.60±0.2		11.91±0.3	−	12.29±0.3		12.12±0.4
Segment	3.49±0.2	3.45±0.2		3.60±0.3	−	4.31±0.4		**3.37±0.2**
Sonar	25.58±2.2	25.67±1.9		26.17±1.7		26.56±1.8		**25.26±1.5**
Vehicle	21.59±1.3	21.64±1.2	−	25.61±1.4	+	**15.57±0.9**		21.87±1.1
Votes	**3.52±0.5**	4.05±0.6		3.52±0.5		3.68±0.4		3.52±0.5
Waveform	16.80±0.3	16.84±0.4		16.12±0.5		17.42±0.4	+	**14.82±0.4**
Wine	2.17±0.8	2.17±0.8		2.12±0.8	+	**0.50±0.5**		1.08±0.5

The error rate of *Ltree* is, on average, smaller than *Ltree WS*, although the differences are not significant. The effect of smoothing on tree size is more noticeable. *Ltree* is significantly better than *Ltree WS* on 2 datasets and never worse (see Table 4.11). These results indicate that the main influence of *smoothing* is on tree size.

Table 4.10: Summary of Ltree Variants (error-rate)

	Ltree	*Ltree WS*	*Mtree*	*Qtree*	*LgTree*
Arithmetic Mean	14.36	14.46	15.01	14.40	14.03
Geometric Mean	10.76	10.87	11.24	9.85	10.22
Average Rank	2.56	3.39	3.56	3.39	2.06

	Ltree / Ltree WS	*Ltree / Mtree*	*Ltree / Qtree*	*Ltree / LgTree*
Nr. of wins	12 / 3	11 / 5	12 / 6	5 / 12
Nr. of signif. wins(t-test)	0 / 0	3 / 0	7 / 4	0 / 2
Nr. of signif. wins(signal)	0 / 0	3 / 0	5 / 3	0 / 2

Table 4.11: Summary of Ltree Variants (Nr. of Leaves)

	Ltree	*Ltree WS*	*Mtree*	*Qtree*	*LgTree*
Arithmetic mean	18.9	19.2	17.5	17.6	17.4
Geometric mean	8.9	8.3	8.3	8.7	7.9
Average Rank	3.3	3.3	2.4	3.6	2.4

	Ltree / Ltree WS	*Ltree / Mtree*	*Ltree / Qtree*	*Ltree / LgTree*
Nr. of wins	9 / 5	2 /12	9 / 6	3 / 11
Nr. of signif. wins(t-test)	2 / 0	1 / 3	3 / 4	1 / 5
Nr. of signif. wins(signal)	2 / 0	1 / 3	3 / 3	1 / 3

4.5.4.2 The Effect of Downward Propagation of the New Attributes

There are three reasons that justify the downward propagation of the new attributes. The first one is related to the method used to deal with continuous attributes mainly used on decision tree induction. Nominal variables are used at most once on a decision tree, because in all subsets obtained by splitting a dataset on a nominal attribute, the value of this attribute is constant. A continuous attribute can be used more than once in the tree, because a test on a continuous attribute produces a binary split. In general, on both subsets the attribute is not constant and future splits on this attribute could be useful.

The second reason is related to the fact that each linear combination discriminates only one class. At each node only one attribute is selected. If it is one of the new attributes, this discriminates only one of the classes. The other attributes that discriminate the other classes could be useful later in the tree building process.

The third reason is that at deeper nodes new attributes contain terms based on previously built attributes. That is, the linear combinations that we get at deeper nodes may contain terms that are linear combinations constructed earlier. The downward propagation of the attributes gradually increases the complexity of the attributes generated at deeper nodes. We believe that this feature could be useful for modeling very complex spaces, although there is a danger of overfitting.

To test the hypothesis that downward propagation of the new attributes is useful, we have developed system *Mtree*. The new attributes are still continuous but are *not* propagated downward the tree. The new attributes are a linear composition of the ordered original attributes. We observe that the performance of *Ltree*, in terms of error rate, outperforms *Mtree* on 11 datasets, 3 with statistical significance. With respect to the number of leaves, *Mtree* generates smaller trees in 12 datasets, 3 with

statistical significance. This results indicate that the improvement verified with *Ltree* is due to its ability to generate more complex models.

Controlling the depth of the constructive operator. The use of more powerful representations does not necessarily lead to better results. It can lead to an increase of instability (variance). To address this problem, *Ltree* uses two heuristic rules that limit the applicability of the constructive operator. The first rule defines the depth beyond which the constructive operator cannot be applied. This is a user settable parameter that by default is 5. The second rule considers the number of examples that fall at a node and the number of attributes. The constructive operator is applied only if the number of examples is greater than 3 times[8] the number of attributes.

In almost all the datasets, we have observed a significant decrease in the error rate when using the constructive operator *near the root* of the tree. Constructing new attributes at a depth greater than 6, seemed to have a very small effect on the error rate. For small datasets this is due to the second rule that disables the use of the constructive operator whenever the number of examples is insufficient when compared to the number of attributes. Using the constructive operator at depths greater than 10, lead to a small increase of the error rates, probably due to *overfitting*.

4.5.4.3 The Effect of Using Different Attribute Constructors

We have analyzed the effect of replacing in *Ltree* the attribute constructor by two other discriminant functions. In the first variant, system *Qtree*, we use a *quadratic composition* of the ordered attributes with downward propagation. This means that deeper nodes may contain quadratic composition of quadratic terms. The second variant, system *LgTree*, uses the *logistic discriminant* as constructive operator. In all the other aspects the system is similar to *Ltree*.

The comparative results of *Ltree*, *Qtree*, and *LgTree* are presented in table 4.9. With respect to the datasets used in our study and in relation to the *error rates*, *LgTree* achieves better results overall than the other variants. It is significantly better than *Ltree* in 2 datasets. With respect to the number of leaves *LgTree* generates significantly smaller trees than *Ltree* in 5 datasets and generates larger trees in 1 dataset.

[8] A user settable parameter.

4.5.5 Bias-Variance Analysis

We have carried out also Bias-Variance analysis with the objective of determining the components of the error. This was done for all the algorithms that use different variants of the discriminant functions, *Ltree*, *Qtree*, and *LgTree*. Figure 4.7 presents the decomposition obtained for each dataset. A summary of the average results relative

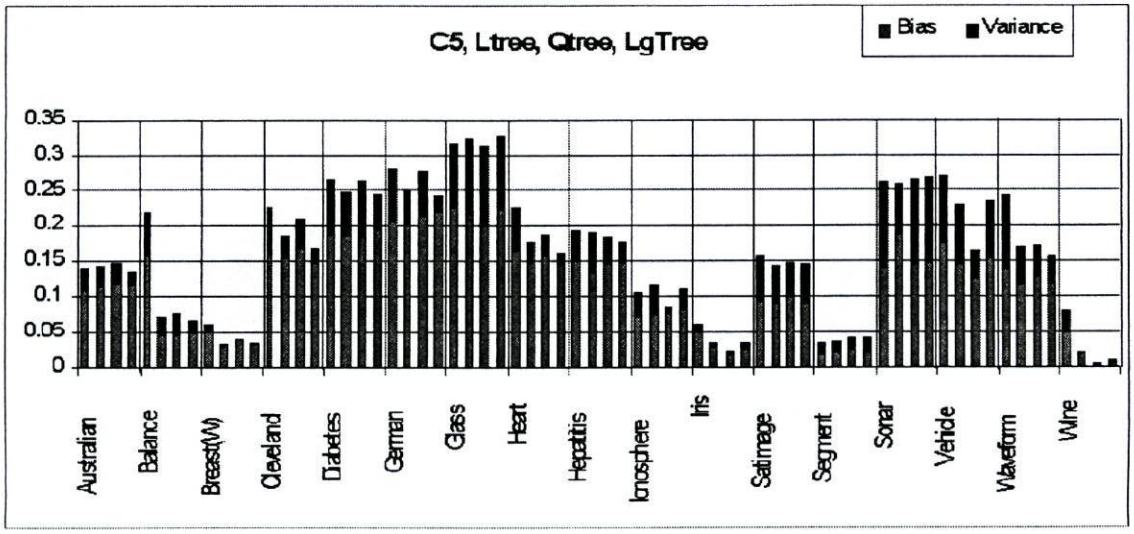


Figure 4.7: Bias Variance decomposition of the error rate. For each dataset the decomposition is shown for C4.5, Ltree, Qtree, and LgTree in this order

to bias-variance decomposition is presented in table 4.12. In comparison to *C4.5* we observe that all the 3 variants of multivariate trees reduce both components. *Qtree* is the algorithm that reduces mostly the *Bias* component, while *LgTree* is the algorithm that reduces mostly the *Variance* component.

Table 4.12: Summary of Bias-Variance decomposition

	C4.5	Ltree	Qtree	LgTree
Bias	12.54	11.19	11.03	11.22
Variance	5.93	4.26	4.26	3.77

4.5.6 How Far from the Best Algorithm?

We have measured the distance of the error rate of an algorithm to the best algorithm on each dataset in terms of the *error margin* (Bernoulli distances) defined earlier.

Table 4.13 presents the average of distances for all datasets. There is strong evidence that if we need to use a learning classifier on new data, and do not have further information, we should first use *Ltree*.

Table 4.13: Averages of Distances to the Best Algorithm

LgTree	*Ltree*	*Qtree*	*C45Obl*	*Mtree*	*Quest*	*LMDT*	*CART*	*OC1*	*C4.5*
0.87	1.17	1.19	1.64	1.72	2.0	3.15	3.54	3.95	4.39

4.5.7 Model Complexity - Minimum Description Length

In this section we show a comparison of the sizes of the different models generated by *Ltree* and C4.5. Because the models use different representation languages, any simple measure, such as "number of nodes", does not capture the complexity of each model. In this study, we analyze the complexity in terms of the code size following the *Minimum Description Length*. We have used the coding schema described in section 3.4.2.

Table 4.14 presents the results. For each dataset, we averaged the code size of the models generated in all the cross-validation experiments. We observe that the code sizes of *Ltree* and *C4.5* are roughly comparable across datasets. *C4.5* generates smaller code sizes in 11 datasets, while *Ltree* generates smaller code sizes in 8 datasets, using *paired t-tests*, with significance level set at 95% (see the right part of table 4.14).

4.5.8 Learning Time

The training time, that is the time needed by the learning algorithm to generate a predictor, is also an important criteria that have to be considered when comparing different algorithms. Here comparisons are less clear as results may strong depend on the implementation details as well on underlying hardware[9]. However the order of magnitude of time complexity is still a useful indicator. Figure 4.8 shows the learning times of multivariate algorithms for each dataset on a log-scale. Table 4.15 presents the relative training times of each algorithm. All training times have been divided by the training time of C4.5 that is used as reference. These results provide an evidence that *Ltree* is much faster than other known multivariate trees including

[9] All algorithms have run on a *Pentium 166MHz, 32M* machine under Linux.

Table 4.14: Minimum Description Length

Dataset	*Tree size*			*Tree + Errors*		
	Ltree		*C4.5*	*Ltree*		*C4.5*
Australian	476.10 ±104.1	+	336.03 ±104.5	1209.07±67.8	+	985.37±66.7
Balance	756.33 ±208.6	−	1138.33 ±151.2	1003.53±177.1	−	1833.93±101.8
Banding	375.17 ±185.8	−	555.20 ±95.0	541.47±146.0	−	678.63±69.2
Breast(W)	211.30 ±1.8		224.43 ±48.8	391.70±25.3	−	509.07±29.3
Cleveland	335.67 ±102.2	−	476.00 ±98.2	660.67±53.0	−	690.33±69.0
Credit	273.63 ±173.0		322.70 ±96.0	1008.40±96.8		976.67±58.0
Diabetes	657.23 ±385.4		643.43 ±230.8	2106.80±249.9	+	1852.23±93.1
German	1354.20 ±625.5		1468.90 ±214.6	3041.80±313.4	+	2656.27±114.3
Glass	1028.47 ±436.2	+	692.17 ±73.9	1253.60±431.3	+	861.93±62.5
Heart	258.93 ±20.3	−	330.80 ±89.4	538.23±17.4		520.67±55.5
Hepatitis	97.73 ±51.9	−	184.67 ±54.1	246.97±21.3	−	279.33±34.2
Ionosphere	867.13 ±53.0	+	339.13 ±71.0	942.13±39.1	+	393.60±49.1
Iris	361.50 ±78.4	+	108.70 ±12.7	379.50±77.7	+	137.37±13.4
Mushroom	205.23 ±8.1	−	226.23 ±15.4	205.23±8.1	−	226.23±15.4
Satimage	17690.70 ±1474.0	+	11046.80 ±426.6	16994.53±679.1	+	13204.93±328.3
Segment	7044.17 ±215.4	+	1373.70±87.1	7328.87±86.6	+	1623.40±83.0
Sonar	298.00 ±39.8	−	332.37 ±31.8	348.70±27.7	−	365.93±32.1
Vehicle	4021.93 ±396.7	+	2464.10 ±416.3	4655.07±343.4	+	3266.10±164.6
Votes	57.20 ±11.2		56.17 ±4.2	159.80±7.7	−	165.83±11.6
Waveform	4740.73 ±635.4		4676.37 ±369.8	7353.90±408.7	+	5979.47±164.7
Wine	871.70 ±7.4	+	144.40 ±15.9	872.03±7.0	+	164.57±19.7
Mean MDL	1999		1292	2440		1780

CART, OC1, LMDT and Quest. As Jordan *et al.* [JJ94] have pointed the training time for divide and conquer algorithms is often orders of magnitude faster than gradient based algorithms. This is one of the aspects why *Ltree* is significantly faster than other oblique decision trees like CART, OC1 and LMDT that use gradient descent approaches to determine hyper-planes.

4.6 Conclusions

We described a new and efficient method for the construction of multivariate decision trees. It exploits constructive induction. The method explores the power of decision trees to split a problem into sub-problems with the power of discriminant functions that build useful decision surfaces. The method can be used either as a *pre-processing* step, as in *C45Oblique*, or incorporated in the decision tree induction, as in *Ltree*.

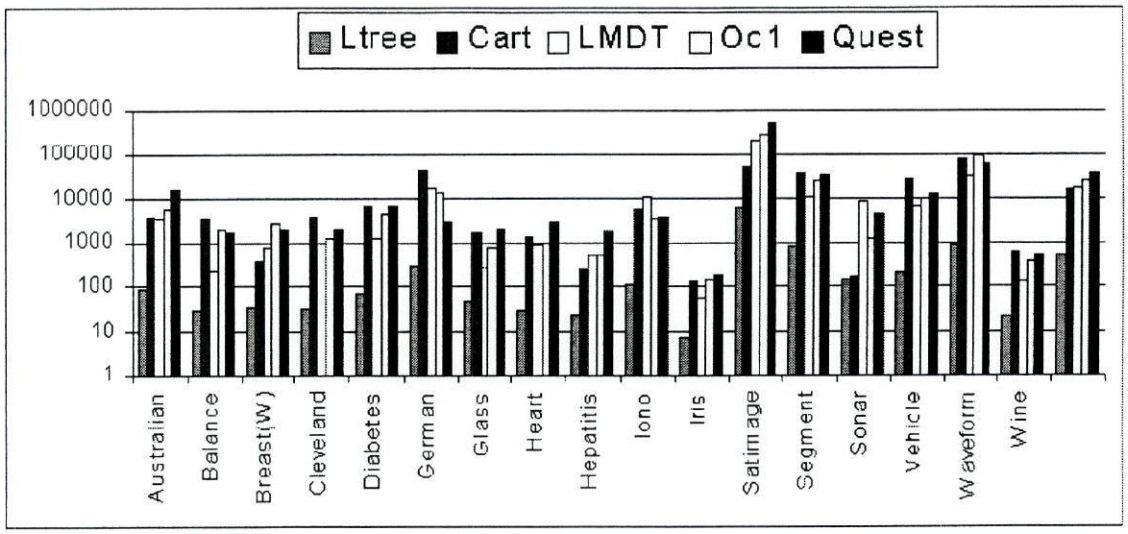


Figure 4.8: The learning time of multivariate algorithms for each dataset.

Table 4.15: Average of relative running times

C4.5	*MTree*	*C45Oblique*	*Ltree*	*Qtree*	*LMDT*	*LgTree*	*Quest*	*CART*	*OC1*
1	1.5	1.54	2.3	3.9	15.8	29.2	32.3	93.8	136.1

The overall results of *C45Oblique*, measured both in terms of error rate and learning time, compare favorably with other approaches. The advantage of *C45Oblique* is that it is a rather simple system incorporating two well-known techniques (*C4.5* and *linear discriminant*). It can be easily implemented without changing those systems. However, system *Ltree* obtains more consistent and somewhat better results both in terms of error rate and number of leaves.

There are two main features that characterize *Ltree*. The first one is the use of constructive induction. When building the tree, new attributes are computed as linear combinations of the previous ones. The new attributes built are propagated downward the tree. New attributes built at lower nodes contain terms based on attributes built higher up. This schema allows building very complex decision surfaces, mainly when using a *quadratic function* as constructive operator. The second aspect is that the number of attributes is variable along the tree, including in two nodes at the same level.

In problems with numerical attributes, attribute combination extends the representational power and relaxes the language bias of univariate decision tree algorithms. The analysis based on Bias-Variance error decomposition [KW96] helps to clarify this

issue. *Ltree* combines a *linear discriminant* that is known to have *high bias*, but *low variance* with a decision tree, known to have *low bias* but *high variance*. This is the desirable composition. We use constructive induction as a way of improving the bias controlling instability.

Another aspect of *Ltree* is the following. When classifying an example, the system outputs an appropriate class probability distribution. It takes into account not only the distribution at the leaf where the example falls, but also a combination of the distributions of the nodes on the path that the example follows.

We have shown that the methodology implemented in *Ltree* achieves good accuracy when compared with other oblique decision trees systems. This is done in an efficient manner leading to reduced learning times.

At each decision node *Ltree* performs two inductive steps: the first one consists of building the discriminant function, the second one consists of applying the decision tree criteria. In the first step the discriminant decision rule is not used (i.e. not used to produce classifications). All decisions, such as stopping, choosing the splitting attribute, etc, are delayed to the second inductive step. The final decision is made by the decision tree criteria. Using Wolpert's terminology [Wol92] the constructive step performed at each decision node represents a bi-stacked generalization. From this point of view, the proposed methodology can be seen as a general architecture for combining algorithms by means of *constructive induction*, a kind of *local* bi-stacked generalization. Other algorithms (e.g. a *neural net*) could replace the constructive operator used. This architecture can be used with *regression problems* using *principal component analysis* as attribute constructor. The attribute constructors used in this chapter are mainly oriented to continuous attributes. Following chapters we extend the proposed architecture, and provide a general method for combining classifiers.

Chapter 5

Multiple Models

Using multiple classifiers for increasing learning accuracy is an active research area. In this section we present a summary of the work done in the area of generating multiple models and combining predictions from different classifiers. The first observation that can be made when dealing with multiple models is that combining identical estimators is useless. A necessary condition for the ensemble approach to be useful is that individual estimators should exhibit a substantial level of disagreement. To obtain uncorrelated classifiers several strategies are presented here. Some methods use a particular learning algorithm trained with different distributions of the training set, giving rise to so called Bagging and Boosting methods. Other methods use different learning algorithms, as in Stacked generalization. The strengths and limitations of each method are discussed in detail.

5.1 Multiple Models

The term *multiple models* or *ensemble of classifiers* is used to identify a set of classifiers whose individual decisions are combined in some way (typically by voting) to classify new examples [Die97]. The first observation that can be made when dealing with multiple models is that combining identical estimators is useless. Thus a necessary condition for the ensemble approach to be useful is that individual estimators have a substantial level of disagreement.

Hansen and Salamon [HS90] first introduced the hypothesis that the ensemble of models is most useful when its member models make errors independently with respect to one another. They proved that when all the models have the same error rate and that the error is less than 0.5 and they make errors completely independently that the expected ensemble error must decrease monotonically with the number of models.

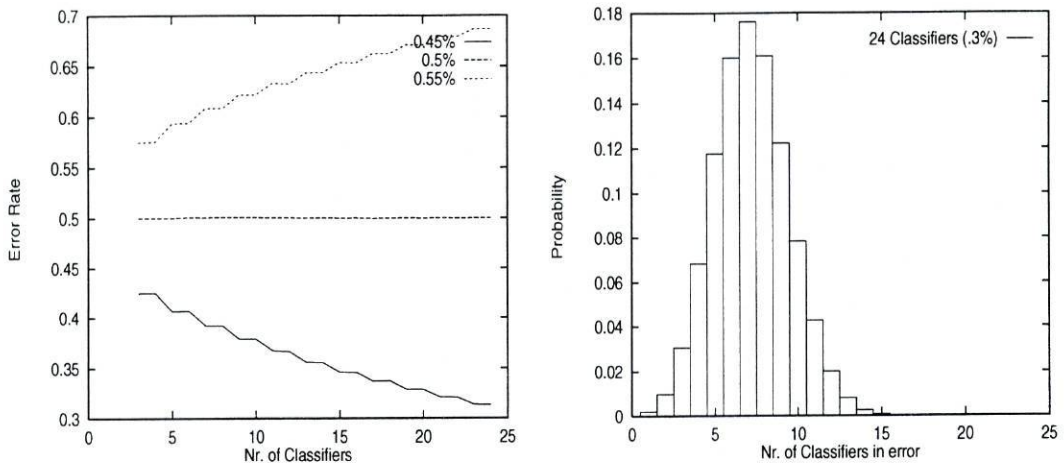


Figure 5.1: (a) Error rate versus nr. of classifiers in an ensemble. (b) Probability that exactly n of 24 classifiers will make an error.

Figure 5.1(a) shows the variation of the error rate obtained by varying the number of classifiers in an ensemble. This is a simulation study, in a two class problem. The probability of observing each class is 50%. A varying number of classifiers, from 3 to 24, is used to classify, by uniform vote, each example. The classifiers have the same probability of making an error, but errors are independent of one another. When this probability is 45%, the error rate of the ensemble monotonically decreases. When it is 55% the error rate monotonically increases, and when it is equal to 50% the error rate of the ensemble stays constant. If the error rates of n classifiers are all equal to $p < 0.5$ and if the errors are independent, then the probability that the majority vote is wrong can be calculated as follows. It is equal to the area under the curve of

binomial distribution showing the probability that more than $n/2$ classifiers are wrong [Die97]. Figure 5.1(b) shows this area for a simulation of 24 classifiers, each having an error of 30%. The area under the curve for more than 12 classifiers is 0.02, which is much less than the error rate of the individual classifiers (0.3).

Tumer and Ghosh [TG95, TG96a, TG96b] have shown how the error rate obtained by a combiner is related to the error rate of a single classifier. The equation that relates both is:

$$Error_{ensemble} = \frac{1 + \rho(N-1)}{N} Error + Error_{Optimal\ Bayes} \tag{5.1}$$

where ρ denotes the correlation among classifier errors, and $Error_{Optimal\ Bayes}$ is the error rate obtained using the Bayes rule assuming that all the conditional probabilities are known. If $\rho = 0$ the error of the ensemble decreases proportionally to the number of classifiers. When $\rho = 1$ the error of the ensemble is equal to the error of a single classifier.

Two models are said to make a *correlated error* when they *both* classify an example of class i as belonging to class j, $i \neq j$. Ali and Pazzani [AP96] present a precise definition of correlated errors: Given an ensemble of classifiers $E = \{\hat{f}_1, ..., \hat{f}_T\}$. Let $\hat{f}_i(x) = y$ denote that the model i as classified example x to class y. Let $f(x)$ denote the true class for x. The *error correlation* between two classifiers i and j is defined as the probability that models $\hat{f}_i$ and $\hat{f}_j$ make the same error:

$$\phi_{ij} = p(\hat{f}_i(x) = \hat{f}_j(x), \hat{f}_i(x) \neq f(x)). \tag{5.2}$$

The degree to which the errors in F are correlated, $\phi_e(F)$, has the following definition:

$$\phi_e(E) = \frac{1}{T(T-1)} \sum_{i=1}^{T} \sum_{j \neq i}^{T} \phi_{ij} \tag{5.3}$$

where T represents the number of models in the ensemble. Ali's definition of *error correlation* does not satisfy the property that the correlation between an object and itself should be 1. We prefer to define the error correlation between pairs of classifiers as the conditional probability of the two classifiers make the same error given that one of them make an error. This definition of *error correlation* lies in the interval $[0 : 1]$ and the correlation between one classifier and itself is 1. Our definition is:

$$\phi_{ij} = p(\hat{f}_i(x) = \hat{f}_j(x) | \hat{f}_i(x) \neq f(x) \vee \hat{f}_j(x) \neq f(x)). \tag{5.4}$$

Taking into account that ϕ_{ij} is symmetric, equation 5.3 can be rewritten as:

$$\phi_e(F) = \frac{2}{T(T-1)} \sum_{i=1}^{T} \sum_{j>i}^{T} \phi_{ij} \tag{5.5}$$

The *error correlation* metric was used as a measure of the *diversity* of an ensemble in [Ska97].

Although the assumptions of the theoretical analysis differs from real world applications, two main ideas that have arisen from this study and that are relevant for us too are[1]:

- Combine models that make uncorrelated errors;
- Each model must perform better than a random guess.

Two questions arise when dealing with *multiple models.* The first one is – *how to combine the predictions of the different models?* The second problem is – *how to generate different models?* We will address these in the following sections.

5.2 Combining Classifiers Predictions

When combining classifiers predictions we can contrast voting versus non-voting methods and dynamic versus static methods.

Voting methods use counts or weighted counts to classify a query point. *Static* methods all base classifiers are consulted to classify a query example while *dynamic* methods characterize the area of expertise of the base classifiers and for a query point only ask the opinion of the experts.

5.2.1 Voting versus Non-voting Methods

5.2.1.1 Voting Methods

Voting is the most common method used to combine classifiers. As pointed out by Ali and Pazzani [AP96], this strategy is motivated by the Bayesian learning theory which stipulates that in order to maximize the predictive accuracy, instead of using just a single learning model, one should ideally use all admissible models in the hypothesis space[2]. The vote of each hypothesis should be weighted by the posterior probability of

[1] Ali and Pazzani [AP96] use more restrictive rules. They require that individual classifiers should make negatively correlated errors and should be accurate.

[2] Domingos [Dom97a] argues that the success of multiple models approach is primarily due to variance reduction and not to being a closer approximation to the Bayesian learning theory.

that hypothesis given the training data. Several variants of the voting methods can be found in the machine learning literature. One is *uniform voting* where the opinion of all base classifiers contributes to the final classification with the same strength. Another common variant is *weighted voting*, where each base classifier has an associated weight, that can change over the time, and strengthens thus the classification attributed to a "good" classifier.

Ho *et al.* in [HHS94] use a *linear discriminant* to compute the weights in a schema similar to *Stacked Generalization* [3].

5.2.1.2 Non Voting Methods

One improvement on uniform vote is obtained when each classifier can produce *class-probability* estimates instead of a simple classification decision. A *class-probability* estimate for data point x is the probability that the true class is k. The class probabilities of all the models can be combined, and so the class probability of the ensemble is: $\frac{1}{L}\sum_{i=1}^{L} p_i$. This method is known in the literature as *distribution summation*. Kitler [Kit98] presented a theoretical perspective on this problem. He examined the sensitivity of the various combination rules to estimation errors, and presented a framework unifying some of the most common used rules, like the *product rule*, the *sum rule*, the *min rule*, the *max rule*, and the *median rule*.

5.2.2 Dynamic versus Static Methods

The distribution of the error rate over the instance space is usually not homogeneous. Depending on the classifier, the error rate will be more concentrated on certain regions of the instance space than in others. *Dynamic* methods will take into account the *query point* and perform a *model selection* to classify the given *query point*. Here we will consider three dynamic methods: the Model Applicability Induction, the Composite Learner, and the SCANN method, which are described in the following in more detail.

5.2.2.1 Model Applicability Induction

Ortega [Ort95] presents the approach called *Model Applicability Induction* for combining predictions from multiple models. The approach consists of characterizing the

[3] *Stacked Generalization* will be presented later in this chapter.

Input: A training set T,
A learning algorithm ϕ
A meta learning algorithm $\Im$
[1] Set $Level_1$ data := {}
[2] **For** each example $\vec{x} \in T$
[3] Generate a classifier $\phi(T - \vec{x})$
[4] **If** ϕ correctly classifies $\vec{x}$
[5] Insert $\{\vec{x}, +\}$ in $Level_1$ data
[6] **Else**
[7] Insert $\{\vec{x}, -\}$ in $Level_1$ data
[8] Learn a classification rule (referee) for $Level_1$ data using $\Im$

Figure 5.2: Model Applicability Induction: the algorithm to generate referees.

situations in which each model is able to make correct predictions. This is done by learning a *referee* for each available model.

Figure 5.2 presents the general algorithm to build a *referee*. The $Level_1$ data is a two class problem. The positive examples are the examples correctly classified by ϕ. The misclassified examples constitute the negative class. The classification rule generated using $Level_1$ data characterizes the examples that classifier ϕ classified correctly. In future instances these referees are first consulted to select the most appropriate prediction model and the prediction of the selected model is then returned.

5.2.2.2 Composite Learner

Ting [Tin96, Tin97] observe that induced models will have relatively poor performance in some regions of the instance space. He proposes a method to characterize these regions for naive Bayes and Instance based learners. The metric for characterizing the predictive accuracy is defined as:

$$Typicality(x) = \frac{Inter\ Concept\ Distance}{Intra\ Concept\ Distance} \tag{5.6}$$

where the *inter concept distance* of an example $\vec{x}$ is defined as its average distance to examples of different classes, and the *intra concept distance* of an example is defined as its average distance to other instances of the same class.

Once predictive accuracy for each model has been characterized, we can combine models by selecting the model with high predictive accuracy for a given query point. The method was successfully applied to several datasets from the UCI repository using a naive Bayes and an Instance based learner.

5.2.2.3 SCANN - Correspondence Analysis followed by a Nearest Neighbor

Merz [Mer98] presents a method for combining models for classification and regression problems. The method consists of transforming a (M x N) matrix of predictions, where each row corresponds to a training example, and each column to a learned model, into a new, low dimensional instance space. This transformation is done using *Correspondence Analysis* techniques. A *Singular Value Decomposition* is used to decompose the matrix of predictions into a set of uncorrelated "basic models" accounting for all the patterns in the original matrix. The decomposition can detect redundant models, and characterize the areas of the example space where each model is superior. The SVD-based representation aids in avoiding the problems associated with correlated predictions without discarding any learned models.

A nearest neighbor strategy is then used to classify test examples in the new representation space.

5.3 Generating Different Models

Several methods for generating multiple models have been described in the literature. Some methods generate classifiers from the same model class, that is using the same representational language, other methods use classifiers from different model class.

5.3.1 Combining Homogeneous Classifiers

In this section we analyze methods that combine models generated by a single algorithm. *Diversity* is one of the requirements when using multiple models. Various strategies has been proposed for generation of different classifiers using the same learning algorithm. Most of them manipulate the training set to generate multiple hypotheses. The learning algorithm runs several times, each time using a different

Input: A data set D,
A learning algorithm $\Im$
A constant L

[1] *For* i := 1 to L
[2] D':= bootstrap sample from D
[3] $H_i = \Im(D')$
[4] *End For*
[5] $H^*(x) = argmax_{y \in Y} \sum_{l=1}^{L} C_i(x) = y$
Output Classifier C^*

Figure 5.3: The Bagging Algorithm

distribution of the training examples. This technique works especially well for *unstable* learning algorithms - algorithms whose output classifier undergoes major changes in response to small changes in the training data.

5.3.1.1 Bagging

Breiman [Bre96b, BFOS84] describes a technique called *bagging*, that produces replications of the training set by sampling with replacement. Each replication of the training set has the same size as the original data. Some examples do not appear in it while others may appear more than once. Such a training set is called a *bootstrap replicate* of the original training set, and the technique is called *bootstrap aggregation* (from which the term *bagging* is derived). For a training set with m examples, the probability of an example being selected is $1 - (1 - 1/m)^m$. For a large m, this is about $1 - 1/e$. Each bootstrap replicate contains, on the average, 36.8% $(1/e)$ of duplicated examples. From each replication of the training set a classifier is generated. All classifiers are used to classify each example in the test set, usually using a uniform vote scheme.

Why does bagging work? Intuitively, taking a majority of several hypotheses, has the effect of reducing the random variability of individual hypotheses. Decision tree is an unstable algorithm and it is exactly the kind of algorithm for which we can obtain large improvements with bagging. When inducing a decision tree there are at least two situations affected by bagging. The first one is the choice of the splitting

attribute at each node. If two or more attributes evaluate similarly with respect to the given evaluation function a small change in the training data can change the chosen attribute. All the descendent subtree will also change. Bagging affects also the choice of a *cut-point*, for *continuous* attributes. C4.5 selects a value in the set of values that appear in the training set. Again a small change in the training set can lead to different cut-point. Aggregating the classifiers by voting could lead to a more complex decision surface. Figure 5.4 illustrate this last situation.

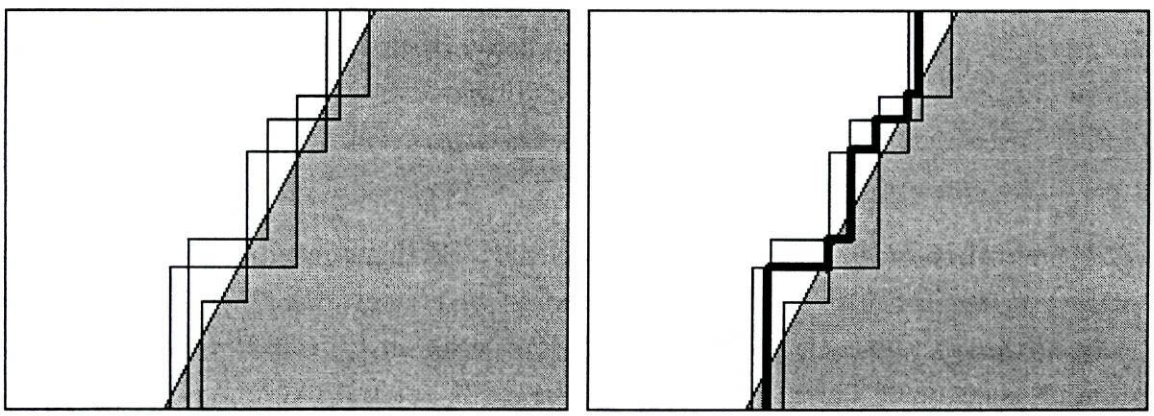


Figure 5.4: The left figure represents the individual decision surfaces of three classifiers. The right figure presents, in bold, the decision surface obtained by uniform vote of the three classifiers.

Breiman [Bre96b] presents a more formal justification. In classification problems, predictor $\Im(x, L)$ predicts a class label $j \in \{1, ..., c\}$. Denote $P(\Im(x, L) = j)$ as the probability of $\Im(x, L)$ predicting class j for the example x when trained with many independent replicates of L. Denote $P(j|x)$ as the probability that input x generates class j. Then the probability that the predictor correctly classifies x is:

$$\sum P(j|x)P(\Im(x, L) = j)$$

If predictor $\Im(x, L)$ is good - in the sense that it is order-correct[4] for most inputs x - then bagging can transform it into a nearly optimal predictor. On the other hand, poor predictors can be transformed into worse ones. The same behavior regarding stability holds. Unstable classifiers are usually be improved by Bagging. Bagging stable classifiers is not a good idea.

[4]A learner is order-correct for an example x if, given many different training sets, it predicts the correct class for x more often than any other.

Domingos [Dom97c] discusses two possible explanations to the impressive gains in accuracy of bagging classifiers. He concludes:

> *Given the empirical evidence, it is unlikely that bagging works because it is an approximation to Bayesian model averaging, and it is plausible that it works at least in part because it corrects for an overly-strong simplicity bias in the underlying learner.*

Some classification methods provide estimates of probabilities $p(j|x)$ that correspond to probability of an object with prediction vector x belongs to class j. If such probabilities are available then the class corresponding to x is estimated as $argmax_j p(j|x)$. For such methods, a natural competitor to bagging by voting is to average the $p(j|x)$ over all bootstrap replications.

How many bootstrap replicates are sufficient? Breiman intuition is: *"more replicates are required with an increasing number of classes"* [Bre96b]. He also notes that *"bagging is almost a dream procedure for parallel computing"* assuming that the running times for a large number of bootstraps is not too relevant. In a simulation experiment varying the number of bootstraps, it was verified that a moderate number of samples(10) is usually sufficient.

In conclusion, bagging is a simple and easy way to improve an existing method. All that it needs is adding a loop in front that selects the bootstrap sample and sends it to the learning algorithm and a back end that does the aggregation of votes. What one loses, in comparison with trees, is a simple and interpretable structure. What one gains is increased accuracy.

Another sampling method that can be used to construct training sets consists of leaving out disjoint subsets of the training data [Die97, HKS96]. For example, the training set can be randomly divided into 10 disjoint subsets. Then 10 overlapping training sets can be constructed by dropping out a different subset of these 10 subsets. The same procedure is employed to construct training sets for 10-fold cross validation, so ensembles constructed in this way are sometimes called *cross-validated committees.*

5.3.1.2 Boosting

Schapire first proposed a general method to convert a *weak learner* into one that achieves arbitrarily high accuracy [Sch90]. The algorithm originally developed was

Input: A data set D,
A learning algorithm $\Im$
A constant L

[1] Initialize for all i: $w_1(i) = 1/m$ — initialize the weights
[2] For $l = 1$ *to* L
[3] For all i: $p_l(i) := wl(i)/\sum_i w_l(i)$
[4] $h_l = \Im(p_l)$ — call the learning algorithm
[5] $e_l = \sum_i p_l(i)[h_l(i) \neq y_i]$ — compute the error of e_l
[6] if $e_l > 1/2$ then
[7] $L = l - 1$
[8] goto 12
[9] $\beta_l := e_l/(1 - e_l)$
[10] For all i: $w_{l+1}(i) := w_l(i)\beta_l^{1-[h_l(x_i)\neq y_i]}$ — compute new weights
[11] End for
Output $h_f(x) = argmax_{y\in Y} \sum_{l=1}^{L} (log\frac{1}{\beta_l})[h_l(x) = y]$

Figure 5.5: The ADABOOST.M1 Algorithm

based on a theoretical model known as the *weak learning model.* This model assumes that there exist *weak* learning algorithms than can do slightly better than random guessing regardless of the underlying probability distribution D used in generating the examples. The work of Schapire shows how to *boost* these weak learners to achieve arbitrarily high accuracy.

The *boosting* algorithm of Freund and Schapire [Sch90, FS96] maintains a weight for each example in the training set that reflects its importance. Adjusting the weights causes the learner to focus on different examples leading to different classifiers. Boosting is an iterative algorithm. In each iteration the weights are adjusted in accordance with the performance of the corresponding classifier. The weight of the misclassified examples is increased. The final classifier aggregates the learned classifiers in each iteration by weighted voting. The weight of each classifier is a function of its accuracy.

Boosting is an active research area. Recently Schapire and Freund presented AdaBoost (from *Adaptative Boosting*) algorithm [FS96] presented in figure 5.5. Like *bagging* this algorithm generates a set of classifiers that participate in voting. Boosting generates classifiers sequentially. In each iteration, the algorithm changes the weights of the

training instances taking into account the errors of previously built classifiers.

There are two reasons for the improvement in performance verified by boosting. The first is that it generates a hypothesis whose error rate is small by combining many hypotheses whose error rate may be large. It seems that boosting may be helpful on learning problems having either of the following two properties. The first one, which holds for many real-world problems, is that the observed examples tend to have varying degrees of difficulty. For such problems, the boosting algorithm tends to generate distributions that concentrate on the harder examples. The second property is that the learning algorithm should be sensitive to changes in the training examples so that significantly different hypotheses could be generated for different training sets. This property is related to the second reason for improvement verified with boosting - variance reduction. Intuitively, taking a weighted majority over many hypotheses has the effect of reducing the random variability of the combined hypothesis.

Schapire *et al.* [SFBL97] extend boosting for multi-class problems using *Error Output Codes*. Margineantu *et al.* [MD97] propose a method for selecting a subset of the hypotheses obtaining nearly the same level of performance as the entire set.

5.3.1.3 Option Trees

Option trees [Bun90, KK97], described earlier in section 2.2.2.2, can also be seen as a method to combine multiple homogeneous classifiers. In comparison with bagging, an *option tree* is deterministic and does not rely on random bootstrap samples. It also uses all the available data to build the tree. Unlike boosting it is easy to produce a parallel version of the algorithm.

5.3.1.4 Error-correcting output codes

The method of *Error-correcting output codes - ECOC* [DB95] was originally designed to solve multi-class problems by solving two-class problems. ECOC represent classes with a set of output bits, where each bit encodes a binary classification task corresponding to a unique partition of the classes. Algorithms that use ECOC learn a function corresponding to each bit. All functions are then combined to generate class predictions. Table 5.1 exemplifies how ECOC encodings differ from two other common output representations. In each of these output representations, each class $c_i \in C$ is assigned a unique *codeword* $s_i = \{s_{i1}, ..., s_{il}\}$ of l code-letters.

Table 5.1: Example of Output Representations

Class Name	Atomic	One per Class	ECOC
Setosa	1	100	00000
Virginica	2	010	11100
Versicolor	3	001	00111

Ricci *et al.* [RA98] present a study on applying ECOCs using nearest neighbor classifiers. The authors conclude:

> *ECOCs drastically reduce the bias component of the error at the cost of moderately increasing the variance.*

5.3.2 Combining Heterogeneous Classifiers

5.3.2.1 Stacked Generalization

Wolpert [Wol92] proposed the *Stacked Generalization* framework. This is a layered architecture. The classifiers at the *level-0* receive as input the original data and each classifier outputs a prediction. Successive layers receive as input the predictions of the immediately preceding layer and the output is passed to the next layer. A single classifier at the most top level outputs the final prediction.

Stacked Generalization is an attempt to minimize the generalization error by using the classifiers in higher layers to learn the type of errors made by the classifiers immediately below. In this perspective it can be seen as an extension to model selection methods, namely *cross-validation*, which use a "winner take all" strategy. The single classifier with the lowest cross-validation error is selected. The idea behind stacked generalization is that there may be more intelligent way to use a set of classifiers. The role of the higher classifiers is to learn how previous classifiers make mistakes, in which classes they agree or disagree and use this knowledge when making predictions.

Most work on the stacking architecture, for example [Wol92, TW97, Ska97, Bre96c], concentrates on a two-layer architecture, which is illustrated in figure 5.6. There are two different phases: the training phase and the application phase. The learning phase consists of the following steps:

1. Train each of the level-0 classifiers using leave-one-out cross-validation as follows. For each example in the training set leave one out and train on the remaining

examples. After training, classify the left-out example. Form a vector from the predictions of all level-0 classifiers and the actual class of that example.

2. Train the level-1 classifier, using as training set the collection of vectors generated in the previous step. The number of examples in the level-1 data is equal to the number of examples in the training set.

3. In step 1, classifiers are generated using a leave-one out method. To fully explore the training set, all level-0 classifiers are retrained using the entire training set. The generated models are used to classify the examples in the test set.

In the application phase, when a new example is presented, this is classified by all level-0 classifiers. The vector of predictions is then classified by the level-1 classifier which outputs the final prediction for the example.

The general framework is not restricted to the described basic model. For example, Breiman [Bre96c] has verified that in some regression problems better results were obtained using 10-fold cross-validation instead of leave-one-out. Ting [TW97] empirically observes that using probability class distributions as level-1 attributes and a linear discriminant as level-1 classifier better results were obtained.

In all cases Stacking is a more sophisticated technique of cross validation that can reduce the error due to reduction of bias [Bre96c, Ska97].

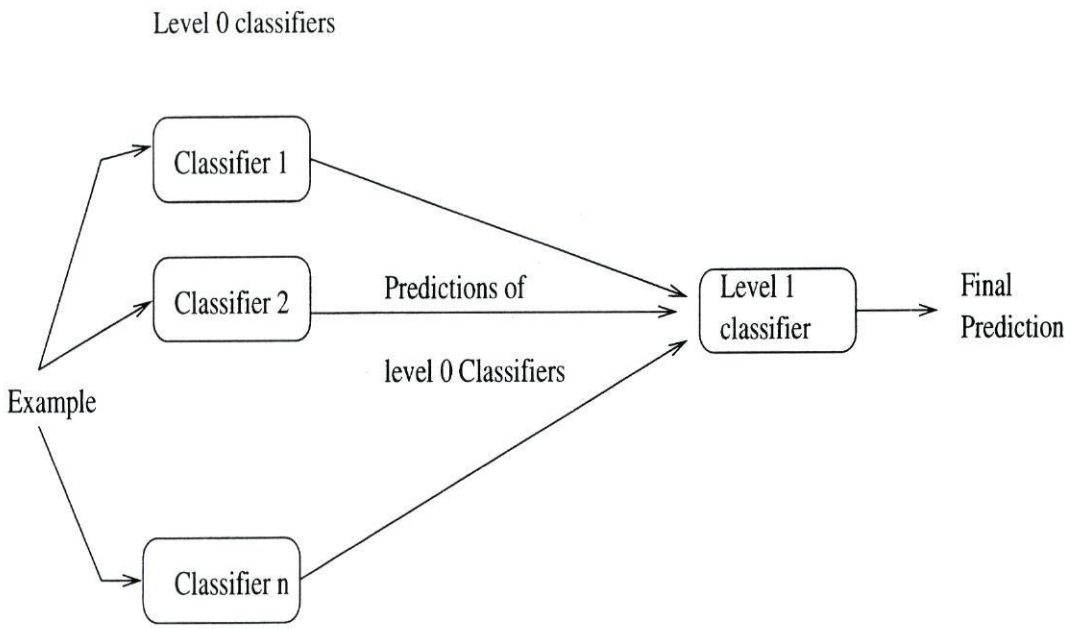


Figure 5.6: Stacked Generalization architecture.

5.3.2.2 Meta Learning

Chan and Stolfo [CS95a] present two schemes for classifier combination: *arbiter* and *combiner*. Both schemes are based on meta learning, where a meta-classifier is generated from meta data, on the basis of the predictions of the base classifiers. An arbiter is also a classifier and it is used to arbitrate among predictions generated by different base classifiers. The training set for the arbiter is obtained from all the available data using a selection rule. An example of a selection rule is *"Select the examples whose classification cannot be predicted consistently using the base classifiers"*. This arbiter, together with an arbitration rule, determines the final classification on the basis of the base predictions. An example of an arbitration rule is *"Use the prediction of the arbiter when the base classifiers cannot obtain a majority"*. Later [CS95b, CS97], this framework was extended using arbiters/combiners in an hierarchical fashion, generating arbiter/combiner binary trees. An arbiter/combiner tree is a hierarchical structure composed of arbiters/combiners that are computed in a bottom-up, binary-tree fashion. An arbiter / combiner is initially learned from the output of a pair of base classifiers and recursively, an arbiter / combiner is learned from the output of two arbiters. For k classifiers there are $log_2(k)$ levels generated.

Skalak [Ska97] discusses methods for combining classifiers. He provides a very thorough analysis of several algorithms, most of which are based on *Stacked Generalization*, and shows where these are able to improve the performance of *Nearest Neighbor* classifiers.

5.4 Hybrid Systems

Results of empirical comparisons of existing learning algorithms [MST94] illustrate that each algorithm has certain *selective superiority*. It is best for some, but not all tasks. This is the motivation behind designing algorithms that can fit the data with *heterogeneous* representations. That is, different regions of the input space are approximated using different types of models.

Brazdil and Torgo [BT90], in a noticeable earlier work, integrate models generated by decision trees and other rule learners using different partitions of the training data. Decision trees are converted to rules by generating a rule for each path in the tree. The final output is a set of rules, generated by selecting the rules with high confidence level from the pool of rules.

Domingos [Dom98a] proposes CMM, a meta-learner that seeks to retain most of the accuracy gains of multiple model approaches, while still producing a single comprehensible model. CMM generates a new training set, composed of a large number of examples generated and classified according to the ensemble, plus the original examples. CMM was used with C4.5RULES as the base learner, and with bagging as the multiple-model methodology. On 26 benchmark datasets, CMM retains on average 60% of the accuracy gains obtained by bagging relative to a single run of C4.5RULES, while producing a rule set whose complexity is typically a small multiple of the complexity of C4.5RULES.

Brodley [Bro95b, Bro93] presents *Model Class Selection - MCS system*, a hybrid algorithm that combines, in a single tree, nodes that are univariate tests, multivariate tests generated by *linear machines* and instance based learners. At each node MCS uses a set of *If-Then* rules to perform a heuristic best-first search for the best hypothesis for the given partition of the dataset. The set of rules incorporates knowledge of experts. One of such rules is given in Fig. 5.7. *MCS* uses a dynamic search control

If the number of instances is less than the capacity of a hyper plane
Then find the best univariate test
Else form a linear test LT_n of all n features

Figure 5.7: MCS rule base: an example.

strategy to perform an automatic model selection. *MCS* builds trees which can apply a different model in different regions of the instance space.

Kohavi [Koh96] presents a hybrid system that generates a univariate decision decision tree using naive Bayes classifiers at the leaves. This approach attempts to utilize the advantages of both decision trees (i.e., segmentation) and naive Bayes (evidence accumulation from multiple attributes). The author claims that the model retains the interpretability of naive Bayes and decision trees, while resulting in classifiers that frequently out perform both constituents, especially on large datasets.

Domingos [Dom97b, Dom96] presents *RISE* system that unifies a rule learner and an instance based learner. Instances are treated as maximally specific rules, and classification is performed using a best-match strategy. Rules are learned by gradually generalizing instances until no improvement in accuracy is obtained.

5.5 Discussion

Earlier results of *boosting* and *bagging* are quite impressive. Quinlan [Qui96a] conducted experiments with boosting and bagging with 10 iterations (i.e. generating 10 classifiers). He observed a reduction of the error rate in between 10% and 19%. Quinlan argued that these techniques are mainly applicable to unstable classifiers. As Breiman [Bre96a] pointed out, both techniques require that the learning system is not stable, resulting in different classifiers when there are small changes in the training set. Bauer and Kohavi [BK99] presented an empirical comparison between *boosting* and *bagging*, and argue that *boosting*, when applied to decision trees, is able to reduce the *bias* component at the cost of increasing tree size. Quinlan [Qui96a] and Bauer and Kohavi [BK99] agree that both approaches substantially improve predictive accuracy. Boosting shows the greater benefit, but it produces severe degradation on some datasets. Ali and Pazzani [AP96] have pointed out that the number of training examples needed by boosting increases as a function of the accuracy of the learned model. Another view is put forward by Bauer and Kohavi [BK99]. They refer that the main problem with *boosting* seems to be its lack of robustness to noise. This is expected, because noisy examples tend to be misclassified, and their weight will consequently increase. They present some examples where the performance of boosting models degraded, when compared to the original model. They also point that bagging improves performance in *all* datasets used in the experimental evaluation. They conclude that although *boosting* is on average better than bagging, it is *not* uniformly better than bagging.

Boosting and bagging require a considerable number of member models because they rely on varying the data distribution to get a diverse set of models from a single learning algorithm. Most studies use a substantial ensemble of classifiers. For example, Bauer and Kohavi [BK99] use 25 classifiers, Freund and Schapire [FS96] and Breiman [Bre98] use 100 classifiers. Recently Quinlan [Qui99] has analyzed boosting with small ensembles of classifiers. He has used 3 classifiers, and proposed a method to merge the 3 trees into one large tree that represents the ensemble.

Zheng [Zhe98] has applied *boosting* with a *naive Bayes* classifier and states:

> *We implemented a boosting algorithm for a naive Bayes classifier using a similar method to that for boosting a decision tree. Although the algorithm achieves higher accuracy than the naive Bayesian classifier in some domains, the overall accuracy improvement over the naive Bayesian classifier*

> *in a large set of natural domains is very marginal. The reason might be that boosting implicitly requires instability of the boosted learning system.*

Effective improvement of a committee of naive Bayes classifiers was obtained using different subsets of attributes. Again citing Zheng:

> *A naive Bayesian classifier is not stable in the sense that a small change on the attribute set could lead to very different classifiers. Moreover, due to the attribute independence assumption, a naive Bayesian classifier built on an attribute subset might perform better than a naive Bayesian classifier created using all the attributes.*

Note that this technique was also used by Skalak [Ska97] and Bay [Bay98] for committees of Nearest Neighbor Classifiers.

Wolpert [Wol92] says that successful implementation of *Stacked Generalization* for classification tasks is a *"black art"*, and the conditions under which stacking works are still unknown:

> *There are currently no hard and fast rules saying what $level_0$ generalizers should we use, what $level_1$ generalizer one should use, what k numbers to use to form the $level_1$ input space, etc.*

Recently, Ting and Witten [TW97] have shown that successful stacked generalization requires the use of output class distributions rather than class predictions. In their experiments only the MLR algorithm (a linear discriminant) was suitable for level-1 generalizer.

5.6 Summary

In this chapter we have presented and discussed the relevant work in a *"hot-topic"* area of machine learning and statistics: combining classification algorithms. We have presented an analysis of the problem, and presented two necessary, although not sufficient, conditions for successful combination of classifiers: they should be diverse and also perform at least better than a random guessing. We have reviewed several relevant methods for combining classifiers including voting, stacking, bagging, and boosting.

Chapter 6

Cascade Generalization

In this chapter we present a new general method for merging classifiers - Cascade Generalization. The basic idea is to run sequentially the given set of classifiers, and at each step perform an extension of the original data set by the insertion of new attributes. The new attributes represent class probability distributions given by the base classifiers. This constructive step extends the representational language for the high level classifiers, relaxing their bias. Cascade Generalization produces a single but structured model for the data that combines the model class *representation of the base classifiers. We have performed an empirical evaluation of Cascade composition of three well known classifiers:* Naive Bayes, Linear Discriminant, *and* C4.5. *Composite models show a significant increase of performance, when compared with the corresponding single models.*

6.1 Introduction

In the conclusions of chapter 4 we have discussed the methodology of discriminant trees as a kind of bi-stacked generalization. This architecture is somewhat specific. It uses discriminant functions at the base level and it is restricted to two classifiers. In this chapter we present a more general method for combining classifiers – *Cascade Generalization* – which can be used to combine more than two classifiers. The proposed method belongs to the family of stacking algorithms in the sense that learning occurs in several layers. Cascade Generalization is sequential in nature. When cascading algorithms the high level classifiers use the original attributes extended with the probability class distributions provided by the base classifiers.

The next section provides a detailed description of the method.

6.2 Cascade Generalization

Consider a learning set $D = (\vec{x_n}, y_n)$ with $n = 1, ..., N$, where $\vec{x_i} = [x_1, ..., x_m]$ is a multidimensional input vector, and y_n is the output variable. Since the focus of this paper is on classification problems, y_n takes values from a set of predefined values, that is $y_n \in \{Cl_1, ..., Cl_c\}$, where c is the number of classes. A classifier $\Im$ is a function that is applied to the training set D to construct a model $\Im(D)$. The generated model is a mapping from the input space X to the discrete output variable Y. When used as a predictor, represented by $\Im(\vec{x}, D)$, it assigns a y value to the example $\vec{x}$. This is the traditional framework for classification tasks. Our framework requires that the predictor $\Im(\vec{x}, D)$ outputs a vector representing conditional probability distribution $[p1, ..., pc]$, where p_i represents the probability that the example $\vec{x}$ belongs to class i, i.e. $P(y = Cl_i|\vec{x})$. The class that is assigned to the example $\vec{x}$, is the one that maximizes this last expression. Most of the commonly used classifiers, such as *naive Bayes* and *Discriminant*, classify each example in this way. Other classifiers, for example *C4.5*, have a different strategy for classifying an example, but it requires few changes to obtain a probability class distribution.

We define a constructive operator $\varphi(\vec{x}, \mathcal{M})$ where $\mathcal{M}$ represents the model $\Im(D)$ for the training data D, while $\vec{x}$ represents an example. For the example $\vec{x}$ the operator φ concatenates the input vector $\vec{x}$ with the output probability class distribution. If the operator φ is applied to all examples of dataset D' we obtain a new dataset D''. The cardinality of D'' is equal to the cardinality of D' (i.e. they have the same

number of examples). Each example in $\vec{x} \in D''$ has an equivalent example in D', but augmented with $\#c$ new attributes, where $\#c$ represents the number of classes. The new attributes are the elements of the vector of class probability distribution obtained when applying classifier $\Im(D)$ to the example $\vec{x}$. This can be represented formally as follows:

$$D'' = \Phi(D', \mathcal{A}(\Im(\mathcal{D}), \mathcal{D}')) \tag{6.1}$$

Here $\mathcal{A}(\Im(\mathcal{D}), \mathcal{D}')$ represents the application of the model $\Im(D)$ to data set D' and represents, in effect, a dataset. This dataset contains all the examples that appear in D' extended with the probability class distribution generated by the model $\Im(D)$.

Cascade generalization is a sequential composition of classifiers, that at each generalization level applies the Φ operator. Given a training set L, a test set T, and two classifiers $\Im_1$, and $\Im_2$, Cascade generalization proceeds as follows: Using classifier $\Im_1$, generates the $Level_1$ data:

$$Level_1 train = \Phi(L, \mathcal{A}(\Im(L), L)) \tag{6.2}$$

$$Level_1 test = \Phi(T, \mathcal{A}(\Im(L), T)) \tag{6.3}$$

Classifier $\Im_2$ learns on $Level_1$ training data and classifies the $Level_1$ test data:

$$\mathcal{A}(\Im_2(Level_1 train), Level_1 test)$$

These steps perform the basic sequence of a cascade generalization of classifier $\Im_2$ after classifier $\Im_1$. We represent the basic sequence by the symbol ∇. The previous composition could be shortly represented by:

$$\Im_2 \nabla \Im_1 = \mathcal{A}(\Im_2(Level_1 Train), Level_1 Test)$$

which, by applying equations 6.2 and 6.3, is equivalent to:

$$\Im_2 \nabla \Im_1 = \mathcal{A}(\Im_2(\Phi(L, \mathcal{A}(\Im_1(L), L))), \Phi(T, \mathcal{A}(\Im_1(L), T)))$$

This is the simplest formulation of *Cascade Generalization*. Some possible extensions include the composition of n classifiers, and the parallel composition of classifiers.

A composition of n classifiers is represented by:

$$\Im_n \nabla \Im_{n-1} \nabla \Im_{n-2} ... \nabla \Im_1$$

In this case, Cascade Generalization generates *n-1* levels of data. The final model is the one given by the $\Im_n$ classifier. This model could contain terms in the form of conditions based on attributes build by the previous built classifiers.

A variant of cascade generalization, which includes several algorithms in parallel, could be represented in this formalism by:

$$\Im_n \nabla [\Im_1, ..., \Im_{n-1}] =$$
$$\mathcal{A}(\Im_n(\Phi_p(L, [\mathcal{A}(\Im_1(L), L), ..., \mathcal{A}(\Im_{n-1}(L), L)])), (\Phi_p(T, [\mathcal{A}(\Im_1(L), T), ..., \mathcal{A}(\Im_{n-1}(L), T)])))$$

The algorithms $\Im_1$, ..., $\Im_{n-1}$ run in parallel. The operator

$$\Phi_p(L, [\mathcal{A}(\Im_1(L), L), ..., \mathcal{A}(\Im_{n-1}(L), L)])$$

returns a new data set L' which contains the same number of examples as L. Each example in L' contains $(n-1) \times \#cl$ new attributes, where $\#cl$ is the number of classes. Each algorithm in the set $\Im_1, ..., \Im_{n-1}$ contributes with $\#cl$ new attributes.

6.3 An Illustrative Example

In this example we will consider the UCI data set *Monks-2* [Tea91]. The *Monks* data sets describe an artificial robot domain and are quite well known in the Machine Learning community. The robots are described by six different attributes and classified into one of two classes. We have chosen the *Monks-2 problem* because it is known that this is a difficult task for systems that learn decision trees in attribute-value formalism. The decision rule for the problem is: **"The robot is O.K. if exactly two of the six attributes have their *first* value".** This problem is similar to *parity* problems. It combines different attributes in a way which makes it complicated to describe in DNF or CNF using the given attributes only.

Some examples of the training data are presented:

head	body	smiling	holding	color	tie	Class
round	round	yes	sword	red	yes	not_Ok
round	round	no	balloon	blue	no	OK

Using 10-fold cross-validation, the error rate of *C4.5* is 32.9%, and of *naive Bayes* is 34.2%. The composite model C4.5 after *naive Bayes*, $C4.5 \nabla naiveBayes$, operates as

follows. The $Level_1$ data is generated, using the *naive Bayes* as the classifier. Naive Bayes builds a model from the original training set. This model is used to compute a probability class distribution for each example in the training and test set. The $Level_1$ is obtained by extending the train and test set with the probability class distribution given by the naive Bayes. The examples shown earlier take the form of:

head	body	smiling	holding	color	tie	P(OK)	P(not Ok)	Class
round	round	yes	sword	red	yes	0.135	0.864	not_Ok
round	round	no	balloon	blue	no	0.303	0.696	OK

where the new attribute *P(OK)* (*P(not OK)*) represent the probability that the example belongs to class *OK*(*not OK*).

C4.5 is trained on the *Level_1* training data, and classifies the *Level_1* test data. The composition $C4.5\nabla NaiveBayes$, obtains an error rate of 8.9%, which is substantially lower than the error rates of both *C4.5* and *naive Bayes*. None of the algorithms in isolation can capture the underlying structure of the data. In this case, Cascade was able to achieve a notable increase of performance. Figure 1 presents one of the trees generated by $C4.5\nabla naiveBayes$.

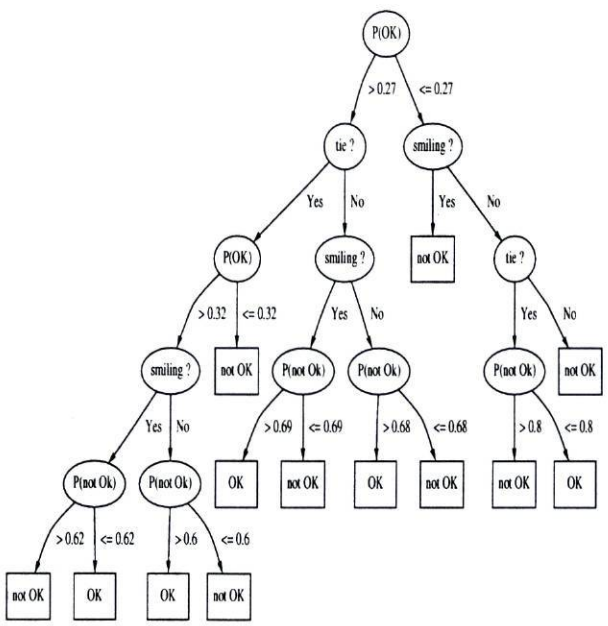


Figure 6.1: Tree generated by C4.5 ∇Bayes

The tree contains a mixture of some of the original attributes (smiling, tie) with some of the new attributes constructed by *naive Bayes* (*P(OK), P(not Ok)*). At the root of the tree appears the attribute *P(OK)*. This attribute represents a particular class

probability (Class = OK) calculated by *naive Bayes*. The decision tree generated by C4.5 uses the constructed attributes given by Naive Bayes, but redefining different thresholds. Because this is a two class problem, the Bayes rule uses $P(OK)$ with threshold 0.5, while the decision tree sets the threshold to 0.27. Those decision nodes are a kind of function given by the Bayes strategy. For example, the attribute *P(OK)* can be seen as a function that computes $p(Class = OK|\vec{x})$ using the Bayes theorem. On some branchs the decision tree performs more than one tests of the class probabilities. In a certain sense, this decision tree combines two representation languages: that of naive Bayes with the language of decision trees. The constructive step performed by *Cascade Generalization*, inserts new attributes that incorporate new knowledge provided by naive Bayes. It is this new knowledge that allows the significant increase of performance verified with the decision tree, despite the fact that naive Bayes cannot fit well complex spaces. In the *Cascade* framework lower level learners delay the decisions to the high level learners. It is this kind of collaboration between classifiers that Cascade Generalization explores.

6.4 Discussion

Cascade Generalization belongs to the family of stacking algorithms. Wolpert defines Stacking Generalization [Wol92] as a general framework for combining classifiers. It involves taking the predictions from several classifiers and using these predictions as the basis for the next stage of classification.

Cascade Generalization may be regarded as a special case of Stacking Generalization mainly due to the layered learning structure. Some aspects that make Cascade Generalization particular, are:

- The new attributes are continuous. They take the form of a probability class distribution. Combining classifiers by means of categorical classes looses the strength of the classifier in its prediction. The use of probability class distributions allows us to explore that information.
- All classifiers have access to the original attributes. Any new attribute built at lower layers is considered exactly in the same way as any of the original attributes.
- Cascade Generalization does not use internal cross-validation. This aspect affects the computational efficiency of Cascade Generalization.

Many of these ideas has been discussed in literature. Ting [TW97] has used probability class distributions as level-1 attributes, but did not use the original attributes. Skalak [Ska97] refers that Schaffer has used the original attributes and class predictions as level-1 attributes, but with disappointing results. In our view this could be explained by the fact that he combines three algorithms with similar behavior from a bias-variance analysis: decision trees, rules, and neural-networks (see section 6.6.2 for more details on this point). Chan and Stolfo [CS95a] have used the original attributes and the class predictions in a scheme denoted *class-attribute-combiner*. In the experiments with two datasets, using two decision trees as base classifiers, the class-attribute-combiner exhibited higher accuracy in one of them.

Exploiting all these aspects is what makes Cascade Generalization succeed. Moreover, this particular combination implies some *conceptual* differences.

- While Stacking is parallel in nature, Cascade is sequential. The effect is that intermediate classifiers have access to the original attributes plus the predictions of low level classifiers. An interesting possibility, that has not been explored in this dissertation, is to provide the classifier$_n$ with the original attributes plus the predictions provided by classifier$_{n-1}$ only.

- The ultimate goal of Stacking Generalization is combining predictions. The goal of Cascade Generalization is to obtain a model that can use terms in the representation language of lower level classifiers. In the *Cascade* framework lower level learners delay the final decision to the high level learners. In general, none of the new attributes can, in isolation, determine a class [1]. This is only possible when considering the set of new attributes introduced by a given classifier.

- *Cascade Generalization* provides rules to choose the low level of classifiers and the high level classifiers. This aspect will be developed in the following sections.

6.5 Empirical Evaluation

To evaluate the proposed methodology we have considered three base algorithms: a decision tree (C4.5), naive Bayes, and linear discriminant. We have evaluated all pairwise combinations plus two combinations of all the classifiers. For each dataset we have estimated the error rate of a particular combination using 10-fold cross-validation.

[1] Except for two class problems.

Table 6.1: Results of Cascade Generalization (error rates)

Dataset	*Bay*∇*Bay*		*Bay*∇*Dis*		*Bay*∇*C4.5*		*Dis*∇*Dis*		*Dis*∇*Bay*		*Dis*∇*C4.5*
Australian	14.69±0.5	+ +	13.61±0.2	+	14.16±0.6		14.06±0.1	+ +	**12.72±0.4**		+ 14.15±0.7
Balance	7.06±1.1	+	8.37±0.1	− −	23.38±1.0	+	8.41±0.1	+ −	11.44±0.8	−	22.23±0.8
Banding	22.36±0.9		21.99±0.8	+ +	**18.76±1.2**		23.28±1.4		22.01±1.6		+ 22.33±1.7
Breast	2.83±0.1	− +	3.26±0.1	− +	3.42±0.2		4.13±0.1	+	2.75±0.1	−	5.08±0.4
Cleveland	17.28±0.9		**16.03±0.4**	− +	20.35±1.5		16.07±0.5		16.35±0.5	−	21.77±1.9
Credit	14.91±0.4	+ +	**13.35±0.3**		13.97±0.6		14.22±0.1	+ +	13.59±0.4		14.34±0.3
Diabetes	24.71±0.6	+	**22.44±0.3**	−	25.33±0.8		22.71±0.2		23.51±0.6	−	25.99±0.8
German	− 25.48±0.6	+	23.17±0.6	−	28.56±0.5		**23.03±0.5**		23.73±0.6	−	28.58±0.7
Glass	37.48±1.7		35.79±1.7	+	**30.63±2.8**		36.25±1.4		35.89±1.6	+	31.63±2.8
Heart	16.67±0.7		16.30±0.5	−	21.74±1.5		16.37±0.4	+	**15.56±0.6**	−	22.89±1.9
Hepatitis	15.95±1.6	+	17.52±1.3		18.44±1.9		21.60±2.0	+	16.30±1.2		21.15±1.8
Ionosphere	9.76±0.7	+ +	9.14±0.3	+ +	**8.57±0.8**		13.38±0.8	+	10.42±0.4	+	10.47±1.2
Iris	3.80±0.5	−	3.27±0.9	+	3.67±1.0		**2.00±0.0**	− +	3.00±0.4	−	4.73±0.9
Monks-1	25.21±0.4	−	33.33±0.0	+	3.56±1.8	−	41.07±1.5	+	25.01±0.0	+ −	20.21±4.2
Monks-2	+ 30.31±3.0		34.33±0.9	−	34.19±0.6		35.06±0.8		34.07±0.6	−	34.21±0.3
Monks-3	1.76±0.8	− +	14.13±0.3	+	0.00±0.0	+	20.76±0.8	+	2.77±0.0	−	22.80±0.3
Mushroom	+ 1.85±0.0	+	3.13±0.0	+	0.00±0.0		6.86±0.0	+ +	1.77±0.0	−	6.86±0.0
Satimage	+ 18.82±0.1	+ −	16.57±0.1	+ −	15.61±0.1	+	15.59±0.1	+ +	14.84±0.1	+	13.63±0.4
Segment	+ 9.41±0.2	+ +	7.91±0.1	+ −	3.78±0.3	+	7.93±0.1	− +	9.29±0.1	+	3.27±0.2
Sonar	25.59±1.4	+ +	23.72±1.1	+ +	**21.84±2.0**		24.81±1.2	+	24.93±1.4		25.96±2.1
Vehicle	39.16±1.0	+ −	25.34±0.7	+ −	28.52±0.8		22.00±0.3	− +	23.54±0.9	− +	26.33±1.2
Votes	10.00±0.3	+	5.28±0.2	+ −	4.41±0.3		5.43±0.2	+	5.43±0.2	+	3.56±0.5
Waveform	+ 16.42±0.3	+	15.24±0.2	− +	21.80±0.7	+	**14.45±0.2**	− +	16.93±0.4	−	24.65±0.4
Wine	2.62±0.6		**1.06±0.6**	− +	4.14±0.9		1.31±0.7		2.01±0.7	−	5.83±1.2

The process was repeated 10 times, each time using a different permutation of the dataset. The final estimate for the error rate is the average of the 10 runs of 10-fold cross-validation.

Tables 6.1 and 6.2 present the results of all pairwise combinations of the three base classifiers and the most promising combination of the three models. Each column corresponds to a *Cascade Generalization* combination. For each combination we have conducted *paired t-tests*. All composite models are compared against its components using *paired t-tests* with significance level set to 99.9%. The +(−) signs indicate that the combination (e.g. C4∇Bay) is significantly better than the component algorithms (i.e. C4.5 and Bayes). The results are summarized in Table 6.3. The first line shows the arithmetic mean across all datasets. It shows that the good combinations use C4.5 at top level, as in *C4.5*∇*naive Bayes* or *C4.5*∇*Discrim*∇*naive Bayes*. This is confirmed by the second line that shows the geometric mean. The third line that shows the average rank of all base and cascade combinations, computed for each dataset by assigning rank 1 to the most accurate algorithm, rank 2 to the second best and so on. In the remainding three lines a particular cascade combination is compared against the top-level algorithm used in the combination. The first one of

Table 6.2: Results of Cascade Generalization (error rates)

Dataset	*C4.5*∇*C4.5*	*C4.5*∇*Dis*	*C4.5*∇*Bay*	*C4.5*∇*Disc*∇*Bay*	*C4.5*∇*Bay*∇*Disc*
Australian	14.74±0.5	13.99±0.9	15.41±0.8	14.24±0.5	15.34±0.9
Balance	21.87±0.7 + +	5.42±0.7 + +	**4.78±1.1** + + +	5.34±1.0 + + +	6.77±0.6
Banding	23.77±1.7	21.73±2.5	22.75±1.8	21.48±2.0	22.18±1.5
Breast	5.36±0.5 +	4.13±0.1 +	**2.61±0.1** + +	2.62±0.1 + +	2.69±0.2
Cleveland	22.27±2.2 −	19.93±1.0 +	18.31±1.1	18.25±2.2 − −	20.54±1.6
Credit	14.21±0.6	13.85±0.4	15.07±0.7	14.84±0.4	13.75±0.6
Diabetes	26.05±1.0 + −	24.51±0.9 −	26.06±0.7 −	25.02±0.9 − −	25.48±1.4
German	28.61±0.7 + −	24.60±1.0 + −	26.20±1.1 + − −	26.28±1.0 + −	25.92±1.0
Glass	32.02±2.4	36.09±1.8	33.60±1.6	34.68±1.8	35.11±2.5
Heart	23.19±1.9 +	18.48±1.5 −	19.30±1.9 + − −	18.67±1.0 + − −	18.89±1.1
Hepatitis	20.60±1.5	19.89±2.3 +	16.63±1.3 + +	15.97±1.9	17.21±2.1
Ionosphere	10.21±1.3 +	10.79±0.8	11.55±1.0 +	11.28±0.8 +	10.98±0.6
Iris	4.80±0.9 −	3.53±0.8	5.00±0.8 −	4.53±0.9 −	4.13±1.1
Monks-1	2.70±0.8 +	3.52±1.8 +	1.49±1.7 + + +	**0.78±1.2** + +	1.20±1.2
Monks-2	32.87±0.0 +	32.87±0.0 + +	8.99±2.6 + + +	8.99±2.6 + + +	**8.89±2.8**
Monks-3	**0.00±0.0** +	**0.00±0.0** − +	0.60±0.4 − + +	0.60±0.4 − + +	0.81±0.5
Mushroom	**0.00±0.0** +	**0.00±0.0** − +	0.14±0.0 − + +	0.15±0.0 − + +	0.04±0.0
Satimage	13.58±0.5 + +	**12.18±0.4** +	13.06±0.4 + + +	12.83±0.3 + + +	12.22±0.3
Segment	3.21±0.2 +	**3.09±0.1** +	3.67±0.3 + +	3.44±0.3 + +	3.40±0.2
Sonar	28.02±3.2	24.75±2.9	24.36±1.9	24.45±1.8	23.83±2.1
Vehicle	26.96±0.8 +	22.36±0.9 +	28.35±1.3 + − +	23.97±0.9 + + −	24.28±1.0
Votes	+ **3.17±0.5** − +	4.41±0.5 +	3.70±0.3 − +	4.62±0.7 − + +	4.45±0.5
Waveform	24.66±0.3 + −	16.86±0.3 + +	17.30±0.5 + − +	16.75±0.4 + − +	15.94±0.3
Wine	6.99±0.7 + −	4.25±0.6 +	2.97±0.9 + −	2.50±0.6 + −	2.26±0.7

these (line 4) shows the number of datasets in which the top-level algorithm was more accurate than the corresponding cascade combination, versus the number in which it was less. The fifth line considers only those datasets where the error rate difference was significant at the 1% level, using paired t-tests. The last line shows the *p-values* obtained by applying the Wilcoxon matched-pairs signed-ranks test. The smaller the value, the more significant is the difference. The results show that on some datasets some variants of Cascade Generalization achieve significant improvements with respect to all the individual components.

6.5.1 Comparison with Stacked Generalization

We have compared various versions of Cascade Generalization to Stacked Generalization, as defined in Ting [TW97]. In our re-implementation of *Stacked Generalization* the *level_0* classifiers were C4.5 and Bayes, and the *level_1* classifier was *Discrim*. The attributes for the *level_1* data are the probability class distributions, obtained

Table 6.3: Summary of results of Cascade Generalization (error rates)

Measure	Bayes	Bay∇Bay	Bay∇Dis	Bay∇C4	Disc	Disc∇Disc	Disc∇Bay	Disc∇C4
1 Arithmetic Mean	16.68	16.42	16.01	15.37	17.13	17.11	15.33	18.0
2 Geometric Mean	12.57	11.97	12.08	10.47	13.28	13.11	11.32	14.71
3 Average Rank	8.56	8.46	5.85	7.06	8.08	7.81	6.08	9.52

	Bay∇Bay/Bay	Bay∇Dis/Bay	Bay∇C4/Bay	Disc∇Disc/Disc	Disc∇Bay/Disc	Disc∇C4/Disc
4 Nr. of Wins	11/13	16/8	14/9	9/4	16/8	8/13
5 Significant Wins	5/1	11/3	11/8	5/1	9/4	10/6
6 Wilcoxon Test	0.50	0.04	0.26	0.30	0.06	0.25

Measure	C4.5	C4.5∇C4.5	C4.5∇Bay	C4.5∇Dis	C4.5∇Dis∇Bay	C4.5∇Bay∇Dis
1 Arithmetic Mean	16.23	16.24	13.41	14.22	13.01	13.18
2 Geometric Mean	11.28	11.07	7.91	9.66	7.58	7.42
3 Average Rank	9.38	8.85	7.5	5.67	6.04	6.12

Measure	C4.5∇C4.5/C4.5	C4.5∇Bay/C4.5	C4.5∇Dis/C4.5	C4.5∇Dis∇Bay/C4.5	C4.5∇Bay∇Dis/C4.5
4 Nr. of Wins	13/7	18/3	14/10	17/7	17/7/
5 Significant Wins	1/0	9/1	8/2	11/3	9/3
6 Wilcoxon Test	0.31	0.02	0.00	0.00	0.00

from the *level_0* classifiers using a 5-fold stratified cross-validation[2]. Table 6.4 shows, in the first column, the results of *Stacked Generalization*. *Stacked Generalization* is compared, using *paired t-tests*, to *C4.5∇Discrim∇naive Bayes* and *C4.5∇naive Bayes∇Discrim* in this order. The $+(-)$ sign indicate that for this dataset the Cascade model performs significantly better (worse). Table 6.5 presents a summary of results. They provide evidence that the generalization ability of Cascade Generalization models is competitive with Stacked Generalization that computes the *level_1* attributes using internal cross-validation. This aspect affects of course the learning times. Both Cascade models are at least three times faster than Stacked Generalization.

6.5.1.1 The Type of $level_1$ Attributes in Cascade Generalization

Ting and Witten [TW97] have shown that successful implementations of Stacked Generalization should use, as $level_1$ attributes, the output class probabilities of $level_0$ models. In this subsection we examine how this issue affects Cascade Generalization. We have used the combination *C4.5∇Bayes∇Discrim* and compared the standard form, where the base classifiers output class probabilities, against its lesioned version where the base classifiers output categorical class predictions. Table 6.6 presents a summary of the results. It shows that using class probabilities is somewhat better than the use of categorical class predictions.

[2]We have also evaluated Stacked Generalization using C4.5 at top level. The version that we have used is somewhat better. Using C4.5 at top level the average mean of the error rate is 15.14.

Table 6.4: Comparison of Cascade Generalization versus Stacked Generalization

Dataset	*Stacked Gener.*		*C4.5∇Disc∇Bay*		*C4.5∇Bay∇Disc*
Australian	13.99±0.5		14.24±0.5		15.34±0.9
Balance	7.76±0.9	+	5.34±1.0		6.77±0.6
Banding	21.45±1.2		21.48±2.0		22.18±1.5
Breast	2.7±0.2		2.62±0.1		2.69±0.2
Cleveland	16.75±0.9		18.25±2.2	−	20.54±1.6
Credit	13.44±0.6	−	14.84±0.4		13.75±0.6
Diabetes	23.62±0.5	−	25.02±0.9		25.48±1.4
German	24.72±0.3	−	26.28±1.0		25.92±1.0
Glass	31.38±1.8		34.68±1.8		35.11±2.5
Heart	16.19±0.9	−	18.67±1.0	−	18.89±1.1
Hepatitis	15.33±1.1		15.97±1.9		17.21±2.1
Ionosphere	9.7±1.2		11.28±0.8		10.98±0.6
Iris	4.13±1.0		4.53±0.9		4.13±1.1
Monks-1	3.74±2.1	+	0.78±1.2		1.20±1.2
Monks-2	32.87±0.1	+	8.99±2.6	+	8.89±2.8
Monks-3	2.24±0.8	+	0.60±0.4	+	0.81±0.5
Mushroom	2.93±0.1	+	0.15±0.0	+	0.04±0.0
Satimage	13.11±0.4		12.83±0.3	+	12.22±0.3
Segment	3.32±0.2		3.44±0.3		3.40±0.2
Sonar	24.81±1.1		24.45±1.8		23.83±2.1
Vehicle	27.72±0.8	+	23.97±0.9	+	24.28±1.0
Votes	3.65±0.4	−	4.62±0.7	−	4.45±0.5
Waveform	17.14±0.4		16.75±0.4	+	15.94±0.3
Wine	2.23±0.9		2.50±0.6		2.26±0.7

6.5.2 Rank of Algorithms

As has been pointed out earlier various aggregate measures can be used to rank the different algorithms. Table 6.7 presents an ordered list of algorithms, ordered by the mean of error rates. The table 6.8 presents an ordered list of algorithms, ordered by average distance to the best algorithm for each dataset. Independently of the used criteria to rank the algorithms, at the top appears a composite model. The most promising combinations use *C4.5* (e.g. C4.5∇Bayes) as top level algorithm. The new attributes built by *Discrim* and *naive Bayes* express relations between attributes that

Table 6.5: Summary of comparison against Stacked Generalization (error rates)

	Stacked Generalization	*C4.5∇Discrim∇Bayes*	*C4.5∇Bayes∇Discrim*
Arithmetic Mean	13.95	13.01	13.18
Geometric Mean	9.93	7.58	7.42
Average Rank	1.8	2.1	2.1

	C4.5∇Disc∇Bay vs Stack.G.	*C4.5∇Bay∇Disc vs Stack.G.*
Nr. of Wins	10 / 14	9 / 15
Signif. Wins	6 / 5	6 / 3
Wilcoxon Test	0.90	0.70

Table 6.6: The type of $level_1$ attributes. Summary of Results.

	C4.5∇Bayes∇Discrim Class Probabilities	*C4.5∇Bayes∇Discrim Categorical Class Predictions*
Arithmetic Mean	13.01	14.02
Average Rank	1.46	1.54
Nr. Wins	13	11
Nr.Signif. Wins	3	2

are outside the scope of DNF algorithms like C4.5. These new attributes systematically appear at the root of the composite models.

Table 6.7: Average error rates of classifiers

C4.5∇Dis∇Bay	C4.5∇Bay∇Dis	C4.5∇Bay	C4.5∇Dis	Dis∇Bay	Bay∇C4.5	Bay∇Dis	C4.5
13.01	13.18	13.41	14.22	15.33	15.37	16.01	16.23
	C4.5∇C4.5	Bay∇Bay	Bayes	Discr	Dis∇Dis	Dis∇C4.5	
	16.24	16.42	16.68	17.13	17.11	18.0	

Table 6.8: Average distance to the best algorithm

C4.5∇Dis∇Bay	C4.5∇Bay∇Dis	C4.5∇Bay	C4.5∇Dis	Bay∇C4.5	Dis∇Bay	C4.5∇C4.5	C4.5
1.93	1.93	2.11	2.26	4.29	4.48	4.58	4.68
	Bay∇Dis	Dis∇Dis	Bay∇Bay	Discr	Bayes	Dis∇C4.5	
	4.97	5.41	5.64	5.90	6.02	6.64	

6.6 Why does Cascade Generalization Work?

Cascade re-represents the given instance space into a new, higher-dimensional space. In principle this could turn the given learning problem into a more difficult one. This phenomenon is known as the *curse of dimensionality* [Mit97]. In this section we analyze the behavior of Cascade Generalization along three dimensions: the error correlation, the bias-variance analysis, and Mahalanobis distances.

6.6.1 Error Correlation

In order to achieve good results when combining classifiers it is important that combined models make uncorrelated errors. In this section we use equations 5.3 and 5.4 to estimate how the errors made by the base classifiers are correlated. Note that when

combining a set of F classifiers the value of $\phi_e(F)$ lies in the interval [0,1]. It is 1 when the classifiers always make the same error and it is 0 when the classifiers always make different errors. The formula that we use provides higher values that the one used by Ali and Pazzani in [AP96]. As it was expected the lowest degree of correlation is between *decision trees* and *Bayes* and between *decision trees* and *discrim.* They use very different representation languages. The error correlation between *Bayes* and *discrim* is a little higher. Although they use somewhat similar representation languages, the search strategies are quite different.

Table 6.9: Error Correlation between base classifiers

Dataset	*C4-Bayes*	*C4-Discrim*	*Bayes-Discrim*
Australian	0.51	0.55	0.48
Balance	0.20	0.19	0.40
Banding	0.32	0.33	0.45
Breast(W)	0.39	0.31	0.35
Cleveland	0.41	0.45	0.55
Credit	0.48	0.52	0.43
Diabetes	0.44	0.50	0.54
German	0.45	0.44	0.63
Glass	0.28	0.34	0.31
Heart	0.41	0.44	0.58
Hepatitis	0.39	0.33	0.44
Ionosphere	0.25	0.29	0.24
Iris	0.24	0.24	0.27
Monks-1	0.14	0.06	0.40
Monks-2	0.96	0.95	0.94
Monks-3	0.00	0.00	0.12
Mushroom	0.00	0.00	0.22
Satimage	0.23	0.24	0.30
Segment	0.10	0.10	0.39
Sonar	0.27	0.31	0.29
Vehicle	0.22	0.23	0.17
Votes	0.31	0.31	0.30
Waveform	0.26	0.29	0.40
Wine	0.12	0.01	0.10
Mean	0.32	0.32	0.40

6.6.2 The Bias-Variance Decomposition

The base algorithms used in the experimental evaluation have different behavior from the point of view of Bias-Variance analysis. A decision tree is known to have low bias but high variance, while naive Bayes and linear discriminant are known to have low variance but high bias [Bre98].

Our experimental evaluation has shown that the most promising combinations use

a decision tree as high level classifier, and naive Bayes or linear discriminant as low level classifiers. Based on this results we suggest the following strategy to combine classifiers:

- At low level, use algorithms with low variance.
- At high level, use algorithms with low bias.

To give further evidence for this, we measured the bias and the variance of C4.5, naive Bayes and C4.5∇naive Bayes in the datasets under study. The results are shown in figure 6.2.

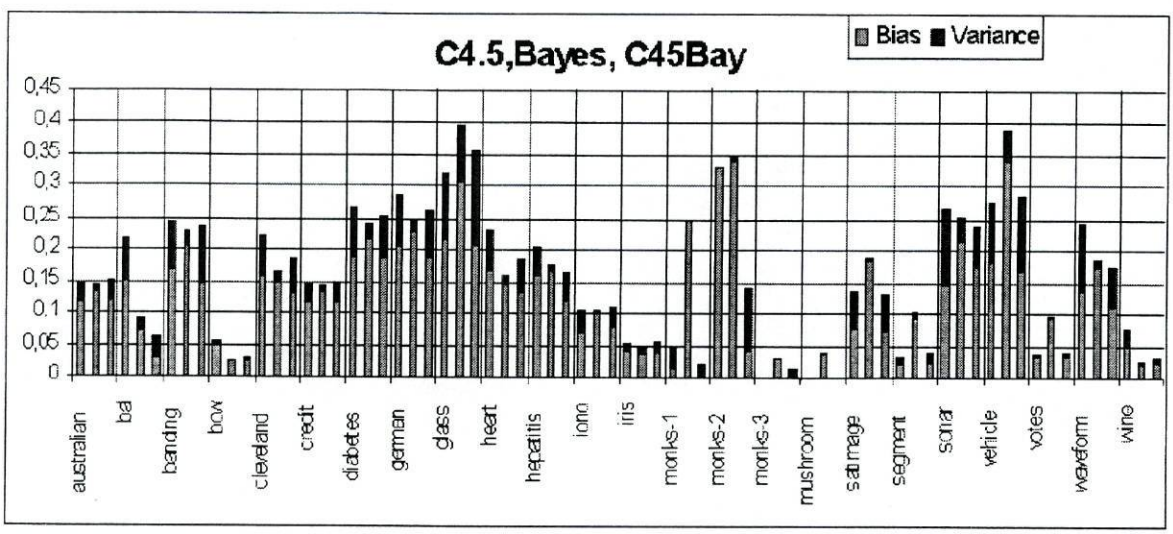


Figure 6.2: Bias-Variance decomposition of the error rate for C4.5, Bayes and C4.5∇Bayes for different datasets.

Table 6.10: Bias Variance decomposition of error rate

	C4.5	Bayes	C45∇Bayes
Variance	4.8	1.59	4.72
Bias	11.53	15.19	8.64

The benefits of the Cascade composition are well illustrated in datasets like Balance-scale, Hepatitis, Monks-2, Waveform, and Satimage.

A summary of the results, by averaging over all datasets, is presented in table 6.10. Comparison between Bayes and C4.5∇Bayes shows that the latter combination obtains a strong reduction of the bias component at costs of increasing the variance component.

C4.5∇Bayes reduces both bias and variance when compared to C4.5. The reduction of the error is mainly due to the reduction of bias.

6.6.3 The effect on Between-Classes Distance

Consider that each class defines a single cluster[3] in an *Euclidean space*. For each class i, the centroid of the corresponding cluster is defined as the vector of attribute means $\bar{x}_i$, which is computed from the examples of that class. The shape of the cluster is given by the covariance matrix S_i.

Using the *Mahalanobis* metric [DG84] we can define two distances:

1. The *within-class* distance. It is defined as the *Mahalanobis* distance between an example and the centroid of its cluster. It is computed as:

$$(\vec{\bar{x}_i} - \vec{x})^T S_i^{-1} (\vec{\bar{x}_i} - \vec{x}) \tag{6.4}$$

   where $\vec{x}$ represents the example attribute vector, $\vec{\bar{x}_i}$ denotes the centroid of the cluster corresponding to class i, and S_i is the covariance matrix for class i.

2. The *between-classes* distance. It is defined as the *Mahalanobis* distance between two clusters. It is computed as:

$$(\vec{\bar{x}_i} - \vec{\bar{x}_j})^T S_{pooled}^{-1} (\vec{\bar{x}_i} - \vec{\bar{x}_j}) \tag{6.5}$$

   where $\vec{\bar{x}_i}$ denotes the centroid of the cluster corresponding to class i, and S_{pooled} is the pooled covariance matrix using S_i and S_j.

Figure 6.3 illustrates geometrically the between and within clusters distances.

The intuition behind the within-class distance is that smaller values lead to more compact clusters. The intuition behind the between-classes distance is that larger values lead us to believe that the groups are sufficiently spread in terms of separation of means.

Table 6.11 lists all the datasets with all numeric attributes and shows the average of the between-classes and the within-class distances. The last line shows the *Average Ratio*. It is computed as the average ratio between the value of the extended dataset and the value of the original dataset. We observe that while the within-class distance almost

[3]This analysis assumes that there is a single dominant class for each cluster. Although this may not always be satisfied, it can give insights about the behaviour of Cascade composition.

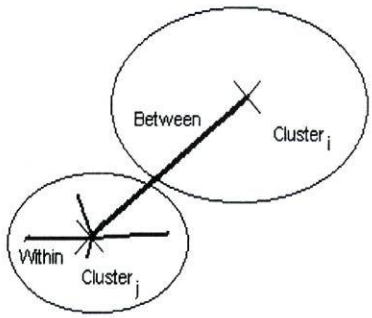


Figure 6.3: Between and within cluster distances.

remains constant the between-classes distance increases. For example when using the constructive operator Dis∇Bay the between-classes distance almost doubles.

6.6.4 A Detailed Analysis on Satimage Dataset

In this section we provide a detailed analysis of the behavior of Cascade Generalization in a specific problem. We focus the analysis on one dataset – *Satimage*. This is a medium size dataset, consisting of 6435 examples with 36 continuous attributes and 6 classes. Error rates and number of leaves of base and composite models are shown in table 6.12.

6.6.4.1 Error rates

Figure 6.4(a) shows the error rates of C4.5 and naive Bayes. Each point in the figure corresponds to one particular run of 10*10-fold cross-validation. The figure contains thus 100 points. The diagonal line corresponds to the hypothesis that both algorithms have identical error rates. In this case, if a point is above the diagonal it means that error rate of naive Bayes was higher than that of C4.5. We observe that in all runs the error rate of naive Bayes is higher than the error rate of C4.5. The figure shows also a line that corresponds to the linear fitting of the points using `gnufit`. It also illustrates that the trend is towards higher values of the error of naive Bayes.

Figure 6.4(b) plot the error rate of C4.5 against C4.5∇Bayes. We observe that 3/4 of the points are below the diagonal indicating that the combination C4.5∇Bayes has lower error rate. The linear fit is below the diagonal line indicating the trend towards lower values of the error of the composite model. This analysis confirms that the trend

Table 6.11: Average Between-classes and Within-class distances.

	Between Classes				Within Class			
	Original	Ext. Dis	Ext. Bay	Ext. Dis∇Bay	Original	Ext. Dis	Ext. Bay	Ext. Dis∇Bay
Australian	1.36	1.37	1.37	1.38	3.04	3.04	3.04	3.10
Balance	2.29	3.92	3.71	6.36	1.89	2.24	2.34	2.60
Breast(W)	4.28	4.57	6.88	6.96	2.64	2.76	2.75	2.82
Cleveland	2.21	2.30	2.49	2.56	3.50	3.63	3.63	3.74
Diabetes	1.38	1.41	1.60	1.61	2.64	2.81	2.80	2.97
German	1.30	1.30	1.37	1.37	4.74	4.85	4.85	4.95
Glass	4.96	6.94	27.58	44.25	2.66	3.13	3.07	3.28
Heart	2.23	2.34	2.54	2.55	3.50	3.63	3.63	3.75
Hepatitis	2.11	2.17	2.56	2.74	4.15	4.26	4.26	4.36
Ionosphere	2.48	2.92	3.46	3.65	5.20	5.28	5.31	5.39
Iris	9.61	11.84	11.41	11.58	1.86	1.97	1.98	2.10
Satimage	4.98	7.61	6.41	7.23	5.82	6.07	6.08	6.22
Segment	12.22	17.77	19.23	19.69	3.19	3.39	3.42	3.55
Sonar	2.67	3.46	5.95	6.97	7.64	7.70	7.70	7.77
Vehicle	3.68	6.04	4.44	6.10	4.14	4.40	4.43	4.68
Votes	4.62	5.41	4.65	5.51	3.70	3.79	3.81	3.89
Waveform	2.98	3.82	3.87	3.95	4.53	4.74	4.71	4.89
Wine	6.99	7.17	11.31	15.64	3.46	3.46	3.53	3.53
Average Ratio	1	1.23	1.57	1.94	1	1.05	1.05	1.09

Table 6.12: Error rates and mean nr. of leaves of base and composite models

Algorithm	Error Rate	Nr. of Leaves
C4.5	13.65	280.59
Bayes	19.05	
Discrim	16.01	
C45∇Bayes	12.96	264.98
C45∇Discrim	12.18	242.71

observed when considering the *average* error rate is also apparent for the majority runs of the cross-validation. The same observation applies when using Discrim as attribute constructor, which is illustrated in figure 6.5.

6.6.4.2 Bias-Variance analysis

The table 6.13 presents the Bias-Variance decomposition for the base and Cascade combinations.

Table 6.13: Bias Variance decomposition of error rate

	C4.5	Bayes	Discrim	C45∇Bayes	C45∇Discrim
Variance	6.8	0.99	1.1	6.39	6.38
Bias	9.2	18.7	15.9	8.76	8.43

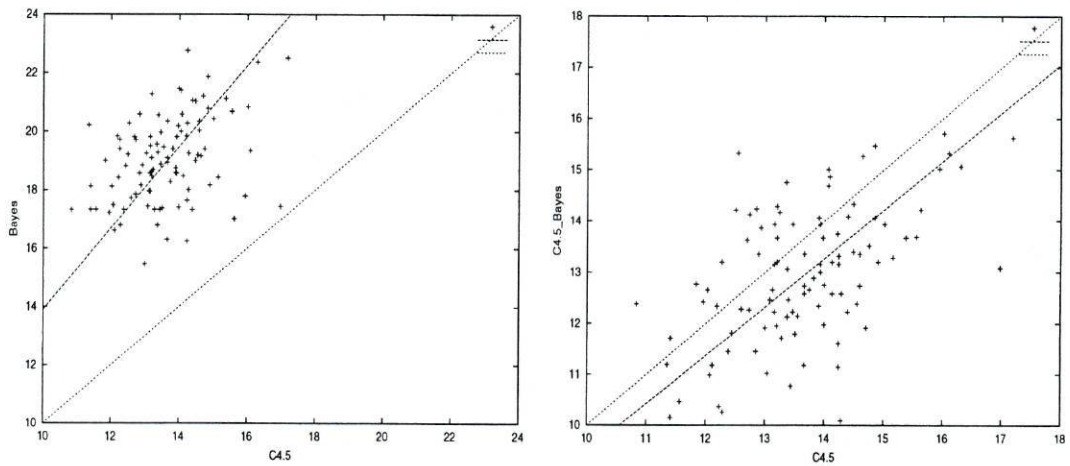


Figure 6.4: (a) C4.5 versus Bayes and (b) C4.5 versus C4.5∇Bayes

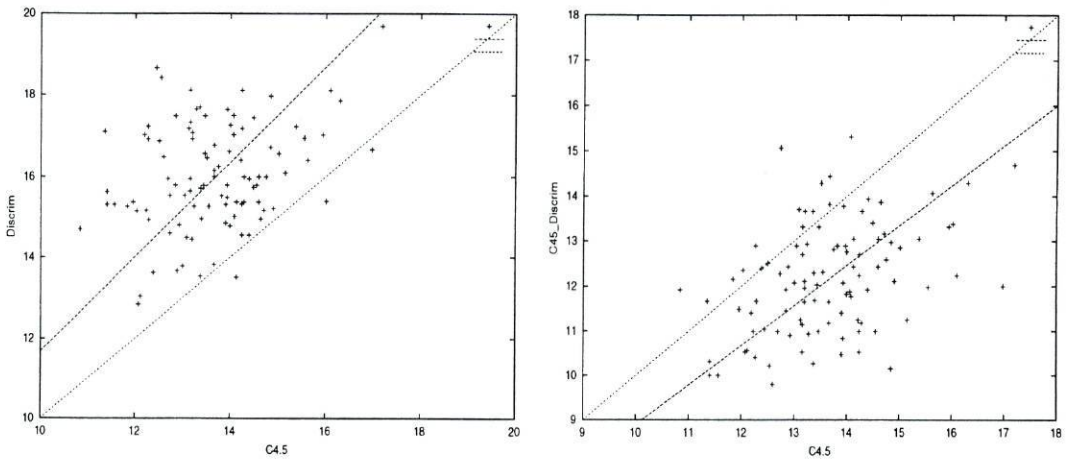


Figure 6.5: (a) C4.5 versus Discrim and (b) C4.5 versus C4.5∇Discrim

This results provide strong evidence that both C4.5∇Discrim and C4.5∇Bayes improves both bias and variance when compared with C4.5. When comparing Cascade combinations with naive Bayes and Discrim we observe a strong reduction on the bias component at costs of increasing in the variance component.

6.6.4.3 Between-Classes Distances

Table 6.14 presents the results for the between-classes distances for all pairs of classes. The first line shows the results relative to the original dataset. The second line shows the results on the extended dataset using *Discrim*, and the last line shows the results on the extended dataset using *naive Bayes*. In both extended datasets we verify an increase of the between-classes distances. This means that there is an increase of

the spread of the classes. Consequently the learning problem on the extended dataset should be easier than on the original dataset. Moreover, we observe that the extension using *Discrim* leads to higher values of the between-classes distances when compared to the extension using *naive Bayes*. On the dataset extended by *Discrim* the increase on the between-classes distances is about 53%, while the increase on the within-class distances is about 4%. This observation is consistent with the fact that lower error rate are obtained using C4.5∇Discrim. Table 6.16 shows a summary of the results.

Table 6.14: Between-classes Distances

Classes	1-2	1-3	1-4	1-5	1-6	2-3	2-4	2-5	2-6	3-4	3-5	3-6	4-5	4-6	5-6
Original Dataset	6.73	6.81	6.48	4.98	7.42	7.24	5.23	3.75	4.80	2.39	5.70	4.33	3.69	2.03	3.00
Extended using Discrim	11.87	10.08	9.70	7.77	10.80	12.48	9.66	4.56	9.82	2.58	9.33	6.11	3.98	2.13	3.28
Extended using Bayes	8.49	8.50	8.34	5.38	9.47	9.52	7.61	5.52	7.08	2.73	7.99	5.40	4.43	2.17	3.50
Extended using Bayes∇Discrim	13.55	10.16	10.14	8.03	13.12	12.67	9.93	5.67	10.27	2.75	9.71	6.43	4.5	2.18	3.61

The table 6.15 shows the effect of extending the original dataset on the within-class distances. Although the clusters becomes less compact, the increase is rather small. It is probably outweighed by the rather more significant increase of the between-cluster distances. The relative increases of both between and within-cluster distances are shown in table 6.16.

Table 6.15: Within-class Distances

Classes	1	2	3	4	5	6
Original Dataset	5.82	5.82	5.79	5.82	5.84	5.82
Extended using Discrim	6.02	6.11	5.97	6.10	6.16	6.04
Extended using Bayes	6.10	6.10	6.00	6.10	6.12	6.03
Extended using Bayes∇Discrim	6.19	6.35	6.16	6.31	6.42	6.24

Table 6.16: Average Distances

	Between-Classes	% Increase	Within-Class	% Increase
Original Dataset	4.98		5.82	
Extended using Discrim	7.61	53%	6.07	4%
Extended using Bayes	6.41	29%	6.08	4.5%
Extended using Bayes∇Discrim	8.18	64%	6.28	8%

6.7 Conclusions

Cascade Generalization belongs to the family of stacking algorithms. The main achievement of our method is the ability to merge different models into a single

one. The final model could use terms of the representational language of the base classifiers. The bias restriction imposed by using a single model is relaxed. Cascade combination provides a single and structured model for the data, which is a strong advantage over the methods that combine classifiers by voting. This is done in an efficient manner leading to reduced learning times. Another advantage of Cascade Generalization is related to the use of probability class distributions. Many common classification algorithms generate categorical classes when classifying examples. Combining classifiers which output categorical classes ignores the information concerning the strength of predictions. The use of class probability distributions provides us with useful information and is explored in our systems.

One of the main problems when combining classifiers is: *Which algorithms should we combine?* We provide a methodology for choosing the *low level algorithms*, *high level algorithms*, and *the type of attributes* to be used in the high level input space. Our work suggests the following strategy:

- Combine classifiers with different behavior from the point of view of *Bias-Variance* analysis.
- At low level, use algorithms with low variance.
- At high level, use algorithms with low bias.

In the *Cascade* framework lower level learners delay the final decision. This is delegated to the high level learners by selecting learners with low bias for the high level, we are able to fit more complex decision surfaces, taking into account the *"stable"* surfaces generated by the low level learners.

How many low level classifiers we should use in a combination is still an open issue. Given equal performance, we would prefer fewer component classifiers, since training and application times will be lower for smaller number of components. Besides, larger number of components may lead to undesirable increase in variance. Larger number of components has also adverse effects on comprehensibility. In our study we have considered at most three components. The version with three components seemed perform better than the two component versions. More research is needed to establish the limits of extending this scenario.

Cascade Generalization can be used *locally* at each iteration of a divide-and-conquer algorithm. This ability allows us to build a new family of multistrategy algorithms. This is the subject of the next chapter.

Chapter 7

Local Cascade Generalization

In this chapter we extend the work of the previous chapter by applying Cascade Generalization locally. *At each iteration of the divide-and-conquer algorithm, a reconstruction of the instance space occurs by the addition of new attributes. Each new attribute represents the probability that an example belongs to the class given by a base classifier. We have implemented three Local Generalization Algorithms. The first merges a linear discriminant with the decision tree, the second merges a naive Bayes with the decision tree, and the third merges a linear discriminant and a naive Bayes with the decision tree. All the integrated algorithms show an increase of performance, when compared with the corresponding single models. Local Cascade Generalization also outperforms other methods for combining classifiers, like Stacked Generalization and competes well against* Boosting, *with statistically significant confidence levels.*

7.1 Introduction

Many classification algorithms use a divide-and-conquer strategy that resolves the given complex problem by dividing it into simpler problems, and then by applying recursively the same strategy to the subproblems. Solutions of subproblems are combined to yield a solution of the original complex problem. This is the basic idea behind the well known decision tree based algorithms: ID3 (Quinlan, 1984), ASSISTANT (Kononenko *et al.*, 1987), CART (Breiman *et al.*, 1984), C4.5 (Quinlan, 1993), etc. The power of this approach derives from the ability to split the hyperspace into subspaces and fit each subspace with different functions. In this chapter we explore *Cascade Generalization* on the subspaces generated by the divide-and-conquer algorithm. The intuition behind this proposed method is the same as behind any divide-and-conquer strategy. The relations that can not be captured at global level can be discovered on the local level.

In the following sections we describe how to apply *Cascade Generalization* locally. We have developed this strategy for decision trees, although it should be possible to use it in conjunction with any divide-and-conquer method, like *decision lists* [Riv87].

7.2 Local Cascade Generalization

Local Cascade Generalization is a composition of classification algorithms that is elaborated when building the classifier for a given task. In each iteration of a divide-and-conquer algorithm, *Local Cascade Generalization* extends the dataset by the insertion of new attributes. These new attributes are propagated down to the subtasks. In this paper we restrict the use of *Local Cascade Generalization* to decision tree based algorithms. However, it should be possible to use it with any *divide and conquer* algorithm. Figure 7.1 presents the general algorithm of *Local Cascade Generalization*, restricted to a decision tree. The method is shortly referred to as *CGTree*.

When growing the tree, new attributes are computed at each decision node by applying the Φ operator . The new attributes are propagated down the tree. The number of new attributes is equal to the number of classes appearing in the examples that fall into this node. This number can vary at different levels of the tree. In general deeper nodes may contain a larger number of attributes than the parent nodes. This could be a disadvantage but the number of new attributes that can be generated decreases rapidly. As the tree grows and the classes are discriminated, deeper nodes also contain

Input: A data set D, a classifier $\Im$
Output: A decision tree

Function CGtree(D, $\Im$)
  If stop criteria(D) = TRUE
    return a leaf with class probability distribution
  $D' = \Phi(D, \mathcal{A}(\Im(D), D))$
  Choose the attribute A_i that maximizes splitting criterion on D'
  For each partition of examples based on the values of attribute A_i
    generate a subtree: $Tree_i$ = CGtree(D'_i, $\Im$)
  return Tree containing a decision node based on attribute Ai,
      storing $\Im(D)$ and descendant subtrees $Tree_i$
End

Figure 7.1: Local Cascade Algorithm based on a Decision Tree

examples with a decreasing number of classes. This means that as the tree grows the number of new attributes decreases.

In order to be applied as a predictor, any *CGTree* must store, in each node, the model generated by the base classifier using the examples that fall at this node. When classifying a new example, the example traverses the tree in the usual way, but at each decision node it is extended by the insertion of the probability class distribution provided the base classifier predictor at this node.

In the framework of local cascade generalization, we have developed a *CGLtree*, that uses the $\Phi(D, \mathcal{A}(Discrim(D), D))$ operator in the constructive step. Each internal node of a *CGLtree* constructs a discriminant function. This discriminant function is used to build new attributes. For each example, the value of a new attribute is computed using the the linear discriminant function. At each decision node, the number of new attributes built by *CGLtree* is always equal to the number of classes taken from the examples that fall at this node. In order to restrict attention to well populated classes, we use the following heuristic: we only consider a $class_i$ if the number of examples, at this node, belonging to $class_i$ is greater than N times the number of attributes[1]. By default N is 3. This implies that at different nodes, different number of classes will be considered leading to addition of a different number

[1]This heuristic was suggested by Breiman et al. [BFOS84]

of new attributes.

In our empirical study we have used two other algorithms that apply *Cascade Generalization* locally. The first one is *CGBtree* that uses as constructive operator

$$\Phi(D, \mathcal{A}(naiveBayes(D), D)),$$

and the second one is *CGBLtree* that uses as constructive operator:

$$\Phi_p(D, [\mathcal{A}(naiveBayes(D), D), \mathcal{A}(Discrim(D), D)]),$$

In all other aspects these algorithms are similar to *CGLtree*.

All versions of *Local Cascade Generalization algorithms* are based on *Dtree*[2]. That is, they use exactly the same splitting criteria, stopping criteria, pruning mechanism, etc. Moreover they share many minor heuristics that individually may appear unimportant, but collectively can make difference.

At each decision node, *CGLtree* applies the *Linear discriminant*, while *CGBtree* applies the *naive Bayes* algorithm. *CGBLtree* applies the *Linear discriminant* to all *ordered* attributes and the *naive Bayes* to all *categorical* attributes. In order to prevent *overfitting* the construction of new attributes is constrained to a depth of 5. In addition, the level of pruning is greater than the level of pruning in *Dtree*.

There is one restriction to the application of the $\Phi(D', \mathcal{A}(\Im(D), D'))$ operator: the induced classifier $\Im(D)$ must return the corresponding probability class distribution for each $\vec{x} \in D'$. Any classifier that satisfies these requisites could be applied. It is possible to imagine a *CGTree*, whose internal nodes are trees themselves. For example, small modifications to C4.5[3] enables the construction of a *CGTree* whose internal nodes are trees generated by C4.5.

7.3 An Illustrative Example

Figure 7.2 represents the tree generated by a *CGTree* on the *Monks-2* problem. The constructive operator used is: $\Phi(D, Discrim \nabla Bayes(\vec{x}, D))$. At the root of the tree the *naive Bayes* algorithm provides two new attributes – *Bayes_1* and *Bayes_2*. The

[2]Dtree is an univariate tree. It is similar to the decision tree described in chapter 4.

[3]Two different methods are presented in [Gam98, TW97].

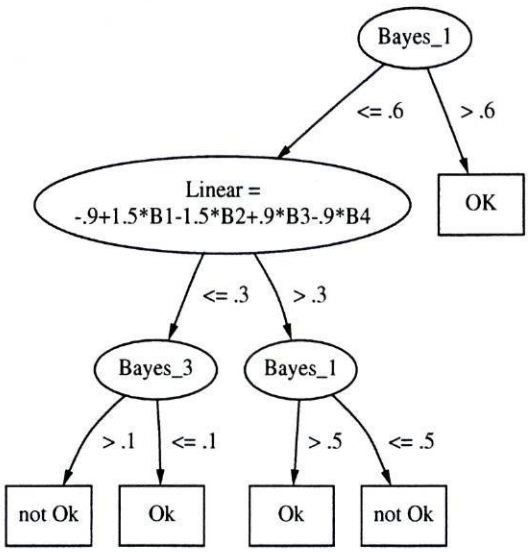


Figure 7.2: Tree generated by a CGTree using Discrim∇Bayes as constructive operator

linear discriminant uses continuous attributes only. There are only two continuous attributes, those built by the *naive Bayes*. In this case, the coefficients of the linear discriminant shrink to zero by the process of variable elimination used by the discriminant algorithm. The *gain ratio* criterion chooses the *Bayes_1* attribute as a test. The dataset is split into two partitions. One of them contains only examples from class *OK*: a leaf is generated. In the other partition two new *Bayes* attributes are built (Bayes_3, Bayes_4) and so a linear discriminant is generated based on these two *Bayes* attributes and on those built at the root of the tree. The attribute based on the linear discriminant is chosen as test attribute for this node. The dataset is segmented and the process of tree construction proceeds.

This example illustrates two points:

- The interactions between classifiers: The linear discriminant contains terms built by *naive Bayes*. Whenever a new attribute is built, it is considered as a regular attribute since then. Any attribute combination built at deeper nodes can contain terms based on the attributes built at upper nodes.

- Re-use of attributes with different thresholds. The attribute *Bayes_1*, built at the root, is used twice in the tree with different thresholds.

7.4 Relation to Other Work on Multivariate Trees

With respect to the final model, there are clear similarities between *CGLtree* and *Multivariate trees* of Brodley and Utgoff [BU95] and John [Joh96]. Any multivariate tree is topologically equivalent to a three-layer *inference network* [Lan96]. The constructive ability of our system is similar to the *Cascade Correlation Learning architecture* of Fahlman *et al.* [FL90]. Also the final model of *CGBtree* is related with the *recursive naive Bayes* presented by Pat Langley [Lan93a]. This is an interesting feature of *Local Cascade Generalization*: it unifies in a single framework several systems from several different research areas.

7.4.1 Relation to Linear Tree

Local Cascade Generalization is an extension of the method proposed in chapter 4 to build multivariate trees. In Discriminant Trees, the constructive operator incorporated a single discriminant function. In Local Cascade composition this restriction was relaxed. We can use *any* classifier as constructive operator. Moreover, a composition of several classifers, like in *CGBLtree*, could be used.

In chapter 4 of this dissertation, we have compared system *Ltree*, similar to *CGLtree*, with *CART* [BFOS84], *OC1* [MKS94] and *LMDT* [BU95]. The focus of this chapter is on methodology for combining classifiers. Consequently, we only compare our algorithms against other methods that generate and combine multiple classifiers.

7.5 Empirical Evaluation

We have evaluated three variants of Local Cascade Generalization – *CGBtree*, *CGLtree*, and *CGBLtree* described in the previous section. We have used twenty-four datasets from the UCI repository. As usual, we estimate the error rate using 10*10-fold cross-validation.

7.5.1 Analysis of the Error Rates

In this section we analyze the performance, in terms of error rates, of *Local Cascade Generalization* in comparison to global *Cascade Generalization*. In another set

of experiments we compare *Local Cascade Generalization* to two other methods for combining classifiers: a variance reduction method, Boosting (here represented by C5.0Boosting) and a bias reduction method, Stacked Generalization.

Table 7.1: Results of (a)Local Cascade Generalization (b)Boosting and Stacked (c) Boosting a Cascade algorithm

Dataset	*CGBtree*	*CGLtree*	*CGBLtree*	*C5.0Boost*	*Stacked G.*	*C5B∇Bayes*
	(vs. Corresponding Cascade Models)			*(vs. CGBLtree)*		*(vs. C5.0Boost)*
Australian	14.42±0.6	14.69±0.9	13.95±0.7	13.21±0.7	13.99±0.5	13.96±0.9
Balance	5.32±1.1 −	8.24±0.5 −	8.08±0.4	− 20.03±1.0	7.76±0.9	+ 4.25±0.7
Banding	26.67±1.1	23.60±1.2	20.69±1.2	+ 17.39±1.7	21.45±1.2	18.38±1.8
Breast(W)	2.62±0.1 +	3.23±0.4	2.66±0.1	− 3.34±0.3	2.7±0.2	3.03±0.2
Cleveland	15.10±1.4 +	16.54±0.8	16.50±0.8	− 18.95±1.3	16.75±0.9	17.86±1.1
Credit	+ 15.36±0.5	14.41±0.8	14.52±0.8	+ 13.41±0.8	13.44±0.6	13.57±0.9
Diabetes	25.60±1.3	24.43±0.9	24.48±0.9	24.58±0.9	23.62±0.5	24.71±1.1
German	25.22±1.2	24.78±1.1	24.88±0.8	25.36±0.8	24.72±0.3	25.20±1.1
Glass	+ 32.99±2.3	34.71±2.3	32.35±2.0	+ 25.06±2.0	31.28±1.8	− 29.23±1.6
Heart	+ 16.37±1.0	16.85±1.2 +	16.81±1.1	− 19.94±1.3	16.19±0.9	+ 17.24±1.5
Hepatitis	16.87±1.1 +	16.87±1.1	16.87±1.1	16.67±1.5	15.33±1.1	15.93±1.3
Ionosphere	9.62±0.9	11.06±0.6	11.00±0.7	+ 6.57±1.1	9.7±1.2	7.71±0.7
Iris	4.73±1.3	2.80±0.4 +	2.80±0.4	− 5.68±0.6	4.13±1.0	5.07±0.8
Monks-1	− 9.88±3.5	6.80±3.3 −	8.53±3.0	+ 0.00±0.0 +	3.74±2.1	0.33±0.3
Monks-2	− 15.21±3.4	33.19±1.7	11.88±3.3	− 35.76±1.0 −	32.87±0.1	+ 3.64±1.7
Monks-3	0.39±0.4 −	0.92±0.5	0.39±0.4	0.00±0.0 −	2.24±0.8	0.63±0.3
Mushroom	− 0.11±0.0 −	0.96±0.1 −	0.24±0.0	+ 0.00±0.0 −	2.93±0.1	0.01±0.0
Satimage	+ 11.83±0.2	11.99±0.3 +	11.99±0.3	+ 9.25±0.2 −	13.11±0.4	9.21±0.2
Segment	4.36±0.2	3.19±0.3	3.20±0.3	+ 1.79±0.1	3.32±0.2	− 2.09±0.1
Sonar	26.23±1.7	25.26±1.5	25.50±1.6	+ 19.25±2.2	24.81±1.1	23.02±1.2
Vehicle	28.75±0.8	21.21±0.9 +	21.32±0.9	− 23.71±0.7 −	27.72±0.8	24.29±1.3
Votes	3.29±0.4	4.30±0.5 +	3.26±0.4	4.11±0.4	3.65±0.4	4.30±0.4
Waveform	16.54±0.7 +	15.74±0.5 +	16.12±0.5	− 17.45±0.4 −	17.14±0.4	+ 15.58±0.3
Wine	2.30±0.5 +	1.20±0.6 +	1.20±0.6	− 3.45±1.0	2.23±0.9	2.94±0.7

7.5.1.1 Comparison of Local and Global Cascade Combinations

In this subsection we compare *Local Cascade Generalization* algorithms with the corresponding global Cascade models. For example, *CGLtree* is compared against *C4.5∇Discrim.* In Table 7.1a the columns 2,3,4 presents the results of *Local Cascade Generalization.* Each column corresponds to a particular Cascade Generalization algorithm. Each algorithm is compared against its similar *Cascade* model using *paired t-tests.* The +(−) sign means that the error rate of the local model is, significantly lower (higher) than the correspondent global model. Table 7.2 presents a summary of the results of this comparison. There is some evidence of the advantages of applying locally Cascade Generalization.

Table 7.2: Summary of results comparing Global versus Local Cascade Generalization

	C4.5∇Bayes	CGBtree	C4.5∇Discrim	CGLtree	C4.5∇Discrim∇Bayes	CGBLtree
Arithmetic Mean	13.41	13.74	14.22	14.04	13.01	12.88
Geometric Mean	7.91	8.27	9.66	9.20	7.58	7.87
Average Rank	4.08	3.54	3.5	3.50	3.65	2.73
Distance to the Best	1.24	0.94	0.92	0.64	0.92	0.41
Nr. of wins	10	14	12	12	7	17
Signif. Wins (t-tests)	3	4	3	5	3	7
Wilcoxon Test	0.54		0.68		0.13	

7.5.1.2 Comparison of Local Cascade Generalization with Boosting and Stacked Generalization

Table 7.1b presents the results of *C5.0Boosting* with the default parameter of 10, that aggregates 10 trees. This table also shows the results of *Stacked Generalization* as it is defined in [TW97] and discussed in the previous chapter. Both *C5.0Boosting* and *Stacked Generalization* are compared against *CGBLtree*, using *paired t-tests* with the significance level set to 99.9%. A $+(-)$ sign means that *C5.0Boosting* or *Stacked Generalization* performs significantly better (worse) than *CGBLtree*. In this study, *CGBLtree* performs significantly better than *Stacked Generalization*, in 6 datasets and only on one dataset it performs worse (at this confidence level). A summary of results is presented in table 7.3.

Table 7.3: Summary of results (Error Rates) of Local Cascade Generalization, Boosting and Stacked Generalization

	CGBtree	*CGLtree*	*CGBLtree*	*C5Boost*	*Stacked Gen.*	C5Boost∇Bayes
Arithmetic Mean	13.74	14.04	12.88	13.54	13.95	11.78
Geometric Mean	8.27	9.20	7.87	8.83	9.93	5.83
Average Rank	4.02	3.92	3.23	3.33	3.46	3.04
Distance to the Best	2.07	1.74	1.51	1.73	2.03	0.80

	CGBLtree / C5Boost	*CGBLtree / Stacked Gen.*
Nr. of wins	12 / 12	14 / 10
Nr. of significant wins (t-tests)	9 / 9	6 / 1
Nr. of significant wins (signal)	5 / 8	6 / 1
Wilcoxon Test	0.99	0.50

The comparison of *CGBLtree* and *C5.0Boosting* indicates that our algorithm has significantly better performance in 9 datasets and is worse also in 9 datasets. It is interesting to note that in 24 datasets there are 18 significant differences. This is an evidence that *C5.0Boosting* and *Cascade Generalization* have rather different behavior. The improvement observed with *C5.0Boosting* is mainly due to the reduction of the *variance* component of the error rate. With *Cascade* algorithms the improvement is

mainly due to the reduction of the *bias* component.

7.5.1.3 A Step Ahead - Boosting a Global Cascade Algorithm

Taking into account the previous results, we have done a first experiment on boosting a Cascade combination. In this case we have used the combination *C5.0Boosting*∇*naive Bayes*. Table 7.1c presents the results in the last column. It improves over *C5.0 Boosting* on 4 datasets and is worse in 2. This seems a quite good result, and we intend, in the near future, to *boost CGBLtree*.

7.5.2 Analysis of the Number of Leaves

To have insights about model complexity of local Cascade models we have measured the number of leaves of its models. Table 7.4 presents a summary of the results. The results of using *C5.0* are presented for comparison. Any local Cascade tree, generates a model with half the number of leaves than the models generated by *C5.0*.

Table 7.4: Summary of results (Nr. of Leaves) of Local Cascade Generalization

	CGBtree	*CGLtree*	*CGBLtree*	*C5.0*
Arithmetic Mean	19.54	20.16	19.87	40.73
Geometric Mean	9.90	11.11	10.31	20.94
Average Rank	2.06	2.15	2.08	3.71

7.5.3 Learning Times

We have measured the learning times, that represents another dimension when comparing classifiers. Figure 7.3 shows the time needed by each algorithm in each dataset. Table 7.5 presents the average of learning times needed to run the experiments reported in this chapter[4]. Our results demonstrate that any *CGTree* is faster than *C5.0Boosting*. *C5.0Boosting* is slower because it generates 10 trees with increased complexity. Also, any *CGTree* is faster than *Stacked Generalization*. This is due to the internal cross validation used in *Stacked Generalization*.

[4] *C5.0* and *C5.0 Boosting* have run on a *Sparc 10* machine. All the other algorithms have run on a *Pentium 166MHz, 32Mb* machine under *Linux*.The running times of *C5.0* and *C5.0 Boosting* were reduced by a factor of 2 as suggested in: `www.spec.org`.

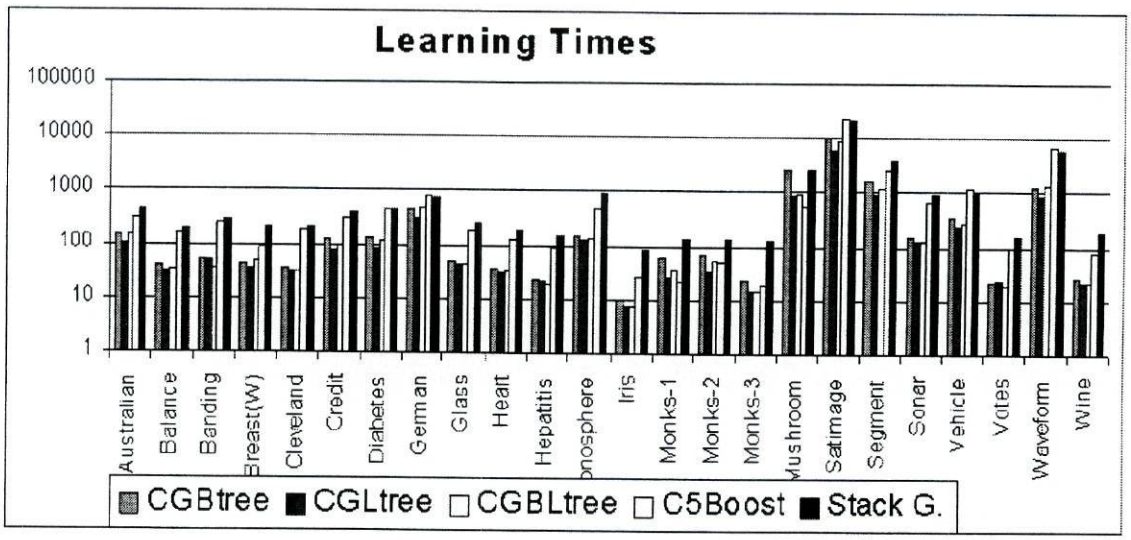


Figure 7.3: Learning Curves and Learning Times on Satimage Dataset.

Table 7.5: Comparison of relative learning times of local Cascade Classifiers

C5.0	CGBtree	CGLtree	CGBLtree	C5.0Boost	Stacked Gen.
1	2.4	2.8	3.9	5.47	5.51

We have also constructed the learning curves of local Cascade algorithms and C5.0. Figure 7.4 shows the dependence of the error rates and learning times on the number of training examples for Satimage dataset. The figure on left shows the dependence of error rate on the number of examples. The figure on right shows the dependence of learning times on the number of examples. With respect to the learning time our results suggest that the behavior of local cascade trees is similar to a standard decision tree (C5.0). We observe that after 200 examples the curve has approximately the form of a line, following the same slope as C5.0. This means that the logarithm of the difference of times between C5.0 and CGTree is constant.

7.6 Concluding Remarks

The systems based on *Local Cascade Generalization* can be seen as a composition of learning classifiers that exploit the constructive ability of *Cascade Generalization* at each iteration of a divide-and-conquer algorithm. Our empirical evaluation suggests that this an efficient method to combine classifiers. It competes well against standard methods, like *Stacked Generalization* and *Boosting*, in terms of accuracy, with

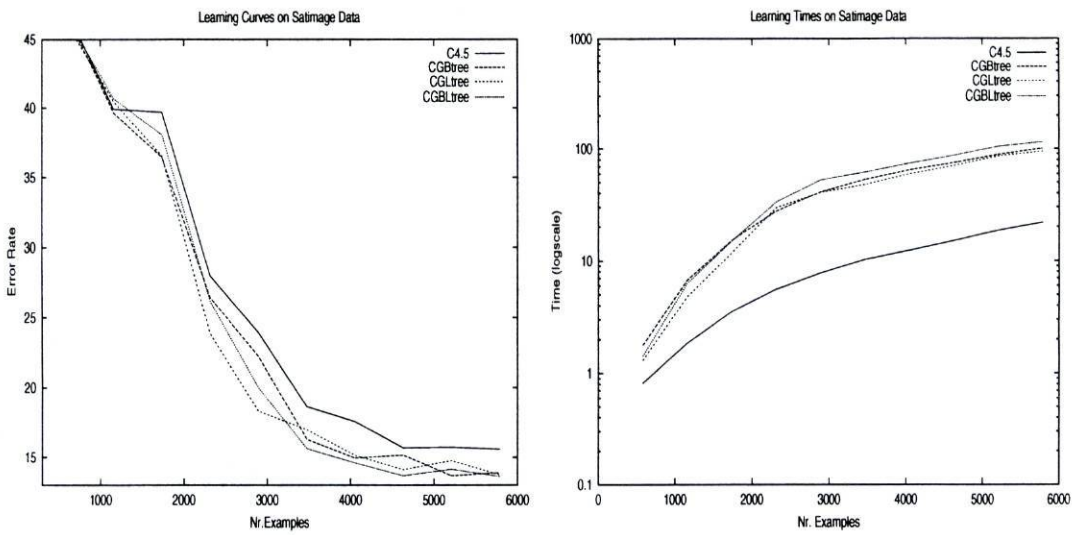


Figure 7.4: Learning Curves and Learning Times on Satimage Dataset.

advantages in the interpretability of learned models requiring reduced learning times.

With respect to the generated models, there are interesting similarities with neural nets. In neural nets, the output of a neuron serves as the input for the neurons in the following layer. In *Cascade Generalization* the output of a base classifier is processed by the following classifiers. The advantage of *Cascade Generalization* is that it uses different classifiers, leading to different types of generalization. Moreover, the models of *Cascade Generalization* are simpler than those of neural nets and hence easier to interpret.

Chapter 8

Conclusions

We briefly summarize some of the results presented in this dissertation. We highlight the major contributions of this work in the area of machine learning, and discuss future work.

8.1 Concluding Remarks

In this chapter we briefly summarize the main contributions of this dissertation to the area of machine learning. The last section is dedicated to some of the open issues and possible future developments of this work.

The main contribution of this dissertation is that it provides a new and general method for combining learning models by means of constructive induction. The basic idea of the method is to use the learning algorithms in sequence. At each iteration a two step process occurs:

1. A model is built using a base classifier.
2. The instance space is extended by the insertion of new attributes. These are generated by the built model for each given example.

The constructive step generates terms in the representational language of the base classifier. If the high level classifier chooses one of these terms, its representational power has been extended. The *bias* restrictions of the high level classifier is relaxed by incorporating terms of the representational language of the base classifiers. This is the basic idea behind *Ltree* and the *Cascade Generalization* architecture.

The method is presented in two parts.

1. *As method for building multivariate trees*
   In the first part of the dissertation we have presented system *Ltree*, a multivariate decision tree. *Ltree* uses as constructive operator a *linear discriminant* function. It was the precursor of the *Cascade* architecture. Because the particular form of the constructive operator used, *Ltree* only inserts $\#cl - 1$ new attributes where $\#cl$ represents the number of classes. In general, it is not possible to follow this schema with other types of base classifiers.

   We have shown that our system has advantages concerning accuracy and learning times when compared to other well known multivariate trees, like *CART*, *LMDT*, and *OC1*.

2. *A general method for combining learning models*
   In the second part of the dissertation we present a general framework to combine classifiers. The method *Cascade Generalization* is an extension of the method

presented on the first part. The base classifiers are not restricted to *discriminant functions* but are generalized to other types of classifiers. We define the conditions that the base classifier must satisfy so that it can be used in this framework, and define criteria for selecting the suitable types of low and high level classifiers.

We have empirically shown that the method improves the accuracy of the base classifiers and competes well against other methods that combine classifiers, namely the state of the art method - Boosting.

We have examined two different schemes of combining classifiers. The first one loosely couples classifiers whilst the second one tightly couples classifiers.

1. *Loose coupling: Base classifier(s) pre-process data for another stage*
   This framework can be used to combine most of the existing classifiers without changes, or with rather small changes. The method only requires that the original data is extended by the insertion of the probability class distribution that must be generated by the base classifier.

   System *C45Oblique* builds multivariate trees and compete well against *C4.5*.

2. *Tight coupling through Local constructive induction*
   In this framework two or more classifiers are coupled locally. Although in this dissertation we have used only *Local Cascade Generalization* in conjunction with decision trees the method could be easily extended to other *divide-and-conquer* systems, like *decision lists*.

Most of the existing methods such as *Bagging* and *Boosting* that combine learned models, use a voting strategy to determine the final outcome. Although this leads to improvements in accuracy, it has strong limitations - loss in interpretability. Our models are easier to interpret particularly if classifiers are loosely coupled. The final model uses the representational language of the high level classifier, possibly enriched with expressions in the representational language of the low level classifiers.

When Cascade Generalization is applied locally, the models generated are more difficult to interpret than those generated by loosely coupled classifiers. The new attributes built at deeper nodes, contain terms based on the previously built attributes. This allows us to built very complex decision surfaces, but it affects somewhat the interpretability of the final model.

One interesting feature of *local Cascade Generalization* is that it provides a single framework for a collection of different methods. Our method can be related to several paradigms of machine learning. For example there are similarities with multivariate trees [BU95], recursive discriminants [LV88], neural networks [FL90], recursive Bayes [Lan93a], and multiple models, namely Stacked Generalization [Wol92].

The unified framework is useful because it overcomes some superficial distinctions and enables us to study more fundamental ones. From a practical perspective the user's task is simplified, because his aim of obtaining better accuracy can be achieved with a single algorithm instead of several ones. At the same time he may have a reasonable confidence that the accuracy will not decrease on new unseen data.

Besides the main contributions mentioned earlier we would like to highlight the following three aspects of our work, that improve existing techniques:

- The schema for computing class distributions by propagating probabilities along a decision tree. Although these methods have been used earlier [Bun90, Qui92], they are usually heuristic based. We provide a well founded method to solve the problem.

- The improvement in the definition of *error correlation*. We have shown that the Ali and Pazzani's [AP96] definition does not satisfy all the desirable properties and proposed a new, more consistent, definition.

8.2 Limitations and Future Work

Some open issues, which could be explored in future, involve:

- From the perspective of *bias-variance* analysis the main effect of the proposed methodology is a reduction on the bias component. It should be possible to combine the *Cascade* architecture with a *variance* reduction method, like *bagging* or *boosting*.

- The theoretical analysis in the area of model combination, suggests that we should combine models that perform better than a random guess. *Cascade* algorithms have no restriction on the used models. All models are constructed, although it is possible and desirable to restrict the use of generated models.

Other problems that involve basic research include:

- Why does *Cascade Generalization* improve performance?
  Our experimental study suggests that we should combine algorithms with complementary behavior from the point of view of *bias-variance* analysis. Other forms of complementarity can be considered, for example the *search bias*. So one interesting issue to be explored is: given a dataset, can we *predict* which algorithms are complementary?

- When does *Cascade Generalization* improve performance?
  In some datasets *Cascade* was not able to improve the performance of base classifiers. Can we characterize these datasets? That is, can we *predict* under what circumstances *Cascade Generalization* will lead to an improvement in performance?

- How many base classifiers should we use?
  The general preference is for smaller number of base classifiers. Under what circumstances can we reduce the number of base classifiers without affecting performance?

- The *Cascade Generalization* architecture provides a method for designing algorithms that use multiple *representations* and multiple *search* strategies within the induction algorithm. An interesting line of future research should explore *flexible* inductive strategies using several diverse representations. It should be possible to extend *local Cascade Generalization* to provide a dynamic control and this make a step in this direction.

Vail's Second Axiom:
The amount of work to be done increases in proportion to the amount of work already completed.
Unix fortune

Appendix A

Datasets used

A.1 Datasets Description

All the datasets used in the experimental evaluation are available from the *University of California, Irvine, (UCI) Repository of Machine Learning Databases* [BKM99]. Some of them were been used as in the *StatLog* project[MST94]. For more information on the Repository, consult `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Table A.1: Datasets Characteristics

Dataset	*Examples*	*Classes*	*Attributes*	*Missing Values*	*Default accuracy*
Australian	690	2	14	None	55.5%
Balance	625	3	4	None	46.8%
Banding	238	2	29	Yes(42%)	63.5%
Breast(W)	699	2	9	Few(2%)	65.5%
Cleveland	303	2	13	Few(2%)	54.5%
Credit	690	2	15	Few(5%)	55.5%
Diabetes	768	2	8	None	65.1%
German	1000	2	24	None	70.0%
Glass	213	6	9	None	35.5%
Heart	270	2	13	None	55.6%
Hepatitis	155	2	19	Yes (48%)	79.4%
Ionosphere	351	2	33	None	64.1%
Iris	150	3	4	None	33.3%
Monks-1	432	2	6	None	50%
Monks-2	432	2	6	None	67%
Monks-3	432	2	6	None	53%
Mushroom	8124	2	22	Yes (30%)	51.8%
Satimage	6435	6	36	None	23.8%
Segment	2310	7	18	None	14.3%
Sonar	208	2	60	None	53.4%
Vehicle	846	4	18	None	25.8%
Votes	435	2	16	Yes (47%)	61.4%
Waveform	2581	3	21	None	34.9%
Wine	178	3	13	None	39.9%

- *Australian*

  The problem concerns credit card applications. All attribute names and values have been changed to meaningless symbols to protect confidentiality of the data.

  This dataset is interesting because there is a good mix of attributes – continuous, nominal with small numbers of values, and nominal with larger numbers of

values. There were originally a few missing values, but these have all been replaced by the overall median. This dataset was used in the StatLog project.

- *Balance*
  This data set was generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced. The attributes are the left weight, the left distance, the right weight, and the right distance. The correct way to find the class is the greater of $left\_distance \times left\_weight$ and $right\_distance \times right\_weight$. If they are equal, it is balanced. This is a three class problem, with four numerical attributes. The dataset contains 625 records.

- *Banding*
  Rotogravure printing involves rotating a chrome-plated, engraved copper cylinder in a bath of ink, scraping off the excess ink, and pressing a continuous supply of paper against the inked image with a rubber roller. Sometimes a series of grooves – called a *band* –appears in the cylinder during printing, ruining the finished product. *Banding* results in the press being shut down for an average of about 1.5 hours. Many process features were thought to contribute to banding, but there was no documentation conclusively associating them with the problem. The plant of R.R. Donnelley and Sons at Gallatin, Tennessee, has collected data from several hundred printing jobs, some affected by banding and others not. This dataset concentrates on the data for one particular printing press. Each case describes characteristics of one job and the control settings on the printing hardware.

- *Breast Cancer (Wisconsin)*
  This one of the breast cancer databases available at UCI. This breast cancer database was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg. The problem is to predict whether a tissue sample taken from a patient's breast is malignant or benign. There are two classes, nine numerical attributes and 699 observations.

- *Cleveland*
  The problem consists of predicting the presence of heart disease given the results of various medical tests carried out on a patient. The dataset contains 13 attributes and 303 records.

- *Credit*

This problem concerns credit card applications. All attribute names and values have been changed to meaningless symbols to protect confidentiality of the data.

- *Diabetes*
  The problem is to predict whether a patient would test positive for diabetes given a number of physiological measurements and medical test results. The diagnostic, binary-valued variable investigated is whether the patient shows signs of diabetes according to World Health Organization criteria (i.e., if the 2 hour post-load plasma glucose was at least 200 mg/dl at any survey examination or if found during routine medical care). The patients in the database are females at least twenty-one years old of Pima Indian heritage, living near Phoenix, Arizona, USA. There are two classes, seven numerical attributes and 768 records.

- *German*
  This is a dataset in the area of credit. The goal is to predict if a customer as a good or bad credit risk. It was used in the StatLog project. The dataset contains 1000 records with 24 numerical attributes.

- *Glass*
  The task is to identify a glass sample taken from the scene of an accident as one of six types of glass. This dataset was created in the Central Research Establishment, UK. Each case consists of 9 chemical measurement on one of 6 types of glass. There are 214 cases.

- *Heart*
  This is the StatLog version of the Cleveland database available from the UCI. The purpose of this dataset is to predict the presence or absence of heart disease given the results of various medical tests carried out on a patient. There are two classes, seven numerical attributes, six categorical attributes and 270 examples.

- *Hepatitis*
  The task for this domain is to predict from test results whether a patient will live or die from hepatitis. Each instance is described by 19 features (7 numeric and 12 boolean). The dataset contains 155 records.

- *Ionosphere*
  This radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of

structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere.

Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this database are described by 2 attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal.

- *Iris*
  This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are not linearly separable from each other.

- *Monks*
  The Monks problems are three artificial problems originally used to compare the performance of algorithms [Tea91]. In the given domain robots are characterized by six attributes:

  1. Head-shape $\in$ {round, square, octagon}
  2. Body-shape $\in$ {round, square, octagon}
  3. Is-smiling $\in$ {yes, no}
  4. Holding $\in$ {sword, ballon, flag}
  5. Jacket-color $\in$ { red, yellow, green, blue}
  6. Has-tie $\in$ {yes, no}

  Each problem contains 432 examples. The target concept for each of the three problems are:

  - Monk 1
    (Head-shape = Body-shape) or (Jacket-color = red)
  - Monk 2
    Exactly two of the features have their *first* value
  - Monk 3
    (Jacket-color = green and holding = sword) or (Jacket-color $\neq$ blue and body-shape $\neq$ octagon)

- *Mushrooms*

  This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom. Nevertheless the *odor* feature predicts the class with over 98% of accuracy.

- *Satimage*

  The database consists of the multi-spectral values of pixels in 3x3 neighbourhoods in a satellite image, and the classification associated with the central pixel in each neighbourhood. The aim is to predict this classification, given the multi-spectral values. In the sample database, the class of a pixel is coded as a number. There are six classes, thirty-six attributes and 6435 records. It was used in the Statlog project.

- *Segmentation*

  The problem consists on identifying an outdoor image. The instances were drawn randomly from a database of 7 outdoor images. The images were hand segmented to create a classification for every pixel as one of brickface, sky, foliage, cement, window, path or glass. It was used in the Statlog project. There are seven classes, nineteen numerical attributes and 2319 records.

- *Sonar*

  This is the data set used by Gorman and Sejnowski in their study of the classification of sonar signals using a neural network. The task is to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock.

  The file contains 111 patterns obtained by bouncing sonar signals off a metal cylinder at various angles and under various conditions and it contains 97 patterns obtained from rocks under similar conditions. The transmitted sonar signal is a frequency-modulated chirp, rising in frequency. The data set contains signals obtained from a variety of different aspect angles, spanning 90 degrees for the cylinder and 180 degrees for the rock.

  Each pattern is a set of 60 numbers in the range 0.0 to 1.0. Each number represents the energy within a particular frequency band, integrated over a

certain period of time. The integration aperture for higher frequencies occur later in time, since these frequencies are transmitted later during the chirp.

The label associated with each record contains the letter "R" if the object is a rock and "M" if it is a mine (metal cylinder). The numbers in the labels are in increasing order of aspect angle, but they do not encode the angle directly.

- *Vehicle*
  The problem is to classify a given silhouette as one of four types of vehicle, using a set of features extracted from the silhouette. The original purpose was to find a method of distinguishing 3D objects within a 2D image by application of an ensemble of shape feature extractors to the 2D silhouettes of the objects. Measures of shape features extracted from example silhouettes of objects to be discriminated were used to generate a classification rule by means of computer induction. This object recognition strategy was successfully used to discriminate between silhouettes of model cars, vans and buses viewed from constrained elevation but all angles of rotation. The goal is to classify a given silhouette as one of four types of vehicle, using a set of features extracted from the silhouette. The vehicle may be viewed from one of many different angles.

- *Votes*
  The problem is to predict the party affiliation (Democrat or Republican) of a member of the USA Congress based on sixteen votes. This data set includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the Congressional Quarterly Almanac. Each vote has three possible values: yes, no, or neither. There are two classes, sixteen categorical attributes and 435 records.

- *Waveform*
  This is an atificial domain with three classes based on three waveforms. Each class consists of a random convex combination of two of these waveforms with random noise added. It was defined in the CART book [BFOS84]. The version that we use contains twenty-one numerical attributes and 2581 records.

- *Wine*
  This data contains the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

References

[AP96] K. Ali and M. Pazzani. Error reduction through learning multiple descriptions. *Machine Learning, Vol. 24, No. 1*, 1996.

[Bay98] Stephen D. Bay. Combining nearest neighbor classifiers through multiple feature subsets. In Jude Shavlik, editor, *Proceedings of the 15th International Conference -ICML'98.* Morgan Kaufmann, 1998.

[BEHM90] A. Blumer, A. Ehrenfeucht, D. Haussler, and M.Warmuth. Occam's razor. In Jude Shavlik and Thomas Dieterich, editors, *Readings in Machine Learning.* Morgan Kaufmann Publishers, 1990.

[BFOS84] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth International Group., 1984.

[BH94] P. Brazdil and R. Henery. Analysis of results - cap.10. *Machine Learning, Neural and Statistical Classification*, 1994.

[BJ77] G. Bhattacharyya and R. Johnson. *Statistical concepts and methods.* New York, Willey and Sons, 1977.

[BK99] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105–139, 1999.

[BKM99] C. Blake, E. Keogh, and C.J. Merz. UCI repository of Machine Learning databases, 1999.

[BM94] K. Bennett and O. Mangasarian. Multicategory discrimination via linear programmig. *Optimization: Methods and Software, 3*, 1994.

[BN92] Wray Buntine and Tim Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8:75–85, 1992.

[Bra84] Ivan Bratko. *Prolog, Programming for Artificial Intelligence.* Addison-Wesley Publishing Company, 1984.

[Bre96a] L. Breiman. Bias, variance, and arcing classifiers. Technical report 460, Statistics Department, University of California, 1996.

[Bre96b] Leo Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[Bre96c] Leo Breiman. Stacked regressions. *Machine Learning*, 24:49–64, 1996.

[Bre98] L. Breiman. Arcing classifiers. *The Annals of Statistics*, 26(3):801–849, 1998.

[Bro93] C. Brodley. Addressing the selective superiority problem: Automatic algorithm / model class selection problem. In P. Utgoff, editor, *Machine Learning, Proceedings of the 10th International Conference.* Morgan Kaufmann, 1993.

[Bro95a] C. Brodley. Automatic selection of split criterion during tree growing based on node location. In A. Prieditis and S. Russel, editors, *Machine Learning, Proceedings of the 12th International Conference.* Morgan Kaufmann, 1995.

[Bro95b] C. Brodley. Recursive automatic bias selection for classifier construction. Machine Learning, 20, 1995.

[BT90] P. Brazdil and L. Torgo. Knowledge acquisition via knowledge integration. In B. Wielinga and et al., editors, *Current Trends in Knowledge Acquisition.* IOS Press, 1990.

[BU92] C. Brodley and P. Utgoff. Multivariate versus univariate decision trees. Coins technical report, 92-8, University of Massachusetts, 1992.

[BU95] Carla E. Brodley and Paul E. Utgoff. Multivariate decision trees. *Machine Learning*, 19:45–77, 1995.

[Bun90] Wray Buntine. *A theory of Learning Classification Rules.* PhD thesis, University of Sydney, 1990.

[Cat91] J. Catlett. On changing continuous attributes into ordered discrete attributes. In Y. Kodratoff, editor, *European Working Session on Learning -EWSL91.* LNAI 482 Springer Verlag, 1991.

[CJ97] W. Cohen and D. Jensen. Overfitting explained. In *Preliminar papers of the 6th International Workshop on Artificial Intelligence and Statistics*, 1997.

[CKB87] B. Cestnik, I. Kononenko, and I. Bratko. Assistant 86: a knowledge-elicitation tool for sophisticated users. In I. Bratko and N. Lavrac, editors, *European Working Session on Learning -EWSL87*. Sigma Press, Wilmslow, England, 1987.

[CN89] Peter Clark and Tim Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.

[Coh95] W. Cohen. Fast effective rule induction. In A. Prieditis and S. Russel, editors, *Machine Learning, Proceedings of the 12th International Conference*. Morgan Kaufmann, 1995.

[CS95a] P. Chan and S. Stolfo. A comparative evaluation of voting and meta-learning on partitioned data. In A. Prieditis and S. Russel, editors, *Machine Learning Proc of 12th International Conference*. Morgan Kaufmann, 1995.

[CS95b] P. Chan and S. Stolfo. Learning arbiter and combiner trees from partitioned data for scaling machine learning. In *KDD 95*, 1995.

[CS97] P. Chan and S. Stolfo. On the accuracy of meta-learning for scalable data mining. *Journal of Intelligent Information systems*, 8:5–28, 1997.

[Dan95] Wayne Daniel. *Biostatistics: a foundation for analysis in the health sciences*. John Wiley and Sons, Inc., 1995.

[DB95] T. Dietterich and G. Bakiri. Solving multiclass problems via error-correcting output codes. *Journal of Artificial Intelligence Research, Vol. 2*, 1995.

[DG84] W. Dillon and M. Goldstein. *Multivariate Analysis, Methods and Applications*. J. Wiley and Sons, Inc., 1984.

[DH73] R.O. Duda and P.E. Hart. *Pattern Classification and scene analysis*. New York, Willey and Sons, 1973.

[Die96] T. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. Corvallis, technical report nr. 97.331, Oregon State University, 1996.

[Die97] T. Dietterich. Machine learning research: four current directions. *AI Magazine, Vol. 18, No. 4*, 1997.

[DKS95] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In A. Prieditis and S. Russel, editors, *Machine Learning Proc. of 12th International Conference*. Morgan Kaufmann, 1995.

[Dom96] Pedro Domingos. Unifying instance-based and rule-based induction. *Machine Learning*, 24:141–168, 1996.

[Dom97a] Pedro Domingos. Bayesian model averaging in rule induction. In Padhraic Smyth and David Madigan, editors, *Preliminary papers of the Sixth International Workshop on Artificial Intelligence and Statistics*, 1997.

[Dom97b] Pedro Domingos. *A Unified Approach to Concept Learning*. PhD thesis, University of California, Irvine, 1997.

[Dom97c] Pedro Domingos. Why Does Bagging Work? A Bayesian account and its implications. In David Heckerman, Heikki Mannila, and Ramasamy Uthurusamy, editors, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1997.

[Dom98a] Pedro Domingos. Knowledge discovery via multiple models. *Intelligent Data Analysis*, 24(2), 1998.

[Dom98b] Pedro Domingos. Occam's two razors: The sharp and the blunt. In R. Agrawal and P. Stolorz, editors, *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1998.

[Dom98c] Pedro Domingos. A process-oriented heuristic for model selection. In *Proceedings of the 15th International Conference -ICML'98*, 1998.

[DP97] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29:103–129, 1997.

[DR95] S.K. Donoho and L.A. Rendell. Rerepresenting and restructuring domain theories: A constructive induction approach. *Journal of Artificial Inteligence Research*, pages Vol. 2, 411–446, 1995.

[EMS93a] F. Esposito, D. Malerba, and G. Semeraro. Decision tree pruning as a search in the state space. In P. Brazdil, editor, *Machine Learning: ECML93*. LNAI 667, Springer Verlag, 1993.

[EMS93b] F. Esposito, D. Malerba, and G. Semeraro. Decision tree prunning as a search in the state space. In P.Brazdil, editor, *Machine Learning: ECML-93*. LNAI 667, Springer Verlag, 1993.

[EMS97] F. Esposito, D. Malerba, and G. Semeraro. A comparative analysis of methods for pruning decision trees. *Transactions on Pattern Analysis and Machine Intelligence*, 19(5), 1997.

[FI93] U. Fayyad and K Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *13th International Joint Conference of Artificial Intelligence*, 1993.

[FKY96] Jerome H. Friedman, Ron Kohavi, and Yeogirl Yun. Lazy decision trees. In AAAI, editor, *Thirteenth National Conference on Artificial Intelligence*. MIT Press, 1996.

[FL90] Scott E. Fahlman and Christian Lebiere. The Cascade-Correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, February 1990.

[Fri99] J. Friedman. Greedy function approximation: a gradient boosting machine. Technical report, Statistics Department, Stanford University, 1999.

[FS96] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In L. Saitta, editor, *Machine Learning, Proceedings of the 13th International Conference*. Morgan Kaufmann, 1996.

[Fuk90] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, Inc., 1990.

[FW98] Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the 15th International Conference -ICML'98*, 1998.

[Gam98] J. Gama. Combining classifiers by constructive induction. In C. Nedellec and C. Rouveirol, editors, *Machine Learning ECML-98*. Springer Verlag, 1998.

[GC92] O. Gascuel and G. Caraux. Statistical significance in inductive learning. In B.Neumann, editor, *10th European Conference on Artificial Intelligence.* John Willey and Sons, 1992.

[GTS98] J. Gama, L. Torgo, and C. Soares. Dynamic discretization of continuous attributes. In Helder Coelho, editor, *Progress in Artificial Intelligence-Iberamia-98.* LNAI 1484, Springer Verlag, 1998.

[HHS94] T. Ho, J. Hull, and S. Srihari. Decision combination in multiple classifier systems. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1994. Vol. 16, N 1.

[HKS93] D. Heath, S. Kasif, and S. Salzberg. Induction of oblique decision trees. In *Proc. of the 13th IJCAI*, pages 1002–1007, Chambery, France, 1993.

[HKS96] D. Heath, S. Kasif, and S. Salzberg. Committees of decision trees. 1996.

[HS90] L.K. Hansen and P. Salamon. Neural networks ensembles. *Transactions on Pattern Analysis and Machine Intelligence*, 12(10), 1990.

[IS96] A. Ittner and M. Schlosser. Non-linear decision tees - ndt. In L. Saitta, editor, *Machine Learning Proc of 13th International Conference.* Morgan Kaufmann, 1996.

[JJ94] M. Jordan and R. Jacob. Hierarchical mixtures of experts and the EM algorithm. *Neural Computing, n.6*, 1994.

[Joh96] G. John. Robust linear discriminant trees. In D. Fisher, editor, *Learning from Data: Artificial Intelligence and Statistics V.* Springer Verlag, 1996.

[Joh97] George John. *Enhancements to the data mining process.* PhD thesis, Stanford University, 1997.

[Jor98] Alipio Jorge. *Iterative Induction of Logic Programs.* PhD thesis, University of Porto, 1998.

[KD95a] E.B. Kong and T. Dietterich. Error-correcting output coding correct bias and variance. In A. Prieditis and S. Russel, editors, *Machine Learning Proc of 12th International Conference.* Morgan Kaufmann, 1995.

[KD95b] E.B. Kong and T. Dietterich. Error-correcting output coding correct bias and variance. In A. Prieditis and S. Russel, editors, *Machine Learning Proc of 12th International Conference.* Morgan Kaufmann, 1995.

[Kit98] J. Kittler. Combining classifiers: A theoretical framework. *Pattern analysis and Applications, Vol. 1, No. 1*, 1998.

[KK97] R. Kohavi and C. Kunz. Option decision trees with majority votes. In D. Fisher, editor, *Machine Learning Proc. of 14th International Conference*. Morgan Kaufmann, 1997.

[KL90] D. Kibler and P. Langley. Machine learning as an experimental science. In Jude Shavlik and Thomas Dieterich, editors, *Readings in Machine Learning*. Morgan Kaufmann Publishers, 1990.

[KM96] Dimitrios Kalles and Tim Morris. Efficient incremental induction of decision trees. *Machine Learning*, 24:231–242, 1996.

[Koh95a] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proc. of the 14th IJCAI*, pages 1137–1143, Montreal, Canada, 1995.

[Koh95b] Ron Kohavi. *Wrappers for Performance Enhancement and Oblivious decision Graphs*. PhD thesis, University of Stanford, 1995.

[Koh96] R. Kohavi. Scaling up the accuracy of naive bayes classifiers: a decision tree hybrid. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1996.

[Kon91] I. Kononenko. Semi-naive bayesian classifier. In Y. Kodratoff, editor, *European Working Session on Learning -EWSL91*. LNAI 482 Springer Verlag, 1991.

[KS96] Ron Kohavi and Mehran Sahami. Error-based and entropy-based discretization of continuous features. In Evangelos Simoudis, Jia Wei Han, and Usama Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1996.

[KSD97] R. Kohavi, D. Sommerfield, and J. Dougherty. Data mining using MLC++, a machine learning library in C++. *International Journal of Artificial Intelligence Tools*, 6 Nr. 4, 1997.

[KW96] Ron Kohavi and David H. Wolpert. Bias plus variance decomposition for zero-one loss functions. In L. Saitta, editor, *Machine Learning, Proceedings of the 13th International Conference*. Morgan Kaufmann, 1996.

[Lan93a] P. Langley. Induction of recursive bayesian classifiers. In P.Brazdil, editor, *Machine Learning: ECML-93.* LNAI 667, Springer Verlag, 1993.

[Lan93b] P. Langley. Induction of recursive bayesian classifiers. In P.Brazdil, editor, *Machine Learning: ECML-93.* LNAI 667, Springer Verlag, 1993.

[Lan96] Pat Langley. *Elements of Machine Learning.* Morgan Kaufmann, 1996.

[LB91] H. Lounis and G. Bisson. Evaluation of learning systems: an artificial data-based approach. In Y. Kodratoff, editor, *European Working Session on Learning -EWSL91.* LNAI 482 Springer Verlag, 1991.

[LM98] Huan Liu and Hiroshi Motoda. *Feature Selection for Knowledge Discovery and Data Mining.* Kluwer Academic publishers, 1998.

[LS97] W. Loh and Y. Shih. Split selection methods for classification trees. *Statistica Sinica*, 7:815–840, 1997.

[LV88] W. Loh and N. Vanichsetakul. Tree-structured classification via generalized discriminant analysis. *Journal of the American Statistical Association*, 1988.

[Mán91] R. López De Mántaras. A distance-based attribute selection measure for decision tree induction. *Machine Learning*, 6:81–92, 1991.

[Mar97] J.K. Martin. An exact probability metric for decision tree spliting and stopping. *Machine Learning*, 28, 1997.

[Mat91] C. Matheus. The need for constructive induction. In *Machine Learning, Proceedings of the 8th International Conference.* Morgan Kaufmann, 1991.

[Mat98] R. Mattison. AnswerTree algorithm summary, 1998.

[MBB94] Katharina Morik, Francesco Bergadano, and Wray Buntine. Guest editorial. *Machine Learning*, 14:137–138, 1994.

[MD97] Dragos Margineantu and Thomas G. Dietterich. Pruning adaptive boosting. In D. Fisher, editor, *Machine Learning, Proceedings of the 14th International Conference.* Morgan Kaufmann, 1997.

[Mer98] Christopher Merz. *Classification and Regression by Combining Models.* PhD thesis, University of California, Irvine, 1998.

[Mic90] R. Michalski. A theory and methodology of inductive learning. In Jude Shavlik and Thomas Dieterich, editors, *Readings in Machine Learning.* Morgan Kaufmann Publishers, 1990.

[Min89a] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4:227–243, 1989.

[Min89b] John Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3:319–342, 1989.

[Mit90a] Tom Mitchell. Generalization as search. In Jude Shavlik and Thomas Dieterich, editors, *Readings in Machine Learning.* Morgan Kaufmann Publishers, 1990.

[Mit90b] Tom Mitchell. The need for biases in learning generalizations. In Jude Shavlik and Thomas Dieterich, editors, *Readings in Machine Learning.* Morgan Kaufmann Publishers, 1990.

[Mit97] Tom Mitchell. *Machine Learning.* MacGraw-Hill Companies, Inc., 1997.

[MKS94] S.K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Inteligence Research*, Vol.2:1–32, 1994.

[MR89] C. Matheus and L. Rendell. Constructive induction on decision trees. In *Proceedings of IJCAI.* Morgan Kaufmann, 1989.

[MST94] D. Michie, D.J. Spiegelhalter, and C. Taylor. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, 1994.

[MSW90] O. Mangasarian, R. Setiono, and W. Wolberg. Pattern recognition via linear programming: theory and applications to medical diagnosis. In T. Coleman and Y. Li, editors, *SIAM*, 1990.

[MT94] R. Michalski and G. Tecuci. *Machine Learning - A multistrategy Approach, Vol. 4.* Morgan Kaufmann, 1994.

[Ort95] J. Ortega. Exploiting multiple existing models and learning algorithms. In *AAAI 96 - Workshop in Induction of Multiple Learning Models*, 1995.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers, Inc., 1988.

[PF96] P. Pompe and A. Feelders. Using machine learning, neural networks and statistics to predict corporate bankruptcy: a comparative study. In Phillip Ein-Dor, editor, *Artificial Intelligence in Economics and Management.* Kluwer Academic Publishers, 1996.

[PH90] Giulia Pagallo and David Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99, 1990.

[PTVF92] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C: the art of scientific computing 2 Ed.* University of Cambridge, 1992.

[Qui79] R. Quinlan. Discovering rules by induction from large collections of examples. In Donald Michie, editor, *Expert Systems in the Microelectronic Age.* Edinburgh University Press, 1979.

[Qui86] R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Qui88] R. Quinlan. Simplifying decision trees. 1988.

[Qui92] R. Quinlan. Learning with continuous classes. In Adams and Sterling, editors, *Proceedings of AI'92.* World Scientific, 1992.

[Qui93] R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, Inc., 1993.

[Qui94] R. Quinlan. Comparing connectionist and symbolic learning methods. In S. Hanson, G. Drastal, and R. Rivest, editors, *Computational Learning theory and natural learning systems.* MIT Press, 1994.

[Qui95] R. Quinlan. Mdl and categorical theories (continued). In A. Prieditis and S. Russel, editors, *Machine Learning Proc of 12th International Conference.* Morgan Kaufmann, 1995.

[Qui96a] R. Quinlan. Bagging, Boosting and C4.5. In *Procs. 13th American Association for Artificial Intelligence.* AAAI Press, 1996.

[Qui96b] R. Quinlan. Improved use of continuous attributes in c4.5. *Journal of Artificial Intelligence Research, 4*, 1996.

[Qui99] R. Quinlan. Miniboosting decision trees. *Journal of Artificial Inteligence Research*, to appear, 1999.

[RA98] F. Ricci and D. Aha. Error-correcting output codes for local learners. In C. Nedellec and C.Rouveirol, editors, *Machine Learning ECML-98*. LNAI 1398, Springer Verlag, 1998.

[RC90] Larry Rendell and Howard Cho. Empirical learning as a function of concept character. *Machine Learning*, 5:267–298, 1990.

[Rip96] Brian D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[Ris83] J. Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11, 1983.

[Riv87] Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2:229–246, 1987.

[Sal97] S. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery, Vol. 1*, 1997.

[Sch90] Robert Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

[Sch93] Cullen Schaffer. Overfitting avoidance as bias. *Machine Learning*, 10:153–178, 1993.

[Sch95] C. Schaffer. A conservation law for generalization performance. In A. Prieditis and S. Russel, editors, *Machine Learning, Proceedings of the 12th International Conference*. Morgan Kaufmann, 1995.

[SFBL97] Robert Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. In D. Fisher, editor, *Machine Learning, Proceedings of the 14th International Conference*. Morgan Kaufmann, 1997.

[Ska97] David Skalak. *Prototype Selection For Composite Nearest Neighbor Classifiers*. PhD thesis, University of Massachusetts Amherst, 1997.

[Sta99] StatSoft. *Electronic Statistics Textbook*. Tulsa, StatSoft Inc., 1999.

[SW97] B. Schulmeister and F. Wysotzki. DIPOL - A hybrid piecewise linear classifier. In G. Nakhaeizadeh and C. Taylor, editors, *Machine Learning and Statistics*. John Wiley and Sons, Inc., 1997.

[Tea91] S. Thrun and et all. The monk's problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, 1991.

[TG95] K. Tumer and J. Ghosh. Classifier combining: analytical results and implications. In *AAAI 96 - Workshop in Induction of Multiple Learning Models*, 1995.

[TG96a] K. Tumer and J. Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection Science, Special issue on combining artificial neural networks: ensemble approaches*, 8, N.3-4, 1996.

[TG96b] K. Tumer and J. Ghosh. Theoretical foundations of linear and order statistics combiners for neural pattern classifiers. Technical Report TR-95-02-98, 1996.

[Tin96] Kai Ming Ting. The characterisation of predictive accuracy and decision combination. In L. Saitta, editor, *Machine Learning, Proceedings of the 13th International Conference*. Morgan Kaufmann, 1996.

[Tin97] K.M. Ting. Decision combination based on characterization of predictive accuracy. *Intelligent Data Analysis, Vol. 2, No. 1*, 1997.

[TW97] K.M. Ting and I.H. Witten. Stacked generalization: when does it work? In *Procs. International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1997.

[UB90] P. Utgoff and C. Brodley. An incremetal method for finding multivariate splits for decision trees. In B. Porter and R. Mooney, editors, *Machine Learning, Proceedings of the 7th International Conference*. Morgan Kaufmann, 1990.

[UB91] P. Utgoff and C. Brodley. Linear machine decision trees. Coins technical report, 91-10, University of Massachusetts, 1991.

[UBC97] Paul Utgoff, Neil Berkman, and Jeffery Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29:5, 1997.

[Utg89] Paul E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161–186, 1989.

[Van93] T. Van de Merckt. Decision trees in numerical attribute spaces. In *Proc. of the 13th IJCAI*, pages 1016–1021, Chambery, France, 1993.

[Vap95] Vladimir Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, 1995.

[WI98] Sholom Weiss and Nitin Indurkhya. *Predictive Data Mining, a practical Guide.* Morgan Kaufmann Publishers, 1998.

[WK91] S. Weiss and C. Kulikowski. *Computer systems that learn-Classification and prediction methods from Statistics, Neural Nets, Machine Learning and Expert systems.* Morgan Kaufmann Publishers, Inc., 1991.

[WM94] Janusz Wnek and Ryszard S. Michalski. Hypothesis-driven constructive induction in AQ17-HCI: A method and experiments. *Machine Learning,* 14:139–168, 1994.

[Wol92] D. Wolpert. Stacked generalization. In Pergamon Press, editor, *Neural Networks Vol.5,* 1992.

[WP93] C.S. Wallace and J.D. Patrick. Coding decision trees. *Machine Learning,* 11:7–22, 1993.

[YW94] S. Yip and G. Webb. Incorporating canonical discriminant attributes in classification learning. In *Proceedings of the 10th Canadian Conference on Artificial Intelligence.* Morgan Kaufmann, 1994.

[Zhe96] Z. Zheng. *Constructing New Attributes for Decision Tree Learning.* PhD thesis, University of Sydney, 1996.

[Zhe98] Z. Zheng. Naive bayesian classifier committees. In C. Nedellec and C. Rouveirol, editors, *Machine Learning ECML-98.* Springer Verlag, 1998.