Luís Fernando Raínho Alves Torgo

INDUCTIVE LEARNING OF TREE-BASED REGRESSION MODELS

Tese submetida para obtenção
do grau de Doutor em Ciência de Computadores

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

Dedication

Para o meu avô.

Acknowledgements

I would like to thank my supervisor, Pavel Brazdil, for all his support since I started my research on Machine Learning. He has managed to provide me with the ideal mixture of research guidance and freedom to follow my own intuitions. Thanks also for all his useful comments and suggestions that certainly improved this thesis. Finally, I would like also to thank him for all his effort in management tasks that provided me and my colleagues with such excellent working conditions.

Thanks to all my colleagues at LIACC for such pleasant working atmosphere. Thanks in particular to all those in NIAAD (machine learning group) for useful comments and discussions. A special word to my two colleagues with whom I share the office: Alípio Jorge and João Gama. Finally, I would like to thank my colleague Joaquim Pinto da Costa for his support with questions concerning statistics.

I would like to thank the Faculty of Economics of the University of Porto and the Portuguese Government (Programa Ciência and PRAXIS XXI) for all financial support that made this thesis possible. Thanks also are due to the European Community for its financial support to several research projects and networks on which I was involved.

Thanks to all my good friends for all nice moments that allowed me to escape from my research preoccupations. Their friendship made things much easier.

Special thanks to Catarina. Her love and joy are an unlimited source of happiness.

Finally, a very special thanks to my family and in particular my parents. Without their love and unquestionable support nothing of this would ever be possible.

Abstract

This thesis explores different aspects of the induction of tree-based regression models from data. The main goal of this study is to improve the predictive accuracy of regression trees, while retaining as much as possible their comprehensibility and computational efficiency. Our study is divided in three main parts.

In the first part we describe in detail two different methods of growing a regression tree: minimising the mean squared error and minimising the mean absolute deviation. Our study is particularly focussed on the computational efficiency of these tasks. We present several new algorithms that lead to significant computational speed-ups. We also describe an experimental comparison of both methods of growing a regression tree highlighting their different application goals.

Pruning is a standard procedure within tree-based models whose goal is to provide a good compromise for achieving simple and comprehensible models with good predictive accuracy. In the second part of our study we describe a series of new techniques for pruning by selection from a series of alternative pruned trees. We carry out an extensive set of experiments comparing different methods of pruning, which show that our proposed techniques are able to significantly outperform the predictive accuracy of current state of the art pruning algorithms in a large set of regression domains.

In the final part of our study we present a new type of tree-based models that we refer to as *local regression trees*. These hybrid models integrate tree-based regression with local modelling techniques. We describe different types of local regression trees and show that these models are able to significantly outperform standard regression trees in terms of predictive accuracy. Through a large set of experiments we prove the competitiveness of local regression trees when compared to existing regression techniques.

Contents

List of Tables

List of Figures

List of Symbols

X_i is an input (predictor) variable (or attribute).

A is the set of attributes of a problem.

a is the number of attributes (or predictor variables) of a problem, that is $a = \#A$.

χ_i is the domain of the variable X_i .

Y is a goal variable.

Υ is the domain of variable Y.

$\mathbf{x}_i$ is a vector of attribute values having as domain $\chi = \chi_1 \times \ldots \times \chi_a$.

$\langle \mathbf{x}_i, y_i \rangle$ is a case or example with domain $\chi \times \Upsilon$.

$D = \{\langle \mathbf{x}_i, y_i \rangle\}_{i=1}^{n}$ is a data set or sample of cases.

n is the number of training cases of a problem.

$\mathbf{D} = \mathbf{X} \mid \mathbf{Y}$ is a matrix representation of a data set, where $\mathbf{X}$ is a $n \times a$ matrix containing the n vectors of a attributes values, and $\mathbf{Y}$ is a column vector containing the n goal variable values.

$r(\boldsymbol{\beta}, \mathbf{x})$ is a regression model with parameters $\boldsymbol{\beta}$, which maps an attribute vector into a goal variable value, *i.e.* $r : \chi \rightarrow \Upsilon$.

$r(\boldsymbol{\beta}, \mathbf{x}_i)$ is the predicted value of the model $r(\boldsymbol{\beta}, \mathbf{x})$ for a case $\langle \mathbf{x}_i, y_i \rangle$.

T is a tree-based model.

t is a node of a tree T.

T_t is a sub-tree of T rooted at node t.

t_L is the left child node of node t.

t_R is the right child node of node t.

l is a leaf node of a tree.

$\tilde{T}$ is a set containing all leaf nodes of a tree T.

$\#\tilde{T}$ is the number of leaves of a tree, *i.e.* the cardinality of set $\tilde{T}$.

P_i is a path from the root node t_0 to a node t_i. We represent this path by the conjunction of logical tests of the nodes in the path.

D_t is the sub-set of the training sample D falling in node t, that is $D_t = \{\langle \mathbf{x}_i, y_i \rangle \in D : \mathbf{x} \rightarrow P_t\}$.

Chapter 1

Introduction

This dissertation addresses a particular methodology to deal with a data analysis problem usually known as multivariate regression. A *multivariate regression problem* can be loosely defined as the study of the *dependency relation* between a *goal* (*dependent*, *response*) variable and a set of *independent* (*predictor*, *input*) variables. The goal of multivariate regression analysis is to *obtain a model of this relationship*. This model can be used either for *understanding* the interactions between the variables of the domain under study, or to *predict* the value of the goal variable of future instances of the same problem. Regression models are usually obtained through an *inductive process* that uses examples (*the training sample*) of an unknown regression relationship. Consider for instance a collection of observations regarding some financial indicators of thousands of companies and their respective stock values. We could be interested in obtaining a regression model that relates the stock value with some of these indicators. This model could then be used to predict the stock value of a new company for which we know the value of the indicators. Moreover, if our model is somehow intelligible it could reveal some yet unknown relationships in this domain.

The existence of a functional relationship between a target variable and a set of predictor variables is common in real world data analysis (Drapper & Smith, 1981).

Finding a model of such relationship is very important because it allows us to predict and comprehend the behaviour of our data. These and other arguments justify the existence of so many research works carried out about this data analysis problem. Still, the majority of studies about regression analysis concern methods of generating a *multivariate linear model* between the response variable and the predictors. Although being one of the most successful data analysis techniques, linear regression has obvious limitations due to the linear structure that is imposed on the data. To deal with data with more complex structure one needs more sophisticated tools. *Non-parametric approaches to regression* are able to deal with a wide range of regression domains by not imposing any pre-defined global form to the regression surface (Hardle, 1990). These methods assume some functional form only at a *local level*, meaning that they do not try to fit one single model to all given sample. This methodology leads to a regression surface that is globally non-parametric (*i.e.* an approximation that does not have a fixed global functional form). This is an important feature if one wishes to develop a tool that is applicable to a wide range of regression problems. That is one of the goals of the work carried out in this dissertation.

The approach to regression analysis followed in this thesis is usually known as *tree-based regression*. The regression models obtained with this methodology can be represented in the form of a tree. This tree consists of a hierarchy of nodes, starting with a top node known as the *root node*. Each node of the tree contains *logical tests* on the predictor variables, with the exception of the bottom nodes of the hierarchy. These latter are usually known as the leaves of the tree. The leaves contain the predictions of the tree-based model. Each path from the root node to a leaf can be seen as a conjunction of logical tests on the predictor variables. These conjunctions are logical representations of "sub-areas" of the overall regression surface being approximated. For each of these local areas different values of the goal variable can be predicted. Although the form of the logical tests imposes some restrictions on the type of local areas that can be represented, the number of local areas and their "size" is not pre-defined. Because of this regression trees are able to represent a wide range of regression surfaces through a flexible composition of smaller surfaces. The main features of tree-based regression are its *wide range of applicability*, *low computation time* and its *comprehensibility* due to the use of a symbolic representation of the regression surface. These models are studied in several research fields like social sciences, statistics, engineering and artificial intelligence.

Most existing methods of growing a regression tree try to minimise the *mean squared error* of the resulting model. This method has important computational advantages stemming from the mathematics behind this criterion. However, mean squared error is known to be quite sensitive to *outliers* (*i.e.* data points that diverge a lot from the bulk of data). These data items may distort the mean squared error statistic possibly leading to wrong decisions in the process of growing a tree. We see this as an opportunity for considering other growth criteria that do not suffer from this effect.

As we have mentioned, comprehensibility is one of the main advantages of regression trees. However, as these models do not assume any fixed form of the regression surface, they can easily *overfit the given training sample*, by generating too many local areas to approximate the unknown surface. This leads to models that are unnecessarily large with consequences in terms of comprehensibility. Moreover, overfitting may also cause *lower predictive accuracy*, particularly with unreliable data (*i.e. noisy* data). *Pruning* is the usual method of avoiding overfitting in regression trees. Pruning is usually regarded as a search through the space of all possible pruned trees of an overly large tree that overfits the training data. Current approaches to this search problem use methods relying either on *estimates of the true prediction error* of the trees (*e.g.* Breiman *et al.*, 1984) or other criteria like *minimising the binary description length* of the tree model (Robnik-Sikonja & Kononenko, 1998).

The non-parametric features of regression trees usually lead to competitive predictive accuracy in a large set of domains. However, regression trees can be seen as fitting constants in a set of partitions of the input space (the multidimensional space defined by the predictor variables). As we have mentioned, a regression tree is a logic representation of this set of partitions. In each of these regions of the input space regression trees assume that the unknown regression function takes a constant value. This leads to a *highly non-smooth set of local regions*, that all together form the regression surface approximation provided by a regression tree. In this thesis we explore the possibility of introducing a further level of detail within these partitions by *using smother models* within each leaf of the trees.

1.1 Objectives

The main objective of this thesis is to study different aspects of tree-based regression with the aim of improving the accuracy of these models, while trying to retain their main features concerning applicability, processing time and comprehensibility of the models. Our study is focused on deriving trees from large sets of data and thus we are particularly concerned with methods that are sufficiently efficient. We start our study by exploring computationally efficient methods of growing regression trees using two different error criteria. We improve the computational efficiency of the method of growing least squares regression trees, and we consider the alternative of growing trees that minimise the mean absolute deviation. This latter methodology aims at overcoming difficulties with data outliers. We then address the important question of avoiding overfitting of the training data with the objective of providing a better compromise between accuracy and size of the models. Finally, we describe a new type of models, called local regression trees, that permit to attain higher accuracy than existing approaches to tree-based regression, through the use of smoother models in their leaves. Throughout this thesis we have extensively compared our proposals with existing methods. These comparisons were carried out using several data sets and different training sample sizes.

1.2 Main Contributions

This dissertation presents a thorough study of tree-based regression models. The study is divided in three main parts. The first addresses the generation of tree models. We review several existing techniques of growing least squares (LS) and least absolute deviation (LAD) regression trees. We describe new algorithms for inducing LAD trees that increase the efficiency of this task. We present a theoretical study concerning the possibility of extending a theorem described by Breiman *et al.* (1984), to LAD trees. This extension is relevant because the results of the theorem significantly improve the computational efficiency of finding the best discrete split for a tree node. We also describe an experimental comparison between LS and LAD regression trees. This comparison shows that these two methodologies have different application goals. While LS trees try to minimise the square error of the tree resulting in a model with lower probability of committing large prediction errors, LAD trees minimise the average absolute deviation, which leads to models that on average tend to be more accurate but may occasionally commit larger prediction errors.

The second part of this thesis addresses overfitting avoidance within regression trees. We review the major existing pruning techniques and then focus our study on pruning by tree selection. We describe two new methods of exploring the large space of pruned trees that entail accuracy advantages when compared to existing approaches. We present a new method of using Cross Validation estimates, which extends their applicability in the context of pruning by tree selection. We extend the use of *m* estimators (Cestnik, 1990) within regression trees. We describe an *m* estimator of the Mean Absolute Deviation that allows using this method with LAD trees. We also derive an expression of the standard error of *m* estimates, which allows using the 1-SE rule (Breiman *et al.*,1984) during pruning, with large advantages in terms of tree size. Finally, we present a new error estimator (*ChiEst*) for LS trees based on the sampling distribution properties of the mean squared error, that provides very competitive accuracy with low computation costs.

We have carried out an extensive empirical comparison of different methods of pruning by tree selection. We have concluded that selection based on error estimates obtained either with Cross Validation or our *ChiEst* method are the best ways to achieve higher accuracy in a large set of benchmark domains. However, we have also observed that these methods entail some costs in terms of average tree size. With respect to computation time our *ChiEst* method achieved significant advantages when compared to other existing pruning algorithms.

We have also compared our pruning proposals with three state-of-the-art pruning algorithms. We have observed significant accuracy advantages of our methods at the cost of some tree size, particularly when compared to CART (Breiman *et al.*,1984) and CORE (Robnik-Sikonja & Kononenko,1998) pruning algorithms.

Regarding local regression trees we describe three variants of these models: kernel trees; partial linear trees; and local linear trees. We have confirmed our hypothesis concerning the increased predictive accuracy of these models when compared to standard regression trees. However, we have also observed that this advantage comes at the cost of larger computational requirements and loss of some of comprehensibility of the models. We also show that local regression trees can be seen as a means to overcome some of the limitations of the local models we have integrated in their leaves. In effect, the focusing effect provided by local regression trees is largely beneficial in terms of computation time when compared to the use of local models per see. Finally, in an empirical comparison with three of the most well-known existing regression methodologies, we have observed that partial linear trees are among the most competitive in terms of predictive accuracy.

1.3 Organisation of the Thesis

This dissertation is organised as follows. Chapter 2 presents an overview of the existing work on learning by examples within the propositional framework. This description is particularly focused on regression methods. We provide an overview of the major regression techniques used in the research fields of statistics, neural networks and machine learning.

Chapter 3 presents two methods of growing regression trees. The first is the method of growing a tree by minimising the mean squared error of the resulting model. We discuss several computational issues of this methodology and present some algorithms that improve its efficiency. We then address the question of growing regression trees by trying to minimise the mean absolute deviation. We present several algorithms for growing this type of trees and discuss its computational difficulties. We also provide several techniques that allow a large computational speed-up of the task of growing these trees.

Chapter 4 addresses one of the most important aspects of tree-based regression: overfitting avoidance. We follow the method of pruning by selecting from a set of previously generated pruned trees. We describe two new algorithms for generating these pruned trees. We study several alternative ways of evaluating tree models and apply them in the context of pruning by tree selection. We show the results of an empirical comparison between different methods of pruning by tree selection. Finally, we compare our pruning proposals with three of the most successful existing pruning algorithms.

In chapter 5 we describe a new type of tree-based regression models, named local regression trees. These models integrate regression trees with local modelling techniques. The integration is carried out using local models in the leaves of regression trees. We describe three variants of local regression trees: kernel trees; partial linear trees; and local linear trees. We carry out a series of experiments designed with the goal of finding out the advantages and disadvantages of local regression trees.

We also include a series of Annexes that provide further details on the work carried out in the thesis. Annex A describes the material and methods used in the experimental parts of the thesis. This annex also includes details concerning the form of presenting experimental results used throughout the thesis. Annex B presents the full tables of results of several experiments carried out in thesis that were not included in the main text for clarity reasons.

Chapter 2

Inductive Learning

This chapter presents an overview of the research in inductive learning. We briefly describe the various aspects of this scientific field and present some examples of applications. We then focus on a particular learning task usually known as regression, which is the main topic of this thesis. We present an overview of the existing approaches to this problem within Machine Learning as well as other research fields.

2.1 Introduction

Various definitions of machine learning appear in the field literature. Simon (1983) defined it as "*changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time*". Langley (1996) defined learning as "*the improvement of performance in some environment through the acquisition of knowledge resulting from experience in that environment*". Finally, Mitchell (1997) presented a more operational definition by saying that "*a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E*". There are several important notions common to these definitions. First of all, there is the notion of adaptation resulting from new experience. Also relevant is the demand that this adaptation should lead to an improvement of the system performance in other tasks of the same kind.

Michalski (1994) has presented a formal description of the learning task. This author described the Inferential Theory of Learning (ITL) that looks at *learning as a search* through a knowledge space. This search consists of a series of *knowledge transmutations* to achieve a certain learning goal. According to this author, ITL "*strives to characterise logical capabilities of learning systems …by addressing issues like what types of knowledge transformations occur…, what is the validity of knowledge obtained…, how is prior knowledge used…, what knowledge can be derived from the given input and the prior knowledge…*". These aims distinguish ITL from the Computational Learning Theory (COLT), which focuses on the *computational complexity* and *convergence* of the learning algorithms. The knowledge space is defined by the knowledge representation language used to describe the inputs, the background knowledge and the outputs of the learning system. The search carried out by the learner can be seen as a series of inferences from the given knowledge. Michalski (1994) describes three different types of inference strategies - deduction, induction and analogy. While most existing learning systems use only one type of inference strategy, systems exist that are able to perform knowledge transmutations of different type.

Let us consider the following logical entailment,

$$\Gamma \cup B \models C \tag{2.1}$$

where,

Γ is a set of premises;
B is the learner's background knowledge;
and C is a set of consequences.

From the perspective of this inference equation, *deduction* is a knowledge transmutation step that derives C from the premises Γ and the background knowledge B. This step is said to be truth-preserving as C is a logical consequence of $\Gamma \cup B$. Examples of systems that use this type of truth-preserving inferences can be found in Mitchell *et al.* (1983) and DeJong & Mooney (1986).

Inductive inference, on the other hand, consists of hypothesising the premises Γ, given C and B. This inductive step should ensure that Γ together with B entails C. Moreover, this is a falsity-preserving knowledge transmutation because if C is false, then Γ should also be false. *Analogy* or "similarity-based" inference can be seen as a combination of inductive and deductive inference (see for instance, Winston,1980; Carbonell,1986). In this form of inference, an inductive step is made by hypothesising that a new "term" is similar to an existing "term". Then based on this similarity we may deduce new knowledge using deductive steps that were "applicable" to the first term.

Another important distinction made by Michalski (1994) in his framework, is between *conclusive* (*sound*, *strong*) and *contingent* (*plausible*, *weak*) inference. *Conclusive inferences* assume the standard logical entailment used in Equation 2.1, while *contingent inferences* use a weaker form of entailment that states that the consequent is only a *plausible* or *probabilistic* consequence of the premises (i.e. $\Gamma \cup B \mid\approx C$). The work presented in this thesis can be seen as a form of *contingent inductive inference*.

Inductive inferences are used in several learning tasks. For instance, in *clustering*[1] one uses inductive reasoning to try to form groups of observations that are somehow interrelated. In *conceptual clustering* (Michalski & Stepp, 1983; Fisher,1987), the learner tries to discover concepts by grouping the observations in a "meaningful" way. Another learning task using inductive inference is *discovery*, where the goal is to obtain "numeric laws" that explain the given observations of some phenomenon (Lenat, 1970; Langley *et. al.* , 1986). As the learner is not given any guidance regarding which concepts or laws are valid in the domain, clustering is also known as *unsupervised learning*.

In *supervised learning* tasks the learner is given a set of *examples* (or *training cases*) of some *concept*. The learning task consists of using inductive inference to obtain a

[1] In statistics this methodology is sometimes referred as classification, which leads to some confusion because this word is used with another meaning within machine learning.

hypothesis (a model of the concept) that entails these observations. Usually people distinguish two different types of supervised learning tasks: classification and regression. In the case of *classification tasks*[2] the given examples have an attached label that indicates their concept membership. For instance, we may be interested in learning the concept of heart disease. A medical expert (the teacher) could provide our learning system with examples of patients with no heart problems as well as examples of heart diseases. In this example the learning goal is to obtain a model that is able to capture the main "features" of each type of heart disease. In *regression tasks* the "labels" attached to each example are numbers. The examples can be seen as instances of an unknown continuous function. The goal of the inductive system is to learn a general description (or model) of this function. For instance, in a banking institution there is usually a department responsible for quantifying the *risk* of the loan requested by a customer based on a series of indicators. The historical records of these experts' decisions can be used as a basis for inducing a general model of loan risk assessment. The task of inducing regression models based on examples is the main topic addressed in this thesis. In the remaining of this chapter we will briefly describe the general task of supervised learning and then concentrate on the particular case of regression.

2.2 Supervised Learning

In a supervised learning task the learner is given a set of observations (examples, training cases) of some phenomenon that are pre-labelled by a domain expert (the teacher). The learning goal consists of using these labelled observations together with some background knowledge, to obtain a model of this phenomenon. In the particular case of learning systems using inductive inference, this model can be seen as a generalisation of the observations.

---

[2] In statistics this task is usually known as discrimination.

As it was mentioned before the representation language used to describe the cases, the background knowledge and the obtained model is an important design decision, as it determines the knowledge space of the learning task. The choice of this language is another factor that further distinguishes the existing research in supervised learning. The standard approach uses a propositional language where each case is described by a fixed set of *attributes*[3]. An alternative is to use more powerful representation languages that encompass a larger set of learnable problems. Inductive logic programming (ILP) is a recent research field (Muggleton, 1992; Muggleton & De Raedt, 1994; Lavrac & Dzeroski,1994) that uses a subset of first-order logic as representation language[4].

The work presented in this thesis belongs to the area of propositional supervised learning. In propositional learning each training case is described by a set of *a* attributes, $X_1, X_2, \ldots, X_a$. Each attribute X_i has an associated *domain*, χ_i. *Boolean* or *binary* attributes have as domain the two logical constants (*e.g.* χ_{payed} = {TRUE, FALSE}). *Nominal* attributes have as domain a finite set of labels (*e.g.* χ_{color} = {red, white, blue}). *Numeric* attributes can take real or integer values. *Ordinal* attributes take values from a finite set of ordered labels (*e.g.* χ_{size} ={small, medium, large}). Some learning systems are also able to use *structured* attributes (*e.g.* Almuallim *et al.*,1995) were the domain is defined by a hierarchical taxonomy.

In supervised learning each case has an associated label $y \in \Upsilon$ that represents the concept membership of the case. We can see a *training case* as a pair $\langle \mathbf{x}, y \rangle \in \chi \times \Upsilon$, where $\chi = \chi_1 \times \ldots \times \chi_a$, and $\mathbf{x}$ is a vector of the values of the case related to the given attributes $X_1, \ldots, X_a$. It is the expert's responsibility to decide which attributes to include in the description of a case. We can define a *training set* as,

---

[3] Also known as *features* or *input variables.*

[4] Although it may be said that many of the ideas within ILP trace back to earlier work as mentioned in an historical account presented in Sammut (1993). Work on learning relational concept descriptions was investigated by Banerji (1964) and Winston (1970), and most of the theoretical aspects of ILP are based on the work of Plotkin (1970,1971).

$$D = \left\{ \left\langle \mathbf{x}_i , y_i \right\rangle \right\}_{i=1}^{n} \tag{2.2}$$

where,

n is the number of examples given to the system;
$\mathbf{x}_i$ is a vector containing the values of the case i, relative to the attributes X_1, X_2, …, X_a ;
and y_i is the label given by the expert to the case i.

As mentioned before there are two types of supervised learning tasks, classification and regression. The main difference of these two tasks is on the domain of the labels of the cases (*i.e.* γ). In classification γ consists of a finite set of labels (*i.e.* Y is a nominal variable), whereas in regression Y can take real or integer values (*i.e.* $\gamma = \mathbb{R}$ or $\gamma = \mathbb{N}$).

Learning systems using inductive inference proceed by performing a series of knowledge transmutations to the given training set, in order to obtain a generalisation of the cases (a model of the concept under study). Other supervised learners exist that either use other inference mechanisms or even limit themselves to storing the cases (*e.g.* instance-based learners; Aha *et al.*,1991). Depending on the type of knowledge transmutations carried out by the learner we can obtain different types of concept descriptions (or models). In the particular case of learners using inductive inference several alternatives exist within the literature. *Tree-based learners* (*e.g.* Breiman *et al.*,1984; Quinlan, 1986; Cestnik *et al.*, 1987) induce concept descriptions that have the form of a tree hierarchy where each inner node is a test on a attribute and the leaves the predictions of the model. These models are also known as decision (or classification) trees in the case of classification tasks, and regression trees for regression domains. *Rule-based learners* (*e.g.* Michalski, 1983; Michalski *et al.*,1986; Clark & Nibblet, 1989) obtain a set of *IF-THEN* rules, where the conditional is a conjunction of logical tests on the attributes and the conclusion is the predicted label. *Decision graphs* (Oliver, 1993; Kohavi, 1995; Oliveira & Sangiovanni-Vincentelli, 1996) were introduced as a kind of generalisation of decision trees to overcome some of their problems like replication and fragmentation (Kohavi, 1995). *Neural networks* (*e.g.* McClelland & Rumelhart, 1981; Rumelhart *et al.*,1986) obtain a series of weights for the connections in a network of computing units. *Statistical* *approaches* typically obtain models with a functional form, like linear discriminants (Fisher, 1936), generalised additive models (Hastie & Tibshirani, 1990), or regression splines (Friedman, 1991).

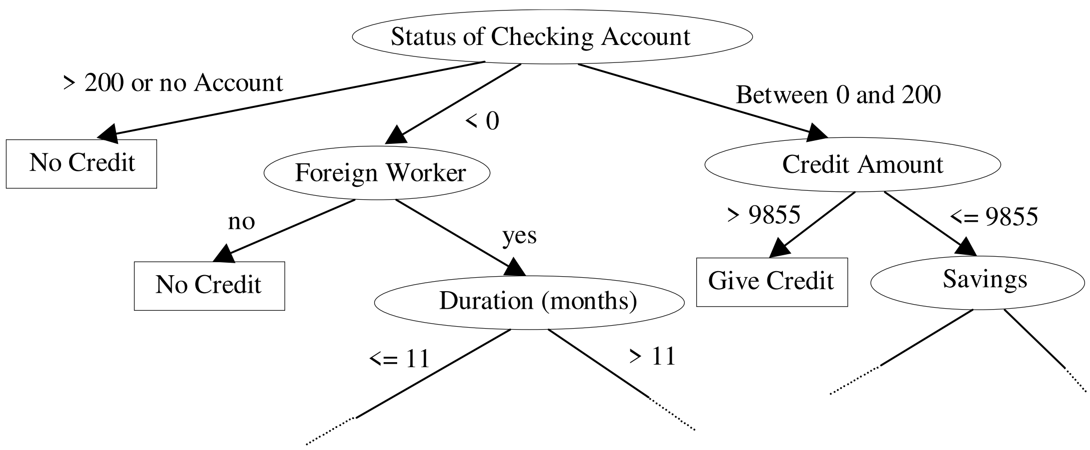
Example 2.1 illustrates a typical supervised learning task (in this case a classification task), and presents two examples of models obtained with different inductive learners using the same training data.

<u>EXAMPLE 2.1</u>

Suppose that a banking institution has 1000 records of past credit requests of its customers. For each request information was stored about 20 different customer "characteristics":

Status of existing account : < 0 ; 0 and 200; 200 and 500; etc.
Duration (in months)
Credit history : no credits; all credits paid back duly; etc.
Purpose : new car; used car; furniture; etc.
Present employment: employed ; unemployed; since less than 1 year; etc.
Personal status and sex : divorced male; married male; etc.
Guarantors : none; co-applicant; guarantor
Age (in years)
Etc.

The institution classified each of these past cases as successful or unsuccessful, depending on whether the client was able to attain his compromises with the banking institution. The goal of this institution is to obtain a model using these past observations to help them in the task of deciding whether or not to concede new credit requests. Giving this data set to a typical decision tree learner we could obtain the following model:



This tree-based model contains a series of conditions on certain input variables that lead to the conclusions. This model can be used to make predictions for a new case by answering the questions following the path from the root of the tree (the top node) to a leaf node (squared boxes). This leaf node contains the prediction of the tree-based model.

Similarly, we could give this data set to a typical rule learner and obtain a set of rules like for instance:

IF Status of existing Checking Account = no account
AND Employed since = between 4 and 7 years
AND Age > 22.5
AND Instalment plans = none
THEN No Credit

IF Duration > 8.5
AND Savings account >= 1000
AND Number of credits on this bank < 1.5
THEN No Credit

IF Credit Amount between 451 and 597
THEN No Credit

IF Credit Amount between 713 and 1118
AND Instalment plans = bank
THEN Give Credit

IF Status of existing Checking Account < 0
AND Duration > 11
AND Purpose = domestic appliances
THEN Give Credit

Etc.

♦

The type of target models is usually known as the hypothesis space bias within machine learning. The hypothesis space consists of all possible models within the selected model description language. Inducing a concept description based on the training data can be seen as a search problem within this hypothesis space (Mitchell, 1982). This search is guided by some *preference bias* that imposes an *ordering* among the hypothesis.

Underlying supervised learning is always the notion of *prediction*. The result of the learning process should allow making predictions for future observations of the same concept. This means that we can look at the output of a supervised learner as a function (or model) that is able to map an unlabelled case (or *testing case*) to a label, *i.e.*

$$f_D: \ \chi \rightarrow \gamma \tag{2.3}$$

Thus, it does not come as a surprise that most learning systems include *predictive performance* as part of their search bias. This means that the inductive knowledge transmutations are somehow biased towards models that ensure better predictive performance on future test cases. However, in many domains it may be advantageous to trade-off some predictive accuracy for simplicity (Bohanec & Bratko, 1994). Because of this, many learning systems also include simplicity as part of their search bias. This means that a key issue in any supervised learner is to select the search bias that better suits the learning goals. This search bias will allow the learner to choose between the possible models within the hypothesis space.

Finally, another important aspect of a supervised learner is the type of knowledge transmutation operators used to "move" through the space of hypothesis. In effect, within each type of inference strategy several alternative search operators exist (Michalski, 1994). Decision trees, for instance, perform a general to specific search[5], which is also the case of some rule learners like AQ11 (Michalski & Chilausky, 1980) and CN2 (Clark & Niblett, 1989). These systems start with very general concept descriptions and proceed by specialising them as a form of "motion" through the space of hypothesis. Other systems perform a specific to general search like for instance Golem (Muggleton & Feng, 1990). There are also cases of systems that use a bi-directional search procedure like RISE (Domingos, 1994; Domingos, 1996) or YAILS (Torgo, 1992, 1993a). These systems have generalisation and specialisation operators that allow them to move in both *directions* of the search space.

In summary we can say that there are three main issues characterising each supervised learning system:

---

[5] That is the reason why some authors use the name Top-Down Induction of Decision Trees (TDIDT).

- The knowledge representation language.
  - For the training cases, the background knowledge and the target model.
- The inference strategy of the learner.
  - The type(s) of strategy(ies), and the used search operators.
- The search bias used by the learner.
  - The function that evaluates the different knowledge states within the search for the target model.

2.3 Regression Problems

Multivariate regression analysis is an old problem that according to Draper and Smith (1980) traces back to the work of Sir Francis Galton (1822-1911), a British anthropologist and meteorologist. Regression analysis can be loosely defined as the application of methods that investigate the relationship between a dependent (or response) variable and a set of independent (or predictor) variables. This study is usually based on a sample of measurements made on a set of objects. This description has a perfect matching with the framework of supervised learning presented in Section 2.2.

In Example 2.2 we describe a simple regression task, together with a possible model obtained with a regression tree learner.

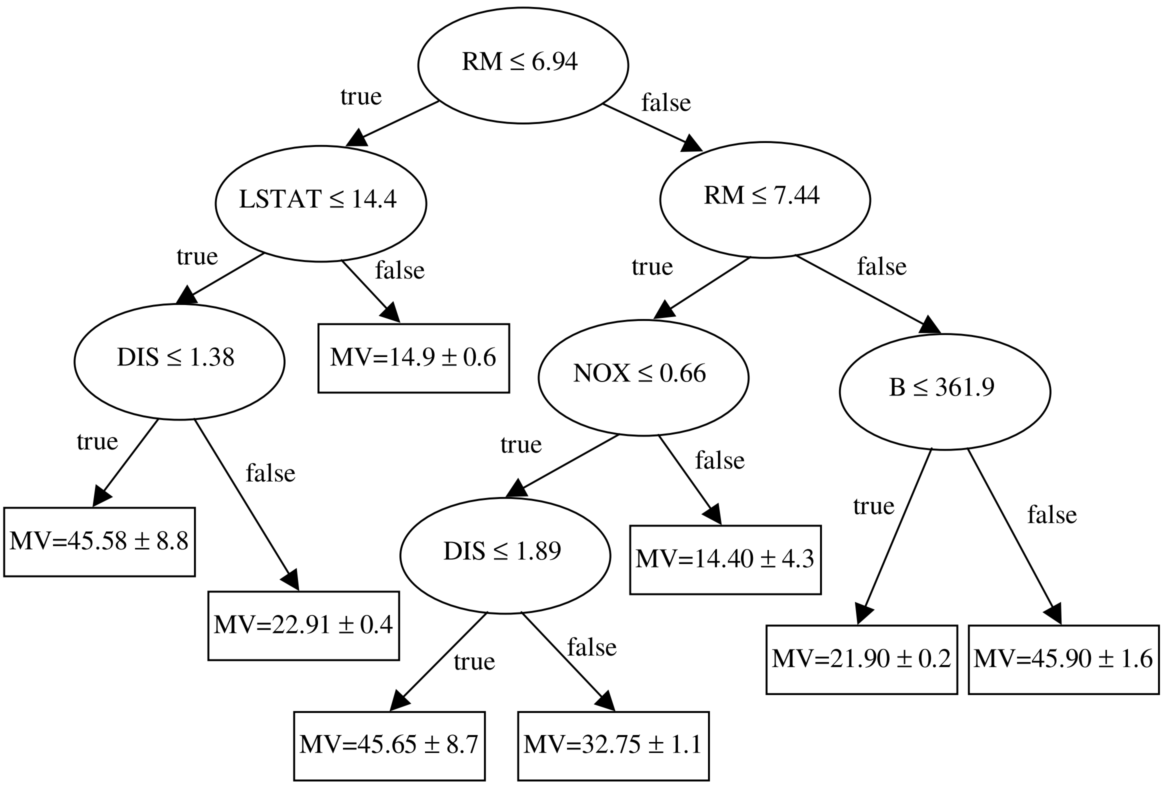
EXAMPLE 2.2

Harrison and Rubinfeld (1978) described an interesting regression application where they tried to predict the housing values in areas of Boston using other variables. Their goal was to check if there was any effect of air pollution concentration (NOX) on these values. The data they have collected consisted of 506 observations each described by the following variables:

MV (target variable) : median value of homes in thousands of dollars.
CRIM : crime rate
ZN : percent of land zoned for lots
INDUS : percent of non-retail business
CHAS : 1 if on Charles River, 0 otherwise
NOX : nitrogen oxide concentration in pphm

RM : average number of rooms
AGE : percent built before 1940
DIS : weighed distance to employment centers
RAD : accessibility to radial highways
TAX : tax rate
P/T : pupil/teacher ratio
B : percent black
LSTAT : percent low-status population

Using the 506 cases we can obtain a regression model that tries to capture the dependence of the housing values on the other variables. This model will not only serve as a prediction tool for future cases, but will also enrich our knowledge about the relations between the variables. If we decided to use a regression tree learner we could obtain the following tree-based model:



This graphical representation of a regression tree has two types of nodes. Rounded nodes represent tests on input variables, while the squared boxes are the leaves of the tree and represent predictions of the target variable. A prediction using this model can be obtained by "dropping" the test case down the tree, following the correct branches according to the outcome of the tests on the input variables. The leaf that is reached has the corresponding prediction.

♦

This example illustrates a typical regression task. However, there are problems where there is more than one target variable. For instance, a recent international competition[6] proposed a regression task where the competitors should predict for each test case the value of seven different target variables. Some learning systems are also able to deal with these learning tasks. Among these we can mention artificial neural networks that can easily handle problems with more than one output variable. Within the ILP community there has also been some attention to the issue of multiple concept learning (De Raedt *et al.*,1993). Another interesting example of dealing with regression problems with multiple response variables can also be found in Breiman and Friedman (1995). In this thesis we only focus on problems with one response variable, which is often encountered in practice. Still, when facing such type of problems one can always follow the strategy of taking them as several different regression tasks. In effect, we have proceeded that way in the above-mentioned competition, and the solution obtained with the tool implementing the ideas described in this thesis (RT) was announced as one of the runner-up winners of the competition.

2.3.1 A Formalisation of Regression Problems

The task of a regression method is to obtain a model based on a sample of objects belonging to an unknown regression function. This sample (the training set) consists of pairs of the form $\langle \mathbf{x}_i, y_i \rangle$ where $\mathbf{x}_i$ is a vector of the values of the attributes (predictor variables) and y_i is the respective value of the response (output).

In the context of regression analysis, matrix notation is often used as it simplifies some formulations. We will adopt this notation when presenting existing regression methods. Let

[6] The 3rd International ERUDIT competition (http://www.erudit.de/erudit/activities/ic-99/).

$\mathbf{X}$ be the input matrix whose i-th row is the input vector $\mathbf{x}_i$. If there are n vectors, $\mathbf{X}$ is a matrix with dimension $n \times a$, where a is the number of attributes. We will collect the target values of the input vectors in a $n \times 1$ matrix, $\mathbf{Y}$. We can also represent a data set D as a $n \times (a+1)$ matrix $\mathbf{D}$.

We can look at a *regression learning system* as a function that maps a data set D into a regression model that we will denote as $r_D(.)$. A *regression model* is also a function that maps an input vector $\mathbf{x}_i \in \chi$ into a real number $y \in \Upsilon$. Regression analysis is mainly concerned with estimating or predicting the mean value of the dependent variable Y based on the values of the independent variable(s) X_i ,

$$E\left(Y \middle| X_1, \ldots, X_a\right) \tag{2.4}$$

where

$E(.)$ denotes statistical expectation.

As the true underlying regression function is usually unknown the estimates are based on a sample of this function (the training set).

The regression relationship between the attributes and the target variable that is usually assumed is described by,

$$y_i = r(\beta, \mathbf{x}_i) + \varepsilon_i \tag{2.5}$$

where,

$r(\beta, \mathbf{x})$ is a regression model on the input variables $\{X_i\}_{i=1}^{a}$ with parameters β ;

and ε_i are observation errors.

The main goal of a regression method is to search for the "best" model parameters β according to a selected preference criterion. As it was mentioned in section 2.2, this search bias can include both the estimated prediction error (*i.e.* performance on future test cases), and the simplicity of the model. However, it should be mentioned that most existing regression tools only use prediction error to guide the search for the best model parameters. The following section presents several measures of the error of regression models that can be used to estimate their predictive power.

2.3.2 Measuring the Accuracy of Regression Models

In order to obtain an estimate of the predictive performance of a model we can use it to predict the label of a set of cases. In the case of regression models we can use them to obtain numeric predictions of the target variable. This evaluation is possible if we know the true value of the label of these cases. Using this true value we can compare it to the model prediction and thus quantify its performance. As the target variable of regression problems is numeric, the existing error measures revolve around the difference between the predicted and true values of this variable.

Mean Absolute Deviation (MAD) is an error measure that quantifies the error of a model by averaging the absolute deviations of its predictions, *i.e.*

$$MAD(r) = \frac{1}{n}\sum_{i=1}^{n}\left| y_i - r(\beta, \mathbf{x}_i) \right| \tag{2.6}$$

where,

$\{\langle \mathbf{x}_i, y_i \rangle\}_{i=1}^{n}$ is a data set ;
and $r(\beta, \mathbf{x}_i)$ is the prediction of the regression model we want to evaluate for the case $\langle \mathbf{x}_i, y_i \rangle$.

For reasons having to do with ease of computations when trying to find a model with minimum error, the measure that is classically used in regression analysis is the Mean Squared Error (MSE), given by

$$MSE(r) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - r(\beta, \mathbf{x}_i) \right)^2 \tag{2.7}$$

Using this error measure as the minimisation criterion for obtaining the parameters of our model we have what traditionally is called *Least Squares Regression* methods.

Another common error measure is the Relative Mean Squared Error (*RMSE*) that is given by,

$$RMSE(r) = \left(\frac{1}{n} \sum_{i=1}^{n} \left(y_i - r(\beta, \mathbf{x}_i) \right)^2 \right) \Bigg/ \left(\frac{1}{n} \sum_{i=1}^{n} \left(y_i - \bar{y} \right)^2 \right) = \frac{MSE(r)}{MSE(\bar{y})} \tag{2.8}$$

where,

$\bar{y}$ is the average Y value of the sample.

This measure gives a relative value of the fitting error. A value between zero and one indicates that r is doing better than just predicting the average Y value.

2.3.2.1 Biased versus Unbiased Error Estimates

Most of existing regression methods assume some form of regression model and search for the model parameters that minimise the chosen error criterion. The equations presented in the previous section must be seen as statistical estimators of the true error of our regression model. As statistics they are based on samples of a population. We would like to have estimates that provide a precise value of the predictive performance of our model in future samples of the same population. Estimates obtained with the same data used to induce the model parameters are known to be unreliable (or biased). As an extreme example consider the 1-nearest neighbour algorithm (Cover & Hart, 1967). This classifier predicts the class label of a new case with the label of the most similar training case. If we use the same training data to evaluate this classifier this leads to an error rate of zero, which most surely is an over-optimistic measure of the true error of this classifier on unseen cases.

Obtaining reliable estimates of the true error of a model is important in many respects. First of all, it enables to estimate its future accuracy on new data. Reliable estimates are also important for model selection. Many learning systems try to generate several alternative models and choose what appears to be the best among them. Decision trees, for instance, may generate a set of pruned trees and use reliable error estimates to choose the right size one (Breiman *et al.*,1984). Reliable error estimates are also important when combining models (Brazdil & Torgo, 1990) or their predictions (*e.g.* Wolpert,1992; Breiman,1996).

The prediction error is most of the times the "driving force" behind the construction of a regression model. Estimating the prediction error of a regression model using the training sample leads to what is usually named a *resubstitution estimate* of the error. Using this type of estimates with a learning method with a reasonably rich hypothesis space will most surely lead to a model capturing statistically irrelevant features of the data (*i.e.* a model that overfits the training data). A model that overfits the training data will hardly generalise over unseen data. These models although providing an almost perfect fit of the training data will perform poorly when faced with new data.

There are several approaches to the problem of obtaining reliable error estimates of models. *Resampling methods* proceed by obtaining the estimates with data not used for inducing the models. Several techniques exist to achieve this, like the *Holdout* (Highleyman, 1962[7]), *Cross Validation* (Mosteller & Wallace, 1963; Stone, 1974), *Bootstrap* (Efron,1979; Efron & Tibshirani, 1993), *etc.* These methods proceed by obtaining multiple samples from the given training sample.

Bayesian estimation methods (Good, 1965) usually obtain the estimates using some form of combination of the prior expectation of the parameter being estimated and the observed value. An example of these techniques are the *m*-probability estimates (Cestnik, 1990). These estimates can be seen as a generalisation of the Laplace's law of succession (Good, 1965).

Finally, a different approach to the problem of estimating a population parameter consists of studying the *sampling distribution properties* of our estimates. Suppose that we are interested in obtaining estimates of the mean value of a variable. If we obtain several samples of size *n* we can calculate for each sample the average value of the variable. Each of these averages is an estimate of the true population mean value. We can look at these

---

[7] According to Ripley (1996) one of the first references on this method.

averages as values of another variable and we can study the properties of the distribution of this new variable (which is called the sampling distribution). In this particular case, it can be proven that the sampling distribution of the average is normal in the limit (known as the *central limit theorem*). Knowing the sampling distribution of our target parameter allows us to draw important conclusions regarding the confidence on any particular estimate.

In Chapter 4 we address in detail the issue of estimating the true error of regression trees as a means to avoid overfitting of tree-based models.

2.4 Existing Regression Methods

This section presents a brief overview of existing work on regression. We describe several different approaches from different research fields, all concerned with the problem of regression.

2.4.1 Statistical Methods

Regression is one of the major topics studied in statistical data analysis. There is an enormous amount of work on several approaches to this problem. We will present a brief overview of the major paradigms without describing all the existing variants.

2.4.1.1 Global parametric approaches

Global parametric methods try to fit a single functional model to all the given training data. This imposes a strong assumption on the form of the unknown regression function, which may lead to lower accuracy if that is not the case. However, these approaches usually fit simple functional models that are easily interpreted and have fast computational solutions.

A classical example of a global approach is the widely used *linear regression model* that is usually obtained using a *Least Squares Error Criterion* that tries to find the vector of parameters $\boldsymbol{\beta}$ that minimises the sum of squared errors,

$$\sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 X_1 + \ldots + \beta_a X_a))^2 \tag{2.9}$$

The minimisation of this equation has an elegant and efficient solution given by,

$$\boldsymbol{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y} \tag{2.10}$$

where,

$\mathbf{X}'$ denotes the transpose of **X**;
and $\mathbf{X}^{-1}$ the inverse matrix of **X**.

There are many variants of this general set-up that differ in the way they develop/tune these models to the data (*e.g.* Draper & Smith, 1981).

2.4.1.2 Non-parametric approaches (Local Modelling)

Non-parametric regression[8] belongs to a data analytic methodology usually known as *local modelling* (Fan, 1995). The basic idea behind local regression consists of obtaining the prediction for a data point **x** by fitting a parametric function in the *neighbourhood* of **x**. This means that these methods are "locally parametric" as opposed to the methods described in the previous section.

According to Cleveland and Loader (1995) local regression traces back to the 19th century. These authors provide a historical survey of the work done since then. The modern work on local modelling starts in the 1950's with the *kernel methods* introduced within the probability density estimation setting (Rosenblatt,1956; Parzen,1962) and within the regression setting (Nadaraya,1964; Watson,1964). *Local polynomial regression* is a generalisation of this early work on kernel regression. In effect, kernel regression amounts to fitting a polynomial of degree zero (a constant) in a neighbourhood. Summarising, we can state the general goal of local regression as trying to fit a polynomial of degree *p* around a query point (or test case) $\mathbf{x}_q$ using the training data in its neighbourhood. This

---

[8] Also known as non-parametric smoothing and local regression.

includes the various available settings like kernel regression (p=0), local linear regression (p=1), *etc*[9].

Local regression is strongly related to the work on instance-based learning (*e.g.* Aha *et al.*,1991), within the machine learning community. Given a case **x** for which we want to obtain a prediction, these methods use the training samples that are "most similar" to **x** to obtain a local model that is used to obtain the prediction. This type of inductive methodologies do not perform any kind of generalisation of the given data[10] and "delay learning" till prediction time[11].

Most studies on local modelling are carried out for the case of one unique input attribute. Still, the framework is applicable to the multivariate case and has been used with success in some domains (Atkeson *et al.*,1997; Moore *et al.*,1997). However, several authors alert for the "danger" of applying these methods in higher input space dimensions[12] (Hardle, 1990; Hastie & Tibshirani,1990). The problem is that with high number of attributes the training cases are so sparse that the notion of local neighbourhood can hardly be seen as local. Another drawback of local modelling is the complete lack of interpretability of the models. No comprehensible model of the training data is obtained. With some simulation work one may obtain a graphical picture of the approximation provided by local regression models, but this is only possible with low number of attributes.

---

[9] A further generalisation of this set-up consists of using polynomial mixing (Cleveland & Loader,1995), where p can take non-integer values.

[10] This is not completely true for some types of instance-based learners as we will see later. Nevertheless, it is true in the case of local modelling.

[11] That is the reason for also being known as lazy learners (Aha, 1997).

[12] The so called "curse of dimensionality" (Bellman, 1961).

2.4.1.3 Additive models

The basic idea of additive models is to take advantage of the fact that a complex regression function may be decomposable in an additive form, where each constituent represents a simple function. Additive models consist of sums of other lower dimensional functions. These models were developed as an answer to the difficulties of local models with high dimensions. Comprehensibility is another goal of these methods. They share the interpretability of global linear regression. As each attribute has an additive effect on the model prediction of the target variable *Y*, they are said to be additive in the attribute effects (Hastie & Tibshirani,1990). This property is intuitively appealing from the user perspective. An *additive model* can be defined by,

$$r(\mathbf{x}) = \alpha + \sum_{j=1}^{a} f_j\left(X_j\right) \tag{2.11}$$

where,

the f_j's are univariate basis functions, one for each attribute.

These models may also be generalised to functions on more than one attribute as well as to nominal attributes. Each basis function f_j can have any arbitrary functional form, like for instance the kernel models that were mentioned in the previous section. A common set-up consists of fitting a one-dimensional kernel to each attribute.

To obtain an additive model we may use an iterative fitting procedure like the backfitting algorithm (Friedman & Stuetzle, 1981).

The main drawback of this methodology is its computational complexity. One has a potentially large set of candidate basis functions to choose from, and afterwards one has to tune the parameters of each of these basis functions (Friedman, 1991). Still, there are some simplifications that allow fast algorithms to be used. An example of an additive model is a *regression tree* (Morgan & Sonquist,1963; Breiman *et al.*,1984), which can be efficiently obtained with a recursive partitioning algorithm. Work on tree-based models (either for classification or regression) also exists in other research fields like pattern recognition

(Swain & Hauska,1977; Dattatreya & Kanal,1985) or machine learning[13] (Hunt *et al.*,1966; Quinlan, 1979; Kononenko *et al.*,1984). In Section 2.4.3.1 we briefly describe the existing work in machine learning.

Another example of an additive model is *projection pursuit regression* (Friedman & Stuetzle, 1981). Projection pursuit provides a model of the form,

$$r(\mathbf{x}) = \sum_{i=1}^{F} f_i\left(\sum_{j=1}^{a} \alpha_{j,i} X_i\right) \tag{2.12}$$

These models can be simply described as additive functions of linear combinations of the attributes. In a way these models are related to the work on regression trees with linear models in the leaves (Breiman & Meisel, 1976; Friedman, 1979; Karalic, 1992; Quinlan, 1992).

Adaptive regression splines (Friedman,1991) are other additive models that can be seen as a generalisation of regression trees introduced to overcome some of their limitations. Friedman (1991) describes these limitations in detail and shows how adaptive regression splines can overcome them. The resulting model is implemented in system MARS which provides a regression model of the form,

$$r(\mathbf{x}) = c_0 + \sum_{i=1}^{p} c_i \prod_{k=1}^{K_i} \left[s_{k,i}\left(X_{v(k,i)} - t_{k,i}\right)\right]_+ \tag{2.13}$$

where,

the $\left[s_{k,i}\left(X_{v(k,i)} - t_{k,i}\right)\right]_+$ are two-sided truncated power basis functions.

This model can be recast into the more intuitive form given by,

$$r(\mathbf{x}) = c_0 + \sum f_m(X_m) + \sum f_{m,n}(X_m, X_n) + \sum f_{m,n,o}(X_m, X_n, X_o) + \ldots \tag{2.14}$$

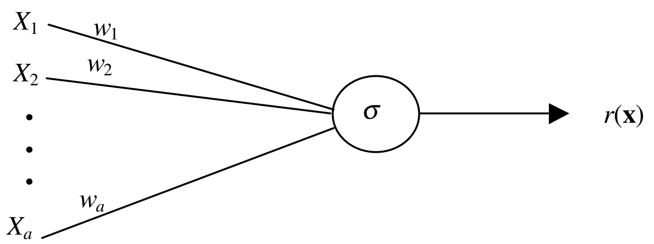
In this equation the first sum is over all basis functions that involve only a single attribute. The second sum uses the basis functions involving two variables, and so on.

---

[13] An extensive survey on tree-based models in the context of classification can be found in Murthy (1996).

2.4.2 Artificial Neural Networks

The work on Artificial Neural Networks (ANN's) has a strong biological motivation. McCulloch and Pitts (1943) proposed the first model of an artificial neuron. Since then many new more complex models have been proposed. The McCulloch-Pitts neural networks were generalised by Ronsenblatt (1958) that introduced the *perceptron networks*. This work was then extended by Minsky and Papert (1969). A perceptron is a computing unit with a threshold (or bias) θ that takes a vector of a real-valued inputs with associated weights, outputting 1 if $\sum_{i=1}^{a} w_i X_i \geq \theta$ and 0 otherwise. This can be represented in a graphical way by,



$r(\mathbf{x}) = \sigma(\theta + \mathbf{w}\ \mathbf{x})$
where,
$$\sigma(y) = \begin{cases} 1 & \text{if } y > 0 \\ 0 & \text{otherwise} \end{cases}$$

Figure 2.1 - A perceptron unit.

Learning with these units consists of finding the weights associated with each input attribute. We start with a random weight vector, and then apply each training case to the perceptron modifying the weights when the predictions are wrong. Weights are modified using the following update rule,

$$w_i = w_i + \Delta w_i \tag{2.15}$$

where,

$\Delta w_i = \eta\left(y_i - r(\mathbf{x})\right)x_i$;
η is called the learning rate;
and $r(\mathbf{x})$ is the prediction of the network for the training case $\mathbf{x}$.

If the training cases are linearly separable and η is sufficiently small this training algorithm is proved to converge (Minsky & Papert,1969). Other learning algorithms exist, like for instance linear programming techniques (Mansfield,1991) or the Karmarkar's algorithm (Karmakar,1984).

In order to relax the requirement of linear separability we can use other training rule. The *delta rule* (or gradient descendent) can be described by the following,

$$w_i = w_i + \Delta w_i \tag{2.16}$$

where,

$$\Delta w_i = \eta \sum_{i=1}^{n} \left(y_i - r(\mathbf{x_i})\right) x_i \ .$$

The delta rule with the gradient descent method for searching the weights that minimise the error of a neuron, are usually applied to unthresholded units. These units do not output a 0 or 1 like the perceptron. Instead their result is given by the linear product of the weights times the attribute values. This means that while for the perceptron $r(\mathbf{x})$ is given by the formula presented in Figure 2.1, for the unthresholded units $r(\mathbf{x})$ is equal to the dot product **w.x**.

Generally, a single computing unit strongly limits the kind of functions we can approximate. *Multilayer networks* are much more powerful representations. In general these consist of an input layer related to the input attributes, one (or more) hidden layers, and an output layer. Notice that ANN's are not restricted to a unique output variable, which is one of their advantages. The *sigmoid unit* is the most common choice as basic unit of multilayer networks. Its behaviour is described by Figure 2.2:

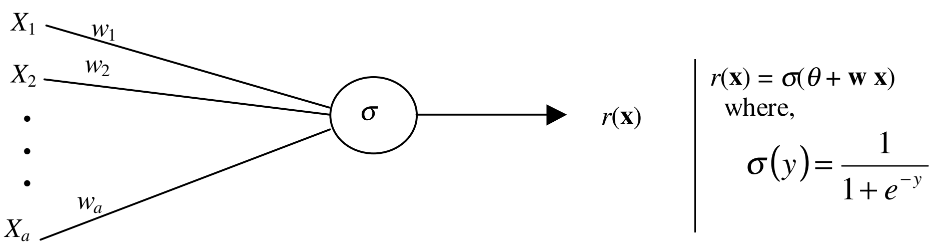


Figure 2.2 - The sigmoid computing unit.

The algorithm generally used to learn multilayer networks of sigmoid units is *backpropagation*. This is probably the most well known neural network learning algorithm. Although usually attributed to Rumelhart and colleagues (1986), this algorithm was, according to Rojas (1996), invented by Werbos (1974, 1994).

A common set-up for applying ANN's in regression consists of using a 3-layered network with one hidden layer (Figure 2.3). Both the input units and the hidden layer units are sigmoids while the output unit is an untresholded unit.

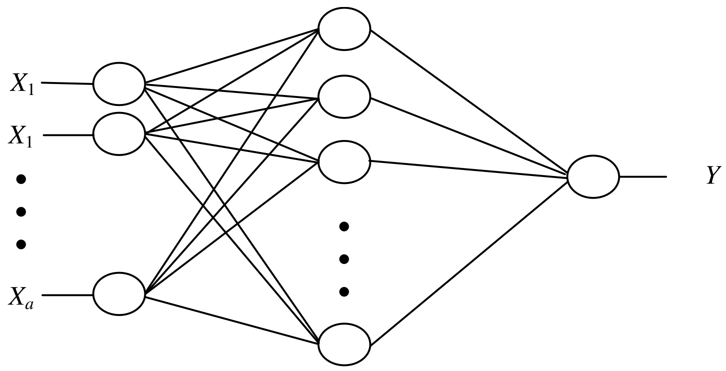


Figure 2.3 - A multilayer architecture for regression.

Usually there are two main criticisms made to artificial neural networks. The first has to do with the slow convergence to a solution. There are many "tricks" that have been proposed to overcome this problem, but still it is a slow method. This is sometimes greatly improved with parallel computer architectures (or even specialised architectures). The other frequent criticism has to do with the fact that the solutions found by the methods are very difficult to interpret by the users. In problems were interpretability is a key factor ANN's are clearly inadequate. However, there are many problems were that is not the case, and successful applications of ANN's abound (*e.g.* Rumelhart *et al.*,1994).

2.4.3 Machine Learning Methods

Machine learning research on supervised induction has traditionally concentrated efforts on classification problems. Work on regression does not abound. In this section we provide a brief overview of some exceptions.

2.4.3.1 Propositional Learning Methods

Regression Trees

Tree-based regression models were also studied within the machine learning community. One of the contributions of the work carried out within this community is the possibility of using different models in the leaves of the trees (*e.g.* Karalic,1992; Quinlan 1992; Torgo, 1997). Other contributions include different pruning methods like the work conducted in systems RETIS (Karalic & Cestnik,1991) and CORE (Robnik-Sikonja & Kononenko, 1998). M5 (Quinlan, 1992, 1993) explores the possibility of combining the predictions of a regression tree with a *k*-nearest neighbour model. Finally, Robnik-Sikonja and Kononenko (1997) describe a new method for guiding the selection of the best attribute during the growth of regression trees.

Regression Rules

Weiss and Indurkhya (1993) have developed a system (SWAP1R) that learns regression rules in a propositional language. The conditional part of the rules consists of a conjunction of tests on the input attributes while the conclusion part contains the associated prediction of the target variable value. Originally these predictions consisted of the average *Y* value of the cases satisfying the conditional part, but later the possibility of using *k*-nearest neighbours was added (Weiss & Indurkhya, 1995). This system has the particularity of dealing with regression by transforming it into a classification problem. In effect, the target values of the training cases are pre-processed by grouping them into a set of user-defined bins. These bins act as class labels in the subsequent learning phase. This means that the learning algorithm induces a classification model of the data, where the classes are numbers representing the obtained bins[14]. These numbers can then be used to make numeric predictions for unseen cases. This idea of transforming a regression problem into a

[14] The authors use the median as the representative of each bin to avoid outliers effects.

classification task was taken further in the work of Torgo and Gama (1997), where a system called RECLA acts as a generic pre-processing tool that allows virtually any classification system to be used in a regression task.

Torgo (1995) has developed a propositional regression rule learner (R^2) that also uses an IF-THEN rule format. This algorithm can use several different functional models in the conclusion of the rules[15]. R^2 uses a covering algorithm that builds new models while there are uncovered cases to fit. The model is chosen from the model lattice that includes all possible regression models for the conclusion part of the rules. The result of this first step is a rule with empty conditional part and the built model as conclusion. This rule is then specialised by adding conditions to restrict the model domain of applicability with the goal of improving its fit. This restriction is guided by an evaluation function that weighs both the fitting as well as the degree of coverage of the rule. This means that the system is looking for rules with good fit but covering as many cases as possible.

CUBIST[16] is a recent commercial system developed by Ross Quinlan. It learns a set of unordered IF-THEN rules with linear models in the conclusion part. Up to now there is no published work on CUBIST but from our experience with the system it looks like it is a rule-based version of M5 (Quinlan, 1992,1993)[17]. It seems to be a kind of C4.5rules (Quinlan, 1993) for M5. The system is able to deal with both continuous and nominal variables, and obtains a piecewise linear model of the data. CUBIST can also combine the predictions of this model with *k*-nearest neighbour predictions (as M5 does).

---

[15] The actual implementation only used averages and linear models, but the system is easily extendible to handle other models due to its algorithm.

[16] More information on this system can be found at the URL, http://www.rulequest.com.

[17] Prof. Ross Quinlan has recently personally confirmed that although he also mentioned that there was more of it behind CUBIST than just a simple transformation form trees to rules.

Instance-based Learners

Instance-based learners are similar to the local modelling approaches described earlier. In effect, the basic philosophy behind these systems is also that of making predictions based on the most similar training instances. Most of the existing work simply stores all training instances without performing any kind of generalisation and then uses a distance metric to find the neighbours of each test case. These systems are in effect variants of the k-nearest neighbour method (Fix & Hodges,1951; Cover & Hart,1967). However, other authors have described systems that perform some generalisations of the cases, like for instance NGE (Salzberg, 1988,1991), or RISE (Domingos, 1994, 1996). Still, all these works deal with classification problems. Among the few exceptions is the work of Kibler *et al.* (1989), and of Connel and Utgoff (1987), which deal with regression. The former uses basically a kernel model with the neighbourhood defined by the distance to the k^{th} nearest neighbour. This means that the k most similar instances are used to obtain the predictions of a new case. Each of these neighbours enters in the prediction calculation with a weight proportional to its distance to the case in question. The work of Connel and Utgoff uses a similar strategy with the difference that all training instances contribute to the prediction.

2.4.3.2 First-order Logic Approaches

Within the field of Inductive Logic Programming (ILP) some recent work has also focussed on the problem of regression. FORS (Karalic, 1995, Karalic & Bratko, 1997) induces a model that has the form of a Prolog program. This program consists of clauses of the form $f(Y,X_1,\ldots,X_a)$ that represent the obtained regression model. The system is able to receive, as input, samples of the unknown regression function and background knowledge, which constitutes a major advantage. FORS uses a covering algorithm at the top level that keeps generating new clauses followed by the removal of the cases covered until a termination criterion is met. The clause generation step follows a general to specific search that keeps adding new literals to the clause. FORS uses several pre-pruning mechanisms for controlling clause generation. Among them is the use of Minimum Description Length (MDL) (Rissanen,1982).

FFOIL (Quinlan, 1996) is another example of an ILP system able to deal with regression. FFOIL is a derivation of the FOIL system (Quinlan, 1990). FFOIL follows a similar covering algorithm as FOIL's. It starts with an empty program and keeps adding clauses until all cases are covered. Each added clause starts with an empty body and literals are appended as a form of specialising the clause. A function with *k* arguments (the input variables) is represented by a *k*+1-ary relation where one argument holds the function outcome. The result obtained by FFOIL consists of a Prolog program with clauses of this relation.

First order approaches to regression have a much larger search space than their propositional counterparts. This fact has two contradictory effects. While being able to find solutions not available to propositional systems, this increased expressiveness has a strong impact in the computational complexity of these systems. This makes them hardly applicable to extremely large domains that are found in some applications (like in a typical Data Mining situation). However, computation power grows at a very fast rate and ILP may well become the major trend in Machine Learning approaches to regression[18].

[18] Although a sceptical could say that data size is increasing even faster.

Chapter 3

Tree-based Regression

This chapter describes two different approaches to induce regression trees. We first present the standard methodology based on the minimisation of the squared error. Least squares (LS) regression trees had already been described in detail in the book by Breiman *et. al.* (1984). Compared to this work we present some simplifications of the splitting criterion that lead to gains in computational efficiency. We then address the alternative method of using a least absolute deviation (LAD) error criterion to obtain regression trees. Although mentioned in the book of Breiman and colleagues (1984), this methodology was never described in sufficient detail. In this chapter we present such a description. The LAD criterion is known to be more robust to skewed distributions and outliers than the LS criterion used in standard regression trees. However, the use of the LAD criterion brings additional computational difficulties to the task of growing a tree. In this chapter we present algorithms based on a theoretical study of the LAD criterion that overcome these difficulties for numeric variables. With respect to nominal variables we show that the theorem proved by Breiman *et. al.* (1984) for subset splits in LS trees does not hold for the LAD error criterion. Still, we have experimentally observed that the use of the results of this theorem as a heuristic method of obtaining the best split does not degrade predictive accuracy. Moreover, using this heuristic brings significant gains in computation efficiency.

3.1 Tree-based Models

Work on tree-based regression models traces back to Morgan and Sonquist (1963) and their AID program. However, the major reference on this research line still continuous to be the seminal book on classification and regression trees by Breiman and his colleagues (1984). These authors provide a thorough description of both classification and regression tree-based models. Within Machine Learning, most research efforts concentrate on classification (or decision) trees (Hunt *et al.*,1966; Quinlan, 1979; Kononenko *et al.*,1984). Work on regression trees started with RETIS (Karalic & Cestnik, 1991) and M5 (Quinlan, 1992). Compared to CART (Breiman *et. al.*,1984), RETIS uses a different pruning methodology based on the Niblet and Bratko (1986) algorithm and *m*-estimates (Cestnik, 1990). With respect to M5 (Quinlan, 1992), its novelty results from the use of linear regression models in the tree leaves[19]. A further extension of M5 was described in Quinlan (1993). This extension consisted in combining the predictions of the trees with *k* nearest neighbour models.

Tree-based regression models are known for their simplicity and efficiency when dealing with domains with large number of variables and cases. Regression trees are obtained using a fast divide and conquer greedy algorithm that recursively partitions the given training data into smaller subsets. The use of this algorithm is the cause of the efficiency of these methods. However, it can also lead to poor decisions in lower levels of the tree due to the unreliability of estimates based on small samples of cases. Methods to deal with this problem turn out to be nearly as important as growing the initial tree. Chapter 4 addresses this issue in detail.

In spite of their advantages regression trees are also known for their instability (Breiman, 1996). A small change in the training set may lead to a different choice when building a node, which in turn may represent a dramatic change in the tree, particularly if the change occurs in top level nodes. Moreover, the function approximation provided by

---

[19] Which was also done in a subsequent version of RETIS (Karalic, 1992).

standard regression trees is highly non-smooth leading to very marked function discontinuities. Although there are applications where this may be advantageous, most of the times the unknown regression function is supposed to have a certain degree of smoothness that is hardly captured by standard regression trees. In Chapter 5 we describe hybrid tree models that improve the smoothness of tree-based approximations. In spite of this drawback, regression trees do not assume any particular form for the function being approximated thus being a very flexible regression method. Moreover, the obtained models are usually considered easily comprehensible.

In Section 3.2 of this chapter we explore methods of inducing regression trees using the least squares (LS) error criterion. The use of this criterion leads to several improvements in terms of computational efficiency resulting from the mathematical base behind it. Namely, thanks to the theorem presented by Breiman *et al.* (1984), we can devise an efficient method for dealing with nominal attributes. Moreover, we present a fast incremental updating method to evaluate all possible splits of continuous attributes with significant computational gains. These splits are known to be the major bottleneck in terms of computational efficiency of tree learning algorithms (Cattlet, 1991).

In the subsequent section we present a method of inducing regression trees using the least absolute deviation (LAD) criterion. The main difference to LS trees lies in the use of medians instead of averages in the leaves and the use of the mean absolute deviation as error criterion. The main advantage of using this methodology is the robustness of the obtained models. In effect, medians and absolute deviations are known to be more robust with respect to the presence of outliers and skewed distributions. However, we will see that this methodology poses several computational difficulties. We will present a theoretical analysis of the LAD criterion, and as a result of this analysis we describe a series of fast updating algorithms that improve the computational efficiency of LAD regression trees.

Another criterion that can be used when growing regression trees is *RRelief* (Robnik-Sikonja & Kononenko, 1997). This criterion is particularly suitable for domains where the input variables (or attributes) are known to be dependent. Still, this criterion entails much larger computational complexity than the LAD or LS criteria due to the necessity of calculating distances between training cases.

A regression tree can be seen as a kind of additive model (Hastie & Tibshirani, 1990) of the form,

$$m(\mathbf{x}) = \sum_{i=1}^{l} k_i \times I(\mathbf{x} \in D_i) \tag{3.1}$$

where,

k_i are constants;

$I(.)$ is an indicator function returning 1 if its argument is true and 0 otherwise;

and D_i are disjoint partitions of the training data D such that $\bigcup_{i=1}^{l} D_i = D$ and $\bigcap_{i=1}^{l} D_i = \phi$.

Models of this type are sometimes called *piecewise constant regression models* as they partition the predictor space χ in a set of regions and fit a constant value within each region. An important aspect of tree-based regression models is that they provide a propositional logic representation of these regions in the form of a tree. Each path from the root of the tree to a leaf corresponds to a region. Each inner node[20] of the tree is a logical test on a predictor variable[21]. In the particular case of binary trees there are two possible outcomes of the test, true or false. This means that associated to each partition D_i we have a path P_i consisting of a conjunction of logical tests on the predictor variables. This symbolic representation of the regression function is an important issue when one wants to have a better understanding of the regression surface.

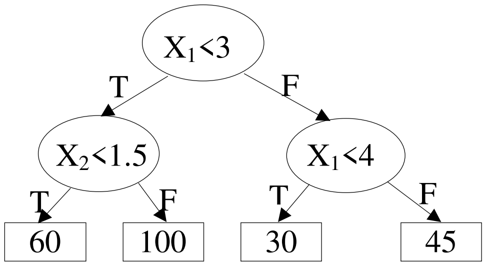
Example 3.1 provides a better illustration of this type of models through a small example of a regression tree:

---

[20] All nodes except the leaves.

[21] Although work exists on multivariate tests (*e.g.* Breiman *et. al.* 1984; Murthy *et. al.*, 1994; Broadley & Utgoff, 1995; Gama, 1997).

<u>EXAMPLE 3.1</u>



$P_1 \equiv X_1 < 3 \wedge X_2 < 1.5$, with $k_1 = 60$

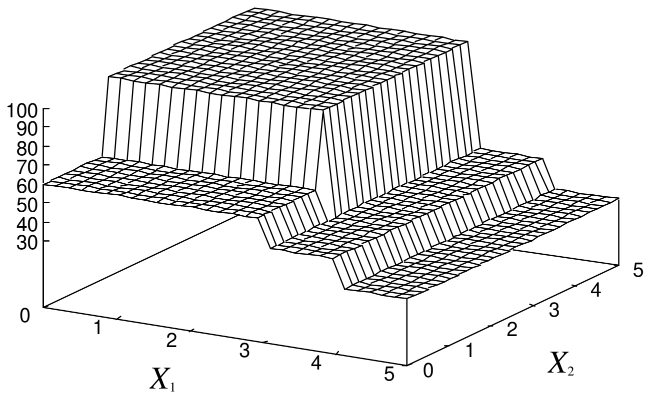
$P_2 \equiv X_1 < 3 \wedge X_2 \geq 1.5$, with $k_2 = 100$

$P_3 \equiv X_1 \geq 3 \wedge X_1 < 4$, with $k_3 = 30$

$P_4 \equiv X_1 \geq 3 \wedge X_1 \geq 4$, with $k_4 = 45$

As there are four distinct paths from the root node to the leaves, this tree divides the input space in four different regions. The conjunction of the tests in each path can be regarded as a logical description of such regions, as shown above.

This tree roughly corresponds to the following regression surface (assuming that there were only the predictor variables X_1 and X_2) :



Using the more concise representation of Equation 3.1 we obtain:

$$m(\mathbf{x}) = 60 \times I(X_1 < 3 \wedge X_2 < 1.5) + 100 \times I(X_1 < 3 \wedge X_2 \geq 1.5) +$$
$$30 \times I(X_1 \geq 3 \wedge X_2 < 4) + 45 \times I(X_1 \geq 3 \wedge X_2 \geq 4)$$

♦

Regression trees are constructed using a recursive partitioning (RP) algorithm. This algorithm builds a tree by recursively splitting the training sample into smaller subsets. We give below a high level description of the algorithm. The RP algorithm receives as input a set of n data points, $D_t = \{\langle \mathbf{x}_i, y_i \rangle\}_{i=1}^{n_t}$, and if certain termination criteria are not met it generates a test node t, whose branches are obtained by applying the same algorithm with two subsets of the input data points. These subsets consist of the cases that logically entail the split test s^* in the node t, $D_{t_L} = \{\langle \mathbf{x}_i, y_i \rangle \in D_t : \mathbf{x}_i \to s^*\}$, and the remaining cases, $D_{t_R} = \{\langle \mathbf{x}_i, y_i \rangle \in D_t : \mathbf{x}_i \not\to s^*\}$. At each node the best split test is chosen according to some local criterion, which means that this is a greedy hill-climbing algorithm.

Algorithm 3.1 – Recursive Partitioning Algorithm.

```
Input  : A set of n data points, { < xi , yi > }, i = 1,...,n
Output : A regression tree

IF termination criterion THEN
       Create Leaf Node and assign it a Constant Value
       Return Leaf Node
ELSE
       Find Best Splitting Test s*
       Create Node t with s*
       Left_branch(t) = RecursivePartitioningAlgorithm({ <xi , yi> : xi → s* })
       Right_branch(t) = RecursivePartitioningAlgorithm({ <xi , yi> : xi ↛ s* })
       Return Node t
ENDIF
```

The algorithm has three main components:

- A way to select a split test (the splitting rule).
- A rule to determine when a tree node is terminal (termination criterion).
- A rule for assigning a value to each terminal node.

In the following sections we present two different approaches to solve these problems. These alternatives try to minimise either the mean squared error or the mean absolute deviation of the resulting tree.

3.2 Least Squares Regression Trees

The most common method for building a regression model based on a sample of an unknown regression surface consists of trying to obtain the model parameters that minimise the least squares error criterion,

$$\frac{1}{n}\sum_{i}^{n}\left(y_i - r(\beta, \mathbf{x}_i)\right)^2 \tag{3.2}$$

where,

n is the sample size;

$<\mathbf{x}_i , y_i>$ is a data point ;

and $r(\beta, \mathbf{x}_i)$ is the prediction of the regression model $r(\beta, \mathbf{x})$ for the case $\langle \mathbf{x}_i, y_i \rangle$.

As we have seen in Chapter 2 this criterion is used in many existing systems. RETIS (Karalic & Cestnik, 1991), M5 (Quinlan, 1992) and CART (Breiman *et. al.*, 1984), all use the least squares (LS) criterion. To our knowledge the only tree induction system that is also able to use the mean absolute deviation is CART.

The following theorem holds for the LS minimisation criterion:

<u>THEOREM 3.1</u>

The constant k that minimises the expected value of the squared error is the mean value of the target variable.

♦

A proof of this theorem can be found in the appendix at the end of this chapter. Based on this theorem the constant that should be assigned to the leaves of a regression tree obtained using the least squares error criterion, is the average of the target values of the cases within each leaf *l*, [22]

$$k_l = \frac{1}{n_l} \sum_{D_l} y_i \tag{3.3}$$

where,

n_l is the cardinality of the set D_l containing the cases in leaf l (*i.e.* $n_l = \#D_l$).

[22] According to the RP algorithm, the cases within any node t of a tree, are the subset of the given training sample that satisfies the conjunction consisting of all tests from the root to that node. We will denote those cases as $D_t = \{\langle \mathbf{x}_i, y_i \rangle \in t\}$.

Some systems like RETIS (Karalic, 1992) and M5 (Quinlan,1992) use other non-constant models in the tree leaves. They use linear polynomials instead of averages. We go back to this issue in Chapter 5, where we address hybrid tree models.

With respect to the splitting rule we restrict our description to the case of binary trees. Each inner node of these trees has two descendent nodes. These inner nodes split the training instances in two subsets depending on the result of a test on one of the input variables. Cases satisfying the test follow to the left branch while the others go to the right branch. The split test is chosen with the objective of improving the fitting error of the resulting tree. Any path from the root node to a node t corresponds to a partition D_t of the input cases. Assuming the constant obtained with Equation 3.3, resulting from the application of the least squares error criterion, we define the fitting error of a node t as the average of the squared differences between the Y values of the instances in the node and the node constant k_t ,

$$Err(t) = \frac{1}{n_t} \sum_{D_t} (y_i - k_t)^2 \tag{3.4}$$

where, k_t is defined by Equation 3.3.

Furthermore, we define the error of a tree T as a weighed average of the error in its leaves:

$$Err(T) = \sum_{l \in \tilde{T}} P(l) \times Err(l) = \sum_{l \in \tilde{T}} \frac{n_l}{n} \times \frac{1}{n_l} \sum_{D_l} (y_i - k_l)^2 = \frac{1}{n} \sum_{l \in \tilde{T}} \sum_{D_l} (y_i - k_l)^2 \tag{3.5}$$

where,

$P(l)$ is the probability of a case falling into leaf l;
n is the total number of training cases;
n_l is the number of cases in leaf l;
and $\tilde{T}$ is the set of leaves of the tree T.

A binary split divides a set of cases in two. The goal of the splitting rule is to choose the split that maximises the decrease in the error of the tree resulting from this division. We define the error of a split s as the weighed average of the errors of the resulting sub-nodes,

$$Err(s,t) = \frac{n_{t_L}}{n_t} \times Err(t_L) + \frac{n_{t_R}}{n_t} \times Err(t_R) \tag{3.6}$$

where,

t_L is the left child node of t defining a partition D_{t_L} that contains the set of cases $\{< \mathbf{x}_i, y_i > \in D_t : \mathbf{x}_i \to s\}$ and n_{t_L} the cardinal of this set;

and t_R is the right child node of t defining a partition D_{t_R} that contains the set of cases $\{< \mathbf{x}_i, y_i > \in D_t : \mathbf{x}_i \not\to s\}$ and n_{t_R} the cardinal of this set.

We are now ready to present the definition of best split for a node t given a set S of candidate splits,

DEFINITION 3.1

The best split s^* is the split belonging to S that maximises

$$\Delta Err(s,t) = Err(t) - Err(s,t)$$

♦

This greedy criterion guides the choice of a split for all inner nodes of an LS regression tree. On each iteration of the RP algorithm all possible splits of each of the predictor variables are evaluated and the one with best ΔErr is chosen.

With respect to the last issue of the tree growing method, that is the stopping rule, the key problem is the reliability of error estimates used for selecting the splits. All the error measures described above are estimates in the statistical sense, as they are functions of the training sample (usually called *resubstitution estimates*). The *accuracy* of these estimates is strongly dependent on the *quality* of the sample. As the algorithm recursively divides the original training set, the splits are being evaluated using increasingly smaller samples. This means that the estimates are getting potentially more unreliable as we grow the tree[23]. It can be easily proven that the value of ΔErr (Definition 3.1) is always greater or equal to zero during tree growth. Apparently we are always obtaining a more precise regression tree model. Taking this argument to extremes, an overly large tree with just one training case in each leaf would have an error of zero. The problem with this reasoning is exactly the

[23] Because the standard error of statistical estimators is inversely proportional to the sample size.

reliability of our estimates due to the amount of training cases upon which they are being obtained. Estimates based on small samples will hardly generalise to unseen cases thus leading to models with poor predictive accuracy. This is usually known as overfitting the training data as we have seen in Section 2.3.2.1.

There are two alternative procedures to minimise this problem. The first consists of specifying a reliable criterion that tries to determine when one should stop growing the tree. Within tree-based models this is usually called *pre-pruning*. The second, and most frequently used procedure, is to grow a very large (and unreliable) tree and then *post-prune* it. Pruning of regression trees is an essential step for obtaining accurate trees and it will be the subject of Chapter 4. With a post-pruning approach the stopping criteria are usually very "relaxed", as there will be a posterior pruning stage. The idea is not to "loose" any potentially good post-pruned tree by stopping too soon at the initial growth stage. A frequently used criterion is to impose a minimum number of cases that once reached forces the termination of the RP algorithm. Another example of stopping criteria is to create a leaf if the error in the current node is below a fraction of the error in the root node.

3.2.1 Efficient Growth of LS Regression Trees

The computational complexity of the recursive partitioning (RP) algorithm used for growing regression trees is highly dependent on the choice of the best split for a given node. This task resumes to trying all possible splits for each of the input variables. The number of possible splits of a variable is strongly dependent on its type. We give below a more detailed version of the RP algorithm used for growing a LS regression tree:

Algorithm 3.2 - Growing a LS Regression Tree.

Input : A set of n data points, { < $\mathbf{x}_i$, y_i > }, i = 1,...,n
Output : A regression tree

```
IF termination criterion THEN
       Create Leaf Node and assign it the average Y value of the n data points
       Return Leaf Node
ELSE
       S* = <arbitrary split>
```

```
        FOR all variables Xv DO
              IF Xv is a nominal variable THEN
                    BestSplitXv = TryAllNominalSplits({< xi,yi >}, Xv)
              ELSE IF Xv is a numeric variable THEN
                    BestSplitXv = TryAllNumericSplits({< xi,yi >}, Xv)
              ENDIF
              IF BestSplitXv is better than s* THEN
                    s* = BestSplitXv
              ENDIF
        ENDFOR
        Create Node t with s*
        Left_branch(t) = GrowLStree({< xi,yi > : xi → s* })
        Right_branch(t) = GrowLStree({< xi,yi > : xi ↛ s* })
        Return Node t
ENDIF
```

The major computational burden of this algorithm lies in the part where we try all possible splits of a variable. Each trial split has to be evaluated, which means that we need to obtain the model of the resulting sub-nodes to calculate their error (*c.f.* Equations 3.4 and 3.6). Assuming the constant model defined in Equation 3.3, we need to calculate two averages (for each branch of the split) to evaluate each split (Definition 3.1). Equation 3.4 is in effect similar to the formula for calculating the variance of a variable[24]. This calculation involves passing through the data twice, once to obtain the average and the second time to calculate the squared differences. This cost can be reduced using the equivalent formula[25],

$$Err(t) = \frac{\sum_{D_t} y_i^2}{n_t} - \left(\frac{\sum_{D_t} y_i}{n_t} \right)^2 \tag{3.7}$$

This calculation can be carried out using a single pass through the data. Even using this formula the cost of evaluating each trial split would still be $O(n_t)$. We propose to reduce this cost using a simplification that enables and incremental evaluation of all splits of a variable. According to the formula given in Definition 3.1 the best split s^* is the one that minimises the value given by Equation 3.6. Using the formula in Equation 3.7 we get,

[24] The only difference is that for obtaining unbiased estimates of the variance based on a sample one usually divides the sum of squares by n-1 and not by n.

[25] We should note, however, that this formulation brings potential round-off errors (Press *et. al.* 1992).

$$Err(s,t) = \frac{n_{t_L}}{n_t} \times \left(\frac{\sum_{D_{t_L}} y_i^2}{n_{t_L}} - \left(\frac{\sum_{D_{t_L}} y_i}{n_{t_L}} \right)^2 \right) + \frac{n_{t_R}}{n_t} \times \left(\frac{\sum_{D_{t_R}} y_i^2}{n_{t_R}} - \left(\frac{\sum_{D_{t_R}} y_i}{n_{t_R}} \right)^2 \right)$$

To simplify notation let SS_L and SS_R be equal to $\sum_{D_{t_L}} y_i^2$ and $\sum_{D_{t_R}} y_i^2$, respectively, and S_L and S_R be equal to $\sum_{D_{t_L}} y_i$ and $\sum_{D_{t_R}} y_i$, respectively, leading to

$$Err(s,t) = \frac{SS_L}{n_t} - \frac{S_L^2}{n_{t_L} n_t} + \frac{SS_R}{n_t} - \frac{S_R^2}{n_{t_R} n_t}$$

$$= \frac{1}{n_t}(SS_L + SS_R) - \frac{1}{n_t}\left(\frac{S_L^2}{n_{t_L}} + \frac{S_R^2}{n_{t_R}} \right)$$

♦

It is easy to see that the first term of this formula is constant whatever the split is that is being evaluated. This is so because $D_t = D_{t_L} \cup D_{t_R}$, so $\sum_{D_{t_L}} y_i^2 + \sum_{D_{t_R}} y_i^2 = \sum_{D_t} y_i^2$, which means that $SS_L + SS_R$ is always constant and equal to $\sum_{D_t} y_i^2$. This means that the only difference among different candidate splits is in the last term.

This simplification we have derived has important consequences on the method used to evaluate and select the best split of a node. Using these results we can present a new definition for the best split s^* of a variable, which has significant advantages in terms of computation efficiency when compared to the previous one (Definition 3.1). Note, however, that this is only valid assuming a constant model of the form given by Equation 3.3 (*i.e.* assuming a least squares error criterion). As our goal is to minimise the expression derived above we get the following new definition for the best split of a node:

DEFINITION 3.2

The best split s^* is the split belonging to S that maximises the expression

$$\frac{S_L^2}{n_{t_L}} + \frac{S_R^2}{n_{t_R}}$$

where, $S_L = \sum_{D_{t_L}} y_i$ and $S_R = \sum_{D_{t_R}} y_i$

♦

This definition enables a fast incremental evaluation of all candidate splits S of any predictor variable as we will see in the following two sections.

3.2.2 Splits on Continuous Variables

We will now present an algorithm that finds the best split for continuous variables using the results of Definition 3.2. Assuming that we have a set of n_t cases whose sum of the Y values is S_t, Algorithm 3.3 obtains the best split on a continuous predictor variable X_v.

Algorithm 3.3 - Finding the best split for a continuous variable.

```
Input : nt cases, sum of their Y values ( St ), the variable Xv
Output : The best cut-point split on Xv

Sort the cases according to their value in Xv
SR = St; SL = 0
nR = nt ; nL = 0
BestTillNow = 0
FOR all instances i DO
        SL = SL + yi; SR = SR - yi
        nL = nL + 1   ; nR = nR - 1
        IF ( Xi+1,v > Xi,v ) THEN                    % No trial if values are equal
                NewSplitValue = (SL^2 / nL) + (SR^2 / nR)
                IF ( NewSplitValue > BestTillNow ) THEN
                        BestTillNow = NewSplitValue
                        BestCutPoint = ( Xi+1,v + Xi,v ) / 2
                ENDIF
        ENDIF
ENDFOR
```

This algorithm has two main parts. The first consists of sorting the instances by the values of the variable being examined, which has an average complexity of $O(n_t \log n_t)$ using Quick Sort. This sorting operation is necessary for running through all trial cut-point values[26] in an efficient manner. We only need to try these cut-point values as they are the only ones that may change the value of the score given by Definition 3.2, because they modify the set of cases that go to the left and right branches. The second relevant part of the algorithm is the evaluation of all candidate splits. The number of trial splits is at most n_t - 1 (if all instances have different value of the variable X_v). Without the equation given in Definition 3.2 we would have to calculate the "variance" of each of the partitions originated by the candidate split. This would involve passing through all data points (using the optimised formula of Equation 3.7) which is $O(n_t)$. This would lead to a worst case complexity of $O(n_t(n_t -1))$ for the second part of the algorithm. Our optimisation given by the formula in Definition 3.2 leads to a worst case complexity of $O(n_t - 1)$ as the "variance" calculation is avoided. Notice that this is only valid for the least squares error criterion that leads to the given simplification. If other criteria were used the complexity could be different particularly if no similar incremental algorithm could be found. With the existence of this fast and incremental method of computing the error gain of a split the complexity of the algorithm is dominated by the sorting operation.

3.2.3 Splits on Discrete Variables

Splits on nominal (or discrete) variables usually involve trying all possible tests of the form $X_v = x_v$, where x_v is one of the possible values of variable X_v. If there are many possible values this usually leads to larger trees. An alternative is not to use binary trees and have one branch for each possible value of the variable. This has the disadvantage of an increased splitting of the training samples, which leads to potentially less reliable estimates sooner than the alternative that involves binary splits. Yet another possible alternative is to consider tests of the form $X_v \in \{x_v, ...\}$. This solution has additional computational costs although it can improve the comprehensibility of the resulting trees and it does not split too much the training cases. Breiman *et. al.* (1984) proved an interesting result (see their

[26] A cut-point value is the value tested in a continuous variable split (*e.g.* X < 10).

Theorem 4.5, Proposition 8.16 and Section 9.4) that changes the complexity of obtaining this type of splits from $O(2^{\#\chi_v -1})$ into $O(\#\chi_v - 1)$, where $\#\chi_v$ is the cardinality of the domain of variable X_v. The method suggested by Breiman *et. al.* (1984, p.247) involves an initial stage where the instances of the node are sorted as follows. Assuming that B is the set of values of X_v that occur in the current node t (*i.e.* $B = \{ b : \mathbf{x}_i \in t \wedge \mathbf{X}_{i,v} = b \}$), and defining $\bar{y}(b_i)$ as the average Y value of the instances having value b_i in variable X_v, we sort the values such that,

$$\bar{y}(b_1) \leq \bar{y}(b_2) \leq \ \ldots \ \leq \bar{y}(b_{\#B})$$

Having the variable values sorted this way, Breiman and his colleagues have proven that,

<u>DEFINITION 3.3</u> (BREIMAN ET AL., 1984)[27]

The best split on discrete variable X_v in node t is one of the $\#B$-1 splits

$$X_v \in \{ b_1 , b_2 , \ldots , b_h \} , \ h = 1, \ldots, \#B\text{-}1$$

♦

This definition results from a theorem that was proved by Fisher (1958) for the case of the least squares error criterion for regression and was extended by Breiman and his colleagues (1984, Sec. 9.4) for a larger class of concave impurity (error) functions. Chou (1991) furthers generalised these results to an arbitrary number of bins (*i.e.* not only binary splits) and to other error functions.

With this method we only have to look for $\#B$-1 subsets instead of $2^{\#B-1}$. Notice that we still need to "pass through" all data to obtain the values $\bar{y}(b_i)$, plus a sorting operation with $\#B$ elements. Before presenting the algorithm for discrete splits we provide a simple example to illustrate this method.

---

[27] The proof of this theorem is given in Section 9.4 (p.274) of Breiman *et. al.* (1984). A much simpler demonstration based on Jensen's inequality can be found in Ripley (1996, p.218).

EXAMPLE 3.2

Suppose that we have the following instances in a node t :

COLOR	...	*Y*
green	...	24
red	...	56
green	...	29
green	...	13
blue	...	120
red	...	45
blue	...	100

leading to the averages

$$\bar{y}(green) = (24 + 29 + 13)/3 = 22$$

$$\bar{y}(red) = (56 + 45)/2 = 50.5$$

and $\bar{y}(blue) = (120 + 100)/2 = 110$

If we sort the values according to their respective average Y values we obtain the ordering <*green*, *red*, *blue*>. According to Breiman's theorem the best split would be one of the #*B*-1 (in this case 2 = 3-1) splits, namely $X_v \in \{green\}$ and $X_v \in \{green, red\}$.

♦

Having the instances sorted according to the method explained above, we use the following incremental algorithm similar to the one presented for continuous variables.

```
Algorithm 3.4 – Finding the best subset split for a discrete variable.

Input : n cases, sum of their Y values ( St ), the variable Xv
Output : An ordered set of values of Xv and a partition of this set

Obtain the average Y value associated to each value of Xv
Sort the values of Xv according to the average Y associated to each value
SR = St ; SL = 0
nR = nt ; nL = 0
BestTillNow = 0
FOR each value b of the obtained ordered set of values DO
       YB = sum of the Y values of the cases with Xv = b
       NB = number of the cases with Xv = b
       SL = SL + YB; SR = SR – YB
       nL = nL + NB   ; nR = nR – NB
       NewSplitValue = (SL^2 / nL)  + (SR^2 / nR)
       IF ( NewSplitValue > BestTillNow ) THEN
              BestTillNow = NewSplitValue
              BestPosition = position of b in set of ordered values
       ENDIF
ENDFOR
```

The complexity of this algorithm is lower compared to the case of continuous variables. In effect, it is dominated by the number of values of the attribute (#*B*). The exception is the part of sorting the values according to their average *Y* value. The sorting in itself is *O*(#*B* log #*B*) but to obtain the average *Y* values associated to each value *b* we need to run through all given instances ($O(n_t)$), which is most probability more complex than the sorting operation, unless there are almost as different values as there are instances.

3.2.4 Some Practical Considerations

The considerations on computational complexity described in the previous sections, indicate that the key computational issue when building least squares regression trees is sorting. We confirmed this in practice by looking at the execution profiles of our tree induction system, RT. We observed that more than 50% of the CPU time was being spent inside of the Quick Sort function. We have tried other sorting algorithms like Heap Sort (*e.g.* Press *et al.*, 1992) but no significant differences were observed. The weight of this sorting operation is so high that even the implementation of the Quick Sort algorithm is a key issue. In an earlier version of our RT system we used a "standard" recursive implementation of this algorithm. This standard implementation has difficulties when the data is already almost sorted. When we finally used the implementation given by Press *et al.* (1992) we have noticed dramatic improvements in the computation time for large data sets. This means that when dealing with huge data sets the presence of continuous variables can become overwhelming due to the necessary sorting of their values for finding the best cut-point. This was already mentioned in Jason Cattlet's Ph.D. thesis (1991) in the context of classification trees. The author described some techniques that try to overcome this problem, like attribute discretisation, which is often explored within Machine Learning (see Dougherty *et. al.* 1995 for a survey), and sub-sampling to avoid sorting all values (called by the author peepholing).

We have carried out a simulation study with two artificial data sets (*Fried* and *Mv*)[28] to observe the behaviour of our RT system with respect to computation time. In this experiments we have generated training samples with sizes from 1000 to 150000 cases. For each sample we have generated one LS regression tree, recording the respective CPU time[29] taken to carry out this task. The results for each of the data sets are shown in Figure 3.1:

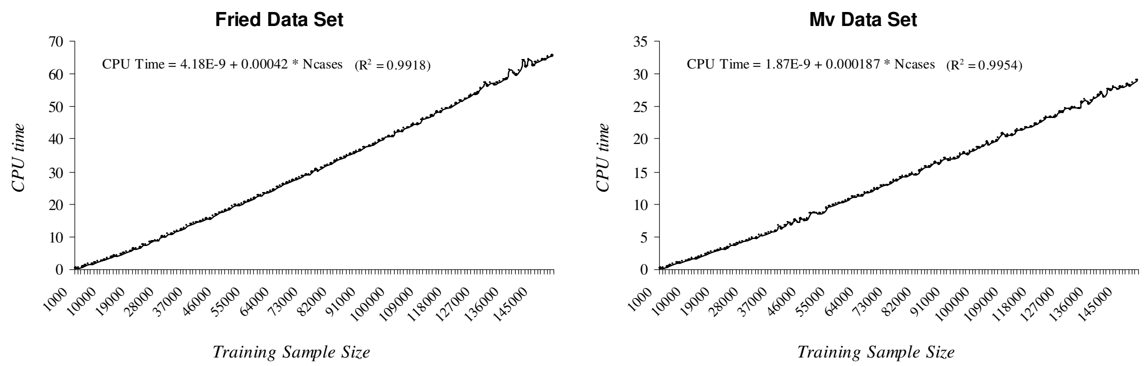


Figure 3.1 - Computation time to generate a LS tree for different sample sizes.

These graphs show a clear linear dependence of the CPU time on the number of training cases. We also present two linear models obtained with the results of these experiments, confirming that the proportion of variance explained by the respective linear relations is very significant ($R^2 > 0.99$). This simulation study confirms the validity of the speed-ups we have proposed to the generation of least squares regression trees. They demonstrate that our RT system can easily handle large data sets and, moreover, they show a desirable linear dependence of the necessary computation time on the sample size.

[28] Full details of these data sets can be found in Appendix A.2.

[29] The experiments were carried out in a dual Pentium II 450MHz machine running Linux.

3.3 Least Absolute Deviation Regression Trees

In their book on classification and regression trees, Breiman and colleagues (1984) mentioned the possibility of using a least absolute deviation (LAD) error criterion to obtain the best split for a node of a regression tree. However, at the time the method was not yet fully implemented (Breiman *et al.*,1984, p.258), so no algorithms or results were given. LAD regression trees use as selection criterion the minimisation of the absolute deviation between the model predictions and the *Y* values of the cases. The use of this criterion leads to trees that are more robust with respect to the presence of outliers and skewed distributions. This is the main motivation for studying LAD regression trees. Least squares (LS) regression trees do not have this nice property. In effect, the squared differences amplify the effect of the error of an outlier. Moreover, the presence of outliers can strongly influence the average, thus leading to values in the leaves that are not "representing" correctly the corresponding training cases.

Building a regression model based on a sample $\{\langle \mathbf{x}_i, y_i \rangle\}_{i=1}^{n}$ using the least absolute deviation error criterion consists of finding the model parameters that minimise the estimated mean absolute deviation,

$$\frac{1}{n}\sum_{i=1}^{n}\left| y_i - r(\beta, \mathbf{x}_i)\right| \tag{3.8}$$

where, $r(\beta, \mathbf{x}_i)$ is the prediction of the model $r(\beta, \mathbf{x})$ for the case $\langle \mathbf{x}_i, y_i \rangle$.

The constant *k* that minimises the estimated mean absolute deviation of the observations with respect to *k*, is the median *Y* value. Minimising the mean absolute deviation to a constant *k* corresponds to minimising the statistical expectation of $| y_i - k |$.

<u>THEOREM 3.2</u>

The constant k that minimises the expected value of the absolute deviation to a continuous random variable Y, with probability density function f(y), is the median of the variable Y, ν_Y.

♦

A formal proof of this theorem can be found in the appendix at the end of this chapter. As a consequence of this theorem LAD trees should have medians at the leaves instead of averages like LS regression trees. The median is a better statistic of centrality than the average for skewed distributions. When the distribution is approximately normal the median and the average are almost equivalent. This generality can be seen as an advantage of LAD trees over LS trees. However, we will see that LAD trees bring additional computational costs, which can make them less attractive for extremely large data sets.

Growing a LAD regression tree involves the same three major questions mentioned before with respect to LS trees: a splitting rule; a stopping criterion; and a rule for assigning a value to all terminal nodes. Moreover, the algorithm driving the induction process is again the Recursive Partitioning algorithm. With respect to the stopping rule the same considerations regarding overfitting can be made for LAD trees. In effect, the overfitting avoidance strategy is independent of the error criterion selected for growing the trees. We can stop earlier the growth of the tree, or we may post-prune an extremely large tree. With respect to the value to assign to each leaf, we have seen that it is the median of the Y values of the cases falling in the leaf. This leads to some differences in the splitting rule. Using the definition presented in Equation 3.6 for the error of a split, we can define the best split for a node t using the least absolute deviation criterion as,

DEFINITION 3.4

The best split s^* is the split belonging to the candidate set of splits S that maximises

$$\Delta Err(s,t) = Err(t) - Err(s,t)$$

which using Equations 3.6 and 3.8 turns out to be equivalent to minimising

$$\frac{n_{t_L}}{n_t} \times \frac{1}{n_{t_L}} \sum_{D_{t_L}} \left| y_i - \nu_{t_L} \right| \quad + \quad \frac{n_{t_R}}{n_t} \times \frac{1}{n_{t_R}} \sum_{D_{t_R}} \left| y_i - \nu_{t_R} \right|$$

or equivalently minimising $\sum_{D_{t_L}} \left| y_i - \nu_{t_L} \right| \quad + \quad \sum_{D_{t_R}} \left| y_i - \nu_{t_R} \right|$

where, ν_{t_L} and ν_{t_R} are the medians of the left and right sub-nodes, respectively.

♦

Thus the best split minimises the sum of the absolute deviations to the respective medians of the left and right branches. Obtaining the median of a set of values involves sorting them, the median being the "middle" value after the sorting. Without any computational simplification this would mean that for each trial split we would need to sort the cases by *Y* value in each sub-branch to obtain the medians and then "pass through" the data again to obtain the two sums of absolute deviations. This would have an average complexity of $O(n^2 \log n)$ for each trial split. Furthermore, this would have to be done for all possible splits of every variable. For instance, a continuous variable has potentially n-1 trial cut-points, which would lead to a average complexity proportional to $O(n^3 \log n)$. This represents too much computation even for simple problems. We will present algorithms that overcome this serious limitation of LAD trees.

3.3.1 Splits on Continuous Variables

As we have seen in Algorithm 3.3 the first step before trying all possible splits of a continuous variable X_i is to sort the instances by the values of this variable. According to Definition 3.4, given a set of cases and a cut-point *V*, we need to obtain the sum of the absolute deviations (*SAD*) of the left and right branches to evaluate the split. For this purpose we need to know the *Y* medians of the cases in each branch of the test. As we have seen the median of a random variable *Y* with probability density function *f*(*y*), is the value ν for which the probability of a value being greater or equal to it is 0.5 (*i.e.* $\int_{-\infty}^{\nu} f(y)dy = \int_{\nu}^{+\infty} f(y)dy$). If we have a sample of such a variable we approximate these probabilities with frequencies. Thus the *sample median* of a set of *n* measurements of a variable *Y*, is the middle value when the observations are arranged in ascending order. If *n* is an odd number, there is a unique middle value, otherwise the median is taken as the average between the two middle points. This means that we need to sort the set of observations to obtain the median. Figure 3.2 shows an example split (X_i < 145) to help clarifying this task:

$X_i < 145$ $\quad$ $X_i \geq 145$

X_i	100	123	130	131	140	150	150	170	175	230
Y values	230	200	10	13	53	234	546	43	23	67
Sorted *Y* values	10	13	53	200	230	23	43	67	234	546

$\nu_L = 53$

$SAD_L = 407$

$\nu_R = 67$

$SAD_R = 708$

Figure 3.2 - A split on $X_i < 145$.

The key issue we face within LAD trees, is how to efficiently compute the sum of absolute deviations for a new trial split $X_i < V'$ (with $V' > V$). Obtaining these *SAD* values for a new cut point involves "moving" some cases from the right branch to the left branch. For instance, in Figure 3.2 if the new trial cut-point is 160 this would mean that 234 and 546 would now belong to the left branch. We would like to avoid having to re-sort the *Y* values for each new trial cut point to obtain the new medians. Moreover, we would also like to avoid passing again through the data to calculate the new *SAD* values. Thus the key to solve the efficiency problems of LAD trees resumes to the following two related problems:

- Given a set of points with median ν and respective *SAD*, how to obtain the new median ν' and the new SAD', when we *add* a new set of data points to the initial set.
- Given a set of points with median ν and respective *SAD*, how to obtain the new median ν' and the new SAD', when we *remove* a set of data points from the initial set.

In the remaining of this section we will describe an efficient method for solving these two problems. The algorithm we will present obtains the new medians and *SAD*s based on the values of the previous trial cut point, which largely improves the efficiency of finding the best LAD split of a continuous variable.

Let *P* be a set containing the ordered *Y* values of a branch. Let us divide this set in two ordered subsets, P^- and P^+. The set P^- contains all observations less or equal to the median, and P^+ the remaining observations. For the left branch of the example in Figure 3.2, we would have $P^- = \{10, 13, 53\}$ and $P^+ = \{200, 230\}$. Given the definition of the median it is easy to see that the following holds:

$$\begin{cases} \#P^- - \#P^+ = 0 & \text{if } \#P \text{ is even} \\ \#P^- - \#P^+ = 1 & \text{if } \#P \text{ is odd} \end{cases} \tag{3.9}$$

If the total number of observations in P is even, the median is the average between the maximum value in P^- and the minimum value in P^+, otherwise the median is the maximum value of P^-. Adding (or removing) a set of observations to P is equivalent to obtaining two new ordered subsets subject to the restrictions given in 3.9. Ordered insertion (removal) in sorted sets can be achieved with computational efficiency using balanced binary trees (AVL trees) (Wirth, 1976). Using these data structures we can efficiently update P^- and P^+ when given a new set of data points B that we wish to add to P. An insertion (removal) in an AVL tree can be done in an average time of the order of $O(\log SizeOfTree)$. This means that the addition (removal) of a set of points B can be done in an average time of the order of $O(\#B \log \#P/2)$. The only problem we face is that when new points are added (removed) the restrictions given in 3.9 may be violated. We thus may need some additional bookkeeping to maintain these constraints. By updating the new subsets subject to the restrictions given in 3.9, we can easily obtain the new median after the addition (removal) of a set of observations.

We now address the issue of obtaining the new sum of absolute deviations, *SAD'*. Let $d(y_1, y_2)$ be defined as

$$d(y_1, y_2) = y_2 - y_1$$

The sum of absolute deviations of a set of points P with median ν_P is given by ,

$$\begin{aligned} SAD_{P,\nu_P} &= \sum_P \left|d(y_i, \nu_P)\right| = \sum_{P^-} d(y_i, \nu_P) + \sum_{P^+} d(\nu_P, y_i) \\ &= \sum_{P^-} \nu_P - \sum_{P^-} y_i + \sum_{P^+} y_i - \sum_{P^+} \nu_P = \sum_{P^+} y_i - \sum_{P^-} y_i + \nu_P\left(\#P^- - \#P^+\right) \end{aligned} \tag{3.10}$$

Using again the example in Figure 3.2, we can confirm this expression observing that

$$SAD_L = |d(10,53)| + |d(13,53)| + |d(53,53)| + |d(200,53)| + |d(230,53)| = d(10,53) + d(13,53) + d(53,53) + d(53,200) + d(53,230) = 407$$

or equivalently,

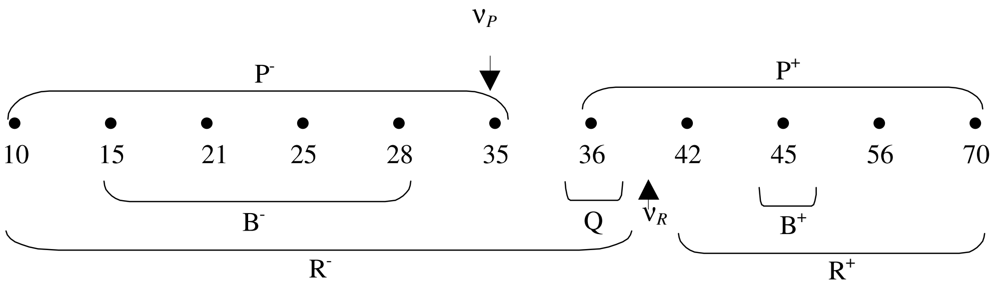
$$SAD_L = (200 + 230) - (10 + 13 + 53) + 53 \times (3 - 2) = 430 - 76 + 53 = 407$$

We will now address the problem of what happens to this *SAD* value when we remove a set of points from *P*. Let *B* be a set of points we want to remove from *P*, originating the set *R* (*i.e.* $R = P \setminus B$). According to 3.10, the sum of absolute deviations of the set *R* is given by:

$$SAD_{R,\nu_R} = \sum_{R^+} y_i - \sum_{R^-} y_i + \nu_R \left(\#R^- - \#R^+\right) \tag{3.11}$$

We will present an alternative formulation for SAD_{R,ν_R} based on the value of SAD_{P,ν_P} and *B*. This will reduce the computational complexity of obtaining SAD_{R,ν_R} from $O(\#R)$ to $O(\#B \log \#R/2)$. Furthermore our solution avoids the second pass through the data, and we manage to obtain SAD_{R,ν_R} as we update the median.

Let us assume that in the set *B* there are more values smaller than the median ν_P, than values above this median. If we denote these two subsets of *B* as B^- and B^+, respectively, this corresponds to saying that $\#B^- > \#B^+$. This implies that the new median after the removal of *B* will be larger than the previous value, *i.e.* $\nu_R > \nu_P$. The following example clarifies this reasoning.



where,

P^- is the set of values smaller or equal to the median of the set *P*;

P^+ is the set of value greater than this median;
B is the set of points we want to remove from P [30];
B^- is the subset of B with values smaller or equal to the median of P;
B^+ is the subset of B containing the values greater than this median;
R is the new set resulting from removing B from P;
R^- is the subset of R containing the values smaller or equal to the median of R;
R^+ is the subset of R with the values greater than this median;
and Q is the subset of R containing values in the interval between the median of P and the median of R.

The following set relations hold under the assumption that $\nu_R > \nu_P$ after removing B from P:

$$R^+ = P^+ - B^+ - Q$$

$$R^- = P^- - B^- + Q$$

where, Q is the set containing all points for which $d(y_i, \nu_P) < d(\nu_R, \nu_P)$.

Using these relations we can re-write Equation 3.11 as :

$$\begin{aligned} SAD_{R,\nu_R} &= \sum_{P^+} y_i - \sum_{B^+} y_i - \sum_{Q} y_i - \sum_{P^-} y_i + \sum_{B^-} y_i - \sum_{Q} y_i + \\ &\quad + \nu_R \left(\#P^- - \#B^- + \#Q - \#P^+ + \#B^+ + \#Q\right) \\ &= \sum_{P^+} y_i - \sum_{P^-} y_i + \sum_{B^-} y_i - \sum_{B^+} y_i - 2\sum_{Q} y_i + \nu_R \left(\#P^- - \#P^+ + \#B^+ - \#B^- + 2\#Q\right) \end{aligned}$$

Similarly, if we assume that $\nu_R < \nu_P$ (*i.e.* $\#B^- < \#B^+$):

$$R^+ = P^+ - B^+ + Q$$

$$R^- = P^- - B^- - Q$$

This leads to,

$$\begin{aligned} SAD_{R,\nu_R} &= \sum_{P^+} y_i - \sum_{B^+} y_i + \sum_{Q} y_i - \sum_{P^-} y_i + \sum_{B^-} y_i + \sum_{Q} y_i + \\ &\quad + \nu_R \left(\#P^- - \#B^- - \#Q - \#P^+ + \#B^+ - \#Q\right) \\ &= \sum_{P^+} y_i - \sum_{P^-} y_i + \sum_{B^-} y_i - \sum_{B^+} y_i + 2\sum_{Q} y_i + \nu_R \left(\#P^- - \#P^+ + \#B^+ - \#B^- - 2\#Q\right) \end{aligned}$$

Finally, if $\nu_R = \nu_P$ (i.e. $\#B^- = \#B^+$), we have :

[30] For instance as a result of a new trial cut-point split (*c.f.* example of Figure 3.2)

$R^+ = P^+ - B^+$

$R^- = P^- - B^-$

leading to,

$$SAD_{R,\nu_R} = \sum_{P^+} y_i - \sum_{B^+} y_i - \sum_{P^-} y_i + \sum_{B^-} y_i + \nu_R\left(\#P^- - \#B^- - \#P^+ + \#B^+\right)$$
$$= \sum_{P^+} y_i - \sum_{P^-} y_i + \sum_{B^-} y_i - \sum_{B^+} y_i + \nu_R\left(\#P^- - \#P^+ + \#B^+ - \#B^-\right)$$

Integrating all three cases into one single formula we get,

$$SAD_{R,\nu_R} = \sum_{P^+} y_i - \sum_{P^-} y_i + \sum_{B^-} y_i - \sum_{B^+} y_i + \left(-\left(\nu_R \neq \nu_P\right)\right)^{(\nu_R > \nu_P)} \times 2\sum_{Q} y_i$$
$$+ \nu_R\left(\#P^- - \#P^+ + \#B^+ - \#B^- + \left(-\left(\nu_R \neq \nu_P\right)\right)^{(\nu_R < \nu_P)} \times 2\#Q\right) \tag{3.12}$$

This formula looks much more complex than Equation 3.11. However, from a computational point of view, it is more suitable for the incremental evaluation of the trial splits. In effect, we are defining the *SAD* of the new set of points (R) as a function of the previous set (P) plus some additional calculations with B and Q. Moreover, as we have to remove the observations in B from the two AVL trees in order to obtain the new median ν_R, we can obtain the summations over B at the same time, thus adding no additional computational cost.

We will now present an algorithm[31] that given a set P and a subset B, obtains the value of the *SAD* in the set resulting from removing B from P. This is one of the steps for the evaluation of a new trial split based on a previous one, as it was mentioned before. Going back to the example in Figure 3.2 this algorithm solves the problem of obtaining the *SAD* of the right branch of the test $X_i < 160$ based on the *SAD* of the test $X_i < 145$. In this example the set B is {234, 546}. The algorithm we present below assumes that we have the values in P stored in two AVL trees P^- and P^+. Furthermore we must have the values of ν_P, $\sum_{P^+} y_i$, $\sum_{P^-} y_i$, $\#P^-$ and $\#P^+$. The algorithm returns the values of ν_R, $\sum_{R^+} y_i$, $\sum_{R^-} y_i$, $\#R^-$, $\#R^+$ plus the updated AVL trees. According to Equation 3.11 we can use these values to

[31] An algorithm with similar objectives was presented in Lubinsky (1995).

calculate SAD_{R,v_R}. To avoid over-cluttering of the algorithm we have omitted some special cases like when the AVL trees turn empty after removing the cases in *B*.

Algorithm 3.5 – Updating the median and SAD after removing a set of points B.

```
Input :
       P+, P-              % AVL's containing the elements in the current partition
       SP+, SP-            % The sum of the Y values of the cases in each AVL
       NP+, NP-            % Number of elements in each AVL
       Med                 % The current median value
       B                   % The data points to be removed
Output :
       The updated values characterising the new partition
       (i.e. updated P+, updated P-, SR+, SR-, NR+, NR-, and NewMED)

SB- = SB+ = NB- = NB+ = SQ = 0
FOR each b in B DO           % removing the B cases
       IF (b <= Med) THEN
              SB- = SB- + b
              NB- = NB- + 1
              P- = AVLremove(b,P-)
       ELSE
              SB+ = SB+ + b
              NB+ = NB+ + 1
              P+ = AVLremove(b,P+)
       END IF
END FOR
NQ = (NB+ - NB-) DIV 2       % DIV stands for integer division, e.g. 5 DIV 2 = 2
NR+ = NP+ - NB+ + NQ         % Obtaining the number of cases in the two new AVL's
NR- = NP- - NB- - NQ
IF ( NR+ > NR-) THEN         % Check consistency with 3.9
       NR+ = NR+ - 1
       NR- = NR- + 1
       NQ = NQ - 1
END IF
IF (NQ > 0) THEN             % Now moving the cases belonging to Q
       FOR i=1 TO NQ DO
              X = AVLmaximum(P-)
              P- = AVLdelete(X,P-)
              P+ = AVLinsert(X,P+)
              SQ = SQ + X
       END FOR
ELSE IF (NQ < 0) THEN
       FOR i=1 TO -NQ DO
              X = AVLminimum(P+)
              P+ = AVLdelete(X,P+)
              P- = AVLinsert(X,P-)
              SQ = SQ + X
       END FOR
END IF
IF (NR- > NR+) THEN          % Calculating the new Median
```

```
        NewMED = AVLmaximum(P⁻)
ELSE
        NewMED = (AVLmaximum(P⁻) + AVLminimum(P⁺)) / 2
END IF
SR⁺ = SP⁺ - SB⁺ + (-(NewMED ≠ MED))^(NewMED > MED)* SQ      % Calculating the SR's
SR⁻ = SP⁻ - SB⁻ + (-(NewMED ≠ MED))^(NewMED < MED)* SQ
```

Following a similar reasoning it would be possible to prove that the sum of absolute deviations of a set *A*, resulting from adding a set of points *B* to our original set *P* can be obtained by,

$$\begin{aligned} SAD_{A,\nu_A} &= \sum_{P^+} y_i - \sum_{P^-} y_i + \sum_{B^+} y_i - \sum_{B^-} y_i + \left(-\left(\nu_A \neq \nu_P\right)\right)^{\left(\nu_A > \nu_P\right)} \times 2\sum_{Q} y_i \\ &+ \nu_A \left(\#P^- - \#P^+ + \#B^- - \#B^+ + \left(-\left(\nu_A \neq \nu_P\right)\right)^{\left(\nu_A < \nu_P\right)} \times 2\#Q\right) \end{aligned} \tag{3.13}$$

This equation leads to a very similar algorithm that updates the median and *SAD* when we add a set of points. In the example mentioned before this algorithm would enable to obtain the value of the *SAD* of the left branch of the test X_i < 160 based on the *SAD* of the test X_i < 145. Due to its similarity to Algorithm 3.5 we do not present it here.

The computational complexity of these algorithms is dominated by the operations in the AVL trees with the cases in *B*. These operations can be done in time proportional to $O(\#B \log \#P/2)$. If we consider all possible splits of a continuous variable we get to an average complexity of $O(n \log n/2)$, where n is the number of observations in the node. This number results from the fact that all observations need to be moved from the right branch to the left branch when we try all possible splits. As we have seen the naïve approach has an average complexity of $O(n^3 \log n)$, which means that our algorithms provide a significant computational complexity decrease for the task of finding the best split of a continuous variable in LAD trees. However, we have seen in Section 3.2.2 that the corresponding complexity in LS regression trees is $O(n)$, which means that even with our optimisations LAD trees are more complex. Still, both type of trees need a previous sorting operation on the values of the variable that is done on average in $O(n \log n)$, which turns out to be the major computational burden.

Having solved the problem of incrementally obtaining the *SAD* of a new split based on a previous one, we are now ready to present the algorithm for obtaining the optimal LAD split for a continuous variable.

Algorithm 3.6 - Best LAD split of a continuous variable.

```
Input : n cases; their median, SAD, and respective AVL's; the variable Xv
Output : The best cut-point split on Xv

Sort the cases according to their value in Xv

Initialise Right with the Input median information (med,sad and avl's)

Set B to the empty set
BestTillNow = 0
FOR all instances i DO
       Add yi to B
       IF ( Xi+1,v > Xi,v ) THEN
              Left = AddToSet(Left,B)
              Right = RemoveFromSet(Right,B)
              RightSAD = Right.R+ - Right.R- + Right.Median * (Right.N- - Right.N+)
              LeftSAD = Left.R+ - Left.R- + Left.Median * (Left.N- - Left.N+)
              NewSplitValue = RightSAD + LeftSAD
              IF ( NewSplitValue > BestTillNow ) THEN
                     BestTillNow = NewSplitValue
                     BestCutPoint = ( Xi+1,v + Xi,v ) / 2
              ENDIF
              Set B to the empty set
       ENDIF
ENDFOR
```

Algorithm 3.6 uses the algorithms we have described before (the call *RemoveFromSet* is Algorithm 3.5, while *AddToSet* is the corresponding algorithm for adding a set). The values returned by these two algorithms enable us to calculate the *SADs* of both branches using Equation 3.11.

We have carried out an experiment with the *Fried* domain in order to confirm the validity of our proposed algorithms as a means to allow growing LAD trees within reasonable computation times. In this experiment we have varied the training sample size from 1000 to 150000 cases. For each size, we have grown a LAD tree storing the respective CPU time taken to carry out this task. The results are shown in Figure 3.3:

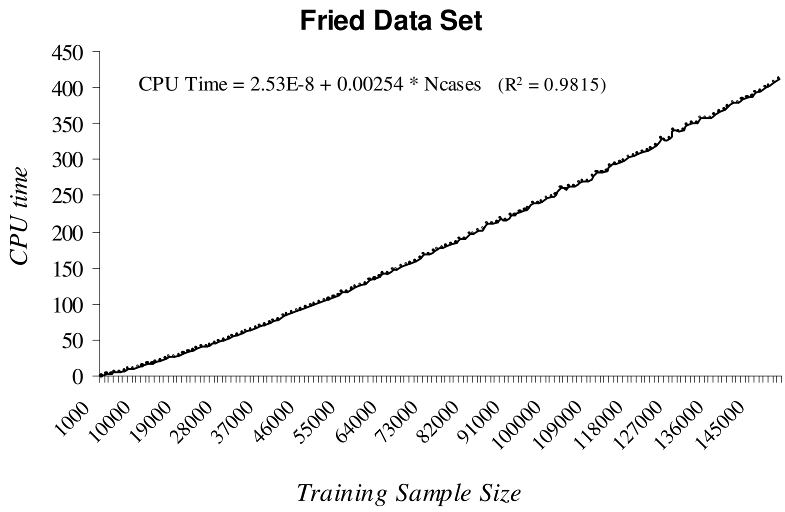


Figure 3.3 - Computation time to grow a LAD tree for different sample sizes of the Fried domain.

This experiment shows that although LAD trees are computationally more demanding than LS trees (*c.f.* Figure 3.1), they still maintain an almost linear dependence of the CPU time with respect to the training sample size. This behaviour clearly confirms that our algorithms are able to evaluate all possible splits of continuous variables in a computationally efficient way.

3.3.2 Splits on Discrete Variables

The best nominal split of the form $X_v \in \{x_v, ...\}$ can be efficiently obtained in LS trees due to a theorem proved by Breiman *et al.* (1984). This theorem reduces the number of tried splits from $2^{\#\chi_v - 1}$ to $\#\chi_v - 1$. Even for a small number of values of the variable this reduction is significant. This means that the question whether this theorem also applies to the LAD error criterion is a key issue in terms of computational efficiency.

According to Definition 3.4 the value of a split is given by the sum of the *SADs* of the left and right branches. A theorem equivalent to the one proved by Breiman *et. al.* (1984) for the LAD criterion would mean that the following hypothesis is true :

HYPOTHESIS 3.1

If P_1, P_2 form a partition resulting from the best split in a discrete variable (*i.e.* P_1, P_2 is an optimal partition) and $\nu_{P_1} < \nu_{P_2}$ then

$$\forall_{B_1 \subset P_1, B_2 \subset P_2} : \quad \nu_{B_1} < \nu_{B_2}$$

♦

If this hypothesis is true, to obtain the best discrete split it would be sufficient to order the values of the discrete variable by their median and the best split is guaranteed to be between two of these ordered values. This is exactly the same procedure followed in Example 3.2 presented in Section 3.2.3, but now with medians instead of averages.

To prove Hypothesis 3.1 it is sufficient to demonstrate that if we exchange any two subsets of the optimal partition we get a worse value of the split. In particular if ψ_{P_1,P_2} is the sum of the *SADs* of the optimal partition (*i.e.* $\psi_{P_1,P_2} = SAD_{P_1} + SAD_{P_2}$), $B_1 = \max\limits_{p_i \in P_1} \nu_{p_i}$ and $B_2 = \min\limits_{p_i \in P_2} \nu_{p_i}$, then if we exchange B_1 with B_2, originating the partition N_1, N_2, we should be able to prove that $\psi_{P_1,P_2} \leq \psi_{N_1,N_2}$. Notice that,

$$N_1 = P_1 - B_1 + B_2 \qquad \text{and} \qquad N_2 = P_2 - B_2 + B_1$$

For instance, let $X_1 \in \{a, b, c\}$ be an optimal split for the variable X_1, with the domain of the variable being $\chi_1 = \{a, b, c, d, e\}$. Let P_1 and P_2 be the sets containing the respective Y values associated with the division entailed by the split. Furthermore, let us suppose[32] that $\nu_a > \nu_c > \nu_b$, and $\nu_e < \nu_d$. If B_1 is the subset of P_1 containing the Y values of the cases for which $X_1 = a$, and B_2 is the subset of P_2 containing the Y values of the cases for which $X_1 = e$, we want to prove that $SAD_{\{b,c,e\}} + SAD_{\{a,d\}} \leq SAD_{\{a,b,c\}} + SAD_{\{d,e\}}$ (*i.e.* that $\psi_{\{b,c,e\},\{a,d\}} \leq \psi_{\{a,b,c\},\{d,e\}}$).

---

[32] ν_i represents the median of the Y values of the cases for which the value of the variable is i.

The derivation of equations for the *SAD* of the sets N_1 and N_2 follows a similar reasoning used for deriving Equations 3.12 and 3.13. However, the expressions are a bit more complex. Our goal is to find an expression for ψ_{N_1,N_2} as a function of ψ_{P_1,P_2}. Namely, we want an expression of the form

$$\psi_{N_1,N_2} = SAD_{N_1} + SAD_{N_2} = SAD_{P_1} + K_1 + SAD_{P_2} + K_2 \tag{3.14}$$

If we manage to obtain expressions for K_1 and K_2 and to prove that their sum is greater or equal to zero, we would be able to obtain a demonstration of Hypothesis 3.1. At the end of this chapter we present a derivation of such an expression for ψ_{N_1,N_2} . However, the expressions we were able to derive for K_1 and K_2 are still too complex for a clear understanding of the behaviour of ψ_{N_1,N_2} . Still, we were able to prove the falsity of Hypothesis 3.1 by finding a counter-example through a large-scale simulation study. This means that its use may lead to the choice of sub-optimal discrete splits for LAD tree nodes. The question that arises is whether the predictive accuracy of LAD trees is affected by these potentially sub-optimal splits. In effect, although we were not able to formally characterise the cases where sub-optimality occurs, we have experimentally observed that these are rare events. In most of our simulation experiments using Hypothesis 3.1 leads to the optimal split. Moreover, if the split is sub-optimal it does not means that the resulting LAD tree will have lower performance in a separate independent test set. We have implemented in our RT system both alternative ways of finding the best nominal split: using Hypothesis 3.1; or trying all possible combinations. Regression data sets with lots of nominal variables do not abound. In our benchmark data sets only *Abalone* and *Mv* include nominal variables. In all experiments we have carried out with these two domains, we never observed any difference in terms of accuracy or tree size[33], between the two alternatives. On the contrary, in terms of computation time there is an overwhelming advantage of using the heuristic method of finding best nominal variable splits outlined by Hypothesis 3.1. We have carried out an experiment with the *Mv* artificial domain to

---

[33] Actually the trees were always the same, meaning that the use of the Hypothesis in these data sets is not leading to sub-optimal splits.

confirm this advantage. We have varied the training sample size from 1000 to 150000 cases, generating two LAD trees. The first (LAD) was obtained by exploring all possible combinations of the discrete variable values, when addressing the task of finding the best sub-set split of a tree node. The latter (LAD, fast) was generated through the use of the results of Hypothesis 3.1 as a heuristic process of finding the best discrete split of a node. The computation time taken to grow these two trees is shown in Figure 3.4:

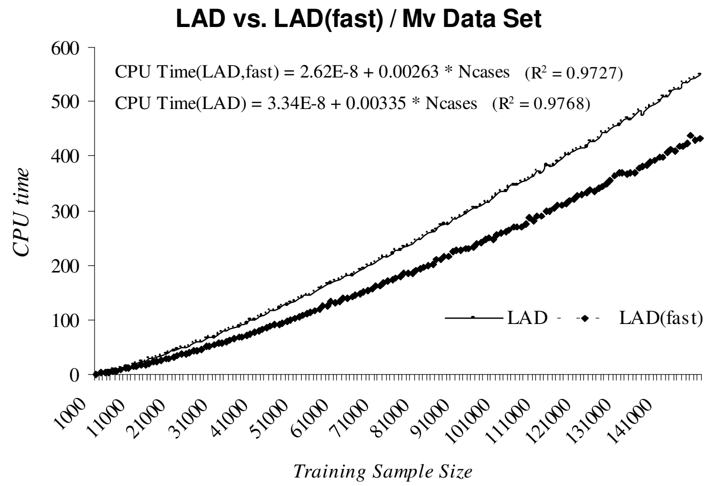


Figure 3.4 - Computation Time of LAD and LAD(fast) trees for different sample sizes.

This experiment confirms the clear computational advantage of using our proposed heuristic process of finding the best discrete split for the nodes of LAD trees. Moreover, we should remark that the three discrete variables of the *Mv* domain, have very few values (two or three). This means that the cost of evaluating all possible discrete splits is not too large when compared to our heuristic process. In domains containing discrete variables with a large set of values (*e.g.* Costa, 1996) the advantage of our proposal would be even more evident. Still, we should remark that both alternatives have a nearly linear behaviour with respect to the relation between CPU time and number of training cases.

3.4 LAD vs. LS Regression Trees

In the previous sections we have described two methods of growing regression trees. These alternatives differ in the criteria used to select the best split for each tree node. While LAD trees try to ensure that the resulting model has the smallest possible deviation from the true goal variable values, LS trees try to minimise the squared differences between the predicted and true values. In this section we address the question of which type of trees should we prefer given a new regression task.

The answer to this question is related to the predictive performance measure that will be used to evaluate the resulting regression models. There are two major groups of prediction error statistics (*c.f.* Section 2.3.2): one based on absolute differences between the predicted values and the true Y values of the cases; and the other based on the squared differences between these values. These two forms of quantifying the prediction error of a model entail different preference biases. In effect, the squared differences are more "influenced" by large prediction errors than the absolute differences. As such, any regression model that tries to minimise the squared error will be strongly influenced by this type of errors and will try to avoid them. This is the case of LS regression trees whose splitting criterion revolves around the minimisation of the resulting mean squared error (*c.f.* Definition 3.1). On the contrary, the minimisation of the absolute differences will result in a model whose predictions will on average be "nearer" the Y value of the training cases. This model is not so influenced by large prediction errors as they are not so "amplified" as when using squared differences. Models obtained by minimising the absolute differences are expected to have lower average difference between the true and predicted values. However, these models will probably commit extreme errors more often than models built around the minimisation of the squared differences. This is the case of LAD trees that use a splitting criterion that tries to minimise the absolute differences (*c.f.* Definition 3.4). From this theoretical perspective we should prefer LAD regression trees if we will evaluate the predictions of the resulting model using the Mean Absolute Deviation (MAD) statistic. On the contrary, if we are evaluating the prediction error using the Mean

Squared Error (MSE) statistic, we should prefer LS regression trees. So, given a new regression problem, which statistic should we use? In applications where the ability to avoid large errors is crucial (for instance due to economical reasons), achieving a lower MSE is preferable as it penalises this type of errors, and thus LS trees are more adequate. In applications where making a few of such extreme errors is not as crucial as being most of the times near the true value, the MAD statistic is more representative, and thus we should theoretically prefer LAD trees.

We should remark that both types of trees are built using estimates of both the MAD and MSE prediction error statistics. Any statistical estimator is prone to error. Thus, if either our estimators are unreliable or our training sample is not representative, the expected theoretical behaviour described above can be misleading. In these cases, we could see a LS tree outperforming a LAD tree in terms of MAD, or a LAD tree outperforming a LS tree in terms of MSE. Still, as we will see by a series of example applications, this is not the more frequent case.

The first application we describe concerns the environmental problem of determining the state of rivers and streams by monitoring and analysing certain measurable chemical concentrations with the goal of inferring the biological state of the river, namely the density of algae communities[34]. This study is motivated by an increasing concern as to what impact human activities have on the environment. Identifying the key chemical control variables that influence the biological process associated with these algae has become a crucial sub-task in the process of reducing the impact of man activities. The data used in this application comes from such a study. Water quality samples were collected from various European rivers during one year and an analysis was carried out to detect

---

[34] This application was used in the 3rd International Competition (http://www.erudit.de/erudit/activities/ic-99/) organised by ERUDIT in conjunction with the new Computational Intelligence and Learning Cluster (http://www.dcs.napier.ac.uk/coil/). This cluster is a cooperation between four EC-funded Networks of Excellence : ERUDIT, EvoNet, MLnet and NEuroNet. The regression system implementing the ideas in this thesis (RT) was declared one of the runner-up winners by the international jury of this competition.

various chemical substances. At the same time, algae samples were collected to determine the distributions of the algae populations. The dynamics of algae communities is strongly influenced by the external chemical environment. Determining which chemical factors are influencing this dynamics represents important knowledge that can be used to control these populations. At the same time there is also an economical factor motivating this analysis. In effect, the chemical analysis is cheap and can be easily automated. On the contrary, the biological part involves microscopic examination, requires trained manpower and is therefore both expensive and slow. The competition task consisted of predicting the frequency distribution of seven different algae on the basis of eight measured concentrations of chemical substances plus some additional information characterising the environment from which the sample was taken (season, river size and flow velocity).

The first regression problem we analyse concerns the task of predicting the frequency distribution of one of the algae (*Alga 6*). Using the 200 available training cases we have grown a LS regression tree. The resulting model is shown in Figure 3.5. If we test this tree on the available testing set consisting of 140 test cases we get a Mean Squared Error (MSE) of 162.286. In alternative, if we evaluate the same model using the Mean Absolute Deviation (MAD) of the model predictions we obtain a score of 7.328. This latter score is more intuitive in the sense that it is measured using the same units as the goal variable. This means that the induced tree (shown in Figure 3.5) makes on average an error of 7.328 in guessing the distribution frequency of the alga.

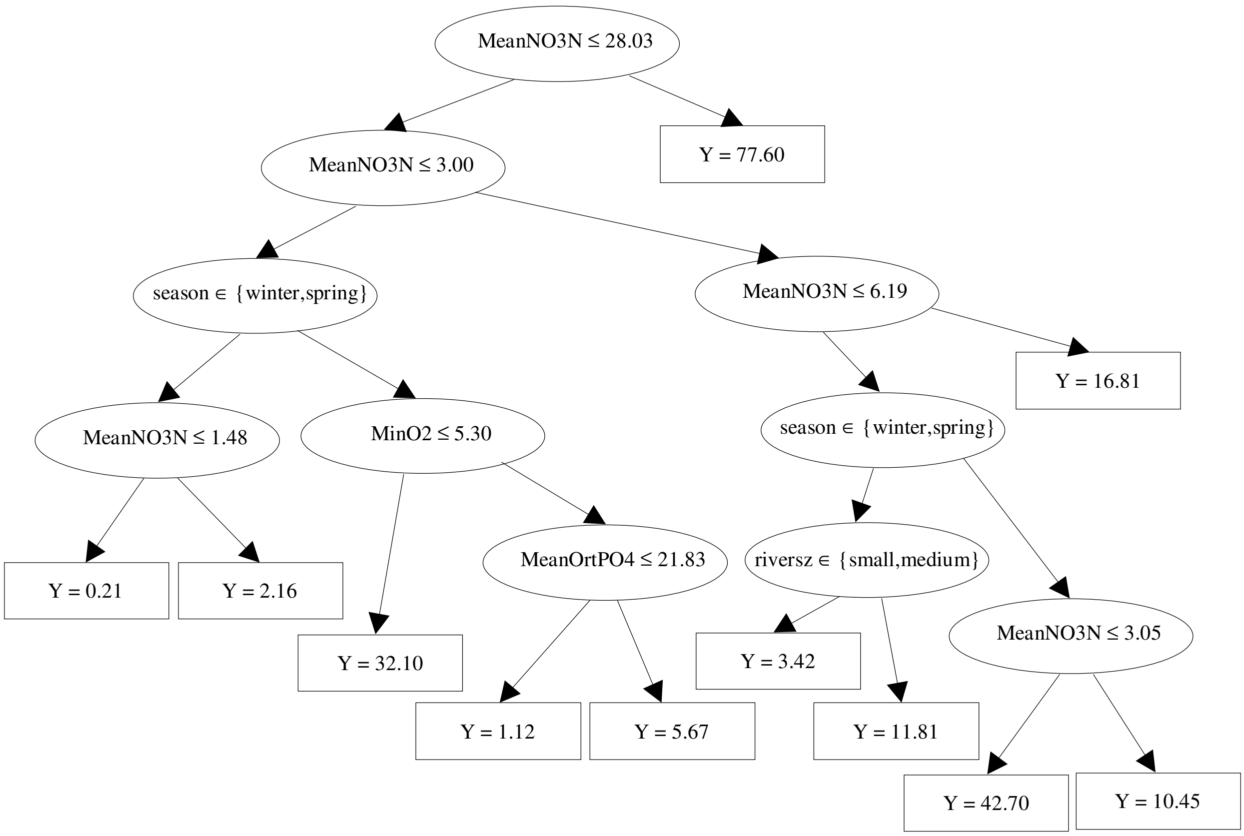


Figure 3.5 - A LS regression tree for the Alga 6 problem[35].
(MSE = 162.286; MAD = 7.328)

Using the same training data we have also grown a LAD regression tree. The resulting model is shown in Figure 3.6. The MSE of this tree on the same testing set is 179.244, which is worse than the MSE of the LS tree shown earlier (MSE = 162.286). However, if we evaluate this LAD tree using the MAD statistic we obtain a score of 6.146, which is better than the MAD of the LS tree (MAD = 7.328). With respect to the training times, both trees are obtained with little computation time due to the small size of the training

[35] Left branches correspond to cases were the node test is true, while right branches correspond to the opposite.

sample. Regarding comprehensibility, both models are acceptable although the LAD tree is considerably smaller.

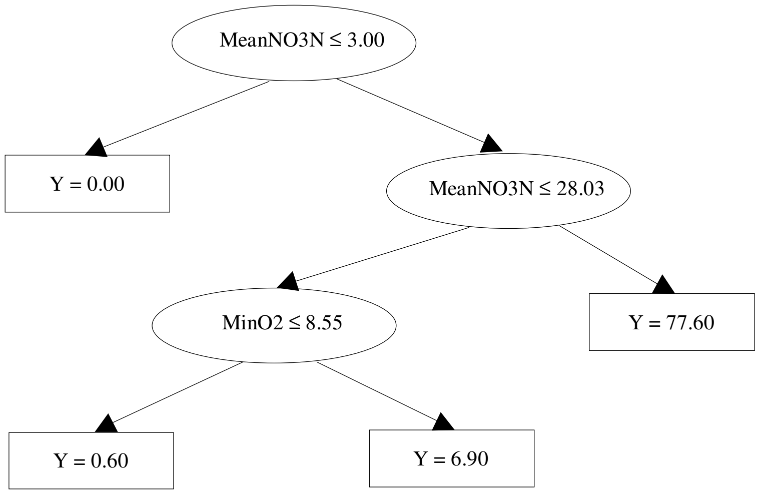


Figure 3.6 - A LAD regression tree for the Alga 6 problem.
(MSE = 179.244; MAD = 6.146)

This example clearly illustrates our previous position concerning which type of model is better. This question depends on the goals of the application. In effect, if one is willing to accept a few exceptionally large errors but give more weight to a model that on average leads to predictions that are nearer the true frequency distribution of the alga, then we should prefer the LAD tree. On the contrary, if we consider that extreme errors are inadmissible because, for instance, they could lead to an environmental disaster, then we should definitely use the LS model. To support these arguments we show in Figure 3.7 the absolute difference between the predicted and true values for both trees on all 140 test cases. As it can be confirmed, the LAD tree makes several very large errors.

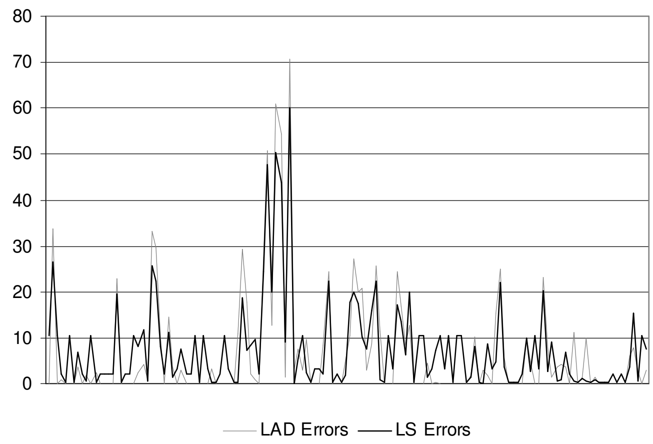


Figure 3.7 - The absolute difference of the errors committed by the LAD and LS trees.

A closer inspection of the distribution of the error size committed by the two models is given in Figure 3.8. This histogram confirms that the LAD tree errors are more often nearer the true value (in 108 of the 140 test cases the error is less than 10) than those of the LS model. In effect, looking at the first two error bins that can be seen as the best scores of both models, we observe that the frequency of errors is more balanced in the case of the LS tree, while the LAD tree is clearly skewed into the bin of smallest errors. Moreover, this histogram also confirms that the LAD tree makes more extreme errors than the LS tree.

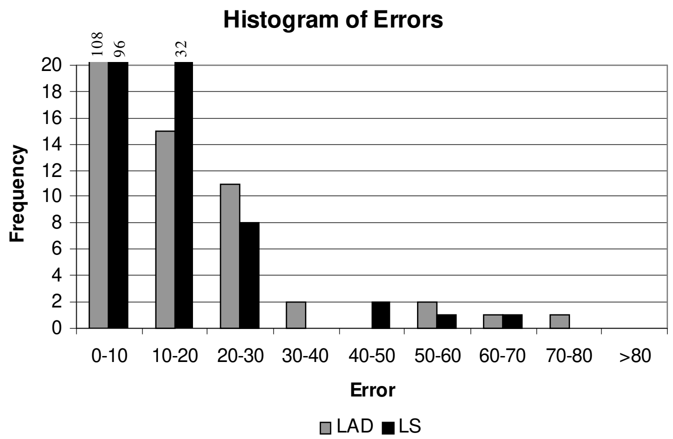


Figure 3.8 - The histogram of the errors of the LAD and LS trees.

We have repeated similar experiments with other data sets and we have observed a similar behaviour. Table 3.1 summarises the results of these experiments in other domains: *Alga 2* of the same competition data; the *Abalone* data set; and the *Pole* domain. Further details concerning these problems can be found in Appendix A.2.

Table 3.1. Results of comparisons between LAD and LS trees.

	Alga 2 200 train cases/140 test cases		*Abalone* 3133 / 1044		*Pole* 5000 / 4065	
	LAD	*LS*	*LAD*	*LS*	*LAD*	*LS*
MSE	109.347	**92.431**	5.894	**5.094**	119.342	**41.612**
MAD	**6.232**	6.422	**1.632**	1.639	**2.889**	2.984
N. Leaves	13	5	51	39	119	97
Error Bins Frequencies						
1^{st} bin 2^{nd} bin	**117** **6**	102 2	**658** 162	452 **280**	**1782** 38	1772 **91**
Last but one Last bin	**2** **1**	0 0	**3** **1**	1 0	**11** **3**	1 0

We observe that depending on the criteria used to evaluate the models (MSE or MAD), either the LAD or LS trees achieve the best score. Moreover, the last lines of the table, showing the frequency distribution of the first two error bins containing the smallest errors and the last two error bins containing the largest errors, confirm a similar error distribution exists as the one shown in Figure 3.8.

The experiments described in this section lead to the following conclusions regarding the applicability of both LS and LAD regression trees. LAD trees are, on average, more accurate than LS trees, although they also commit extreme errors more often. LS trees, on the other hand, are less prone to large prediction errors, while achieving less accurate predictions than LAD trees, on average. Both types of trees are comparable in terms of comprehensibility of the models, but LAD trees are considerably more demanding in terms of computation time. Still, this latter observation can only be regarded as relevant for extremely large training samples.

3.5 Conclusions

In this chapter we have addressed the issue of growing regression trees. We have described in detail two alternative methods of tree induction: one based on the least squares (LS) criterion and the other on the least absolute deviation (LAD) criterion.

Least squares (LS) regression trees had already been described in detail in the book by Breiman *et. al.* (1984). Compared to this work we have presented some simplifications of the splitting criterion that lead to gains in computational efficiency. With these simplifications the task of growing a tree is carried out in computation times that are practically linear with respect to the training sample size.

With respect to LAD regression trees we have presented a detailed description of this methodology. These trees can be considered more adequate to certain types of applications. However, they bring additional computational difficulties to the task of finding the best split of a node. We have presented algorithms that overcome these difficulties for numeric variables, as confirmed by our experiments. With respect to nominal variables we have shown that the theorem proved by Breiman *et. al.* (1984) for subset splits in LS trees does not hold for the LAD error criterion. Still, we have experimentally observed that the use of a heuristic based on the "theorem" does not entail any significant loss in predictive accuracy. Moreover, using this heuristic to find the best discrete split brings very significant gains in computation time as we have observed through a large experiment with a domain containing discrete variables.

We have shown through a set of experiments that both types of trees can be useful depending on the application goals. LAD trees were found to be more accurate on average, while being more susceptible to make large errors. If a few of these errors do not present a problem in the application under consideration then these trees are clearly preferable to LS trees. On the contrary, if extreme errors are unacceptable then LS trees should be the choice. Moreover, these latter trees are obtained in a considerably smaller computation time.

3.5.1 Open Research Issues

A simple formal proof of the falsity of Hypothesis 3.1 would also be useful. This could provide safe indications of when we should or should not use the hypothesis to find the best subset split of a nominal variable in LAD trees.

A further comparative study between LAD and LS trees would be desirable. As Breiman *et. al.* (1984, p.262) mentioned, it is difficult to decide which tree is best. If we use as measure of accuracy on unseen cases the mean squared error (MSE), LS trees will usually have better score as they are grown to minimise this error. If we use instead the mean absolute deviation (MAD) the opposite occurs because LAD trees minimise absolute errors. Apart from extended experimental comparisons a theoretical study of the properties of these two types of trees would certainly help to decide which type of model to use in a new application.

APPENDIX.

PROOF OF THEOREM 3.1.

If Y is a continuous random variable with probability density function $f(y)$, the function that we want to minimise with respect to k is,

$$\phi(k) = E\left[(Y-k)^2\right] = \int_{-\infty}^{+\infty}(y-k)^2 f(y)dy \qquad \text{, as } E[Y] = \int_{-\infty}^{+\infty} y f(y)dy$$

$$= \int_{-\infty}^{+\infty}\left(y^2 - 2yk + k^2\right)f(y)dy$$

$$= \int_{-\infty}^{+\infty} y^2 f(y)dy \; - \; 2k\int_{-\infty}^{+\infty} y f(y)dy \; + \; k^2 \qquad \text{, as } \int_{-\infty}^{+\infty} f(y)dy = 1$$

Minimising with respect to k we have,

$$\frac{\partial}{\partial k}\phi(k) = 0 \quad \Leftrightarrow \quad 0 - 2\int_{-\infty}^{+\infty} y f(y)dy \; + \; 2k = 0 \quad \Leftrightarrow \quad k = \int_{-\infty}^{+\infty} y f(y)dy$$

which by definition is $E[Y]$, *i.e.* the mean value of the variable Y.

♦

PROOF OF THEOREM 3.2.

The function we want to minimise with respect to k is,

$$\phi(k) = E\left(|y\text{-}k|\right) = \int_{-\infty}^{+\infty} |y-k| \, f(y)dy$$

$$= \int_{-\infty}^{k} (k-y)f(y)dy \; + \; \int_{k}^{+\infty} (y-k)f(y)dy$$

$$= k\int_{-\infty}^{k} f(y)dy \; - \; \int_{-\infty}^{k} y f(y)dy \; + \; \int_{k}^{+\infty} y f(y)dy \; - \; k\int_{k}^{+\infty} f(y)dy$$

, as $\int_{k}^{+\infty} f(y)dy = 1 - \int_{-\infty}^{k} f(y)dy$.

So, we have

$$\phi(k) = k\int_{-\infty}^{k} f(y)dy \quad - \quad k \quad + \quad k\int_{-\infty}^{k} f(y)dy \quad - \quad \int_{-\infty}^{k} y\,f(y)dy \quad + \quad \int_{k}^{+\infty} y\,f(y)dy$$
$$= 2k\int_{-\infty}^{k} f(y)dy \quad - \quad k \quad - \quad \int_{-\infty}^{k} y\,f(y)dy \quad + \quad \int_{k}^{+\infty} y\,f(y)dy$$
$$= 2k\,F(k) \quad - \quad k \quad - \quad \int_{-\infty}^{k} y\,f(y)dy \quad + \quad \int_{k}^{+\infty} y\,f(y)dy$$

where, $F(y)$ is the cumulative distribution function of the variable Y.

Now, obtaining the derivative of this function in order to k, and making it equal to zero we get,

$$\frac{\partial}{\partial k}\phi(k) = 2F(k) + 2k\,f(k) - 1 - k\,f(k) - k\,f(k) = 2F(k) - 1$$

so, $\frac{\partial}{\partial k}\phi(k) = 0 \quad \Leftrightarrow \quad F(k) = \frac{1}{2}$

As by definition the cumulative distribution function is equal to ½ for the median of any distribution, the proof is complete.

♦

TENTATIVE PROOF OF HYPOTHESIS 3.1.

To prove Hypothesis 3.1 it is sufficient to demonstrate that if we exchange any two subsets of the optimal partition we get a worse value of the split. In particular if ψ_{P_1,P_2} is the sum of the *SADs* of the optimal partition (*i.e.* $\psi_{P_1,P_2} = SAD_{P_1} + SAD_{P_2}$), $B_1 = \max_{p_i \in P_1} \nu_{p_i}$ and $B_2 = \min_{p_i \in P_2} \nu_{p_i}$, then if we exchange B_1 with B_2, originating the partition N_1, N_2, we should be able to prove that $\psi_{P_1,P_2} \leq \psi_{N_1,N_2}$. Notice that,

$$N_1 = P_1 - B_1 + B_2 \qquad \text{and} \qquad N_2 = P_2 - B_2 + B_1$$

In this appendix we derive an expression for ψ_{N_1,N_2} based on the *SADs* of P_1 and P_2. The derived expression as the form

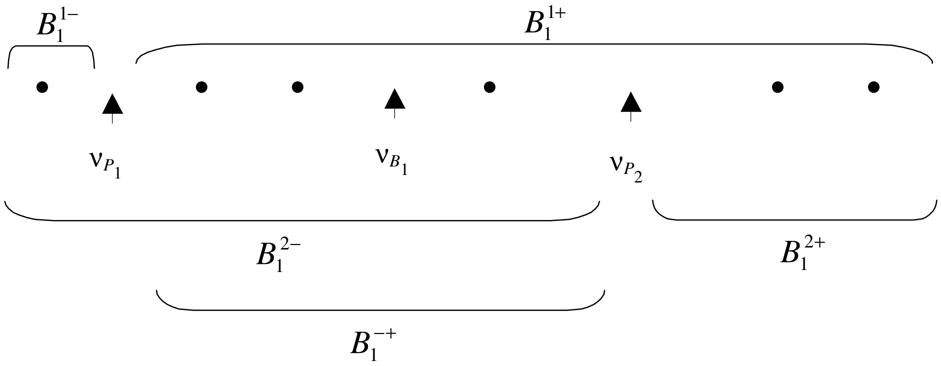
$$\psi_{N_1,N_2} = SAD_{P_1} + K_1 + SAD_{P_2} + K_2$$

Understanding the sign of $K_1 + K_2$, is a fundamental step for the proof of Hypothesis 3.1.

The *SAD* of the set N_1 consisting of removing a set B_1 from a set P_1 and adding to the same set the set B_2, is given by

$$\begin{aligned} SAD_{N_1,\nu_{N_1}} &= \sum_{P_1^+} y_i - \sum_{P_1^-} y_i + \sum_{B_1^-} y_i - \sum_{B_1^+} y_i + \sum_{B_2^+} y_i - \sum_{B_2^-} y_i \\ &+ \left(-\left(\nu_{N_1} \neq \nu_{P_1}\right)\right)^{\left(\nu_{N_1} > \nu_{P_1}\right)} \times 2\sum_{Q_1} y_i + \nu_{N_1}\left(\# N_1^- - \# N_1^+\right) \end{aligned} \tag{3.15}$$

We omit the derivation of this equation, as it is similar to the derivation of Equations 3.12 and 3.13. As we need to obtain an expression for the sum of the two *SADs* (Equation 3.14), it is necessary to introduce notation for differentiating numbers bigger (smaller) than ν_{P_1} from the ones bigger (smaller) than ν_{P_2}. The following figure illustrates this problem for the set B_1 and the adopted notation.



The same kind of notation can be used to describe the relation of set B_2 with both medians. Using Equation 3.15 and the notation presented above, we can derive an expression for our target function ψ_{N_1,N_2},

$$
\begin{aligned}
\Psi_{N_1,N_2} &= SAD_{N_1,v_{N_1}} + SAD_{N_2,v_{N_2}} = \\
&\sum_{P_1^{1+}} y_i - \sum_{P_1^{1-}} y_i + \sum_{B_1^{1-}} y_i - \sum_{B_1^{1+}} y_i + \sum_{B_2^{1+}} y_i - \sum_{B_2^{1-}} y_i \\
&+ \left(-\left(\nu_{N_1} \neq \nu_{P_1}\right)\right)^{\left(\nu_{N_1} > \nu_{P_1}\right)} \times 2\sum_{Q_1} y_i + \nu_{N_1}\left(\#N_1^- - \#N_1^+\right) \\
&+ \sum_{P_2^{2+}} y_i - \sum_{P_2^{2-}} y_i + \sum_{B_2^{2-}} y_i - \sum_{B_2^{2+}} y_i + \sum_{B_1^{2+}} y_i - \sum_{B_1^{2-}} y_i \\
&+ \left(-\left(\nu_{N_2} \neq \nu_{P_2}\right)\right)^{\left(\nu_{N_2} > \nu_{P_2}\right)} \times 2\sum_{Q_2} y_i + \nu_{N_2}\left(\#N_2^- - \#N_2^+\right)
\end{aligned}
$$

as it may be confirmed in the figure presented above the following holds,

$\sum_{B_1^{1-}} y_i - \sum_{B_1^{2-}} y_i = -\sum_{B_1^{-+}} y_i$ and $\sum_{B_1^{2+}} y_i - \sum_{B_1^{1+}} y_i = -\sum_{B_1^{-+}} y_i$

$\sum_{B_{21}^{1+}} y_i - \sum_{B_2^{2+}} y_i = \sum_{B_2^{-+}} y_i$ and $\sum_{B_2^{2-}} y_i - \sum_{B_2^{1-}} y_i = \sum_{B_2^{-+}} y_i$

which leads to,

$$
\begin{aligned}
\Psi_{N_1,N_2} &= \sum_{P_1^{1+}} y_i - \sum_{P_1^{1-}} y_i + \sum_{P_2^{2+}} y_i - \sum_{P_2^{2-}} y_i + \\
&+ 2\times\left(\sum_{B_2^{-+}} y_i - \sum_{B_1^{-+}} y_i\right) + \\
&+ 2\times\left(\left(-\left(\nu_{N_1} \neq \nu_{P_1}\right)\right)^{\left(\nu_{N_1} > \nu_{P_1}\right)} \times \sum_{Q_1} y_i + \left(-\left(\nu_{N_2} \neq \nu_{P_2}\right)\right)^{\left(\nu_{N_2} > \nu_{P_2}\right)} \times \sum_{Q_2} y_i\right) + \\
&+ \nu_{N_1}\left(\#N_1^- - \#N_1^+\right) + \nu_{N_2}\left(\#N_2^- - \#N_2^+\right)
\end{aligned}
$$

We know that[36]

$$
\Psi_{P_1,P_2} = \sum_{P_1^{1+}} y_i - \sum_{P_1^{1-}} y_i + \nu_{P_1} \times ODD(\#P_1) + \sum_{P_2^{2+}} y_i - \sum_{P_2^{2-}} y_i + \nu_{P_2} \times ODD(\#P_2)
$$

$$
\Leftrightarrow \quad \sum_{P_1^{1+}} y_i - \sum_{P_1^{1-}} y_i + \sum_{P_2^{2+}} y_i - \sum_{P_2^{2-}} y_i = \Psi_{P_1,P_2} - \nu_{P_1} \times ODD(\#P_1) - \nu_{P_2} \times ODD(\#P_2)
$$

where,

$$
ODD(i) = \begin{cases} 1 & \text{if } i \text{ is odd} \\ 0 & \text{otherwise} \end{cases}
$$

[36] Notice that due to the restrictions presented in Equation 3.9, $ODD(\#P) = \#P^- - \#P^+$.

So, finally we get

$$\begin{aligned}\Psi_{N_1,N_2} &= \Psi_{P_1,P_2} + \\ &+ 2\times\left(\sum_{B_2^{-+}} y_i - \sum_{B_1^{-+}} y_i\right) + \\ &+ 2\times\left(\left(-\left(\nu_{N_1} \neq \nu_{P_1}\right)\right)^{\left(\nu_{N_1} > \nu_{P_1}\right)} \times \sum_{Q_1} y_i + \left(-\left(\nu_{N_2} \neq \nu_{P_2}\right)\right)^{\left(\nu_{N_2} > \nu_{P_2}\right)} \times \sum_{Q_2} y_i\right) + \\ &+ \nu_{N_1} ODD(\# N_1) - \nu_{P_1} ODD(\# P_1) + \nu_{N_2} ODD(\# N_2) - \nu_{P_2} ODD(\# P_2)\end{aligned} \tag{3.16}$$

This means that to prove Hypothesis 3.1 we need to prove that

$$\begin{aligned}&2\times\left(\sum_{B_2^{-+}} y_i - \sum_{B_1^{-+}} y_i\right) + \\ &+ 2\times\left(\left(-\left(\nu_{N_1} \neq \nu_{P_1}\right)\right)^{\left(\nu_{N_1} > \nu_{P_1}\right)} \times \sum_{Q_1} y_i + \left(-\left(\nu_{N_2} \neq \nu_{P_2}\right)\right)^{\left(\nu_{N_2} > \nu_{P_2}\right)} \times \sum_{Q_2} y_i\right) + \\ &+ \nu_{N_1} ODD(\# N_1) - \nu_{P_1} ODD(\# P_1) + \nu_{N_2} ODD(\# N_2) - \nu_{P_2} ODD(\# P_2) \quad \geq \quad 0\end{aligned} \tag{3.17}$$

The analytic proof of the falsity of 3.17 is rather complex as there are too many variants depending on the relation between the medians and also the cardinalities of the sets involved (P_1, P_2, B_1 and B_2). Being so, we have decided to present a simple example that falsifies Hypothesis 3.1.

Let X_1 be a discrete variable with the following domain : χ_1 = {a, b, c, d, e}. Let us further suppose that we have the following set of cases,

X_1	Y	X_1	Y	X_1	Y	X_1	Y	X_1	Y	X_1	Y
d	-582	a	-143	c	-356	e	-594	b	-138	b	924
d	-289	a	503	c	-94	e	-280	b	98		
d	-274	a	-400	c	79	e	231	b	177		
d	-226	a	-128	c	562	e	601	b	194		
a	568	c	-995	e	-986	e	711	b	717		

This set of cases leads to the following ordering of the values, based on their respective medians,
{d, a, c, e, b} as ν_d (-281.5) < ν_a (-128) < ν_c (-94) < ν_e (-24.5) < ν_b (185.5)

Let us consider the split $X_1 \in \{d,a\}$. The value of this split is given by the sum of the two respective *SADs* and is equal to $\psi_{\{d,a\},\{c,e,b\}} = 2345 + 7481 = 9826$. According to Hypothesis 3.1, if we exchange *a* and *c* we should get a value of the split at most equal to this value, but never smaller. If we make the necessary calculations we obtain the value of $\psi_{\{d,c\},\{a,e,b\}} = 2543 + 7020 = 9563$, which proves that the hypothesis is false. This means that using Hypothesis 3.1 may lead to a sub-optimal nominal split.

Chapter 4

Overfitting Avoidance in Regression Trees

This chapter describes several approaches that try to avoid overfitting of the training data with too complex trees. In the context of tree-based models these strategies are known as pruning methods. Overfitting avoidance within tree-based models is usually achieved by growing an overly large tree and then pruning its "unreliable" branches (also known as post-pruning). Post-pruning can be regarded as a search problem, where one looks for the "best" pruned tree. The pruning techniques we present in this chapter follow the same general strategy as the one used in system CART (Breiman *et al.*,1984). These techniques proceed in two separate stages, where initially a sequence of alternative pruned trees is generated, and then a tree selection process is carried out to obtain the final model. Compared to CART pruning we describe new methods of generating sequences of trees that proved to be advantageous on our benchmark data sets. Moreover, we describe a new tree-matching procedure that extends the applicability of the cross validation selection method used in CART. We extend the use of *m* estimates (Karalic & Cestnik, 1991) by deriving the *m* estimate of the mean absolute deviation, which allows the use of these estimators with LAD trees. We also derive the standard errors of the *m* estimates of both the mean squared error and the mean absolute deviation, which allows the use of the 1-SE rule (Breiman *et al.*,1984) with these estimators. We present a new error estimator for LS regression trees based on the sampling distribution properties of the mean squared error. During a systematic experimental comparison of different methods of pruning by tree selection, this new method together with our new algorithms for generating sequences of pruned trees, proved to be among the most competitive on our benchmark data sets. Finally, we have compared our most promising pruning methods with current state-of-the-art algorithms for pruning regression trees. This comparison revealed that our methods usually lead to more accurate trees in most of our benchmark data sets. However, this advantage is usually associated with larger trees compared to some of the other algorithms. Apart from accuracy gains, one of our new pruning methods has significant advantage in terms of computation efficiency, turning it into a good choice when dealing with large data sets.

4.1 Introduction

The methods described in the previous chapter obtain a tree using an algorithm that recursively divides the given training set. As a consequence of this, the selection of the best splits is based on increasingly smaller samples as the tree grows. The split choices at the lower levels of the tree do often become statistically unreliable although the resubstitution error estimate[37] keeps decreasing. It is usually considered unlikely that this error estimate generalises to unseen cases and the tree is said to overfit the training data. This means that the tree is capturing regularities of the training sample and not of the domain from which the sample was obtained. This is usually taken as the motivation for pruning tree models. However, as Schaffer (1993a) pointed out, pruning can not be regarded as a statistical mean to improve the true prediction error. In effect, it is easy to find real world domains where pruning is actually harmful with respect to predictive

[37] The estimate obtained with the training data, which is used during tree growth.

accuracy on independent and large test samples[38]. On the contrary, as suggested by Schaffer (1993a), pruning should be regarded as a preference bias over simpler models. Understanding the biases of the different pruning methods will provide useful indications on the strategies that suit best the user's preferences.

Post-pruning is the process by which a large tree is grown and then reliable evaluation methods are used to select the "right-sized" pruned tree of this initial model. Post-pruning methods are computationally inefficient in the sense that it is not unusual to find domains where an extremely large tree with thousands of nodes is post-pruned into few hundred nodes. This clearly seems a waste of computation. An alternative consists of stopping the tree growth procedure as soon as further splitting is considered unreliable. This is sometimes known as *pre-pruning* a tree. Pre-pruning has obvious computational advantages when compared to post-pruning. In effect, we may stop the tree growth sooner, and moreover, we avoid the post-pruning process. However, this method incurs the danger of selecting a sub-optimal tree (Breiman *et al.*,1984) by stopping too soon and because of this the usual method of avoiding overfitting is post-pruning.

This chapter starts with an overview of existing techniques of pruning regression trees. We then address a particular type of pruning methodology that works by tree selection from a set of candidate alternative models. We claim that these techniques are more advantageous from an application perspective. We describe several new techniques of pruning by tree selection. Among these we remark two new methods of generating sets of pruned trees based on heuristic estimates of error reliability that we conjecture as being advantageous from a predictive accuracy perspective. We also describe a new error estimation method that we hypothesise as being competitive with resampling estimators with the advantage of being computationally less demanding.

---

[38] Empirical evidence supporting this observation is given in Section 4.4 (Figure 4.23).

4.2 An Overview of Existing Approaches

Post-pruning is the most common strategy of overfitting avoidance within tree-based models. This method consists of trying to obtain a sub-tree of the initial overly large tree, excluding its lower level branches that are *estimated* to be unreliable. As it was mentioned by Esposito *et al.* (1993) pruning can be seen as a search problem. From this perspective two main issues arise when searching for the best pruned tree. The first is the method used to explore the large space of all possible pruned trees, and the second is how to evaluate the different alternatives considered during this process. In this section we briefly describe the main existing methods of pruning regression trees.

4.2.1 Error-Complexity Pruning in CART

CART (Breiman *et al.*, 1984) prunes a large regression tree $T_{\max}$ using a two-stage algorithm called Error-Complexity[39] pruning (Breiman *et al.*,1984, p.233). This method is based on a measure of a tree called error-complexity $EC_\alpha(T)$, which is defined as,

$$EC_\alpha(T) = Err(T) + \alpha \times \#\tilde{T} \tag{4.1}$$

where,

$Err(T)$ is the resubstitution error estimate of tree T;
$\#\tilde{T}$ is the cardinality of the set $\tilde{T}$ containing the leaves of the tree T;
and α is called the complexity parameter and defines the cost of each leaf.

Depending on the cost of each additional leaf (*i.e.* the α value) different sub-trees of $T_{\max}$ minimise the error-complexity measure. Breiman and his colleagues proved that although α can run through a continuum of values there is a sequence of pruned trees such that each element is optimal for a range of α, and so there is only a finite number of "interesting" α values. Furthermore, they developed an algorithm that generates a parametric family of pruned trees $T(\alpha) = \langle T_0, T_1, \ldots, T_n \rangle$, such that each T_i in the sequence is characterised by a different value α_i. They proved that each tree T_i in this sequence is optimal from the *EC*

[39] For classification trees this algorithm is known as Minimal Cost-Complexity.

perspective within the interval [α_i ..α_{i+1}). Using this algorithm, CART generates a sequence of pruned trees by successively pruning the node *t* such that the following function is minimised,

$$g(t,T) = \frac{Err(t) - Err(T_t)}{\#\widetilde{T}_t - 1} \tag{4.2}$$

where,

T_t is the sub-tree of T rooted at node t;
and $\#\widetilde{T}_t$ is the number of leaves of this sub-tree.

The successive *g* function values form the sequence of "interesting" α values. For each of these values a new tree results as the minimising error-complexity tree. We should note that there is no theoretical justification for preferring this set of pruned trees to any other. However, Breiman and his colleagues prove that if one wishes to characterise a tree by a linear combination of its error and a cost for each of its leaves, then this sequence is optimal. By optimal it is meant that for any hypothetical cost per leaf value (α), the sub-tree of $T_{\max}$ that would minimise the expression of Equation 4.1 is included in the sequence generated by this algorithm. However, this does not mean that the sequence of trees $T(\alpha)$ includes the best possible sub-trees of $T_{\max}$ from the perspective of true prediction accuracy, as pointed out by Gelfand *et al.* (1991) and Gelfand & Delp (1991).

The second stage of the Error Complexity pruning method consists of estimating the predictive accuracy of each of the trees in the sequence $T(\alpha)$, and selecting one of the trees based on these estimates. Breiman and his colleagues suggest using a resampling strategy (either a holdout or a cross validation process) to estimate the error of each tree in the sequence. When using *k*-fold Cross Validation (CV), CART divides the given training data into *k* disjoint folds, each containing approximately the same number of observations. For each fold *v* an overly large tree $T_{\max}^{v}$ is learned using the remaining *k*-1 folds. For each of these *k* large trees CART generates a parametric family of pruned trees $T^{v}(\alpha)$, using the method mentioned earlier. Reliable estimates of the error of the trees in each of the *k* sequences are obtained using the fold that was left out of the respective training phase.

This means that for each tree in the k sequences we have an α value plus a reliable estimate of its prediction error. The goal of this CV process is to estimate the prediction error of the trees in the initial sequence $T(\alpha)$. CART obtains the error estimate of each tree $T_i \in T(\alpha)$ by a tree-matching procedure that finds its k "most similar" trees in the k sequences, and defines the error estimate of T_i as the average of the error estimates of these k trees. CART heuristically asserts similarity between trees using their α values. The main danger of this tree-matching process results from the fact that these trees with similar α value are different. Moreover, the tree $T_{\max}$ is obtained with a larger training set and this may lead to a larger set of pruned trees, $T(\alpha)$. Still, CART obtains the k most similar trees of $T_i \in T(\alpha)$ as follows: let $\alpha'_i = \sqrt{\alpha_i \alpha_{i+1}}$; define the k most similar pruned trees in the k sequences $T^v(\alpha)$ as the trees with α value most similar to α'_i. There is no theoretical justification for this heuristic tree-matching process as it was mentioned by Esposito *et al.* (1997).

The other alternative method of obtaining reliable error estimates in CART is using the holdout method. Given a training set a proportion of cases is left aside and the tree $T_{\max}$ is obtained using the remaining cases. The separate set of cases (the holdout set) is then used to obtain unbiased estimates of the prediction error of the trees in the respective sequence $T(\alpha)$.

Breiman and his colleagues describe two alternatives for the final tree selection based on the obtained error estimates. Either to select the tree with lowest estimated error or to choose the smallest tree in the sequence, whose error estimate is within the interval $\hat{Err}_b + S.E.\left(\hat{Err}_b\right)$, where $\hat{Err}_b$ is the lowest error estimate and $S.E.\left(\hat{Err}_b\right)$ is the standard error of this estimate. This latter method is usually known as the 1-SE rule, and it is known to favour simpler trees although possibly leading to lower predictive accuracy (*e.g.* Esposito *et al.*, 1997).

4.2.2 Pruning based on *m* estimates in RETIS

RETIS (Karalic & Cestnik,1991; Karalic, 1992) uses a pruning method based on the Niblett & Bratko (1986) algorithm. Contrary to CART pruning algorithm, this method proceeds in a single-step by running in a bottom-up fashion through all nodes of T_{max}. At each inner node $t \in T_{max}$ the Niblet & Bratko algorithm (*N&B*) compares the error of t and the weighed error of the sub-tree rooted at t (T_t). The weights are determined by the proportion of cases that go to each branch of t. If the error of t is less than the error of T_t the tree T_{max} is pruned at t.

One of the crucial parts of this pruning algorithm is how to obtain the error estimates. Bayesian methods can be used to obtain reliable estimates of population parameters (Good, 1965). An example of such techniques is the *m-estimator* (Cestnik, 1990). This bayesian method estimates a population parameter using the following combination between our prior and posterior knowledge,

$$m\mathrm{Est}(\theta) = \frac{n}{n+m}\zeta(\theta) + \frac{m}{n+m}\pi(\theta) \tag{4.3}$$

where,

$\zeta(\theta)$ is our *posterior* observation of the parameter (based on a size n sample);
$\pi(\theta)$ is our *prior* estimate of the parameter;
and m is a parameter of this type of estimators.

Cestnik and Bratko (1991) used this method to estimate class probabilities in the context of post-pruning classification trees using the *N&B* pruning algorithm. Karalic and Cestnik (1991) extended this framework to the case of least squares (LS) regression trees. These authors have used m-estimators to obtain reliable tree error estimates during the pruning phase. Obtaining the error of an LS tree involves calculating the mean squared error at each leaf node. The resubstitution estimates of the mean and mean squared error obtained with a sample consisting of the cases in leaf l are given by,

$$\bar{y}(D_l) = \frac{1}{n_l}\sum_{i=1}^{n_l} y_i \quad \text{and} \quad MSE_{\bar{y}}(D_l) = \frac{1}{n_l}\sum_{i=1}^{n_l}\left(y_i - \bar{y}(D_l)\right)^2 \tag{4.4}$$

where,

$D_l = \{\langle \mathbf{x}_i, y_i \rangle \in l\}$;
and $n_l = \#D_l$.

Karalic and Cestnik (1991) have derived *m*-estimates of these two statistics. Priors are usually difficult to obtain in real-world domains. The standard procedure to overcome this difficulty consists of using all training set as the source for obtaining the priors. This means that the priors for the mean and the MSE are obtained by estimating their values using all training data. Using equation 4.3 we can obtain the *m* estimate of the mean in a leaf *l* by,

$$
\begin{aligned}
m\mathrm{Est}(\bar{y}) &= \frac{n_l}{n_l+m}\zeta(\bar{y}) + \frac{m}{n_l+m}\pi(\bar{y}) = \\
&= \frac{n_l}{n_l+m}\frac{1}{n_l}\sum_{D_l} y_i + \frac{m}{n_l+m}\frac{1}{n}\sum_{D} y_i = \\
&= \frac{1}{n_l+m}\sum_{i=1}^{n_l} y_i + \frac{m}{n(n_l+m)}\sum_{i=1}^{n} y_i
\end{aligned}
\tag{4.5}
$$

and for the mean squared error,

$$
\begin{aligned}
m\mathrm{Est}(MSE_{\bar{y}}) &= \frac{n_l}{n_l+m}\frac{1}{n_l}\sum_{D_l}(y_i - m\mathrm{Est}(\bar{y}))^2 + \frac{m}{n_l+m}\frac{1}{n}\sum_{D}(y_i - m\mathrm{Est}(\bar{y}))^2 \\
&= \frac{1}{n_l+m}\sum_{i=1}^{n_l} y_i^2 + \frac{m}{n(n_l+m)}\sum_{i=1}^{n} y_i^2 + \\
&+ \frac{n_l}{n_l+m}\left((m\mathrm{Est}(\bar{y}))^2 - 2\times m\mathrm{Est}(\bar{y})\frac{1}{n_l}\sum_{i=1}^{n_l} y_i\right) + \\
&+ \frac{m}{n_l+m}\left((m\mathrm{Est}(\bar{y}))^2 - 2\times m\mathrm{Est}(\bar{y})\frac{1}{n}\sum_{i=1}^{n} y_i\right) \\
&= \frac{1}{n_l+m}\sum_{i=1}^{n_l} y_i^2 + \frac{m}{n(n_l+m)}\sum_{i=1}^{n} y_i^2 + \\
&+ (m\mathrm{Est}(\bar{y}))^2\left(\frac{n_l}{n_l+m} + \frac{m}{n_l+m}\right) - \\
&- 2\times m\mathrm{Est}(\bar{y})\left(\frac{n_l}{n_l+m}\frac{1}{n_l}\sum_{i=1}^{n_l} y_i + \frac{m}{n_l+m}\frac{1}{n}\sum_{i=1}^{n} y_i\right) \\
&= \frac{1}{n_l+m}\sum_{i=1}^{n_l} y_i^2 + \frac{m}{n(n_l+m)}\sum_{i=1}^{n} y_i^2 + (m\mathrm{Est}(\bar{y}))^2 - 2(m\mathrm{Est}(\bar{y}))^2 \\
&= \frac{1}{n_l+m}\sum_{i=1}^{n_l} y_i^2 + \frac{m}{n(n_l+m)}\sum_{i=1}^{n} y_i^2 - (m\mathrm{Est}(\bar{y}))^2
\end{aligned}
\tag{4.6}
$$

♦

From a computational point of view, obtaining the *m* estimate for the MSE in any leaf demands calculating $\sum_{i=1}^{n} y_i$ and $\sum_{i=1}^{n} y_i^2$ for the cases at the leaf and for the whole training set, besides determining n_l , n and m. These values can be easily obtained during tree growth without significant increase in the computation. Thus the computational cost of obtaining *m*-estimates for LS trees reduces to simple arithmetic calculations.

A crucial aspect of *m*-estimates is the value of the parameter *m*. Karalic & Cestnik (1991) mention that the best value of this parameter is domain dependent. However, resampling strategies can be used to automatically tune *m* by evaluating a set of alternatives and choosing the one that obtained best estimated predictive accuracy.

4.2.3 MDL-based pruning in CORE

CORE (Robnik-Sikonja,1997; Robnik-Sikonja and Kononenko, 1998) also uses the *N&B* pruning algorithm mentioned in the previous section. However, instead of comparing the error estimates of each node *t* and its sub-tree T_t, CORE uses the Minimum Description Length principle to guide the decision regarding whether or not to prune any node of a tree.

Classical coding theory (Shannon and Weaver, 1949; Rissanen and Langdon, 1981) tells us that any theory *T* about a set of data *D* can be used to encode the data as a binary string. The main idea behind the use of the Minimum Description Length (MDL) principle (Rissanen, 1982) is that "the simplest explanation of an observed phenomena is most likely to be the correct one" (Natarajan, 1991). Mitchell (1997) describes the MDL principle as a preference for the theory *Th* such that,

$$Th = \underset{Th \in TH}{\arg\min}\ L(Th) + L(D|Th) \tag{4.7}$$

where,

Th is a theory belonging to the space of theories *TH*;
D is a data set;
and *L*(.) represents the binary description length.

This formalisation shows how this theoretical framework provides a way of trading off model complexity for accuracy. In effect, according to this principle we may prefer a shorter theory that makes few errors on the training data to a large theory that perfectly fits the data. Thus this principle can be regarded as a method for guiding overfitting avoidance within inductive learning.

Robnik-Sikonja & Kononenko (1998) describe a coding schema for regression trees[40] that allows using this principle to prune the trees. This coding determines the binary code length of a tree-based model. The binary code of a regression tree consists of the code of the model and of its errors. The pruning algorithm used in CORE runs through the tree nodes using the *N&B* algorithm and at each node t compares its binary code length with the code length of its sub-tree T_t. If the latter is larger the tree $T_{\max}$ is pruned at t.

4.2.4 Pruning in M5

M5 (Quinlan, 1992; Quinlan, 1993) uses a bottom-up method similar to the *N&B* algorithm. M5 can use multivariate linear models in the tree leaves. Because of this, the pruning decision is guided by a criterion different from the ones used in either RETIS or CORE. For each node t, M5 builds a multivariate linear model using the cases in the node and including only the attributes tested in the sub-tree T_t. M5 calculates the Mean Absolute Deviation of this linear model using the cases in t. This value is then multiplied by a heuristic penalisation factor, $(n_t + v)/(n_t - v)$, where n_t is the number of cases in t, and v is the number of attributes included in the linear model. The resulting error estimate is then compared with the error estimate for the sub-tree T_t, and if the latter is larger the sub-tree is pruned.

[40] Full details on this schema can be found in an appendix at the end of this chapter.

4.3 Pruning by Tree Selection

Given an overly large tree T_{max} , the set of all sub-trees of this model is usually too large for exhaustive search even for moderate size of T_{max}. We have seen in the previous section examples of two different approaches to this search problem. The first one considers pruning as a two-stage process. In the first stage a set of pruned trees of T_{max} is generated according to some criterion, while in the second stage one of such trees is selected as the final model. This is the approach followed in CART (Breiman *et al.*,1984). The second type of pruning methods uses a single-step procedure and is more frequent. These latter algorithms run through the tree nodes either in a bottom-up or top-down fashion, deciding at each node whether to prune it according to some evaluation criterion. These two distinct forms of pruning a tree influence the evaluation methods used in the pruning process. When considering two-stage methods, the evaluation of the trees can be seen as a model selection problem, due to the fact that we want to compare alternative pruned trees with the aim of selecting the best one. On the contrary, single-step methods use evaluation at a local level, *i.e.* they need to decide at each node whether to prune it or not. Moreover, two-stage methods have an additional degree of flexibility that we claim to be relevant from the perspective of the practical use of tree-based regression. In effect, they can output the sequence[41] of alternative tree models generated in the first stage together with their evaluation (either an estimate of their prediction error or other criterion like their binary description length). These trees can be regarded as alternative models with different trade-off between model complexity and evaluation score. The system selects one of these trees according to some bias (*e.g.* the lowest estimated error), but without any additional computation cost we can allow the user to inspect and select any other tree that better suits his application needs. We think that this is a very important advantage from an application point of view and because of this the new pruning methods presented in this chapter all follow this two-stage framework.

---

[41] Or part of it as suggested by Breiman *et al.* (1984, p. 310).

Within pruning methods based on tree selection we can make a further distinction, depending on the methods used to generate the set of pruned trees. *Optimal pruning algorithms* (Breiman *et al.*,1984) produce a set of trees decreasing in size by one node, ensuring that each tree in the sequence is the tree with highest accuracy of all possible pruned trees with the same size. Breiman and his colleagues mentioned that an efficient backward dynamic programming algorithm existed but they have not provided it. Bohanec and Bratko (1994) independently developed an algorithm (OPT) also based on dynamic programming that is able to produce a sequence of optimal pruned trees. This algorithm is based on the approach suggested by Breiman *et al.* (1984, p.65). Almuallim (1996) recently presented an improvement of Bohanec and Bratko's algorithm, called OPT-2 that improves the computational efficiency of OPT. Both algorithms were designed for classification trees and domains without noise (Bohanec & Bratko, 1994). According to Bohanec & Bratko (1994) the expected gains in accuracy of optimal algorithms in noisy domains are not high when compared to non-optimal algorithms. We have re-implemented OPT-2 and confirmed this observation. For this reason we do not consider this algorithm in further comparative studies reported in this chapter.

Nested pruning algorithms generate a sequence of trees where each tree is obtained by pruning the previous element in the sequence at some node. These algorithms are obviously more efficient as their search space is smaller, which means that they may miss some good trees found by an optimal algorithm. The main difference between nested pruning algorithms is in the methods used for choosing the next node to prune.

In the following sections we describe in detail the main components of pruning by tree selection algorithms: the generation of a sequence of candidate trees; the evaluation of these candidate models; and the final selection of the tree resulting from the pruning process. Moreover, we will present our novel proposals to both tree generation and evaluation, and describe the results of an extensive experimental comparison of different alternative methods of pruning by tree selection.

4.3.1 Generating Alternative Pruned Trees

In this section we address methods for generating a sequence <T_0, T_1, …, T_n> of nested pruned trees of an overly large tree T_{max}. We describe two existing methods (*Error-Complexity* and *MEL*) and present our two proposals for this task (*MCV* and *LSS*).

The generation of a sequence of trees is the first step of pruning by tree selection. We have already seen in Section 4.2.1 that CART (Breiman *et al.*, 1984) uses an algorithm called *Error-Complexity* (*ErrCpx*) to generate a sequence of nested pruned trees. Error Complexity is an iterative algorithm that starts with the tree T_{max}, which is taken as the first element in the sequence (T_0), and generates the first pruned tree by finding the node $t \in T_{max}$ that minimises,

$$\min_t \left(\frac{Err(t) - Err(T_t)}{\#\widetilde{T}_t - 1} \right) \tag{4.8}$$

where,

T_t is the sub-tree of T rooted at node t;

and $\#\widetilde{T}_t$ is the number of leaves of this sub-tree.

The following pruned trees are obtained using the same method applied to the previous pruned tree in the sequence until a tree consisting only of the root node is reached. Finding the node t at each step involves running through all tree nodes of the current tree, which can be computationally heavy depending on the size of the trees. However, Breiman *et al.* (1984, p.293) have developed an efficient algorithm that avoids running through all tree nodes to find the node to prune at each step. This turns the Error Complexity into an efficient algorithm having an average complexity of $O\left(\#\widetilde{T} \log \#\widetilde{T}\right)$, and a worst case complexity of $O\left(\#\widetilde{T}^2\right)$ according to Bohanec & Bratko (1994).

A simpler method to generate a sequence of nested pruned trees was used in a series of comparisons carried out by Bohanec and Bratko (1994). This method consists of selecting the node t that will lead to the lowest increase of resubstitution error. This notion can be formally stated as finding the node t minimising,

$$\min_t \ \left(Err(t) - Err(T_t) \right) \tag{4.9}$$

We will refer to this method as the *Minimal Error Loss* (*MEL*) algorithm. This method is quite similar to the Error-Complexity algorithm, the single difference being that *MEL* uses a slightly different function to select the next node to prune. Due to this similarity the efficient algorithm described by Breiman and his colleagues can also be used with this method.

As we have mentioned before, one of the motivations for pruning trees is the observation that estimates based on small samples are *potentially unreliable*. By unreliability we mean that the true error of a tree node can be quite far from the value estimated with such small samples. The precision of an estimate is measured by the standard error of the estimate, as we will see in Section 4.3.2.1. This statistic measures the expected variability if more estimates were obtained using other samples of the same population. According to statistical estimation theory, a *consistent estimator* should get more *precise* (*i.e.* have lower standard error) as the sample size grows. Motivated by these considerations we propose the following method for generating a sequence of nested pruned trees. Given a tree T generate a pruned tree by eliminating the node whose error estimate is potentially least reliable. This will lead to a pruned tree of T that is optimal from the perspective of the variability of its estimated error. Now the question is how to determine the potential unreliability of the error in a node. We propose two alternative methods for quantifying the unreliability of the error estimate in a node. The first is motivated by the fact that the standard error of estimators is inversely proportional to the sample size from which the estimates were obtained. It consists of pruning, at each iteration of the algorithm that generates pruned trees, the node t minimising,

$$\min_{t} \left(n_t\right) \tag{4.10}$$

where, n_t is the training sample size in node t.

This can be seen as a naïve form of estimating the unreliability of estimates. We will call this the *Lowest Statistical Support* (*LSS*) algorithm. Apart from its simplicity this method has some computational advantages when compared to other sequence-based methods described here. In effect, the order in which the nodes will be pruned can be obtained with a single pass through the tree[42]. Pruning a particular node does not change this ordering, as the number of cases in the remaining nodes stays the same. This means that to generate a sequence of pruned trees with the *LSS* algorithm we only need to obtain a list of the nodes arranged in ascending order of sample size, and then prune each node in this ordered list to obtain the next pruned tree.

We have analysed another method of estimating the unreliability of the error estimates at each node. The standard procedure in statistics for estimating variability is to use a measure of the spread of the sample. An example of such type of measures is the *Coefficient of Variation* (*e.g.* Chatfield, 1983), which is given by,

$$CV = \frac{s_Y}{\overline{Y}} \tag{4.11}$$

where,

s_Y is the sample standard deviation of Y;
and $\overline{Y}$ is the average Y value.

Using this statistic we can compare the expected variability in the error estimates of different nodes. Having these values we can generate a sequence of nested pruned trees, by pruning at each step of the generation process, the node t with largest coefficient of variation of the mean squared error, that is,

$$\max_t \left(\frac{S.E.(MSE(t))}{MSE(t)} \right) \tag{4.12}$$

where,

MSE can be obtained by any of the estimators that will be described in Section 4.3.2.1.

We will refer to this method as the *Maximal Coefficient of Variation* (*MCV*) algorithm.

Are the four methods of generating sequences of trees (*ErrCpx*, *MEL*, *LSS* and *MCV*) significantly different from each other, *i.e.* do they entail different preference biases that

---

42 Actually, it can even be obtained during the tree growth phase.

can be considered useful for different applications? In order to try to answer this question we have carried out the following experiment. For different training samples we have generated a large tree T_{max} and four different sequences of pruned trees with each method. The goal of this experiment is to compare these sequences. For each sequence we have calculated the average true prediction error over all trees in the sequence, and the lowest true prediction error achieved by one of the trees in the sequence. As we do not know the true regression function for our benchmark domains, we have estimated the true prediction error using large independent test sets. The larger the sets, the more reliable the results of our experiment.

In this experiment we have used the following benchmark domains[43]:

Table 4.1. The basic characteristics of the used benchmark domains.

Data Set	*Basic Characteristics*
Ailerons	13750 cases; 40 continuous variables
Elevators	16559 cases; 18 cont. vars.
2Dplanes	40768 cases; 10 cont. vars.
Mv	40768 cases; 3 nominal vars.; 7 cont. vars.
Kinematics	8192 cases; 8 cont. vars.
CompAct	8192 cases; 22 cont. vars.
CompAct(s)	8192 cases; 8 cont. vars.
Census(16H)	22784 cases; 16 cont. vars.
Census(8L)	22784 cases; 8 cont. vars.
Fried	40768 cases; 10 cont. vars.
Pole	9065 cases; 48 cont. vars.

Ailerons and *Elevators* are two domains with data collected from a control problem, namely flying a F16 aircraft. *2Dplanes* is an artificial domain described in Breiman *et al.* (1984, p.238). *Mv* is an artificial domain containing several variables that are highly correlated. *Kinematics* is concerned with the forward kinematics of an 8 link robot arm. The *CompAct* domains deal with predicting CPU times from records of computer activity in a multi-user university department. The two domains differ in the attributes used to describe the cases. The *Census* domains were designed on the basis of data provided by US

---

[43] Full details of the benchmark domains used throughout the thesis can be found in Appendix A.2.

Census Bureau (1990 US census). The data sets are concerned with predicting the median price of houses in a region based on demographic composition and a state of housing market in the region. They differ in the kind of indicators (variables) used to described the cases. The *Fried* domain is an artificial data set used in Friedman (1990). Finally, the *Pole* domain contains data from a telecommunications problem and was used in a work by Weiss & Indurkhya (1995).

For each of the domains we have repeated the experiment 50 times for different training sample sizes. The results presented are averages of these 50 random samples for each size. Figure 4.1 shows the results of the four methods in terms of lowest "true" error achieved by one of the trees in each sequence, for different training sample sizes.

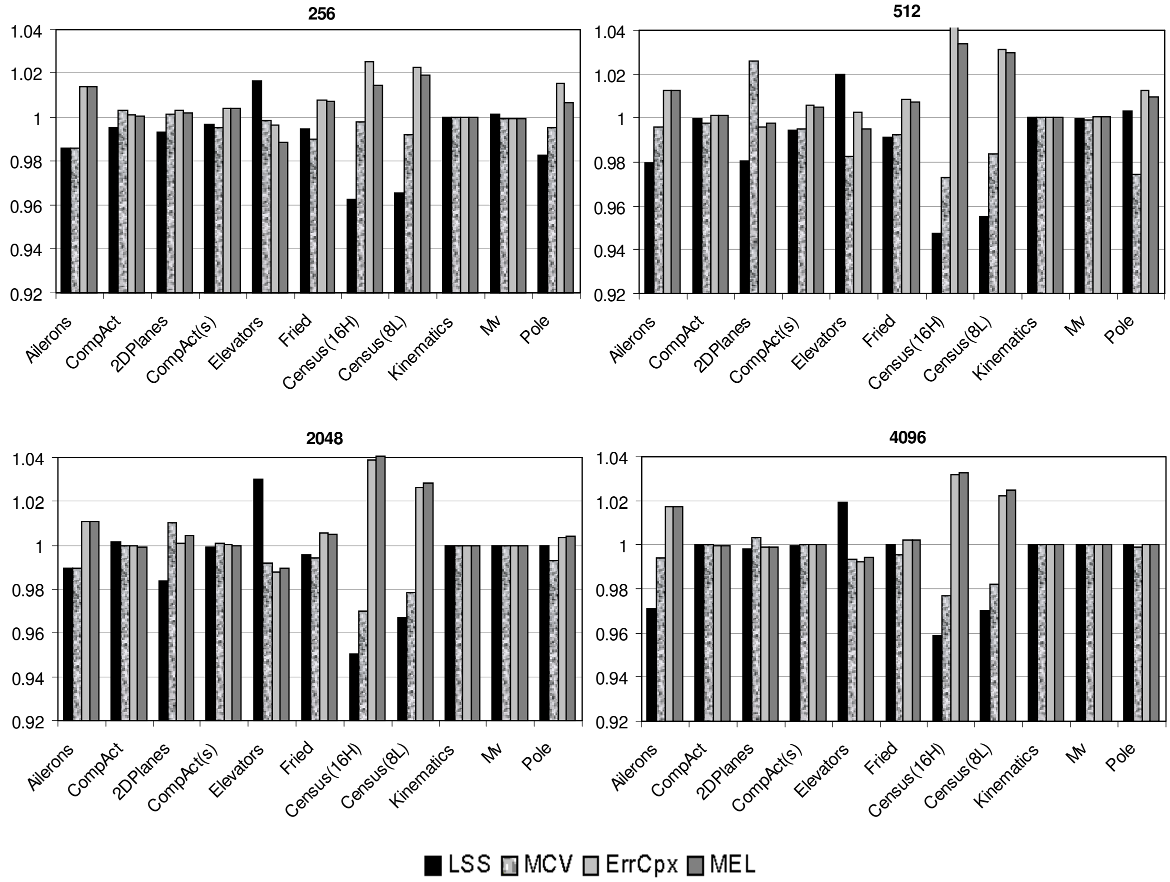


Figure 4.1 - Comparison of the four methods for generating sequences of sub-Trees.

For space reasons the results regarding the average error in each sequence are presented in the Appendix B.4. To be able to present the results for all data sets together, we have normalised each result by dividing it by the average score of the four methods. The results can thus be seen as a kind of ratios with respect to the average score of the four methods.

These experiments show that with the exception of the *Elevators* domain, the *LSS* method generates the more accurate pruned trees from the four methods. Moreover, with the exception of *Elevators*, *Mv* and *Kinematics*, the methods we have proposed generate better pruned trees than the *ErrCpx* and *MEL* methods. With respect to the average error of all trees within each sequence, our experiments show that few differences exist between the four methods (*c.f.* Appendix B.4).Given these results, we can be reasonably confident on the "quality" of the sequences of pruned trees produced by our two methods (*LSS* and *MCV*). Still, generating sequences of trees that include better models does not mean that in the second phase of pruning the existing tree selection methods will be able to choose them. This experiment was carried out under the "ideal" conditions of having access to the "true" prediction error. In Section 4.3.4.1 we will again compare these methods of generating sequences of trees, but now in conjunction with "real" selection methods. Still, based on the results of the experiments presented above, we can say that if we have a reliable method of selecting trees from a sequence it can be advantageous (or certainly not detrimental) to use the sequences generated by both the *LSS* and the *MCV* algorithms, when compared to existing methods.

4.3.2 Methods for Comparing Alternative Pruned Trees

In this section we address the second stage of pruning by tree selection: comparing the generated pruned trees. We will discuss two main methodologies for comparing tree-based models. The first is based on reliable estimates of the error of the models, while the second is based on the minimum description length principle. With respect to error-based selection we described three main strategies for obtaining reliable estimates of the error: methods based on resampling; bayesian estimation; and estimates based on the sampling properties of the error estimates.

4.3.2.1 Comparing Trees using Prediction Error Estimates

An estimator is a function that takes a sample of observations and uses it to estimate an unknown value of a statistical parameter θ. The estimator $\hat{\theta}$ is a random variable with a probability distribution function usually known as the *sampling distribution* of the estimator. An estimator is said to be *unbiased* if its expected value is equal to the true value of the parameter being estimated (*i.e.* $E(\hat{\theta}) = \theta$). This means that with repeated sampling we should obtain the true value of the population parameter by averaging over the different sample estimates. Although being unbiased is an important property of an estimator it does not indicate how *precise* a particular estimate is. In effect, we can have two different unbiased estimators of a parameter θ, one being preferable to the other because its sampling distribution is more tightly spread around the true value of θ. This notion can be captured by a statistic of *spread* applied to the estimates. The resulting statistic is usually known as the *standard error* of an estimator, $\text{S.E.}(\hat{\theta})$. Another important property of an estimator is *consistency*. This property states that with increasing size of the samples our estimates should improve. In summary, we are interested in *consistent*, *minimum variance* (*i.e. precise*) and *unbiased estimators* of the prediction error. In the following sections we describe several estimators of the prediction error of regression trees.

Resampling Methods

The main idea behind *resampling* methods is to use a separate set of data to obtain the reliable estimates. We have already seen a possible way of using these estimators within pruning when we have discussed CART pruning algorithm (Section 4.2.1). Resampling methods can also be used to tune parameters of a learning system. An example of such application consists of finding the "optimal" values of learning parameters to better tune a system to a particular domain (*e.g.* John,1997). This is particularly useful whenever the "optimal" values of these parameters depend on the domain under consideration. In a way the estimation phase of CART pruning can be seen as tuning the *cost per leaf node* (the α value) parameter, to ensure the highest estimated predictive accuracy. These techniques can also be used within the pruning method used in RETIS for obtaining the "best" value of m, and within CORE pruning method to find the "optimal" precision coefficients used in the MDL coding schema.

Our study will be centred on two particular resampling techniques, the *Holdout* and the *Cross-Validation*. Another frequently used resampling technique is the *bootstrap* (Efron,1979; Efron & Tibshirani, 1993) and its variants like the .632 bootstrap or the e0 bootstrap (see for instance Weiss & Kulikowski, 1991 or Kohavi, 1995). The bootstrap method is known to be particular suitable for small samples. With the rapid growth of computational power and the widespread of information technology, the size of data sets is growing at a very fast rate. Research fields like Knowledge Discovery in DataBases (KDD) put a particular emphasis on large data sets, which we share. This was the main motivation for not including the bootstrap method in our study.

Cross Validation Error Estimates

A k-fold cross validation (CV) estimate is obtained by randomly dividing the given training sample in k disjoint folds $D_1,...,D_k$, each containing approximately the same number of observations. For each fold D_f a regression model is constructed using as learning sample $D \backslash D_f$, obtaining the model $r_f(\beta, \mathbf{x})$. This model is then tested on the fold D_f. The same process is repeated for all folds. Finally, the CV error estimate is obtained by averaging the errors of these k models, *i.e.*

$$\hat{E}rr_{kCV} = \frac{1}{n} \sum_{f=1}^{k} \sum_{\{\mathbf{x}_i \in D_f\}} Err_i \tag{4.13}$$

where,

$Err_i = \left| \, y_i - r(\beta, \mathbf{x}_i) \right|$ or $Err_i = \left(y_i - r(\beta, \mathbf{x}_i)\right)^2$, depending on the type of error measure we are using to evaluate our model.

A particular case of this formula occurs when k is set to n. This leads to what is usually known as Leave-One-Out Cross Validation (LOOCV) where n models are constructed using n-1 training cases. This method is computationally expensive so it is used only with very small data sets. The most common set-up in current research within ML is 10-fold CV.

As mentioned by Breiman *et al.* (1984, p.307) it is not clear how to obtain standard error (SE) estimates for the CV estimator since the errors are not independent due to the overlap of the k training sets. If we ignore this dependence we reach a heuristic formulation for the SE of this estimator,

$$S.E.\left(\hat{E}rr_{kCV}\right)=\frac{1}{n}\sqrt{\sum_{f=1}^{k}\sum_{\{\mathbf{x}_i\in D_f\}}\left(Err_i-\hat{E}rr_{kCV}\right)^2} \tag{4.14}$$

The use of CV estimators with tree-based models presents some difficulties. For instance, we have to repeat the learning process k times, which brings additional computation costs. Another problem is how to use the CV estimate to select the best pruned tree. We have seen in Section 4.2.1 that Breiman *et al.* (1984) use the cost per leaf node (the α values) to perform a tree-matching process that allows the use of CV estimates in the pruning process. This method is strongly tied to the error-complexity sequence generation algorithm. It does not make sense to use the α values to perform tree matching with other sequences of trees, because Breiman and his colleagues proved that the optimal sequence is the one provided by the Error-Complexity algorithm. Motivated by the fact that we have studied other algorithms for obtaining sets of pruned trees we have devised an alternative tree-matching method.

The Error-Complexity algorithm produces a parametric sequence of nested trees $T(\alpha) = < T_0, \ldots, T_n>$. Associated with each tree in the sequence there is a α value. Let us denote a tree belonging to this sequence as $T(\alpha_i)$ to reinforce this association between the trees and the respective α values. As we have mentioned in Section 4.2.1, when using k-fold cross validation error estimates, CART also produces k parametric sequences $T^1(\alpha),\ldots,T^k(\alpha)$.

For each tree in these sequences we have a reliable estimate of its error obtained with the respective fold. The tree matching procedure of CART estimates the error of the tree $T(\alpha_i)$ of the main sequence as the average of the error estimates of the trees $T^1\left(\sqrt{\alpha_i\alpha_{i+1}}\right),...,T^k\left(\sqrt{\alpha_i\alpha_{i+1}}\right)$, where $T(\alpha_r)$ is the tree belonging to the sequence $T(\alpha)$ with α value most similar to α_r. The assumption behind this procedure is that the trees $T^1\left(\sqrt{\alpha_i\alpha_{i+1}}\right),...,T^k\left(\sqrt{\alpha_i\alpha_{i+1}}\right)$ have the same true prediction error as $T(\alpha_i)$. In other words, trees with a similar cost per leaf will have similar true prediction error. As mentioned by Esposito *et al.* (1997) there is no theoretical justification to support this.

We now describe an alternative tree-matching method that allows using cross validation error estimates for any sequence of nested pruned trees. Let us assume that trees obtained with samples with approximately the same size will have similar predictive accuracy. Under this assumption it seems reasonable to consider that the error estimate of overly large tree $T_{\max}$ obtained with all training data, should be calculated with the error estimates of the trees $T^1_{\max},...,T^k_{\max}$ (*i.e.* the overly large trees of the k folds). Moreover, as the training sets in the k folds are samples of the same population it is reasonable to assume that they will have the same variance. Based on this argument we can estimate the error of the tree consisting of a single leaf, using the similar trees in the sequences $T^v(\alpha)$[44].

Linear polynomials obtained through the least squares method are usually evaluated by the *proportion of variance they explain*. This statistic is obtained by,

$$\rho^2(r) = \frac{s_Y^2 - Err(r)}{s_Y^2} = 1 - RE(r) \tag{4.15}$$

where,

r is a regression model;
$Err(r)$ is the mean squared error of the model;
s_Y^2 is the sample variance of Y;
and $RE(r)$ is usually known as the relative error of r.

---

[44] Because the error of a tree consisting of a single leaf is given by the variance (s_Y^2) of the training sample (*i.e.* $Err(T_n) = s_Y^2$).

We can calculate similar ρ^2 values for any tree in a sequence. These values range from a maximum value for the tree T_{max} (which is the first element in the sequence, T_0), until the value zero, relative to the tree consisting of a single leaf. These values decrease as the trees get smaller because the trees are nested and have increasing value of resubstitution error (*i.e.* they *explain* less variance of the training sample). This means that we can look at our sequence of trees as a parametric family $T(\rho^2) = \langle T(\rho_0^2), T(\rho_1^2), \ldots, T(\rho_n^2) \rangle$, where $\rho_0^2 > \rho_1^2 > \ldots > \rho_n^2$ and $\rho_n^2 = 0$. Without loss of generality we may re-scale these values to cover the interval [1..0], using a simple linear transformation consisting of dividing the ρ_i^2 values by ρ_0^2 (*i.e.* by the maximum value of ρ_i^2). This leads to the following statistic,

$$\vartheta_i^2 = \frac{\rho_i^2}{\rho_0^2} = \frac{Err(T_n) - Err(T_i)}{Err(T_n) - Err(T_0)} \tag{4.16}$$

where, $\vartheta_0^2 = 1$, $\vartheta_n^2 = 0$, $\vartheta_i^2 > \vartheta_{i+1}^2$ and $0 < \vartheta_i^2 < 1$, $i = 1 \ldots n-1$.

The starting point of our proposed tree matching procedure is a sequence of nested pruned trees, $T(\vartheta^2) = \langle T(\vartheta_0^2), T(\vartheta_1^2), \ldots, T(\vartheta_n^2) \rangle$, and the k cross validation sequences $T^1(\vartheta^2), \ldots, T^k(\vartheta^2)$. Our tree-matching method consists of using the ϑ^2 values to assert similarity between trees in these sequences. Namely, the error estimate of tree $T(\vartheta_i^2)$ is obtained as an average of the error estimates of the k trees $T^1(\vartheta_i^2), \ldots, T^k(\vartheta_i^2)$. The underlying assumption behind this tree matching procedure is that trees explaining the same proportion of variance of the given training sample, are likely to have similar true prediction error on future samples of the same population.

We have compared our tree-matching proposal with the method used in CART. Using the same sequence of trees (the one produced by CART Error-Complexity algorithm) and the same error estimation technique (5-fold CV), we have compared the selected trees in the main sequence for each of the two tree-matching methods. Figure 4.2 shows the sign and the statistical significance of the difference in MSE between the two alternatives, estimated using the DELVE experimental methodology[45].

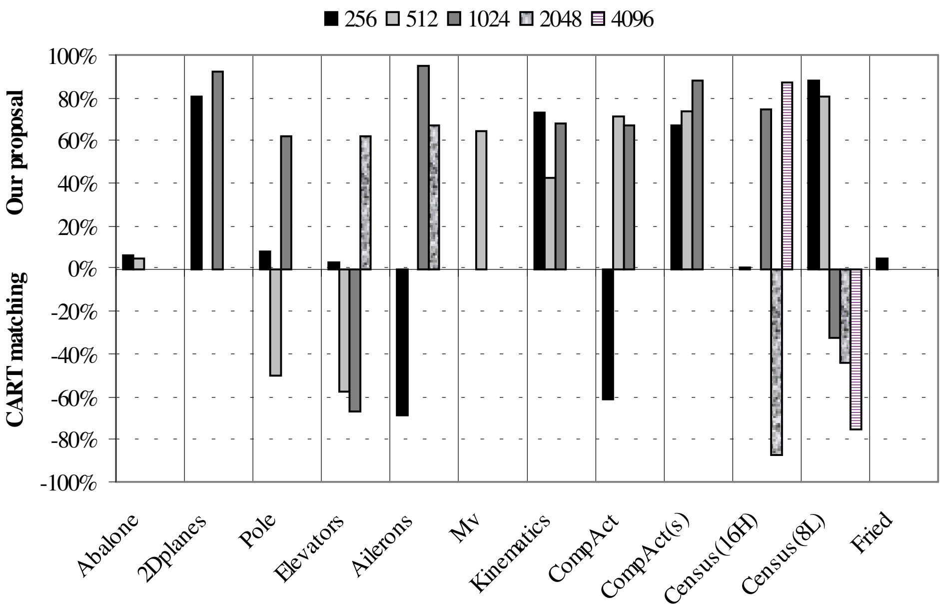


Figure 4.2 - Our tree-matching proposal vs. CART method.

As Figure 4.2 shows there are no statistically significant[46] differences between the choices entailed by the use of the two alternative tree-matching methods. In spite of a tendency for the differences being favourable to our method, we cannot discard with high confidence the hypothesis that both alternatives achieve the same accuracy. Still, we should recall that the single motivation for the introduction of our method was to allow the use of CV estimates with other methods of generating sequences of pruned trees apart from the method used in CART. This is a relevant issue because we have shown in Section 4.3.1

---

[45] Details concerning the experimental methodology and the information described in the figures can be found in Annex A. The complete tables of results of this experiment can be found in Annex B.5.

[46] We consider an observed difference statistically significant if there is at least 95% confidence that the two methods will not achieve similar accuracy on other samples of the same population. Furthermore, if the confidence reaches the 99% level we consider the difference highly significant.

that it is possible to obtain better results with other sequence generation methods that do not produce the same sequence as the Error-Complexity algorithm.

Another relevant issue when applying CV estimators is the number of folds to use. Smaller numbers make the size of the folds larger leading to more reliable estimates. However, as fewer cases are left for training, this also affects adversely the "quality" of the model and thus its error, and hence there is a trade-off between the two factors. Moreover, for large data sets the value of k strongly influences the computation time. We have not carried out any systematic experiment to determine the *optimal* number of folds. In our experiments we have used the value 5 on the basis of empirical observations and also because it is commonly used within ML.

The Holdout Method

With the Holdout method the given training cases are randomly divided into two separate samples. One of the samples is used for training and the other (the holdout sample) to obtain unbiased estimates of the models learned. The usual way data is used by this method in the context of regression trees (*e.g.* system CART by Breiman *et al.*,1984) is described by the Figure 4.3:

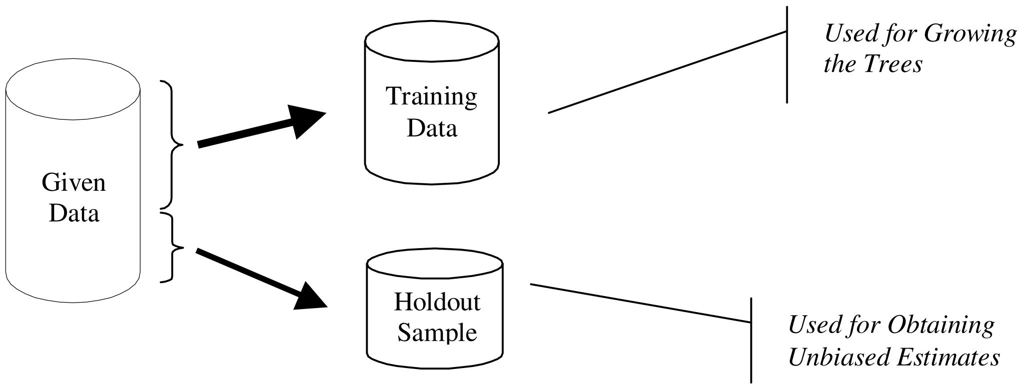


Figure 4.3 - The Holdout Method.

A Holdout estimate is the average prediction error of the model on the cases in the holdout,

$$\hat{Err}_{Hld} = \frac{1}{n_{Hld}} \sum_{i=1}^{n_{Hld}} Err_i \tag{4.17}$$

where,

n_{Hld} is the number of cases in the Holdout sample;

and Err_i is the prediction error of the model $r(\beta,\mathbf{x})$ for case $\langle \mathbf{x}_i, y_i \rangle$.

We are trying to estimate the "true" mean error of a regression model. Assuming that the holdout sample is drawn from the same population as the learning sample, we can say that the Holdout is an unbiased estimator of the mean error of the model. In effect, it is well known that the average is an unbiased estimator of a population mean, and the holdout estimates are obtained by averaging the errors of the model in the holdout sample. With respect to the standard error of the holdout estimates statistical theory tells us that if $X_1, \ldots, X_n$ is a random sample of a distribution with mean μ and variance σ^2 the variance of the average estimator of the mean is given by,

$$\text{Var } \overline{X} = \frac{\sigma^2}{n} \tag{4.18}$$

This means that the standard error of the holdout estimates is given by,

$$S.E.\left(\hat{Err}_{Hld}\right) = \sqrt{\frac{\sigma_E^2}{n}} \tag{4.19}$$

where,

σ_E^2 is the variance of the error Err;

and n is the size of our sample (in this case the size of the holdout).

Using the sample variance estimate[47] we get the following operational formula,

$$S.E.\left(\hat{Err}_{Hld}\right) = \frac{1}{n}\sqrt{\frac{n\sum_{i=1}^{n} Err_i^2 - \left(\sum_{i=1}^{n} Err_i\right)^2}{n-1}} \tag{4.20}$$

---

[47] $s_X^2 = \dfrac{n\sum X^2 - \left(\sum X\right)^2}{n(n-1)}$

The two alternative ways of defining the Err_i's mentioned before (squared or absolute differences) tell us that less credit should be given to the standard error estimates when using the MSE criterion, as we will have powers of four[48], which can be extremely variable. This can be seen as another advantage of LAD regression trees. Equation 4.20 is slightly different from the formula derived by Breiman *et al.* (1984, p.226). The difference results from the fact that the authors have used the biased estimate of the variance, where the denominator is n instead of n-1. This approximation is known to underestimate the true population variance (Chatfield,1983), which means that the values obtained by their formula should be over-optimistic compared to ours.

An important issue when using these estimates is the size of the holdout. This method requires the two samples to be independent which means that we will decrease the number of cases available for training. While one wants a sufficiently large pruning set (holdout), one does not want to remove too many cases from the training set, so as not to harm the quality of the learned trees. The first obvious observation that one can make about this method is that it is clearly inadequate for small samples. In effect, as Weiss and Kulikowski (1991) pointed out, for moderately sized samples this method usually leaves one with insufficient number of cases either for training or pruning. The authors have suggested that a holdout sample with around 1000 observations should be sufficient for most cases. We have experimentally confirmed on our benchmark data sets that this is a reasonable assumption. Using larger holdouts brings little increased precision and, in effect, ends up harming the accuracy of the tree model, because too many cases have been "removed" from the learning sample. In our experiments with the holdout method we have used a similar heuristic, by setting the size of the holdout as follows,

$$n_{Hld} = \min\left(30\% \times n, 1000\right) \tag{4.21}$$

where n is the training sample size.

---

[48] Because for the MSE criterion $Err_i = \left(y_i - r(\beta, \mathbf{x}_i)\right)^2$.

This means that for instance, if the sample size is 500 cases, then we have that min(30%×500,1000) = 150, and thus 150 cases will be left out as holdout while the remaining 350 will be used as training set. On the contrary, if the sample size is 50000, we have that min(30%×50000,1000) = 1000, thus only 1000 cases will be left out as holdout.

m-Estimates

As we have mentioned in Section 4.2.2, Karalic and Cestnik (1991) have presented *m* estimators for the mean and variance of a variable, which can be used to obtain reliable estimates of the error of LS regression trees. Although the authors have used *m*-estimates with the Niblett & Bratko (*N&B*) pruning algorithm, these estimates can also be used to compare alternative regression trees. For instance, given a sequence of trees such as the one produced by the Error-Complexity algorithm used in CART, *m*-estimates could be used to select the final model instead of the resampling techniques used in CART. We will use this method in our experimental comparisons. Moreover, we have extended the work of Karalic and Cestnik (1991) by deriving the standard error associated with these estimators. This allows the use of the 1-SE selection rule (Breiman *et al.*, 1984) with *m*-estimates leading to large benefits in terms of tree size of the selected model.

According to Kendall and Stuart (1969, vol. 1) the standard error of the sample mean squared error is given by,

$$S.E.\left(MSE_{\bar{y}}\right) = \sqrt{\frac{1}{n}\left(\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \bar{y}\right)^4 - \left(\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2\right)^2\right)} \tag{4.22}$$

The standard error is a statistic of the sampling distribution of a population parameter. Using Equation 4.3, we have developed the *m*-estimate of the standard error associated with the sample MSE, which is given by the following equation,

$$mEst\left(S.E.\left(MSE_{\bar{y}}\right)\right)=$$

$$\frac{n_l}{n_l+m}\sqrt{\frac{1}{n_l}\left(\frac{1}{n_l}\sum_{i=1}^{n_l}\left(y_i-mEst(\bar{y})\right)^4-\left(\frac{1}{n_l}\sum_{i=1}^{n_l}\left(y_i-mEst(\bar{y})\right)^2\right)^2\right)}+$$
$$\frac{m}{n_l+m}\sqrt{\frac{1}{n}\left(\frac{1}{n}\sum_{i=1}^{n}\left(y_i-mEst(\bar{y})\right)^4-\left(\frac{1}{n}\sum_{i=1}^{n}\left(y_i-mEst(\bar{y})\right)^2\right)^2\right)} \tag{4.23}$$

The expressions inside the squared roots can be expanded using the following equality

$$\sqrt{\frac{1}{n}\left(\frac{1}{n}\sum_{i=1}^{n}\left(y_i-mEst(\bar{y})\right)^4-\left(\frac{1}{n}\sum_{i=1}^{n}\left(y_i-mEst(\bar{y})\right)^2\right)^2\right)}=$$
$$\frac{1}{n}\sqrt{s_4-4s_3mEst(\bar{y})+4s_2\left(mEst(\bar{y})\right)^2-\frac{1}{n}s_2^2+\frac{4}{n}s_2s_1mEst(\bar{y})-\frac{4}{n}s_1^2\left(mEst(\bar{y})\right)^2} \tag{4.24}$$

where,

$$s_1=\sum_{i=1}^{n}y_i,\quad s_2=\sum_{i=1}^{n}y_i^2,\quad s_3=\sum_{i=1}^{n}y_i^3,\quad s_4=\sum_{i=1}^{n}y_i^4\,.$$

Calculating these s factors brings no significant computational cost as this can be carried out during tree growth. Using the expression given in Equation 4.24 the m-estimate of the standard error of the sample MSE can be calculated in an efficient manner.

We have also extended the use of m-estimates to least absolute deviation (LAD) regression trees. To grow LAD trees we need estimates of the median and of the mean absolute deviation to the median. We have derived m-estimates for these two statistics. Regarding the priors we have followed the same procedure of estimating them at the root of the tree (*i.e.* using all training data). Using Equation 4.3 we can obtain the m estimate of the median in a leaf l as,

$$mEst(\nu)=\frac{n_l}{n_l+m}\nu(D_l)+\frac{m}{n_l+m}\nu(D_n) \tag{4.25}$$

where,

$\nu(D_l)$ and $\nu(D_n)$ are the resubstitution estimates of the medians obtained with the cases in the leaf and root nodes, respectively;
and n_l is the size of training sample in leaf l.

With respect to the mean absolute deviation to this median we have,

$$m\text{Est}(MAD_\nu) = \frac{n_l}{n_l+m}\frac{1}{n_l}\sum_{D_l} |\, y_i - m\text{Est}(\nu)| + \frac{m}{n_l+m}\frac{1}{n}\sum_{D_n} |\, y_i - m\text{Est}(\nu)|$$

$$= \frac{1}{n_l+m}\sum_{D_l} |\, y_i - m\text{Est}(\nu)| + \frac{m}{n(n_l+m)}\sum_{D_n} |\, y_i - m\text{Est}(\nu)|$$

using the equation derived in Section 3.3.1 for the *SAD* of a set of observations we get,

$$= \frac{1}{n_l+m}\left(\sum_{D_l^+} y_i - \sum_{D_l^-} y_i + m\text{Est}(\nu)\times ODD(\#D_l)\right) + \frac{m}{n(n_l+m)}\left(\sum_{D_n^+} y_i - \sum_{D_n^-} y_i + m\text{Est}(\nu)\times ODD(\#D_n)\right) \tag{4.26}$$

The formula derived above needs a pass trough all training data for each estimate of the *MAD*, as we need to obtain the sums of the *Y* values greater and smaller than the *m* estimate of the median. As this estimate is different for each leaf, this needs to be done for all leaves. Thus *m* estimates for LAD trees have a cost proportional to $O(\#\tilde{T} \times n)$, where $\tilde{T}$ is the set of leaves of the tree *T*. We can reduce this cost by obtaining the two summations in an incremental fashion. In effect, during the tree growth these sums get calculated for the resubstitution estimate of the median. Moreover, we already have the observations in two AVL trees D^+ and D^- (see Section 3.3.1). The *m*-estimate of the median is either bigger or smaller than the resubstitution estimate. Thus we only need to update the two sums with the cases in the interval between these two values. This will lead to a complexity proportional to $O(\#\tilde{T} \times k)$, where *k* is much smaller than *n*.

We now address the issue of obtaining the *m* estimate of the standard error associated with the estimate of the mean absolute deviation given above. Kendall and Stuart (1969, vol. 1) refer that the standard error associated with the sample mean deviation about a value ν is given by,

$$S.E.(MAD_\nu) = \sqrt{\frac{1}{n}\left(\sigma^2 + (\nu - \mu)^2 - (\delta_\nu)^2\right)} \tag{4.27}$$

where,

$\delta_\nu = E(|y_i - \nu|)$, i.e. the expected value of the mean absolute deviation.

Using the sample estimates of the σ^2 and δ_ν statistics we get,

$$S.E.(MAD_\nu) = \sqrt{\frac{1}{n}\left(\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2 + (\nu - \bar{y})^2 - \left(\frac{1}{n}\sum_{i=1}^{n}|y_i - \nu|\right)^2\right)} \tag{4.28}$$

We have developed the m estimate of this standard error which is given by the following equation,

$$m\mathrm{Est}(S.E.(MAD_\nu)) =$$

$$\frac{n_l}{n_l + m}\sqrt{\frac{1}{n_l}\left(\frac{1}{n_l}\sum_{i=1}^{n_l}(y_i - m\mathrm{Est}(\bar{y}))^2 + (m\mathrm{Est}(\nu) - m\mathrm{Est}(\bar{y}))^2 - \left(\frac{1}{n_l}\sum_{i=1}^{n_l}|y_i - m\mathrm{Est}(\nu)|\right)^2\right)} +$$

$$\frac{m}{n_l + m}\sqrt{\frac{1}{n}\left(\frac{1}{n}\sum_{i=1}^{n}(y_i - m\mathrm{Est}(\bar{y}))^2 + (m\mathrm{Est}(\nu) - m\mathrm{Est}(\bar{y}))^2 - \left(\frac{1}{n}\sum_{i=1}^{n}|y_i - m\mathrm{Est}(\nu)|\right)^2\right)}$$

Once again we can try to obtain a computationally more efficient formula for the expressions inside the squared roots leading to,

$$\sqrt{\frac{1}{n}\left(\frac{1}{n}\sum_{i=1}^{n}(y_i - m\mathrm{Est}(\bar{y}))^2 + (m\mathrm{Est}(\nu) - m\mathrm{Est}(\bar{y}))^2 - \left(\frac{1}{n}\sum_{i=1}^{n}|y_i - m\mathrm{Est}(\nu)|\right)^2\right)} =$$

$$\frac{1}{n}\sqrt{2nk_1^2 + s_2 - 2s_1k_1 + nk_2^2 - 2nk_2k_1 - \frac{1}{n}k_3^2 + \frac{2}{n}k_2k_3k_5 - \frac{1}{n}k_4^2 + \frac{2}{n}k_2k_4k_5 - \frac{1}{n}k_2^2k_5^2}$$

where,

the s factors are defined as before;

and $k_1 = m\mathrm{Est}(\bar{y})$, $k_2 = m\mathrm{Est}(\nu)$, $k_3 = \sum_{D^+} y_i$, $k_4 = \sum_{D^-} y_i$, $k_5 = ODD(\#D)$.

Although this formula increases the efficiency of the calculation of the standard error, there are still some factors (k_3 and k_4) that need two passes through the data to be obtained. This is the same efficiency problem mentioned when presenting the m estimates of the MAD. However, as these factors are already calculated to obtain the m estimates of the MAD, the calculation of the standard error of these estimates brings no additional computation effort.

Estimates based on Sampling Distribution Properties

Statistical estimation theory is concerned with obtaining unbiased estimates of population parameters. *Point estimates* provide a unique number for the parameter value. Together with this number we are interested in obtaining a standard error of the estimate. Equations 4.22 and 4.27 calculate the standard error associated with both the mean squared error and the mean absolute deviation to the median. *Interval estimates*, on the other hand, provide an interval were we can be sure that in *x*% of the cases the true population parameter lies in. Interval estimates can be obtained if we know the sampling distribution of the parameter being estimated. For instance, the central limit theorem tells us that irrespectively of the distribution of a random variable, the sampling distribution of its mean is normal. This allows us to obtain confidence intervals for the location of the true population mean based on the mean estimated with a single random sample. In the case of regression trees we are interested in obtaining estimates of the true error in each leaf. In our study we have used as error measures either the MSE or the MAD.

For the MSE criterion, the error associated with a leaf can be seen as an estimate of the variance of the cases within it. Statistical estimation theory tells us that the sampling distribution of the variance is the χ^2 distribution (*e.g.* Bhattacharyya & Johnson, 1977), if the original variable follows a normal distribution. According to the properties of the χ^2 distribution, a 100×(1-α)% confidence interval for the true population variance based on a sample of size *n* is given by,

$$\left(\frac{(n-1)s_Y^2}{\chi^2_{\alpha/2,\,n-1}},\quad \frac{(n-1)s_Y^2}{\chi^2_{(1-\alpha/2),\,n-1}} \right) \tag{4.29}$$

where,

s_Y^2 is the sample variance (obtained in a particular tree leaf);
and $\chi^2_{\alpha,n}$ is the tabulated value of the χ^2 distribution for a given confidence level α and *n* degrees of freedom.

This formulation is based on an assumption of normality of the distribution of the variable *Y*. In most real-world domains we cannot guarantee *a priori* that this assumption holds. If that is not the case we may obtain too narrow intervals for the location of the true population variance. This means that the true error in the leaf can be outside of the interval boundaries. However, in the context of pruning by tree selection, we are not particularly interested in the precision of the estimates, but in guaranteeing that they perform a correct ranking of the candidate pruned trees.

The χ^2 distribution is not symmetric, meaning that the middle point of the interval defined by Equation 4.29 does not correspond to the sample point estimate of the variance (Bhattacharyya & Johnson, 1977). In effect, the middle point of this interval is larger than the point estimate. The difference between these two values decreases as the number of degrees of freedom grows, because it is known that the χ^2 distribution approximates the normal distribution[49] when the number of degrees of freedom is sufficiently large. This means that as the sample size (which corresponds to the number of degrees of freedom) grows, the middle point of the interval given in Equation 4.29 will tend to approach the point estimate obtained with the sample. This is exactly the kind of bias most pruning methods rely on. They "penalise" estimates obtained in the leaves of large trees (with few data points) when compared to estimates at higher levels of the trees. Being so, we propose using the middle point of the interval in Equation 4.29 as a more reliable estimate of the variance of any node, which leads to the following estimator of the MSE in a node t,

$$ChiEst(MSE(t)) = MSE(t) \times \frac{n_t - 1}{2} \times \left(\frac{1}{\chi^2_{\alpha/2, n_t - 1}} + \frac{1}{\chi^2_{(1-\alpha/2), n_t - 1}} \right) \tag{4.30}$$

where,

$\frac{n_t - 1}{2} \times \left(\frac{1}{\chi^2_{\alpha/2, n_t - 1}} + \frac{1}{\chi^2_{(1-\alpha/2), n_t - 1}} \right)$ can be seen as a correcting factor of the MSE in a node t.

---

[49] Which is symmetric.

This is a heuristic method of obtaining an estimate of the true mean squared error in a tree node, obtained through the use of a "correcting" factor on the resubstitution estimate of the MSE. This factor is a function of the number of cases from which the resubstitution error was obtained and of the sampling distribution properties of the mean squared error. A similar strategy is followed in C4.5 (Quinlan, 1993a) for classification trees, which applies a "correcting" factor to the resubstitution error rate, based on the binomial distribution. Figure 4.4 shows the value of the correcting factor for different sample sizes and confidence levels of the χ^2 distribution.

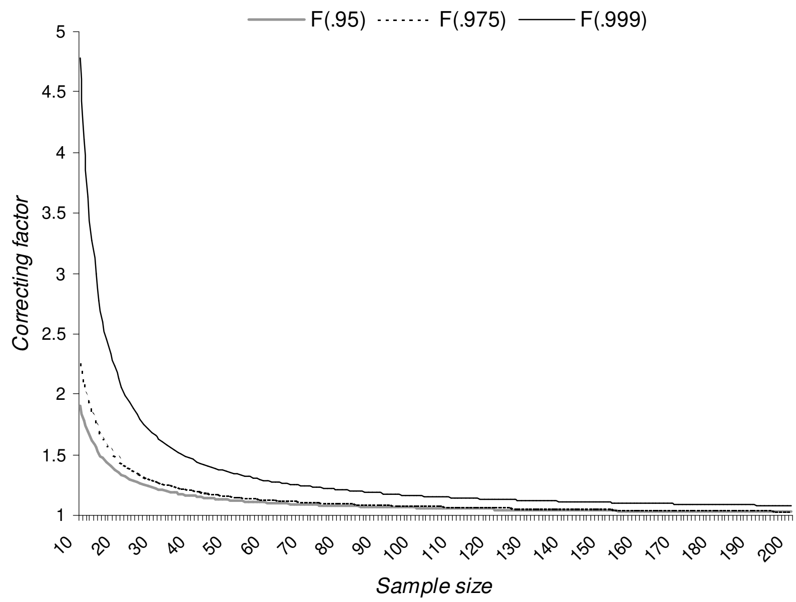


Figure 4.4 –Different values of the "correcting factor" used in the ChiEst estimator.

As it can be seen the larger the confidence level the higher the value of the correcting factor penalising small samples. This means that the higher the confidence level the stronger the preference bias for smaller trees.

Regarding the use of the 1-SE rule we can calculate the standard error of these estimates using Equation 4.22.

We were not able to find the sampling distribution of the Mean Absolute Deviation to the median. Being so, we decided not to use this type of estimators with our LAD regression trees.

4.3.2.2 *Comparing Trees using their Binary Description Length*

Most existing work on pruning is solely guided by reliable estimates of the prediction error. Still, pruning inevitably leads to smaller and thus less complex trees. Pruning has an important effect on model complexity and interpretability as it was pointed out by several authors (*e.g.* Bohanec and Bratko, 1994; Kononenko, 1989). In effect, there is a strong resistance to "black box" models by many human experts. As a result of this some authors have tried to incorporate both the notions of simplicity and prediction accuracy in the preference bias guiding the overfitting avoidance process. Breiman *et al.* (1984) have added a complexity cost to the error estimates leading to the *error-complexity pruning* method used in their CART system. Still, this measure is only used for generating the set of alternative trees considered during the pruning process, while the final selection is solely guided by the minimisation of the estimated error. Both *m*-estimates and *ChiEst* indirectly incorporate a bias for smaller trees by penalising estimates obtained with small samples. The Minimum Description Length (Rissanen, 1978) principle is based on a sound theoretical framework that can incorporate the notions of model complexity and accuracy. This work gave rise to studies of binary coding of tree-based models which is now a well-studied subject. Coding of classification trees was explored for instance by Quinlan & Rivest (1989) and Wallace & Patrick (1993). The work of Kramer (1996) seems to be the first attempt which involves using MDL for selecting a good candidate from a set of different regression trees. This author described the SRT system that learns a particular type of regression trees using a least squares error criterion. The particularity of SRT resides on the use of a relational language for the tests in the nodes of the trees. In effect, the final tree can be translated into a set of relational clauses. SRT builds several trees using different stopping criteria and uses MDL to select the best one. Kramer (1996) describes somewhat vaguely the coding used in SRT. He refers that the length of a tree encoding consists of the sum of the encoding of the tree model plus the encoding of its errors on the training data. The errors are real numbers and are encoded using the method proposed by Rissanen (1982). As for the tree model the author just mentions that he encodes the choices made in each node from all possible literals. As we have seen in Section 4.2.3, Robnik-Sikonja and Kononenko (1998) also use MDL for pruning LS regression trees in the CORE system. The coding schema[50] provided by these authors can be used to obtain the binary description length of any regression tree. We use this code length to compare different pruned trees in the context of pruning by tree selection.

4.3.3 Choosing the Final Tree

This section addresses the final step of pruning by tree selection. After an initial stage consisting of generating a set of alternative pruned trees, we evaluate these alternatives by means of any of the methods described in Section 4.3.1. The goal of these evaluation methods is to provide information that allows choosing one of such models as the final tree obtained by the learning algorithm. Different strategies can be used in this final step of pruning by tree selection.

If we compare the alternative pruned trees using estimates of their true prediction error, the "natural" method of selecting a tree is to choose the model with lowest estimated error. However, Breiman *et al.* (1984) suggested an alternative method biased toward simpler models. This alternative consists of selecting the smallest tree within the interval $\hat{Err}_l + S.E.\left(\hat{Err}_l\right)$, where $\hat{Err}_l$ is the lowest error estimate and $S.E.\left(\hat{Err}_l\right)$ is the standard error of this estimate. This method, usually known as the 1-SE rule, can be generalised to a k-SE rule with $k \geq 0$ [51].

---

[50] Full details regarding this coding schema can be found in the appendix at the end of this chapter.

[51] Notice that with $k = 0$ this rule resumes to selecting the tree with lowest error.

If the trees in the sequence are compared in terms of their binary description length, the application of the Minimum Description Length principle leads to the selection of the model with shortest binary code.

4.3.4 An Experimental Comparison of Pruning by Tree Selection Methods

In this section we describe a set of experiments that compare different approaches to pruning by tree selection. These experiments provide a better understanding of the different biases of the alternative pruning methods we have considered in the previous sections. The conclusions of these experiments allow us to claim that depending on the preference criteria of the user, some methods will be preferable to others in domains with similar characteristics.

4.3.4.1 Comparing Methods of Generating Sets of Pruned Trees

In Section 4.3.1 we have described two new methods of generating sequences of nested pruned trees (*LSS* and *MCV*). In this section we compare these methods with existing alternatives using different ways of selecting the best pruned tree.

Figure 4.5 shows the sign and statistical significance of the estimated MSE difference between our two proposals (*MCV* and *LSS*) and other existing sequence generation methods (*MEL* and *ErrCpx*). In this experiment we have used *ChiEst* with a confidence level of 95%, as the method of selecting one tree from the sequence. All sequence generation algorithms use as "starting point" the same tree $T_{\max}$.

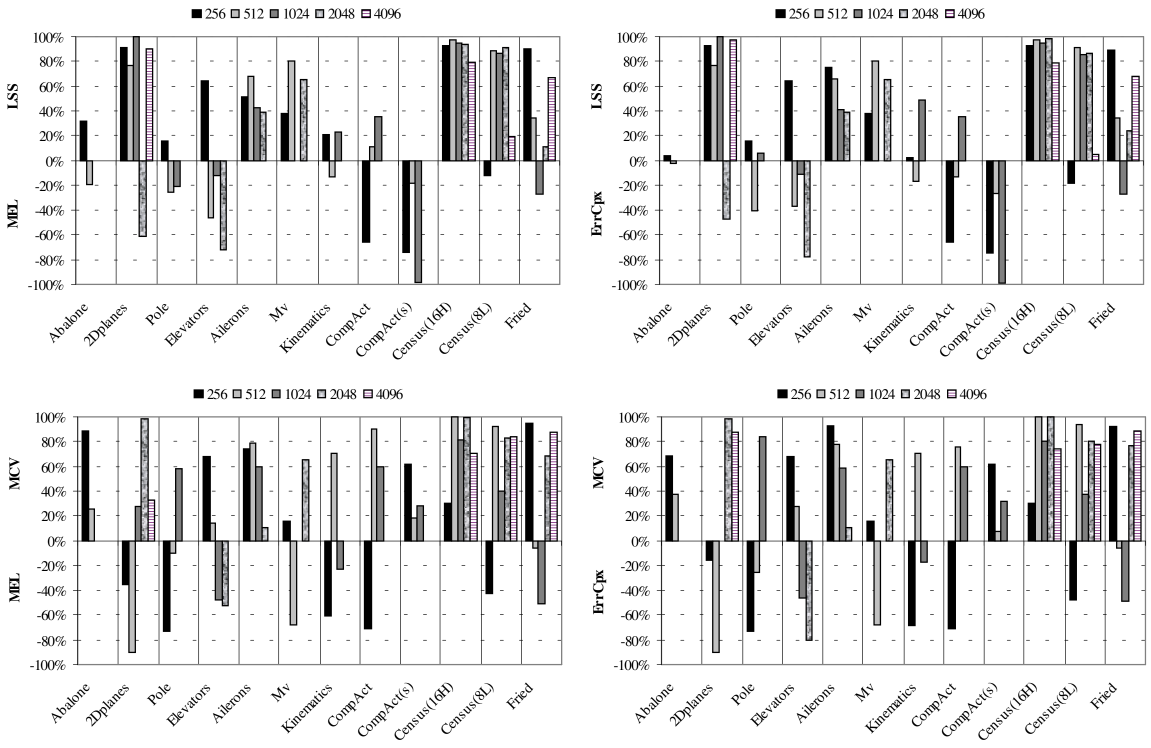


Figure 4.5 – Comparison of LSS with other sequence generation methods using ChiEst(95%) as selection method.

These graphs show a clear advantage of both *MCV* and *LSS* over existing sequence generation methods. In effect, we can observe several statistically significant advantages of our proposals and only with the *LSS* strategy we have observed a significant loss in the *CompAct(s)* domain. These results show that the better potential that we have observed in the experiments reported in Section 4.3.1 (Figure 4.1), can be capitalised by the *ChiEst* selection method.

We have also carried out similar experiments with other tree selection methods. The results are comparable so we do not include them here for space reasons. They can be found in the Appendix B.6.

4.3.4.2 Comparing Methods of Evaluating Trees

Pruning by tree selection includes a stage where a tree is chosen according to some evaluation criteria. In Section 4.3.1 we have reviewed several possible ways of performing this evaluation. In this section we show the results of an experimental comparison of these methods. Here we use as candidate pruned trees the sequence generated with the *LSS* algorithm, which as we have seen in the previous section, is a quite good method overall.

Before presenting the results of the comparison we make a few remarks regarding tuning of the parameters of some tree evaluation strategies. Both *m*-estimates, *ChiEst* and *MDL* selection require that some parameters are set. All of these parameters reflect certain preference bias over the accuracy / tree size trade-off. Ideally, one would like to have a default setting that would "work well" across all domains. Alternatively we can use resampling-based tuning to find out the parameter setting that maximises the expected accuracy on our target domain. Obviously, this tuning strategy only makes sense in case our goal is to maximise predictive accuracy. Still, this is the most common way of proceeding. We have already seen that CART uses such tuning method to find out which cost per leaf (α value) leads to higher estimate of predictive accuracy. We have carried out a set of experiments to obtain a better understanding of the effect of changing the value of the parameters of the different tree evaluation methods.

Tuning of the ChiEst evaluation method

We start our analysis with the *ChiEst* tree evaluation method. The parameter of this error estimator is the confidence level used to obtain the χ^2 distribution values. As Figure 4.4 (p.138) shows, different values of the confidence level lead to different penalisation of the resubstitution estimates. We have carried out a simple experiment to evaluate the effect of the value of the confidence level on the size of the selected tree. This experiment was carried out with the *Abalone*, *Pole*, *CompAct* and *Elevators* data sets. For each domain we have grown a LS regression tree, generated a set of pruned trees using the *LSS* algorithm, and then selected the "best" tree according to a *ChiEst* evaluation carried out using different confidence level values. The result of varying the value of the confidence level from 0.5 to 1 on the relative size of the selected tree when compared to the initially learned tree, is shown in Figure 4.6, for the four domains mentioned above.

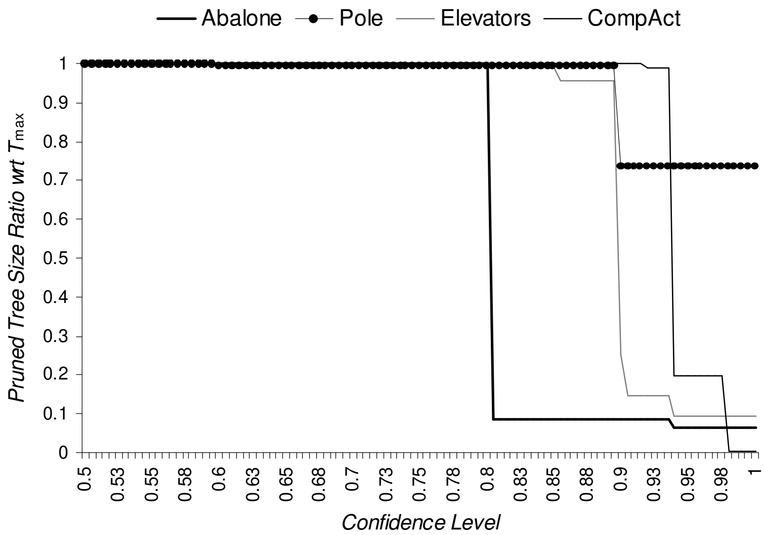


Figure 4.6 - The effect of the value of the confidence level on the pruned tree size.

As we can see the larger the confidence level the smaller the selected pruned tree. However, we can observe that for a wide range of confidence level values the selected tree is the same. This means that the *ChiEst* evaluation method is quite robust to variations on this value. Moreover, we also observe that depending on the domain different levels of pruning are carried out for the same confidence level value.

As we have mentioned we would like to have a fixed setting of the confidence level that was adequate over a wide range of data sets, to avoid the computational burden of having to use resampling-based tuning. We have tried several fixed settings and our experiments lead us to select the value of 0.95. We have carried out a paired accuracy comparison between using resampling-based tuning through 5-fold CV and the fixed setting of 0.95. For CV-based tuning, 16 trial values were used to select the “best” setting. These values range exponentially from 0.5 to 0.994303 using the generating function $CL_i = 1.5 \times e^{-i/22}$, $i = 0..15$. The results of this experiment are shown in Figure 4.7.

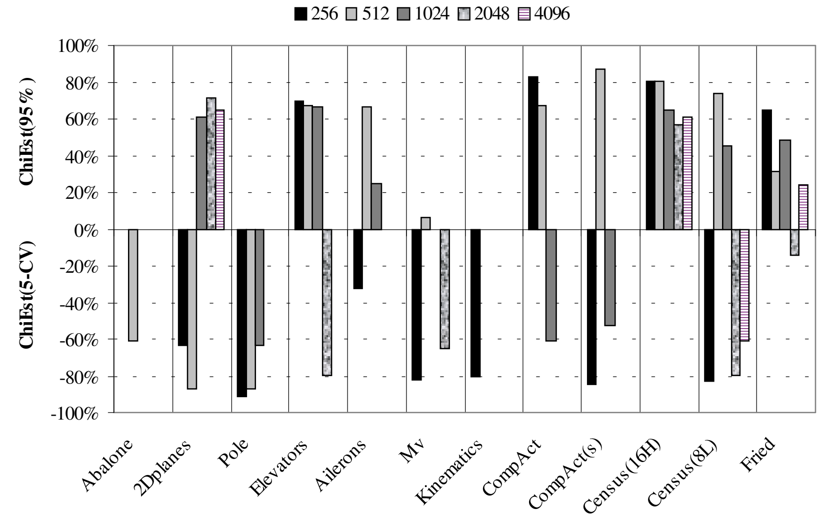


Figure 4.7 - Significance of MSE difference between ChiEst(95%) and ChiEst(5-CV).

This comparison shows that there is no particular advantage in adopting a resampling-based tuning of the confidence level when compared to the fixed set-up of 0.95, at least on these domains. In effect, we have not observed any data set where we could reject with high confidence the hypothesis that both alternatives achieve similar accuracy. Moreover, in several data sets there is a tendency for the fixed setting to perform better. Even more important is the fact that resampling-based tuning is a computationally intensive process, which can be confirmed in Figure 4.8 that shows the tree size and Cpu time ratios between *ChiEst*(CL=95%) and *ChiEst* with the confidence level tuned by a 5-fold CV process.

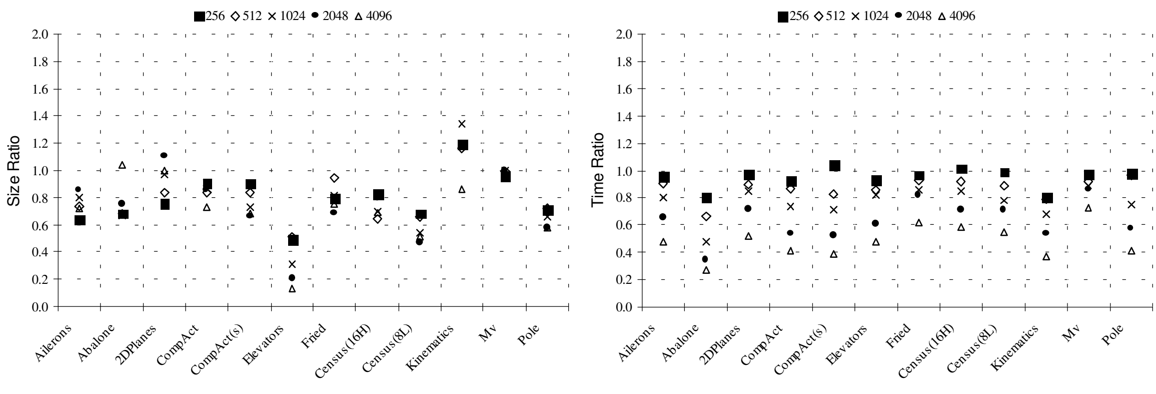


Figure 4.8 - Tree size and Cpu time ratios between ChiEst(95%) and ChiEst(cv).

These results reinforce the argument that for these data sets the fixed value of 95% for the confidence level is the best setting. In effect, not only it has comparable accuracy, but also leads to smaller trees, taking much less computation time. This experimental result is consistent with the graph of Figure 4.6, where we observed that few differences in tree size could be expected for a large range of confidence level values. This may explain why tuning by CV does not produce significantly different results in terms of accuracy from the fixed setting.

Tuning of evaluation based on m-estimates

We now focus on tree evaluation using *m* estimates. With this evaluation mechanism we need to provide the value of the parameter *m*. Setting this value strongly influences the evaluation of candidate trees, thus possibly leading to a different choice of final tree model. We have carried out the same experiment described above for the *ChiEst* method, to observe the behaviour of *m* estimates in the same four domains, when different values of *m* are used. We have varied *m* from 0.05 to 50 in increments of 0.05. Figure 4.9 shows the relative sizes of the selected trees compared with the tree $T_{\max}$ for the different *m* values.

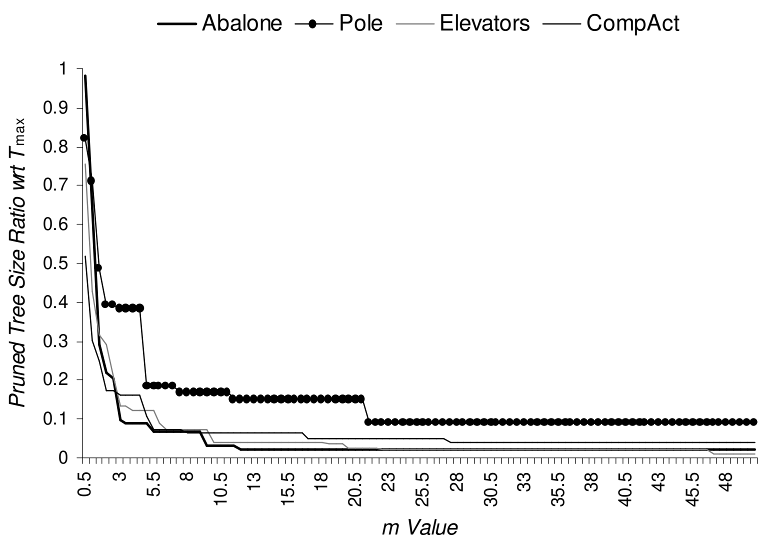


Figure 4.9 - Variation of tree size for different m values.

Figure 4.9 shows that quite different tree sizes can be obtained with slight variations of the *m* parameter value (particularly for small *m* values). Still, the size decreases monotonically with the increase of *m*. This type of monotonous relation was already observed with the coefficient level of *ChiEst* and it is desirable as it can help the user to find the more adequate set-up for his application.

We have also carried out a series of experiments with our benchmark data sets to observe the behaviour of our RT system when using fixed *m* values. We have tried several values for *m* (0.5, 0.75, 1, 2, 3 and 5). Based on the results of these experiments we have observed that while the accuracy results are somehow comparable, there are obvious disadvantages in using small *m* values due to the resulting tree size. Either $m = 2$ or 3 provide the best compromise between size and accuracy on our benchmark data sets. We have compared the results of using the value of 2 for *m* and using 5-fold CV to tune this value for each domain. Figure 4.10 shows the results of this paired comparison. We use 31 trial values of *m* from which the "best" value is selected using 5-fold CV. These values range exponentially from 0.1 to 40.3429 using the generating function $m_i = 0.1 \times e^{i/5}$, $i = 0..30$.

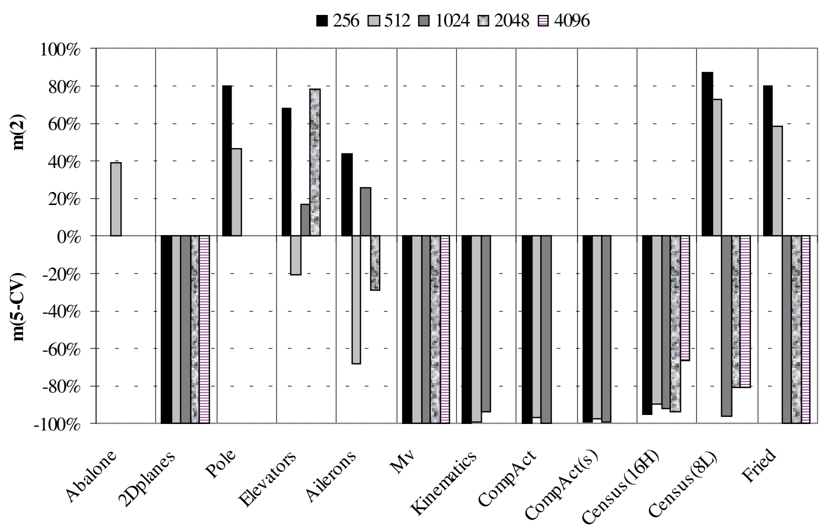


Figure 4.10 - Significance of MSE difference between m(2) and m(5-CV).

As it can be seen in Figure 4.10, tuning *m* by CV leads to significantly more accurate trees on several domains. The results of this experiment show that we can expect with reasonable confidence that tuning *m* by CV is the best strategy for obtaining accurate regression trees post-pruned with *m* estimators.

Figure 4.11 shows the results of this comparison in terms of tree size and computation time ratios. The results in terms of tree size confirm that a fixed value of *m* can be completely inadequate for some domains. Some of the ratios even fall outside of the graph scale (*e.g.* in the *Kinematics* domain using the value of 2 leads to a tree 4 times larger than setting *m* by CV). On other occasions using the value of 2 originates in too simple trees that hardly capture the structure of the domain, leading to poor predictive performance (*c.f.* with the accuracy results on *2Dplanes*, *Mv*, *CompAct*, *CompAct(s)* and *Fried* in Figure 4.10).

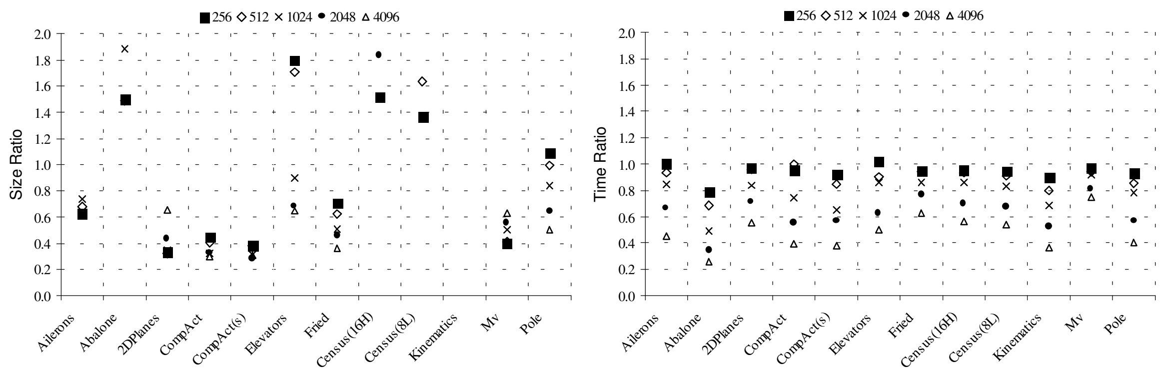


Figure 4.11 - Tree size and Cpu time Ratios for m=2 and m(cv) selection.

With respect to computation times the strategy of tuning *m* by CV has large disadvantage as the sample size grows, which was expected and already happened with the *ChiEst* method.

Tuning of evaluation based on the MDL principle

Finally, we have studied the behaviour of MDL evaluation to identify how it is affected by certain parameters. Here we have considered the parameters that specify the precision of real numbers used for coding the cut-point splits and the errors in the leaves, in accordance with the coding proposed by Robnik-Sikonja & Kononenko (1998). Again using the same four data sets we have post-pruned a large tree using different combinations of values of these two parameters. The size of resulting tree for the different combinations is shown in Figure 4.12.

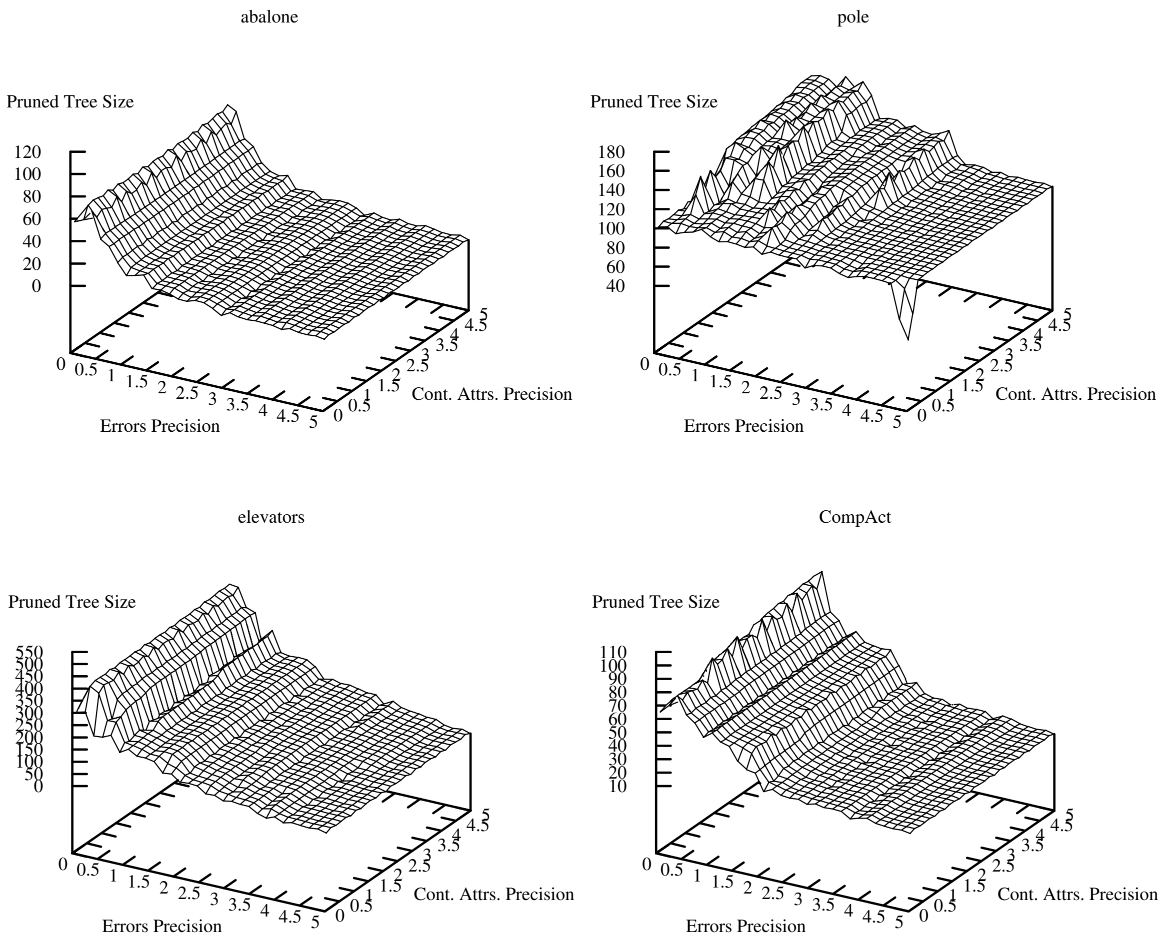


Figure 4.12 - The effect of varying the MDL coding parameters on tree size.

Robnik-Sikonja and Kononenko (1998) claim that the user can easily set the two parameters, as their meaning is intuitive. Although we agree with their position concerning the meaning, the graphs presented show that the effect of varying these values on the size of the resulting selected tree is not always predictable. This is caused by the lack of a clear monotonous relation like the one observed with the parameters of *m* and *ChiEst* estimators, and also by the existence of two parameters instead of a single value to tune.

We have compared a single fixed setting of the two parameters with 5-fold CV tuning. With respect to the fixed setting, after some experimentation, we have selected the value of 0.1 for the precision of the cut-points, and 0.5 for the precision of the errors. This setting seemed to provide the better overall results on our benchmark data sets. Regarding the resampling-based tuning we tried 144 alternatives. These alternatives were generated by exponentially varying the value of the two precision parameters from 0.005 to 7.65 using the function $p_i = 0.005 \times e^{i/1.5}, i = 0..11$. This leads to 12 different precision values per parameter, which after combining originated in the 144 variants (12×12). Figure 4.13 presents the results of this paired comparison using the trees generated by the LSS algorithm as source.

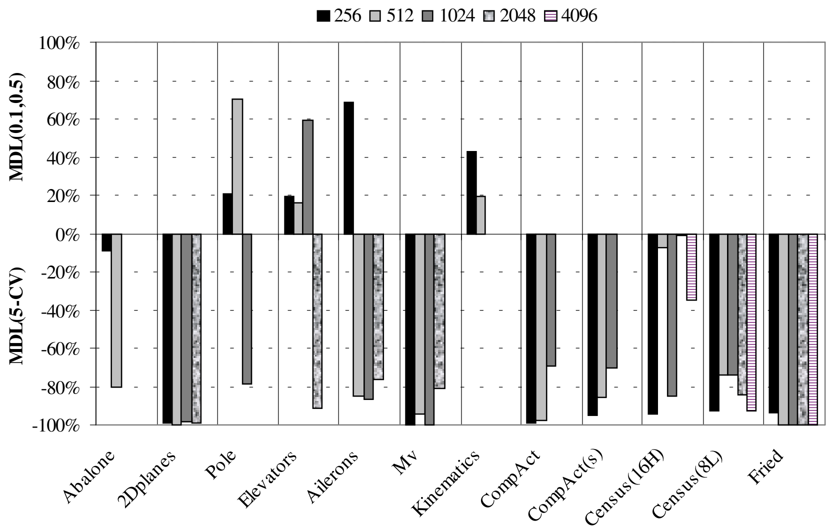


Figure 4.13 - Significance of MSE difference between MDL(0.1,0.5) and MDL(5-CV).

These results lead to the conclusion that CV-based tuning provides a clear advantage in terms of accuracy over this fixed setting on several data sets. The results with respect to tree size and computation time ratios, between MDL with CV-based tuning and the fixed setting are shown in Figure 4.14.

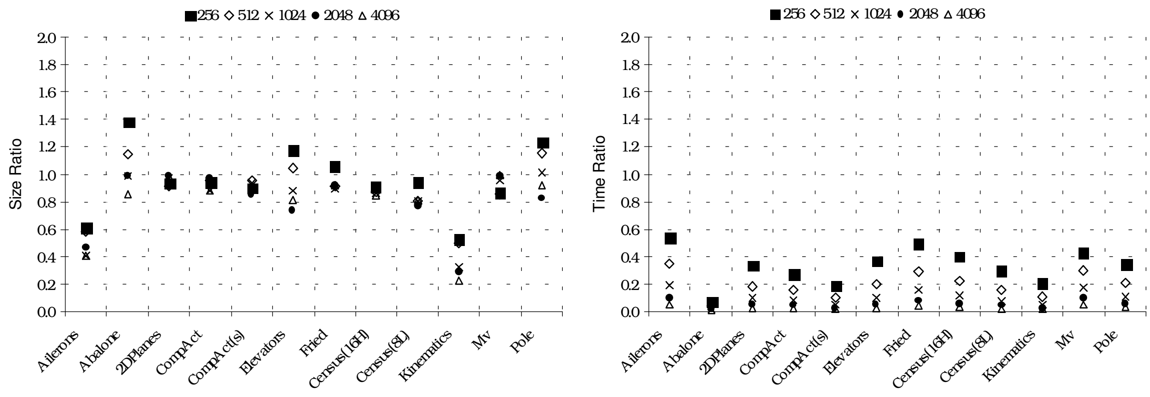


Figure 4.14 - Tree size and Cpu time ratios between MDL(0.1,0.5) and MDL(cv5).

The results in terms of tree size are somehow balanced with a slight advantage of the fixed setting. Regarding computation time we observe that the cost of evaluating the 144 alternatives through 5-fold CV is very high. Still, our experiments indicate that if computation efficiency is not a major concern the best way of using MDL to post prune regression trees is by tuning the precision values using cross validation.

Conclusions regarding tuning of tree evaluation methods

The results of this empirical study of different methods of evaluating trees provide the following indications regarding its use in the context of pruning by tree selection. With respect to *m* estimates and MDL, tuning through resampling is essential to obtain good predictive accuracy in domains with different characteristics. Regarding our *ChiEst* evaluation method, the empirical evidence collected indicates that the method is quite robust to variations on its pruning parameter, and contrary to the other methods we were able to achieve competitive predictive accuracy over all our benchmark data sets using a fixed setting. Although we can not guarantee that this will hold for any data set, this presents an important advantage in terms of computation time as it avoids a costly iterative evaluation process of different alternatives.

Comparing the best settings

We will now present the results of an experimental study whose goal is to determine whether any of the tree evaluation methods is superior to the others. For this purpose we have compared the most promising variants of the different tree evaluation techniques we have considered. Namely, we have compared 5-fold Cross Validation error estimates, with *m* estimates tuned by 5-fold CV, *ChiEst* with 95% as confidence level, and MDL tuned by 5-fold CV. The comparison was carried out using the sequence generated by the *LSS* algorithm as the source for tree selection. Figure 4.15 shows the estimated difference in MSE between 5-fold CV and the other evaluation methods.

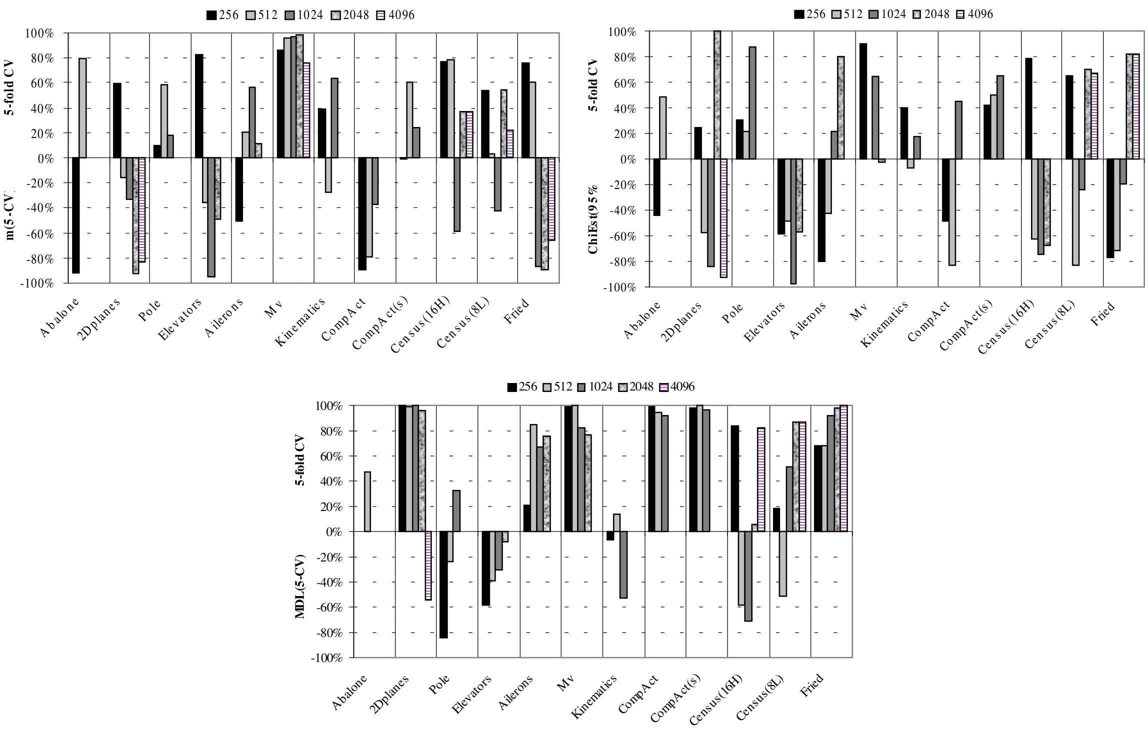


Figure 4.15 - Significance of MSE difference between tree selection methods.

With the exception of MDL selection the differences are most of the times statistically insignificant. Compared to *m* estimates, 5-fold CV has a slight advantage but there are few statistically significant differences. With respect to the comparison with *ChiEst* evaluation, most of the differences are insignificant, but the *ChiEst* method is computationally more efficient as it is the only strategy that grows only one tree. The other methods take more time as they generate and prune several trees, particularly MDL selection tuned by 5-fold CV that needs to evaluate 144 trials (*c.f.* Section 4.3.2.2). Regarding tree size Figure 4.16 shows the ratios between 5-fold CV and the other methods.

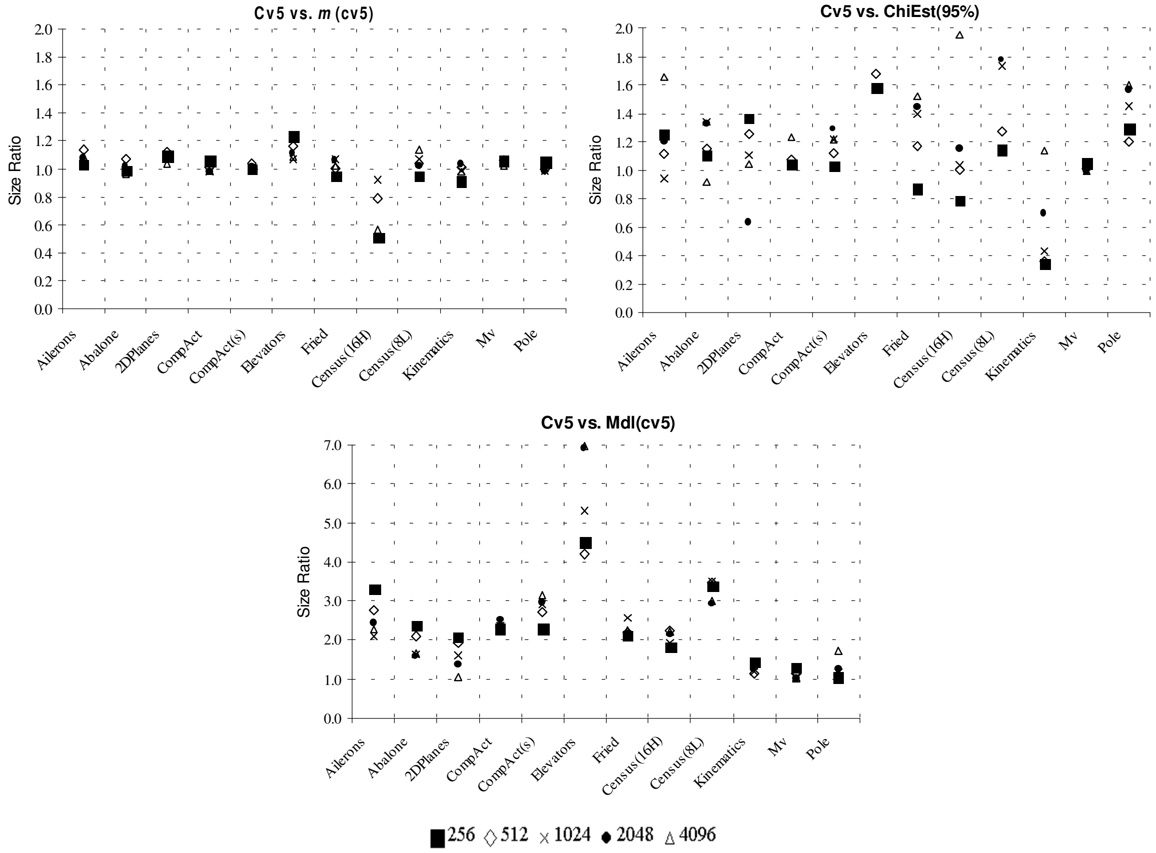


Figure 4.16 - Tree Size Ratio comparison among Selection Methods.

The graphs of this figure show that the preference biases of selection by 5-fold CV estimates and by *m* estimates tuned with CV are very similar. In effect, Figure 4.15 shows that both methods achieve similar accuracy, and Figure 4.16 indicates that the size of the selected trees is also similar. As the computation time of both methods is also comparable there seems to be no particular advantage of one method over the other, at least for the domains considered here. When compared to the *ChiEst* method, 5-fold CV achieves similar accuracy (Figure 4.15), but with trees that are frequently larger as we can observe in Figure 4.16. Moreover, *ChiEst* is much more efficient in terms of computation time as we have already mentioned. This means that for these data sets, both 5-fold CV and *ChiEst* selection have comparable accuracy, but the latter is biased towards smaller trees and it is computationally more efficient. Finally, when compared to MDL selection, 5-fold CV leads to trees that are significantly more accurate in several domains. However, the trees selected by MDL are much smaller as shown in Figure 4.16 (notice the different scale). With respect to computation time CV is preferable, as MDL selection needs to evaluate many trial parameter settings.

4.3.5 Summary

The experimental comparisons carried out in this section have shown that our proposed sequence generation methods (*LSS* and *MCV*) produce more accurate pruned trees (*c.f.* Figure 4.1). Moreover, the tree selection methods we have considered are able to capitalise on this advantage. Thus the use of our tree generation methods proved to be the best form of achieving higher accuracy in pruning by tree selection.

With respect to the evaluation methods we have observed that *m* estimators, *ChiEst* and 5-fold CV have quite similar biases regarding predictive accuracy. However, our *ChiEst* method achieves similar accuracy with smaller trees and much less computation time, which represents an important advantage for large training samples. Regarding selection by MDL we have observed significant losses in predictive accuracy in several domains. Moreover, the method requires a costly tuning process which results in much longer computation times than those of the other methods. However, trees selected by the MDL principle do tend to be significantly smaller, although we can not consider this an advantage in cases where it leads to significant accuracy losses. In effect, looking at these two factors together, we can only consider very interesting the results of MDL selection in both the *Census(16H)* and *Elevators* domains.

Summarising, we can conclude that with the exception of *LSS+m(cv5)* and *LSS+CV5* that behave in a very similar way in all aspects, most of the methods we have evaluated have shown some particular advantage that can be considered an useful bias for some application scenario. Still, when taking the three factors we have considered into account (accuracy, tree size and computation time), we conclude that any of our tree generation methods together with *ChiEst*(*95%*) evaluation provide the best compromise overall for pruning by tree selection.

4.4 Comparisons with Other Pruning Methods

In the previous section we have conducted a thorough study of pruning by selecting from a set of alternative pruned trees. However, as we pointed out in Section 4.2 other pruning methodologies exist. In this section we compare two of the most promising pruning methods we have presented with existing methods of avoiding overfitting in regression trees. Namely, we will compare pruning by tree selection using the *LSS* algorithm together with 5-fold CV and *ChiEst*(95%) evaluation, with CART, RETIS and CORE pruning methods. To ensure a fair comparison of the pruning methodologies all algorithms were applied on the same overly large tree $T_{\max}$. This was made possible because our RT system implements all these pruning variants. With respect to CART pruning we have used as tree selection a 5-fold CV process. For RETIS pruning we have tuned the value of the m parameter using a 5-fold CV process to select from 31 alternatives ranging exponentially from 0.1 to 40.3429 using the generating function $m_i = 0.1 \times e^{i/5}$, $i = 0..30$. Finally, the precision values used in CORE pruning were tuned using 5-fold CV to select from 144 variants obtained using all combinations of 12 values defined by $p_i = 0.005 \times e^{i/1.5}$,$i = 0..11$.

We start by the comparison between our *LSS*+5CV and the other 3 pruning algorithms. Figure 4.17 shows the sign and significance of the observed differences in MSE between our proposal and the others.

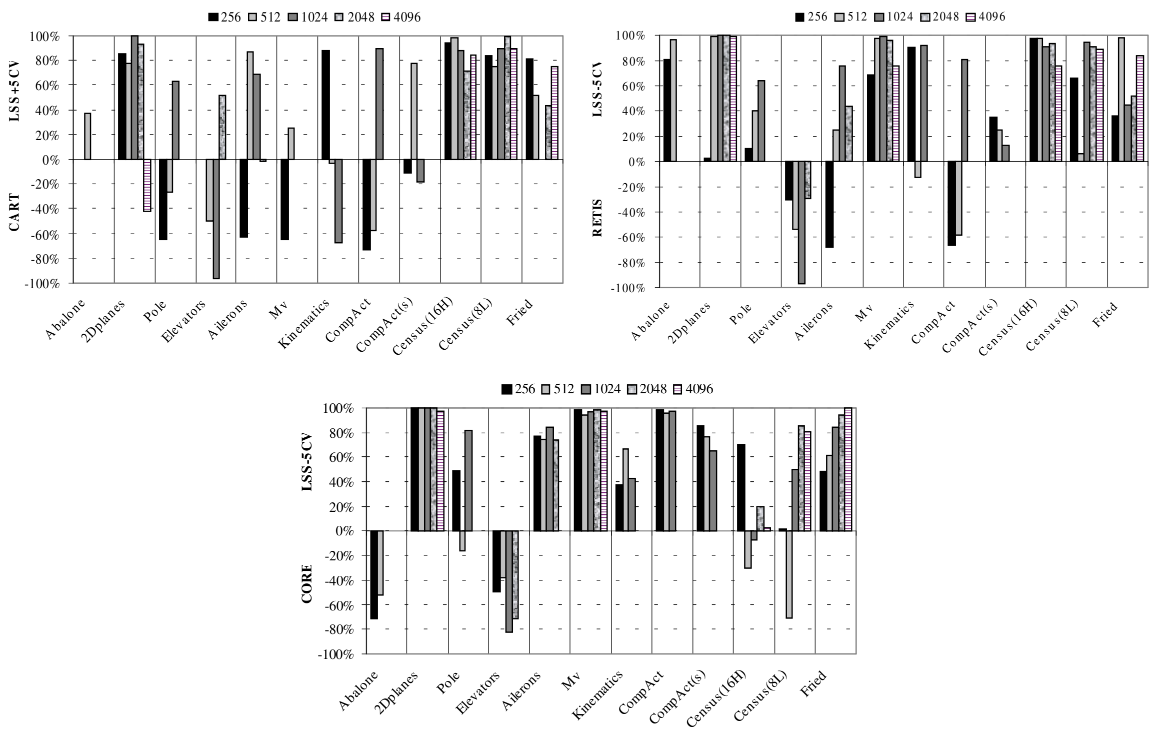


Figure 4.17 - Significance of MSE difference between LSS+5CV and other pruning algorithms.

With the exception of the *Elevators* domain, these graphs show that our pruning method achieves excellent predictive accuracy results when compared to the other pruning strategies. In effect, when considering only the differences that can be regarded as statistically significant, all favour our method. Compared to CART pruning, our method achieves clearly better results in the *2Dplanes*, *Census (16H* and *8L)* and *Fried* domains. The conclusions of the comparison with RETIS pruning are similar although the advantage of our method is more marked and is extended to other domains. In effect, in 39 of the 47[52] experimental set-ups the estimated accuracy difference is favourable to our strategy. With respect to CORE pruning, our method has advantage in 37 of the 47 set-ups, with high statistical significance in several domains. On the contrary, CORE pruning was never found statistically significantly superior to our method, although it achieved better results in both the *Abalone* and *Elevators* domains.

---

[52] Only 47 because from the 12 domains used in our experiments, some of them do not have enough data to carry out the experiments for all sizes we have considered.

Regarding tree sizes the results of the comparison are shown in Figure 4.18.

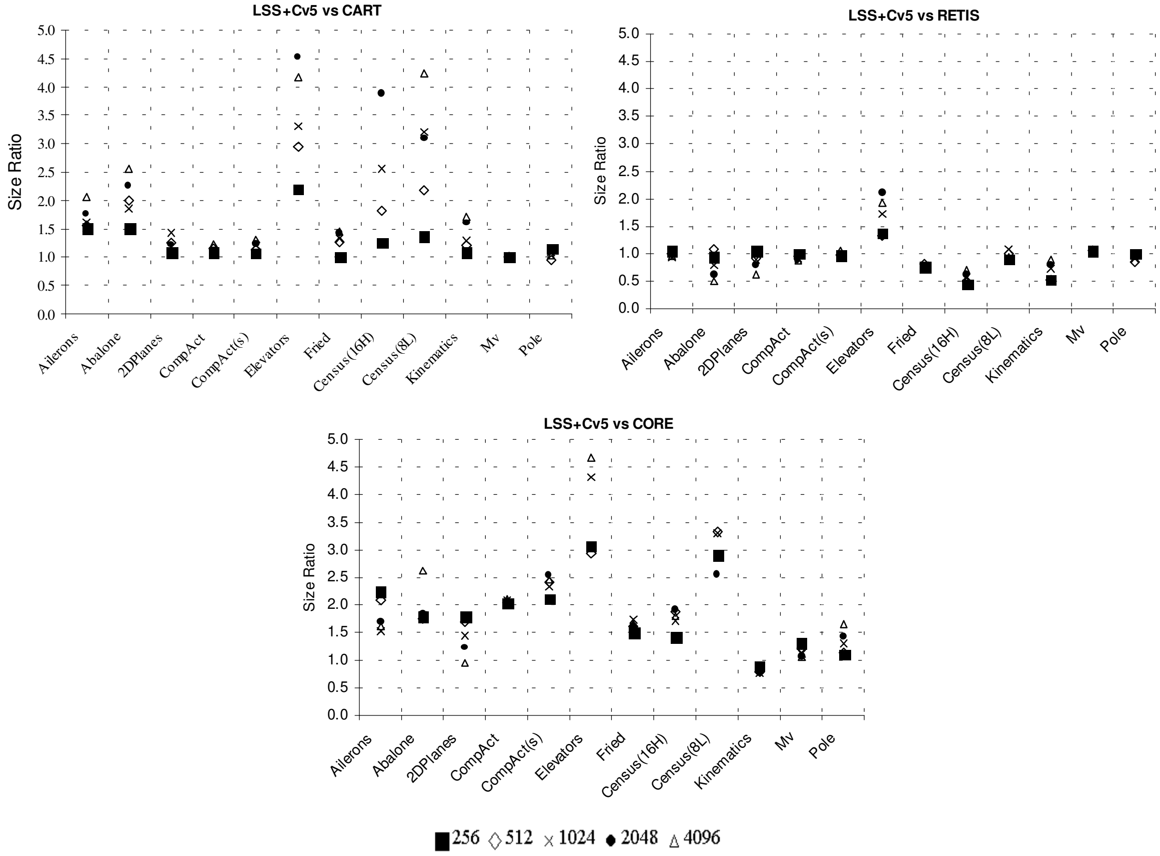


Figure 4.18 - Tree size ratios between LSS+CV5 and other pruning algorithms.

The results of the comparison on tree size indicate that both CORE and CART pruning are clearly biased towards smaller trees. However, we have seen that this benefit comes with a loss of predictive accuracy in several domains, particularly in the case of CORE pruning. With respect to RETIS pruning, our LSS+5CV method has quite similar bias regarding tree size with the exception of the *Elevators* domain. The comparison of computation times revealed similar costs of LSS+5CV, CART and RETIS pruning methods. CORE pruning, however, has significantly larger computation time due to the amount of pruning set-up trials.

With respect to our *LSS+ChiEst(95%)* pruning method, the accuracy comparison with the other three pruning algorithms is shown in Figure 4.19.

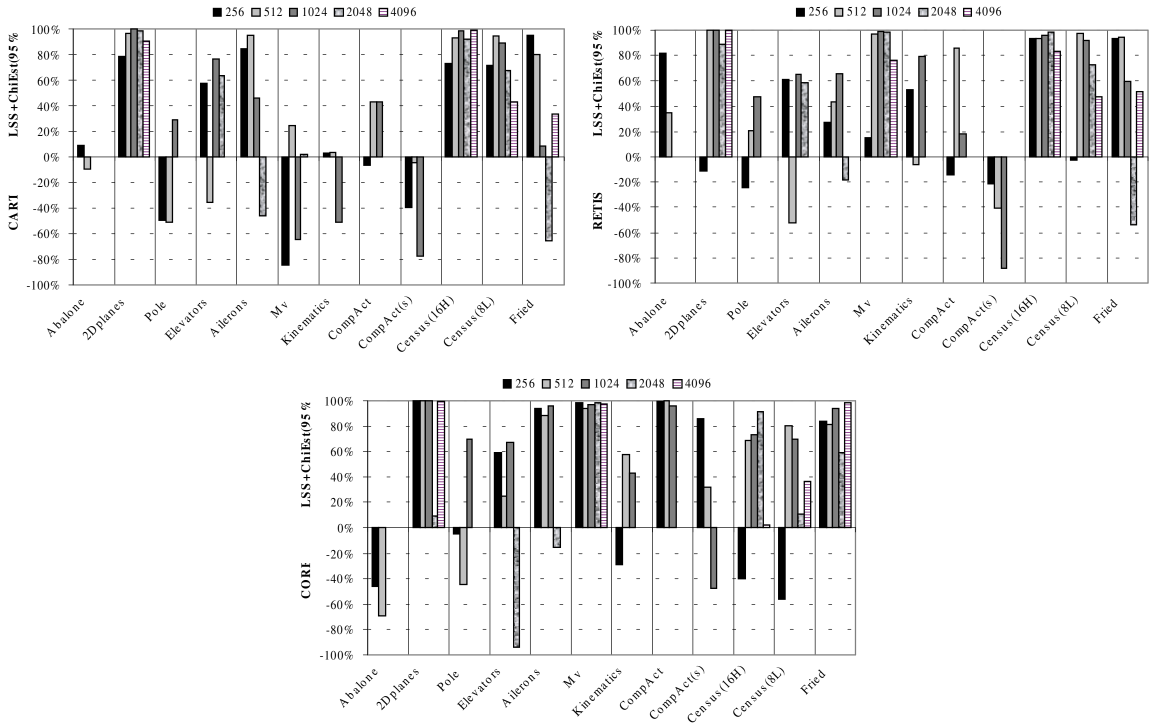


Figure 4.19 - Significance of the MSE difference between LSS+ChiEst(95%) and other pruning algorithms.

Our *LSS+ChiEst(95%)* method also compares quite favourably with the other existing pruning techniques in terms of predictive accuracy on our benchmark domains. Compared to CART pruning, *LSS+ChiEst(95%)* has some difficulties in the *CompAct(s)* domain, although the difference is not statistically significant. It shows advantage in *2Dplanes*, *Census(16H* and *8L)*, *Ailerons, Elevators* and *Fried* domains, often with high significance. Compared to RETIS pruning, the results of our method are even more favourable as it is also significantly better on the *Mv* domain. Finally, compared to CORE pruning, our method has an overall advantage in terms of predictive accuracy with the exception of the *Abalone* data set.

With respect to tree sizes the results of the comparison are shown in Figure 4.20.

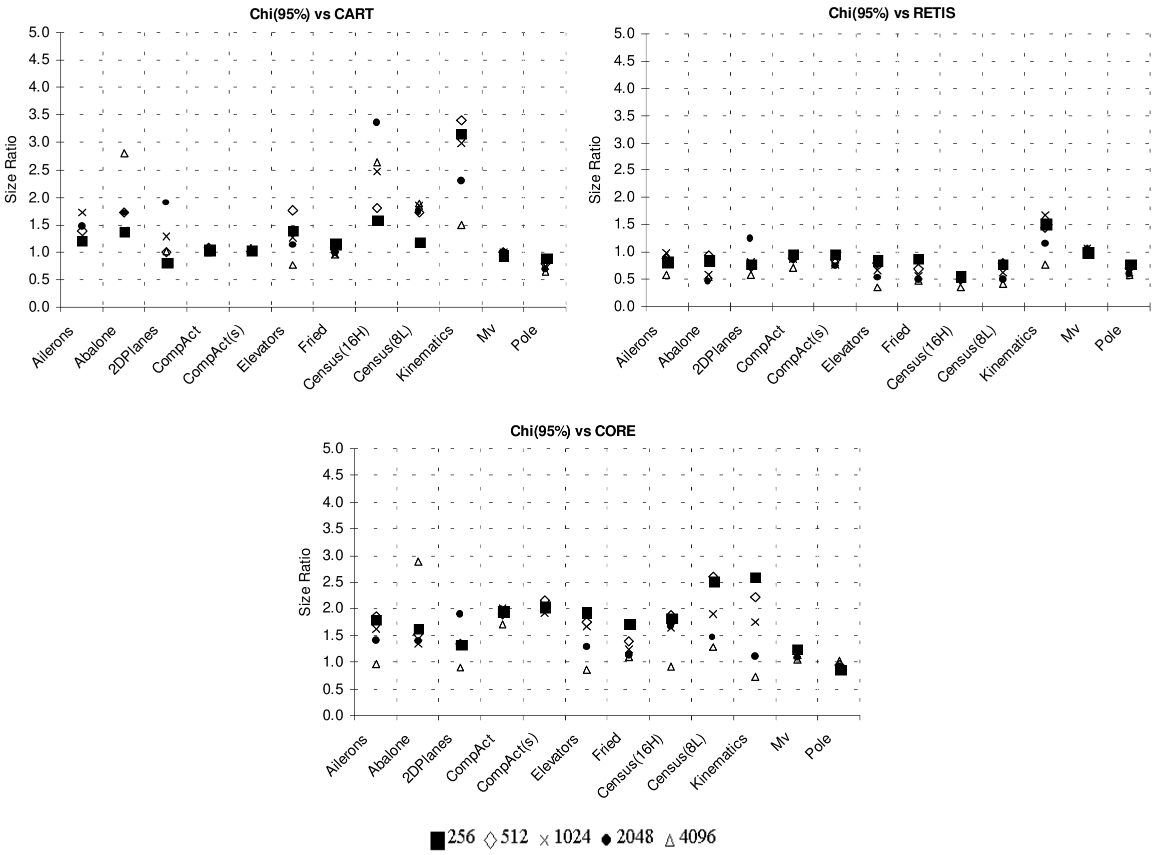


Figure 4.20 - Tree size ratios between LSS+ChiEst(95%) and other pruning algorithms.

Our *LSS+ChiEst(95%)* method is more competitive in terms of tree size than *LSS+5CV*. Still, we continue to observe some disadvantage over CART and CORE pruning in this aspect. Compared to RETIS pruning, *LSS+ChiEst(95%)* has a clear advantage in terms of tree size. Regarding computation time, *LSS+ChiEst(95%)* has an overwhelming advantage as it does not need to learn and prune several trees to tune pruning parameters.

4.4.1 A Few Remarks Regarding Tree Size

In the experiments reported in the previous section our methods clearly did not match the performance of either CART or CORE with respect to tree size. The pruning algorithms of these two systems have a preference bias that favours smaller trees. However, a similar preference bias can be obtained with our methods with the help of the *k*-SE selection rule.

For all the selection methods described in Section 4.3.1 we have given standard error estimates. These allow the use of the *k*-SE rule (Section 4.3.3). The use of this rule will make our methods competitive with CORE and CART pruning in terms of tree size. However, such preference for smaller trees will entail some accuracy loss, as it was the case of CORE and CART pruning. To illustrate this point we present an accuracy comparison of *ChiEst*(95%) using the 0.5-SE and 1-SE rules with CORE pruning.

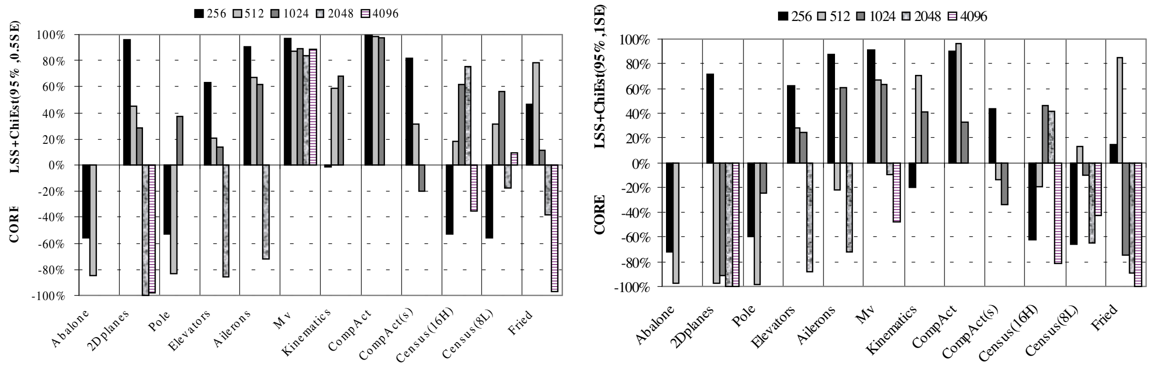


Figure 4.21 - Significance of MSE difference between ChiEst with the k-SE rule and CORE.

Comparing the results to those in Figure 4.19, we confirm the loss of some of the accuracy advantage of our method over CORE pruning. However, the use of this rule can overcome some of the limitations in terms of tree size, as shown in Figure 4.22.

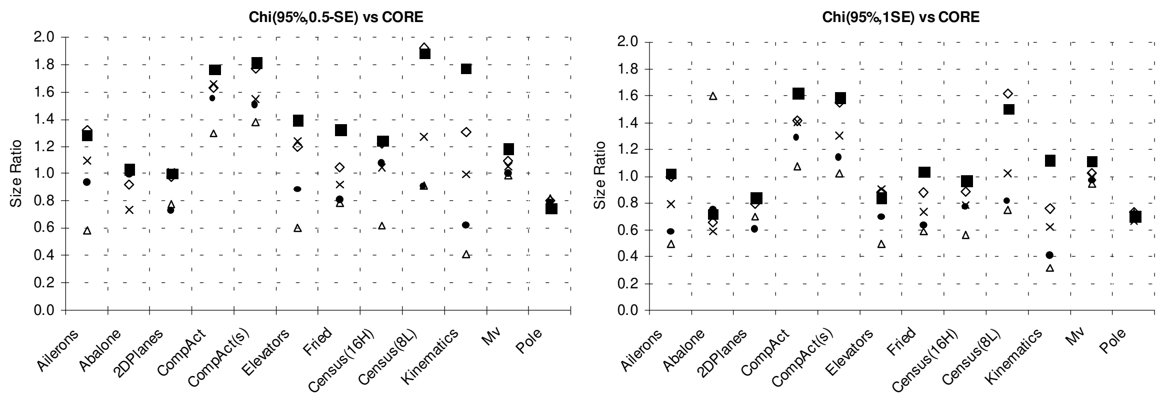


Figure 4.22 - The effect on tree size of the k-SE rule when compared to CORE.

Comparing to the results of Figure 4.20 (notice the different scale) we can see that our method achieves much more competitive results in terms of tree size when employing the SE rule.

4.4.2 Comments Regarding the Significance of the Experimental Results

In this section we have compared two of the most promising pruning by tree selection methods we have presented, with the three most well known methods of pruning regression trees. With respect to predictive accuracy the experiments have shown that our pruning methods achieve better performance on a large set of experimental scenarios. In the light of the arguments of Schaffer (1993a), one may question if it is not the case that the used data sets are just more suited to the preference biases of our methods (*i.e.* are the used domains somehow representative?). In order to answer this reasonable doubt we have carried out a simple experiment in which we obtained a large unpruned tree and compared its accuracy with the accuracy of the tree resulting from pruning it with the CART method. The goal of this experiment is to observe the kind of effect pruning has on all our benchmark domains. The results of this experiment are shown in Figure 4.23.

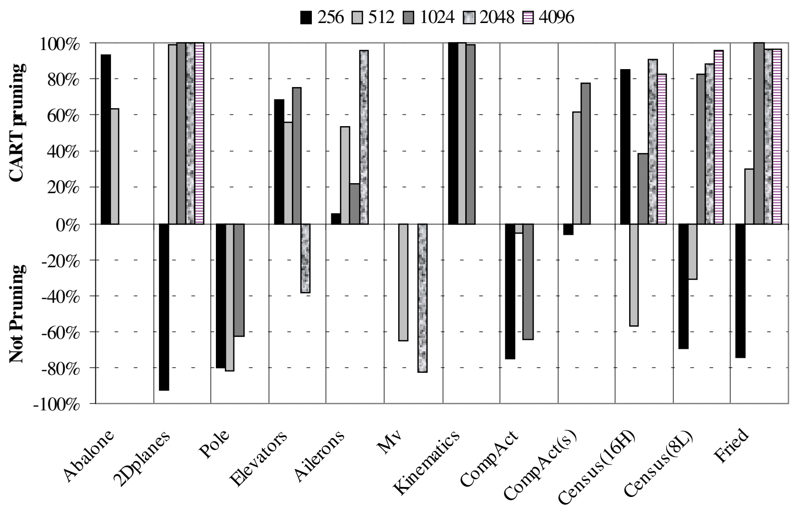


Figure 4.23 - Significance of MSE difference between CART and not pruning.

This graph shows that the "pruning challenges" of our benchmark data sets are quite diverse. In effect, while there are data sets where pruning is clearly beneficial, there are others where that is not so evident (*e.g.* with the *Pole* domain we would do better by not pruning at all!). These results are in agreement with the claims of Schaffer (1993a) on considering pruning as a mere preference bias and not as a statistical mean of achieving higher accuracy. Moreover, the issue whether pruning is beneficial changes with the size of the training samples for several domains. These results indicate that there is a large variety of pruning requirements on our benchmark domains, which increases the confidence on the significance of the accuracy advantages we have observed with our pruning methods. Still, there will obviously exist domains were our methods will perform worse than other pruning algorithms.

4.5 Conclusions

In this chapter we have carried a thorough study of overfitting avoidance within regression trees. We have described the major existing pruning algorithms and presented our approaches to pruning based on tree selection.

We have described several approaches to the generation of sequences of pruned trees and presented two novel methods (*LSS* and *MCV*). Our methods are based on the idea of progressively eliminating nodes where the available sample size does not insure reliable error estimates. The use of this strategy has proven advantageous in our experimental comparisons with existing methods using other strategies, like for instance Error-Complexity sequence generation. We have also studied several techniques for choosing one of such pruned trees. Regarding resampling-based tree selection we have presented a new method of tree-matching, which extends the use of Cross Validation estimates. With respect to selection using *m* estimates we have obtained the standard error of the MSE estimates, which allows the use of the *k*-SE selection rule. Moreover, we have extended the applicability of *m* estimates to LAD regression trees. Finally, we have introduced a new method of estimating the error of a LS tree (*ChiEst*) by using the properties of the sampling distribution of the mean squared error.

We have carried out a systematic experimental evaluation of different ways of generating alternative pruned trees and of selecting the most appropriate one. We have observed that both our two new methods of generating pruned trees and our two new methods of evaluating trees achieved quite competitive results on our benchmark domains. These results are caused by a conjunction of two important factors. Namely, the observation that our generation methods produce more accurate sub-trees and the fact that our tree evaluation methods are able to capitalise on this advantage by correctly ranking the trees according to their estimated prediction error.

We have also compared our most promising pruning algorithms with the three most well known pruning methods. These experiments revealed a marked advantage of our methods in terms of predictive accuracy on several domains. Moreover, we seldom observed the opposite. These advantages need to be weighed with the cost of larger trees. However, through the use of the *k*-SE rule we can minimise this drawback. We have also observed a clear superiority of our method based on the *ChiEst* evaluation in terms of computation time.

4.5.1 Open Research Issues

According to Schaffer (1993a) one of the key research issues within pruning methods is to understand under which conditions are all these techniques beneficial. In particular we would like to know which are the domain characteristics that determine the success of pruning in terms of improving predictive accuracy. In effect, this argument could be extended to learning algorithms in general and not only to pruning methods. One possible path to the solution of this dilemma is to use some kind of meta-learning based on empirical experience with pruning on domains with different characteristics in a similar way as it was done by Brazdil *et al.* (1994). With the obtained meta-knowledge, a pruning algorithm could determine, on the basis of the characteristics of a new domain, which pruning bias would be more adequate. Another possible way is to carry out a theoretical study of the properties of the different pruning methods that would provide better understanding of their applicability. Still, we think that without strong restrictions on the distribution properties of the data sets it will probably be difficult to carry out such study with such highly non-parametric methods as regression trees. Nevertheless, this is clearly an open research question.

APPENDIX.

In this appendix we describe the coding schema proposed by Robnik-Sikonja and Kononenko (1998). These authors code a tree as a sequence of bits encoding each of the tree nodes. For each node they code its type (either a leaf or a split node) and the contents of the node (either the split or the model in the leaf). The code length of a regression tree is the number of bits of this sequence. To code the type of node we only need a single bit indicating if the node is a split node or a leaf. The coding of the node contents depends on the type of node. The binary code of a leaf node consists of the coding of the model (*e.g.* the average *Y* value) followed by the coding of the errors committed by that model. Both the model and its errors are real numbers. For instance, suppose that in a leaf of a LS regression tree we have a set of training cases with the following *Y* values: { 25, 30, 60, 70 }, corresponding to an average of 46.25. The number of bits necessary to code this leaf is equivalent to the length of the encoding of the following real numbers,

$$CodeLen(46.25) + CodeLen(46.25 - 25) + CodeLen(46.25 - 30) + \ldots$$

The coding of real numbers is done following Rissanen (1982). The real number is divided by the required precision ε, and the resulting integer is then coded as a binary string. The code length of the bit string corresponding to a given integer is determined according to the following formulae,

$$CodeLen(0) = 1$$

$$CodeLen(n) = 1 + \log_2(n) + \log_2(\log_2(n)) + \ldots + \log_2(2.865064)$$

where the summation includes only the positive terms.

The computational complexity of calculating the binary description length of a leaf appears large as we need to run through all cases in the leaf to calculate their prediction error and the corresponding code length. However, this can be done with almost no computation cost, during the learning phase. In effect, during this stage we need to run through all cases when creating the nodes and calculating the resubstitution errors, so we can use these cycles to calculate the binary code lengths.

With respect to split nodes their coding depends on the type of split. The first part of the code makes this distinction, while the following bits correspond to the coding of the split. Robnik-Sikonja and Kononenko (1998) describe the coding for several types of split nodes. For instance, to code nominal splits with the form $X_i \in S$, where S is a set of values belonging to χ_i, we have to code information regarding which attribute is being tested and of the set of values in the split. This set of values can easily be represented by a bit string with length corresponding to the possible number of values of the attribute. This leads to the following code length for a nominal split,

$$CodeLen(X_i \in S) = \log_2(\#A) + \#\chi_i \tag{4.31}$$

where A is the set of attributes.

For continuous variable splits the reasoning is similar but instead of the subset of values it is necessary to code a cut-point within the range of values of the attribute being tested. This leads to the following code length,

$$CodeLen(X_i \leq V) = \log_2(\#A) + \log_2\left(\frac{range(X_i)}{\varepsilon}\right) \tag{4.32}$$

where,

$range(X_i)$ is the range of values of the variable X_i ;
and ε is the wanted precision for the cut-points.

Robnik-Sikonja and Kononenko (1998) also describe the coding of splits consisting of conjunctions of conditions and of linear formulae.

The coding schema described above has two parameters, namely the precision used to code the errors at the leaves and the cut-points of continuous splits. The authors suggest using different precision values for these numbers. Robnik-Sikonja and Kononenko (1998) claim that setting these parameters is intuitive for the user depending on his application. Still, CORE is able to use a cross validation process to automatically tune the parameters from a large set of alternative values, selecting the values that ensure better estimated predictive accuracy.

Chapter 5

Local Regression Trees

In this chapter we explore the hypothesis of improving the accuracy of regression trees by using smoother models at the tree leaves. Our proposal consists of using local regression models to improve this smoothness. We call the resulting hybrid models, local regression trees. Local regression is a non-parametric statistical methodology that provides smooth modelling by not assuming any particular global form of the unknown regression function. On the contrary these models fit a functional form within the neighbourhood of the query points. These models are known to provide highly accurate predictions over a wide range of problems due to the absence of a "pre-defined" functional form. However, local regression techniques are also known by their computational cost, low comprehensibility and storage requirements. By integrating these techniques with regression trees, not only we improve the accuracy of the trees, but also increase the computational efficiency and comprehensibility of local models. In this chapter we describe the use of several alternative models for the leaves of regression trees. We study their behaviour in several domains an present their advantages and disadvantages. We show that local regression trees improve significantly the accuracy of "standard" trees at the cost of some additional computational requirements and some loss of comprehensibility. Moreover, local regression trees are also more accurate than trees that use linear models in the leaves. We have also observed that local regression trees are able to improve the accuracy of local modelling techniques in some domains. Compared to these latter models our local regression trees improve the level of computational efficiency allowing the application of local modelling techniques to large data sets. Finally, we have compared local regression trees with three state-of-the-art commercial regression systems, and observed that our models are quite competitive over a wide range of domains.

5.1 Introduction

In this chapter we propose to improve the accuracy of regression trees by using local models in their leaves, which leads to smoother approximations. The motivation for the integration of these two different regression methodologies lies in existing work on model selection and combination. In effect, extensive experimental (*e.g.* Michie *et. al.* 1994) and theoretical (*e.g.* Schaffer, 1994) results have demonstrated that it is difficult to identify a single best inductive method when considering a diverse set of problems. This has been termed the *selective superiority problem* by Broadley (1995) and results from the fact that different algorithms use different preference bias in the search for a model of the data. As proved by Schaffer (no free lunch theorem) it is always possible to find a data set for which any preference bias is inadquate. The same kind of difficulty arises when searching for the best parameter setting for a particular data set within a single learning method. Different parameter settings lead to different models and we need to select one of these. Moreover, there is no clear best setting over all possible domains. There have been two main distinct pathways for dealing with the selective superiority problem: model selection and model combination. Before presenting local regression trees that integrate ideas from local modelling and regression trees, we briefly describe the existing work on these two main strategies for addressing the selective superiority problem.

Model Selection through Resampling

One way of trying to address the selective superiority problem is to use resampling to perform a data-oriented model selection. Using resampling we can obtain estimates of the performance on unseen data of the different models and use these estimates to carry out a selection of the more adequate model (Schaffer, 1993b). However, this can be tedious, if the number of different algorithms is large. Moreover, resampling methods are always limited by the amount of "information" present in the given training data, which may lead to wrong decisions (Schaffer, 1993a). We have confirmed this by observing that pruning guided by resampling estimates may lead to worst trees than not pruning at all (*c.f.* Figure 4.23), meaning that the estimates were misleading.

Model Selection through Meta-learning

Another alternative for addressing the problem of selecting the most adequate algorithm for a given problem consists of trying to obtain meta-knowledge that somehow characterises the applicability of different systems (Aha, 1992; Brazdil *et. al.* 1994; Gama & Brazdil, 1995). Gama and Brazdil (1995) obtained this meta-knowledge with the help of an inductive algorithm. The data used in this meta-learning task contained information of previous experiments carried out with the alternative systems together with a set of attributes characterising each individual data set[53].

Combining Predictions of different Models

One way of dealing with the fact that different algorithms use different preference bias that can be useful in any domain, is to combine the predictions of the resulting models. Earlier work related to the combination of different predictions exploited the notion of redundancy. These notions are related because when we have redundant knowledge we need to solve conflicts and combine evidence (Torgo, 1993b). Cestnik and Bratko (1988) induced a set of redundant rules each characterised by a credibility score, which were then

---

[53] A recently approved European Community research project called METAL (http://www.ncc.up.pt/liacc/ML/METAL/) is expected to give a major boost to this research line.

used to classify new objects. This idea of redundancy was taken further in the work by Gams (1989), Buntine (1990), Kononenko (1991) and Torgo (1993a), among others. Following this initial thread, a series of works with stronger theoretical foundations has emerged. Among these we can mention Stacked Generalization (Wolpert, 1992), Boosting (Freund,1995) and Bagging (Breiman, 1996). All these approaches proved to be successful in terms of leading to a significant accuracy increase. However, all of them have drawbacks in terms of model comprehensibility. In effect, we no longer obtain a single model that can be used to "explain" the predictions.

Combining Models instead of Predictions

Brazdil & Torgo (1990) combined individual models producing a single integrated theory, instead of combining their predictions. Their INTEG system used randomly obtained sub-samples of the training set to generate several alternative models using different learning algorithms. In a subsequent phase the individual models were translated into a common rule-based representation language. Each rule was evaluated using an independent set of data and this evaluation was used to build a single integrated model. The authors reported significant accuracy gains using this method on some benchmark data sets, although this approach requires that certain assumptions have been met. The first is the necessity of a common model representation language to which all individual models must be translatable. The second is the requirement of a separate evaluation sample. While the latter can be considered more or less irrelevant in the light of the current trend in data set sizes, the former could be a problem for some kind of systems (*e.g.* a neural network). Still, this approach has the key advantage of producing a single comprehensible and accurate model of the given training set. Domingos (1997) presented a somewhat related approach that also generates a single final model. His CMM system starts by generating multiple models using variations of the training set. In a second step CMN creates a new training set consisting of all original cases plus a new set of cases appended at the end. This new data consists of randomly generated cases with the goal variable value given by the combined classification of the former individual models. This new training set is then used in a kind of second-level learning step that produces the final model.

Another alternative form of combining models consists of performing a tighter integration of the methodologies. Instead of integrating the models generated by different algorithms, we somehow integrate the methodology behind the algorithms aiming at being able to take advantage of their different preference biases. For instance, the RISE system (Domingos, 1996) integrates instance-based learning with rule-based induction. The integration is achieved by using a single algorithm that does not distinguish between rules and cases and uses the same representation language for both models. Another example of tight integration occurs when multiple models are used to extend the representation language of the learning systems. Gama (1998) describes a methodology that performs a kind of constructive induction that extends the number of attributes describing each case using other models to generate probability class distributions. In the more general framework of Cascade Generalisation (Gama, 1998), several learning systems are applied one after the other using increasingly more complex descriptions of the cases.

In summary, all these systems using tighter integration schemas try to take advantage of the different biases of several basic learning systems. This is the same motivation guiding the work described in this chapter, where we integrate local modelling with regression trees. Section 5.3 describes how we perform such integration. Before presenting such description we analyse in detail local modelling techniques.

5.2 Local Modelling

In Section 2.4.1.2 we have briefly described a set of regression methods usually known as local models that belong to a class of so called 'lazy learning' algorithms. These techniques have as main distinctive feature the fact that they do not obtain any comprehensible model of the given training data that could be stored for future use. In effect, the training phase of such methods consists basically of storing the training instances. Given a query case, local modelling methods search for the training cases in the local neighbourhood of the query case and use them to obtain the prediction. The fact that no rigid global functional form is assumed makes these techniques highly flexible as they can theoretically fit any regression curve.

5.2.1 Kernel Models

Kernel regression (Watson, 1964; Nadaraya, 1964) is a well-known local modelling method. Predictions for query cases are obtained by averaging over the *Y* values of the most similar training cases. The central issue of these models is thus the notion of similarity, which is determined using a particular metric over the input space. Given such metric we can calculate the distance between any two cases. Different distance functions exist like the Euclidean distance, L_p norms, etc. (see for instance Atkeson *et al.*, 1997 for an overview). In our work here we use an Euclidean function defined as,

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{v=1}^{a} w_v \times \delta(\mathbf{X}_{i,v}, \mathbf{X}_{j,v})^2} \tag{5.1}$$

where,

$\mathbf{x}_i$ is an instance vector; $\mathbf{X}_{i,v}$ is the value of variable v on instance i;
w_i is the weight of variable i;
and $\delta(v_1, v_2)$ is the distance between two variable values.

In order to avoid overweighing of discrete variables with respect to continuous variables we use a ramping function (Hong, 1994) for the latter. This leads to the following definition of distance between two variable values:

$$\delta(v_1, v_2) = \begin{cases} 0 & \text{if nominal variable and } v_1 = v_2 \\ 1 & \text{if nominal variable and } v_1 \neq v_2 \\ 0 & \text{if numeric variable and } |v_1 - v_2| \leq T_{eq} \\ 1 & \text{if numeric variable and } |v_1 - v_2| \geq T_{diff} \\ \left(|v_1 - v_2| - T_{eq}\right) \big/ \left(T_{diff} - T_{eq}\right) & \text{if numeric variable and } T_{eq} < |v_1 - v_2| < T_{diff} \end{cases} \tag{5.2}$$

where,

T_{eq} is a threshold for considering two numeric values equal;
and T_{diif} is a similar threshold for difference.

The distance between two numeric values can be better illustrated by Figure 5.1:

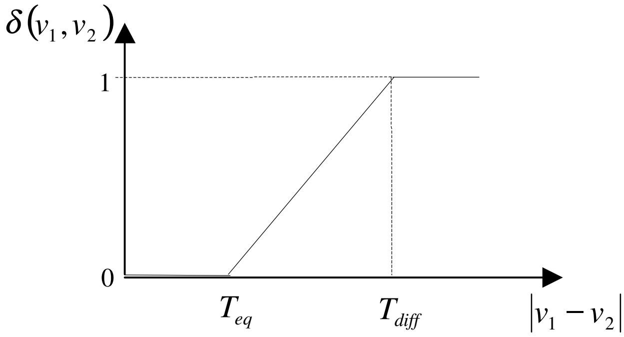


Figure 5.1. The ramping function (Hong, 1994).

Another important issue in the design of a distance function is feature weighing. Distance-based learners like kernel models can be quite sensitive to the presence of irrelevant features. These irrelevant variables can distort the distance between two cases. Feature weighing can help to reduce the influence of irrelevant features by better "tuning" the distance function. Several different approaches exist in the literature (see Wettscherek *et. al.*, 1997, for a review in the context of *k*-Nearest Neighbours). Recently, Robnik-Sikonja & Kononenko (1996) presented a method for estimating variable relevance in the context of regression, based on a previous work on *Relief* (Kira & Rendell, 1992; Kononenko, 1994). They described an algorithm called *RReliefF* and successfully applied it in the context of regression trees. We use this method for estimating the weights of the variables that are used in distance calculations.

The distance function defined above can be used to calculate the distance between a query point and any case belonging to the training data. Kernel models obtain the prediction for a query case using a weighed average of the *Y* values of the training cases within a certain neighbourhood. The size of this neighbourhood is another key issue of kernel regression and local modelling in general. This size is determined by what is usually known as the *bandwidth parameter, h.* Many alternatives exist in the vast literature of kernel modelling (*e.g.* Cleveland & Loader, 1995) that describe how this parameter should be set. The simplest strategy consists of using a maximum distance to the test case $\mathbf{x}_q$ as the bandwidth value (Fan & Marron, 1993). Nearest neighbour bandwidths (Stone,1977; Cleveland,1979), choose the distance to the k^{th} nearest training case as the bandwidth size. This was the method we have adopted. Other hypothesis include the use of optimisation processes that find the best value of the bandwidth either for each test point or globally (*e.g.* Atkeson *et al.*,1997).

All the training points within the specified bandwidth are used to calculate the prediction for a given query case. However, they enter with different weights. Training points that are nearer to the test case are given more importance. This weighing schema is accomplished through the use of what is usually known as the *kernel* (or *weighing*) *function*. We use a gaussian kernel function with the form,

$$K(d) = e^{-d^2} \tag{5.3}$$

where, d is the distance between the query point and the training cases under consideration.

Atkeson *et. al.* (1997) claim that the choice of the kernel function is not a critical design issue as long as the function is reasonably smooth. Still, other alternatives include tricube functions (Cleveland,1979), quadratic functions, etc. Notice that when the bandwidth parameter is set to the distance of the k^{th} nearest training case and we use a uniform kernel[54], kernel regression is in effect a k Nearest Neighbour model (Fix & Hodges,1951; Cover & Hart,1967).

Having defined the major design issues of kernel regression we can obtain the prediction for query point $\mathbf{x}_q$ using the following expression,

$$r(\mathbf{x}_q) = \frac{1}{SKs} \sum_{i=1}^{n} K\left(\frac{d(\mathbf{x}_i, \mathbf{x}_q)}{h}\right) \times y_i \tag{5.4}$$

where,

$d(.)$ is the distance function between two instances (Equation 5.1);
$K(.)$ is a kernel function;

[54] A uniform kernel function gives the same weight to all cases within the bandwidth.

h is a bandwidth value;
<$\mathbf{x}_i$, y_i> is a training instance;
and SKs is the sum of the weights of all training cases, *i.e.* $SKs = \sum_{i=1}^{n} K\left(\frac{d(\mathbf{x}_i, \mathbf{x}_q)}{h}\right)$.

5.2.2 Local Polynomial Regression

Local polynomial regression is a generalisation of kernel regression. In effect, kernel regression can be regarded as fitting a polynomial of degree zero (a constant) in the neighbourhood of a given query point. Kernel regression is known to suffer from the lack of symmetry near the input space boundaries (Hastie & Loader, 1993). These difficulties lead to the development of local polynomial regression where a polynomial of degree p is fitted in the neighbourhood of the query point (Stone, 1977; Cleveland, 1979; Katkovnik, 1979). This includes kernel regression (p=0), local linear polynomials (p=1), and other settings. Within our work on local regression trees we use polynomials of degree 1, that is local linear polynomials.

Fitting a global linear polynomial using a least squares error criterion consists of finding the vector of parameters $\boldsymbol{\beta}$ that minimises the sum of the squared error, *i.e.* $(\mathbf{Y} - \mathbf{X}\beta)'(\mathbf{Y} - \mathbf{X}\beta)$, where $\mathbf{X}'$ denotes the transpose of matrix $\mathbf{X}$. After some matrix algebra the minimisation of this expression with respect to $\boldsymbol{\beta}$ leads to the following set of equations, usually referred to as the *normal equations* (*e.g.* Draper & Smith, 1981),

$$(\mathbf{X}'\mathbf{X})\beta = \mathbf{X}'\mathbf{Y} \tag{5.5}$$

The parameter values can be obtained solving the equation,

$$\beta = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y} \tag{5.6}$$

where $\mathbf{Z}^{-1}$ denotes the inverse of matrix $\mathbf{Z}$.

As the inverse matrix does not always exist this process suffers from numerical instability. A better alternative (Press *et al.*, 1992) is to use a set of techniques known as *Singular*

Value Decomposition (*SVD*), that can be used to find solutions of systems of equations with the form $\mathbf{X\beta} = \mathbf{Y}$.

Regarding local linear polynomials the main difference with respect to global linear regression, is that each training point has an associated weight that is a function of its distance to the query point. In order to fit a local linear polynomial given a query point $\mathbf{x}_q$, we need to find the model parameters that minimise,

$$\sum_{i=1}^{n} \left(Y_i - \beta\ \mathbf{x}_i\right)^2 K\left(\frac{d\left(\mathbf{x}_i, \mathbf{x}_q\right)}{h}\right) \tag{5.7}$$

where,

$K\left(\frac{d\left(\mathbf{x}_i, \mathbf{x}_q\right)}{h}\right)$ is the weight associated to each training case;
$d(.)$ is a distance function;
$K(.)$ a kernel function;
and h the bandwidth.

According to Draper & Smith (1981, p.109) in the case of weighed least squares the normal equations are,

$$\left(\mathbf{X}'\mathbf{W}\mathbf{X}\right)\boldsymbol{\beta} = \mathbf{X}'\mathbf{W}\mathbf{Y} \tag{5.8}$$

where, $\mathbf{W}$ is a diagonal matrix of weights, *i.e.*, $\mathbf{W} = \text{diag}(w_1, w_2, \ldots, w_n)$.

Let us define the following auxiliary value for each training case,

$$v_i = \sqrt{K\left(\frac{d\left(\mathbf{x}_i, \mathbf{x}_q\right)}{h}\right)} \tag{5.9}$$

Using matrix notation the following relation holds,

$$\mathbf{V}'\mathbf{V} = \mathbf{W} \tag{5.10}$$

where,

$\mathbf{V} = \text{diag}(v_1, v_2, \ldots, v_n)$;
and $\mathbf{W} = \text{diag}(w_1, w_2, \ldots, w_n)$.

Using this relation and Equation 5.8 we have,

$$\left(\mathbf{X}'\mathbf{V}'\mathbf{V}\,\mathbf{X}\right)\boldsymbol{\beta} = \mathbf{X}'\mathbf{V}'\mathbf{V}\,\mathbf{Y} \tag{5.11}$$

or by a change of variables,

$$(\mathbf{Z}'\mathbf{Z})\beta = \mathbf{Z}'\mathbf{Q} \tag{5.12}$$

where,

$\mathbf{Z} = \mathbf{VX}$;
and $\mathbf{Q} = \mathbf{VY}$.

This latter equation is in the same format as Equation 5.5, which means that it can be solved using the same numerical procedures (*i.e.* inversion of **Z** or through the SVD method). This means that to obtain the solution for a local linear polynomial given a query point, we just need to calculate the auxiliary weights, v_i (Equation 5.9), and multiply these weights by the matrices **X** and **Y**, obtaining matrices **Z** and **Q** that can be feed in the SVD routine[55].

We also use a backward elimination technique (*e.g.* Drapper & Smith, 1981) to simplify the models obtained through the SVD procedure. This is an iterative technique for progressively eliminating terms of a linear polynomial on the basis of pre-defined thresholds of a *t*-Student test.

5.2.3 Semi-parametric Models

The main idea behind semi-parametric models (*e.g.* Hardle, 1990) is to incorporate in a single model both local (non-parametric) and parametric components. For instance, *partial linear models* (Spiegelman, 1976; Hardle, 1990) integrate a linear component with a kernel model. A partial linear model can be described by,

$$Y = \boldsymbol{\beta}\mathbf{X} + m(\mathbf{x}) \tag{5.13}$$

where,

$\boldsymbol{\beta}\mathbf{X}$ is a least squares linear model with parameters $\boldsymbol{\beta}$;
and $m(.)$ is a kernel regression model.

---

[55] Notice that if we include all training cases in the matrix **X**, and use a uniform weight for all cases, we have in effect a global least squares linear regression model.

One way of using these models is to first generate the parametric component of the model (*i.e.* generate the global linear polynomial) and then apply kernel regression on the residuals (errors) of this model. Here we follow this approach. We start by calculating the linear model using the standard SVD technique. Given a query point we identify the training points within the specified bandwidth. For each of these training cases we calculate the residuals of the linear model,

$$r_i = \boldsymbol{\beta}\mathbf{x}_i - y_i \,, \quad \text{for each } \langle \mathbf{x}_i, y_i \rangle \in \text{bandwidth of } \mathbf{x}_q \tag{5.14}$$

A kernel prediction for the residual of the query point is then obtained with these "training" residuals. Finally, the predicted residual is added to the linear model prediction for the query point, giving the partial linear model prediction. Formally, this corresponds to,

$$r(\mathbf{x}_q) = \boldsymbol{\beta}\mathbf{x}_q - \frac{1}{SKs}\sum_i^n K\left(\frac{d(\mathbf{x}_i, \mathbf{x}_q)}{h}\right) \times (\boldsymbol{\beta}\mathbf{x}_q - y_i) \tag{5.15}$$

where, *SKs* is the sum of the kernel weights.

These models can be seen as performing a local "correction" on the global linear model predictions represented by $\boldsymbol{\beta}\mathbf{x}_q$.

5.3 Integrating Local Modelling with Regression Trees

The methods described in Chapter 3 for inducing a regression tree assume constant values in the tree leaves. In the case of LS regression trees this constant is the average *Y* value of the training cases, while in the case of LAD trees it is the median. We have seen that these constants minimise the respective error criteria (Theorem 3.1 and Theorem 3.2). Regression functions usually have a certain degree of smoothness. Moreover, they tend to be continuous. The approximation provided by regression trees is a kind of histogram-type surface. As an example suppose we want to model the function $Y = \sin(X)$. Let us assume that the only information we give a regression tree learner is a set of 100 cases randomly generated using the function $Y = \sin(X) + N(0,1)$, where $N(0,1)$ is a gaussian noise function with average 0 and variance 1. The approximation provided by a LS regression tree is shown in Figure 5.2:

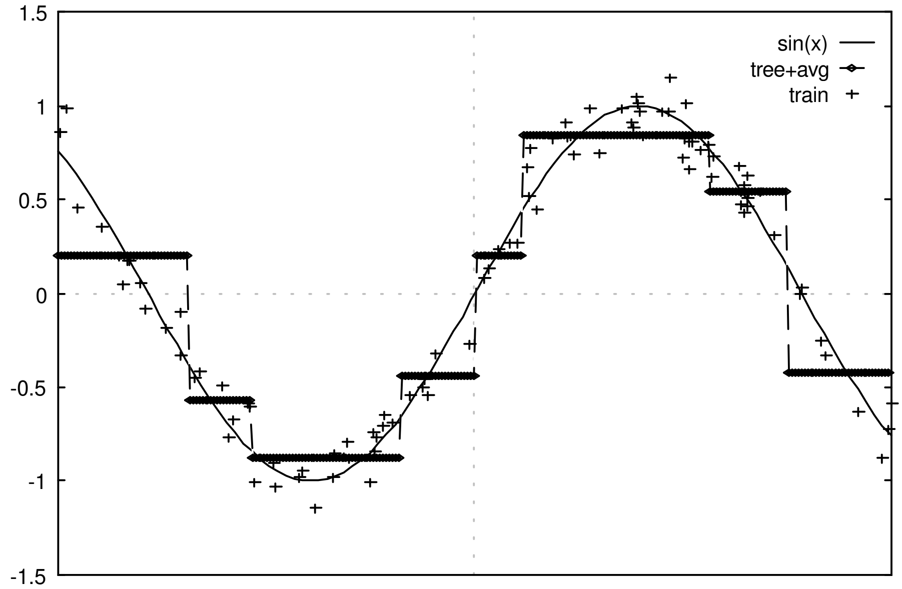


Figure 5.2. The approximation provided by a LS regression tree to the function sin(X).

The approximation of the true function provided by the tree is highly discontinuous and non-smooth curve. These two factors are usually considered the main disadvantages of tree-based regression (Friedman, 1991). In this chapter we describe approaches that integrate trees with certain smoother models so as to minimise these effects. This achieved by using different models in the tree leaves. As a motivating example, Figure 5.3 shows the approximation provided by a regression tree with kernel models in the leaves instead of averages, using exactly the same training data of the example presented above:

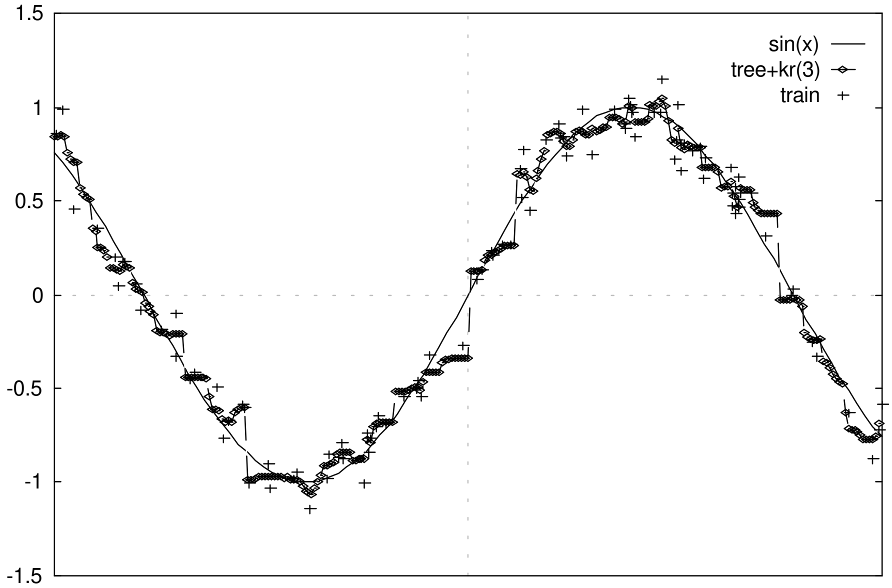


Figure 5.3. The approximation of an LS regression tree with kernel models in the leaves.

Local models by themselves provide highly smooth function approximations. However, they have several drawbacks. Firstly, these techniques do not generate any comprehensible model of the data. Secondly, they demand storing all given training data. Thirdly, they have difficulties in domains where there are strong discontinuities because they assume that nearby data points have similar target variable values. Finally, the computational complexity of local modelling methods is high when the number of training cases is large (*e.g.* Deng & Moore, 1995). This latter problem is particularly serious when using local polynomials. Several techniques exist that try to overcome these limitations. Sampling techniques, indexing schemes like *k*d-trees (Bentley, 1975; Deng & Moore, 1995) or *AD*trees (Moore & Lee, 1998), and instance prototypes[56] (Aha, 1990) are some of the methods used with large samples to decrease the computation time.

We propose to integrate local modelling with regression trees with the objective of overcoming some of the limitations mentioned earlier. On one hand, by using local models in the tree leaves we aim at achieving smoother function approximation, limiting thus the

[56] Which also have the advantage of decreasing the storage requirements.

effects of using constant value approximations. On the other hand, the use of local models in the tree leaves provides a focusing effect that allows calculating the predictions of local models more efficiently. This is achieved by avoiding going through all training points to find the appropriate nearest neighbours, limiting this search to the cases in the leaf reached by the query point. Moreover, the tree structure is a more comprehensible model of the regression surface, which is not the case when using the local models alone. Finally, the use of a tree-structured model allows us to better deal with domains with strong discontinuities. Thus the main goals of our integration proposal are the following:

- Improve standard regression trees smoothness, leading to superior predictive accuracy.
- Improve local modelling in the following aspects:
  - Computational efficiency.
  - Generation of models that are more comprehensible to human users.
  - Capability to deal with domains with strong discontinuities.

5.3.1 Method of Integration

The main decisions concerning the integration of local models with regression trees are how, and when to perform it. Three main alternatives exist:

- Assume the use of local models in the leaves during all tree induction (*i.e.* growth and pruning phases).
- Grow a standard regression tree and use local models only during the pruning stage.
- Grow and prune a standard regression tree. Use local models only in prediction tasks.

The first of these alternatives is more consistent from a theoretical point of view as the choice of the best split depends on the models at the leaves. This is the approach followed in RETIS (Karalic, 1992), which integrates global least squares linear polynomials in the tree leaves. However, obtaining such models is a computationally demanding task. For each trial split the left and right child models need to be obtained and their error calculated. We have seen that even with a simple model like the average, without an efficient incremental algorithm the evaluation of all candidate splits is too heavy. This means that if more complex models are to be used, like linear polynomials, this task is practically unfeasible for large domains[57]. Even with such incremental algorithms we have seen that there is a heavy penalty to pay with a simple model like the median. The experiments described by Karalic (1992) used data sets with few hundred cases. The author does not provide any results concerning the computation penalty of using linear models instead of averages. Still, we claim that when using more complex models like kernel regression this approach is not feasible if we want to achieve a reasonable computation time.

The second alternative is to introduce the more complex models only during the pruning stage. The tree is grown (*i.e.* the splits are chosen) assuming averages in the leaves. Only during the pruning stage we consider that the leaves will contain more complex models, which entails obtaining them for each node of the grown tree. Notice that this is much more efficient than the alternative mentioned above, which involved obtaining the models for each trial split considered during the tree growth. This latter method is the approach followed in M5 (Quinlan, 1992). This system uses global least squares linear polynomials in the tree leaves but these models are only added during the pruning stage. We have access to a version of M5[58] and we have confirmed that this is a computationally feasible solution even for large problems.

In our integration task we have followed the third alternative. In this approach the learning process is separated from prediction tasks. We generate the regression trees using any of the two "standard" methodologies described in Chapter 3. If we want to use the learned tree to make predictions for a set of unseen test cases, we can choose which model should be used in the tree leaves. These can include complex models like kernels or local linear polynomials. Using this approach we only have to fit as many models as there are leaves in the final pruned tree. The main advantage of this approach is its computational efficiency. Moreover, it also allows trying several alternative models without having to re-learn the tree. Using this approach the initial tree can be regarded as a kind of rough approximation of the regression surface which is comprehensible to the human user. On

[57] Particularly with large number of continuous variables.

[58] Version 5.1

top of this rough surface we may fit smoother models for each data partition generated in the leaves of the regression tree so as to increase the predictive accuracy[59].

In our study of integrating smoother models with regression trees we consider the use of the following alternative models in the leaves:

- Kernel models.
- Local linear polynomials.
- Semi-parametric models (partial linear models).

We will refer to the resulting models as local regression trees.

All the alternative models mentioned above are added during the prediction phase. In this aspect our system behaves like a 'lazy learner'. Given a query case we drop it down the tree until a leaf has been reached. Then we use the selected model type to obtain the prediction for the case. As we are using local models this involves obtaining the nearest training cases of the query case. This task is carried out using only the training cases in the leaf reached. This has large computational advantages when compared to approaches that need to consider all training cases when searching for the neighbours, as local modelling techniques do. In the case of partial linear models the computation of the parametric component needs to be done only once for each leaf (not for each query case).

5.3.2 An illustrative example

In this section we present a small example that illustrates the approximations provided by different models in the leaves of a regression tree. The example is simple so as to allow the graphical presentation of the obtained regression surfaces. The goal in this example is to approximate the function $f(X,Y) = \sin(X \times Y) + \cos(X+Y)$. We have randomly generated a training set with 1000 cases using this function plus some gaussian noise. Figure 5.4 shows the function and the training data:

---

[59] This strategy somehow resembles the two-tiered concept representation described in Michalski (1990) for classification problems.

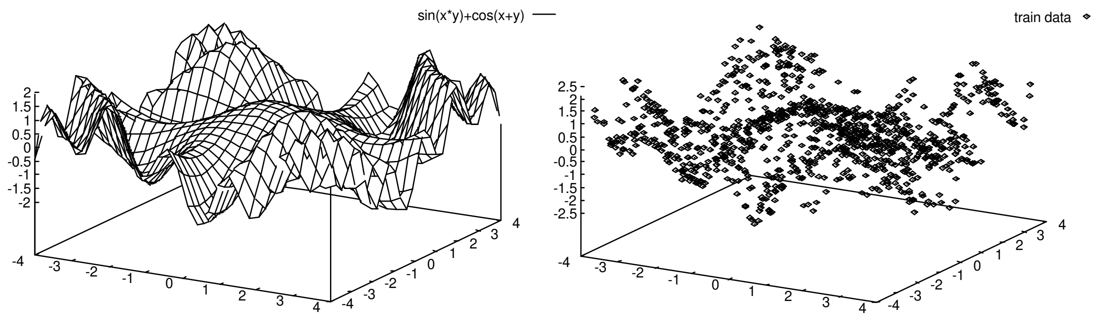


Figure 5.4. The curve and the training data.

We can use the training data to grow a regression tree and then plot the approximations provided when using the different models in the leaves. In Figure 5.5 we show the results when using averages, global linear polynomials (top part), kernel models and local linear polynomials (bottom part) in the leaves of the tree.

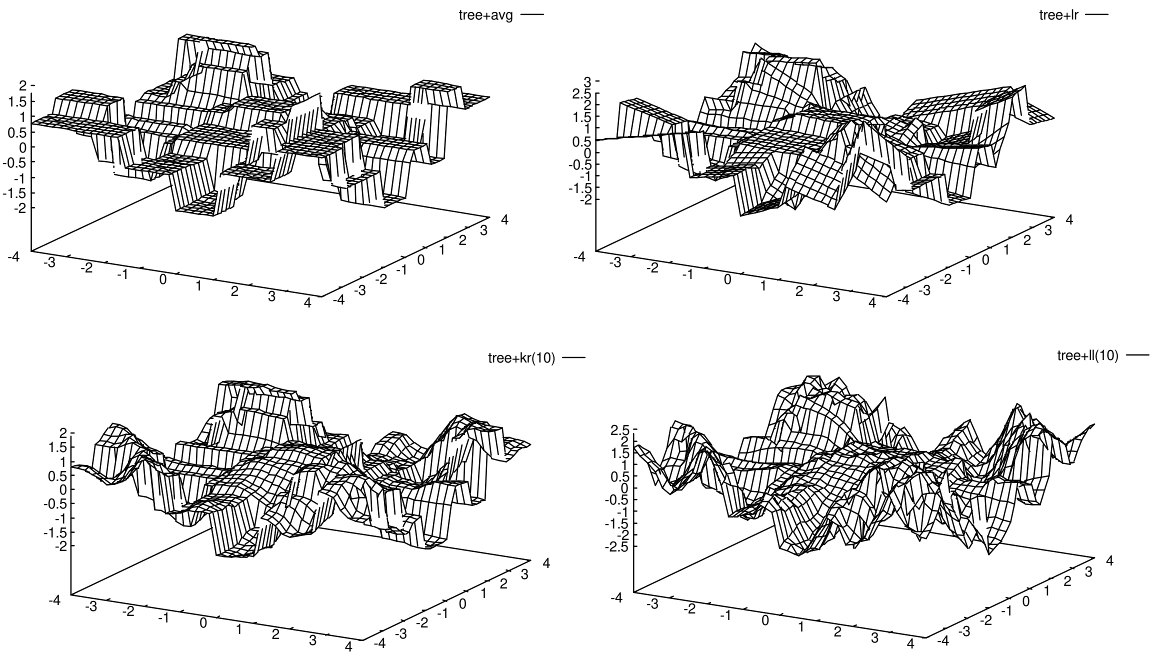


Figure 5.5. Approximations provided by using averages, linear polynomials, kernels and local linear polynomials.

Figure 5.5 shows that when using local models (kernels and local linear polynomials) the approximation is clearly much smoother.

5.3.3 Relations to Other Work

Integrating partition-based methods with other models is not an entirely new idea. Several authors have followed a similar integration schema for classification tasks. Assistant Professional (a commercial version of Assistance 86, by Cestnik *et al.*, 1987) allows the use of a naïve Bayesian classifier in the leaves of classification trees. Smyth *et. al.* (1995) integrate classification trees with kernel density estimators (*e.g.* Silverman, 1986) to obtain non-parametric estimates of class probabilities. The authors report significant improvements over decision trees and kernel density alone for certain class of problems. However, their approach deals with discrete goal variables (*i.e.* classification) and not regression as in our case. Other approaches within the classification scenario are EACH (Salzberg, 1991) and RISE (Domingos, 1996) that generalise instances into exemplars.

Deng & Moore (1995) describe a similar approach but for regression tasks. Their *multires* system integrates *k*d-trees with kernel regression producing what they call kernel regression trees. *K*d-trees (Bentley, 1975) are a method of structuring a set of records each containing a set of measured variables. They are binary trees built in a similar fashion as regression trees. However, while regression trees are built with the goal of grouping data with similar Y values, *k*d-trees try to optimise the storage of these data points in order to achieve faster access time. The consequence is that the splitting criterion of the two approaches is completely different. Moreover, while we use a single leaf of the regression tree to obtain the prediction for a test case, *multires* obtains the prediction by a combination of contributions of several nodes (not necessarily leaves). In *multires* the main issue is obtaining an efficient way of structuring all the training cases to make kernel regression computationally more efficient.

Quinlan (1992) and Karalic (1992) have used global least squares linear models in regression tree leaves. These authors follow a different integration method from ours as we have mentioned before. The M5 system (Quinlan, 1993) is also able to use regression trees and k-NN models. However, this system performs prediction combination instead of integrating the two methodologies like in our proposal. This means that two models are independently obtained and their predictions combined.

The work of Weiss & Indurkhya (1995) integrates a rule-based partitioning method with k-nearest neighbours. However, these authors deal with regression by mapping it into a classification problem. The original Y values are mapped into a set of i intervals. One problem of this approach is that the number of intervals needs to be determined, which can be computationally demanding if one wants to ensure "optimal" accuracy of the resulting models (Torgo & Gama, 1997). Moreover, the search space explored by rule learners is larger than the one of trees. This means that rule learning systems may find solutions that tree learners cannot, but at the cost of computational complexity (Indurkhya & Weiss, 1995). These two latter observations indicate that our hybrid tree learner should be able to cope with larger problems than Weiss & Indurkhya's system. Another important difference when compared to our work involves the type of local models. Weiss & Indurkhya's system uses k-NN while our system includes other more sophisticated local modelling techniques.

5.4 An Experimental Evaluation of Local Regression Trees

In this section we describe a series of experiments that have been designed to check the validity of our hypotheses concerning the advantages of local regression trees. Namely, our aim was to verify whether the variants of our system were able to overcome some of the limitations of both standard regression trees and local models. With respect to standard regression trees, we conjecture that the use of local trees brings a significant increase in accuracy due to the use of smoother models in the tree leaves. Regarding local modelling we hypothesise that local trees have three advantages: providing a comprehensible model of the regression surface; significant advantages in terms of computation time; predictive accuracy advantages in domains with strong discontinuities in the regression surface.

5.4.1 Local Regression Trees vs. Standard Regression Trees

In this section we describe a series of experiments whose objective was to check the validity of the following hypothesis:

The use of smoother models in the tree leaves improves the predictive accuracy of tree-based regression models.

We have also evaluated the costs in terms of computation time of local regression trees when compared to standard trees.

Regarding local regression trees we have considered the use of three different models in the tree leaves: kernel models, local linear polynomials and partial linear models. All these three local modelling variants were described in Section 5.2. Local modelling techniques have many parameters that can be tuned. In the experiments reported below we have not used feature weighing. The bandwidth value was set to the distance of the 10^{th} nearest neighbour[60]. The method of growing and pruning a tree (using *LSS*+*ChiEst*(95%)) was exactly the same for all compared methods (*i.e.* standard and local regression trees). The results of comparing the accuracy of standard regression trees with local trees are shown in Figure 5.6.

---

[60] For local linear polynomials we have used a larger bandwidth (30% of all nearest neighbors) because these local models can lead to wild extrapolations if they are built on the basis of small amounts of training cases.

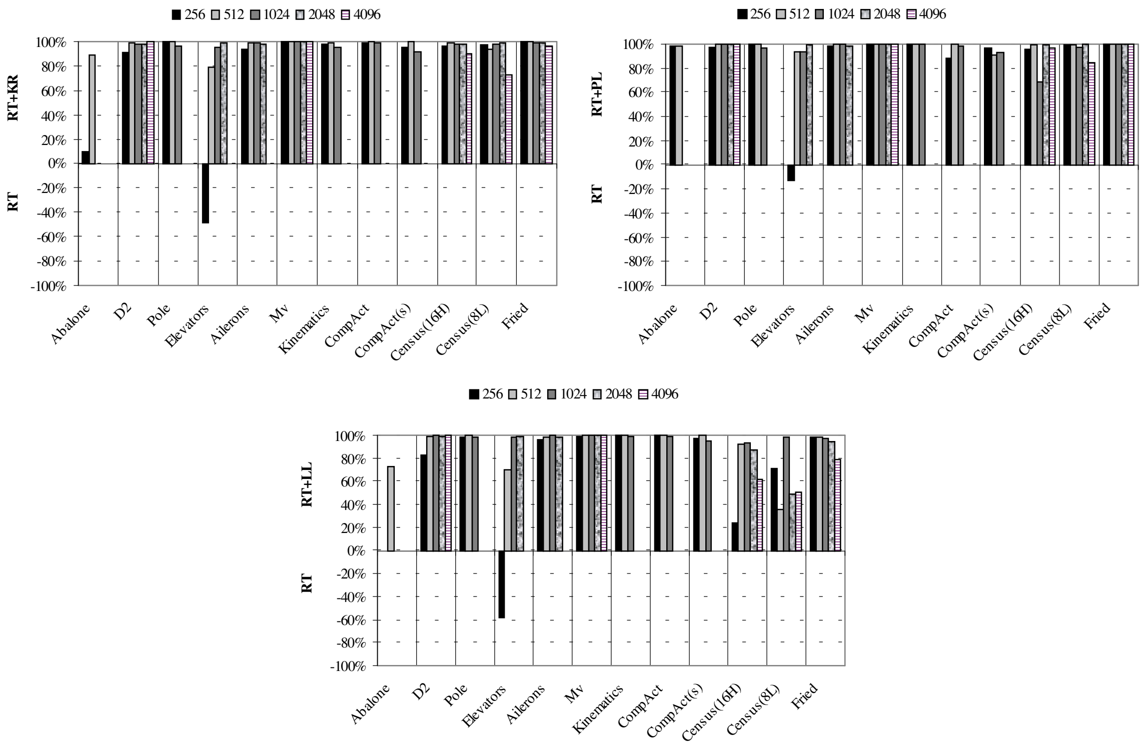


Figure 5.6 – Comparison of standard regression trees (RT) with local regression trees (RT+KR, RT+PL and RT+LL).

As these graphs show there is an overwhelming accuracy advantage of local trees, particularly with kernel (RT+KR) and partial linear models (RT+PL). However, even local trees with local linear polynomials (RT+LL) achieve a significant accuracy advantage in most domains. This experimental comparison shows that local regression trees are able to significantly outperform standard regression trees in terms of predictive accuracy. Standard regression trees (RT) provide a non-smooth approximation. The only difference between RT and the three variants of local trees is that the latter use local models in the leaves. Thus we can claim that these experiments confirm with high statistical confidence our hypothesis concerning the advantages of using smoother models in the leaves.

The accuracy gains reported above have a cost in terms of computation time. Figure 5.7 shows the computation time ratio between a standard *LSS+ChiEst*(*95%*) regression tree and the three local tree variants we have proposed. As the local trees were grown and pruned using the same procedure, the observed differences are only due to the use of local models in the leaves.

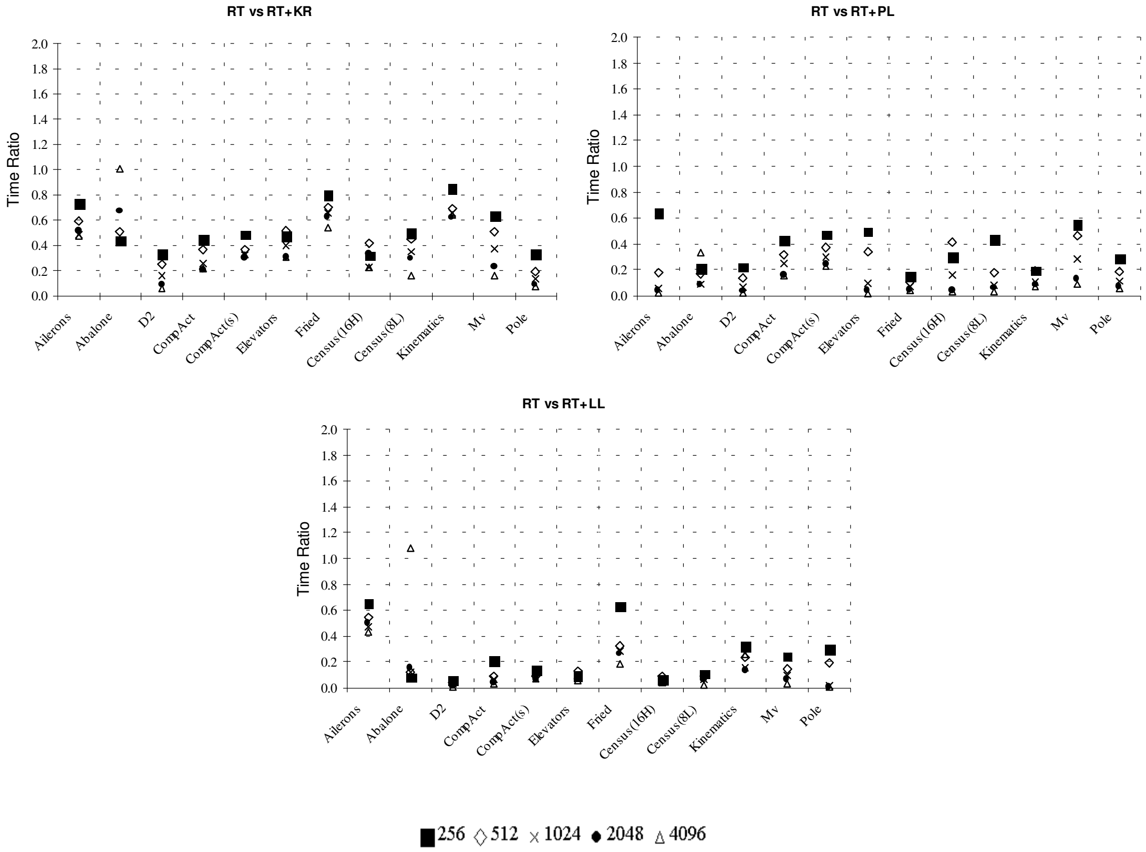


Figure 5.7 - Computation time ratio of standard and local regression trees.

These ratios show that we can expect a considerable increase of computation time due to the use of local models in the leaves. The increase is more marked for large data sets. Still, as we will see in the next section, local regression trees are much faster than the local modelling techniques alone.

5.4.2 Local Regression Trees vs. Local Models

In this section we compare local regression trees with the local models. We compare the accuracy and computation time of our three variants of local regression trees, with the local modelling techniques used alone (*i.e.* applied using all training data instead of with the data in the leaves of a tree). The local modelling techniques are the same used in the experiments reported in the previous section: kernel models, partial linear models and local linear polynomials. The models are applied with the same parameter settings as those used earlier.

The first experiment carried out was designed with the aim of checking the empirical validity of the following hypothesis:

The use of local models in the context of the leaves of a regression tree does not entail an accuracy loss and may even provide useful bias in domains with strong discontinuities.

Figure 5.8 shows the significance of the estimated accuracy difference between the local tree variants and the respective local models alone.

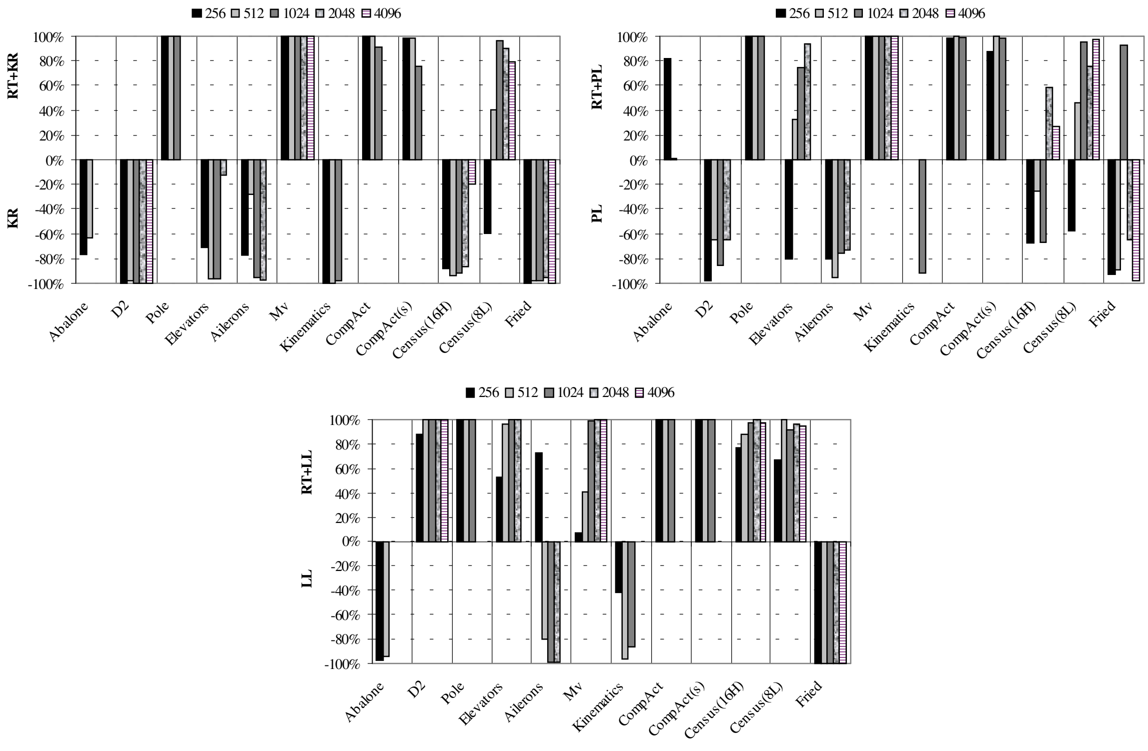


Figure 5.8 – Comparison of Local Trees and Local Models in terms of Significance of MSE difference.

The results of this comparison confirm our hypothesis. In effect, we observe that the two types of models entail quite different biases and both prove to be useful depending on the domain. For instance, we know that the regression surface of the *Mv* artificial domain has strong discontinuities (*c.f.* its definition on Appendix A.2). Our experiments confirm that local regression trees achieve better results in this domain due to the type of approximation provided. The reason for this performance is the following. A strong discontinuity occurs when nearby points of the unknown regression surface have quite different values of the goal variable. A local model is based on the notion of neighbourhood that is measured by a metric in the space of input variables. As such, these models are not able to use the information concerning the fact that two nearby points may have a very different value of the goal variable (as when facing a strong discontinuity). Thus local models will perform the usual smoothing of the goal variable values of the nearby points, which will contribute to a higher prediction error in the vicinity of the discontinuity. Regression trees, on the contrary, are based on the principle of separating points with different goal variable values because this will decrease the variance of the resulting partitions. Being so, when facing a discontinuity, a tree-based model will search for a split in the input variables that ensures a larger variance reduction, which most probably is achieved by separating the points on each "side" of the discontinuity. By separating these points in different partitions, the local models used within each one do not suffer the "bad influence" of the points on the "other side" of the discontinuity.

The other conjecture we have made regarding the comparison of local regression trees with local models was that the former are able to overcome the computation time limitations of local modelling. Figure 5.9 shows the results of comparing the computation time of local trees and the respective local models.

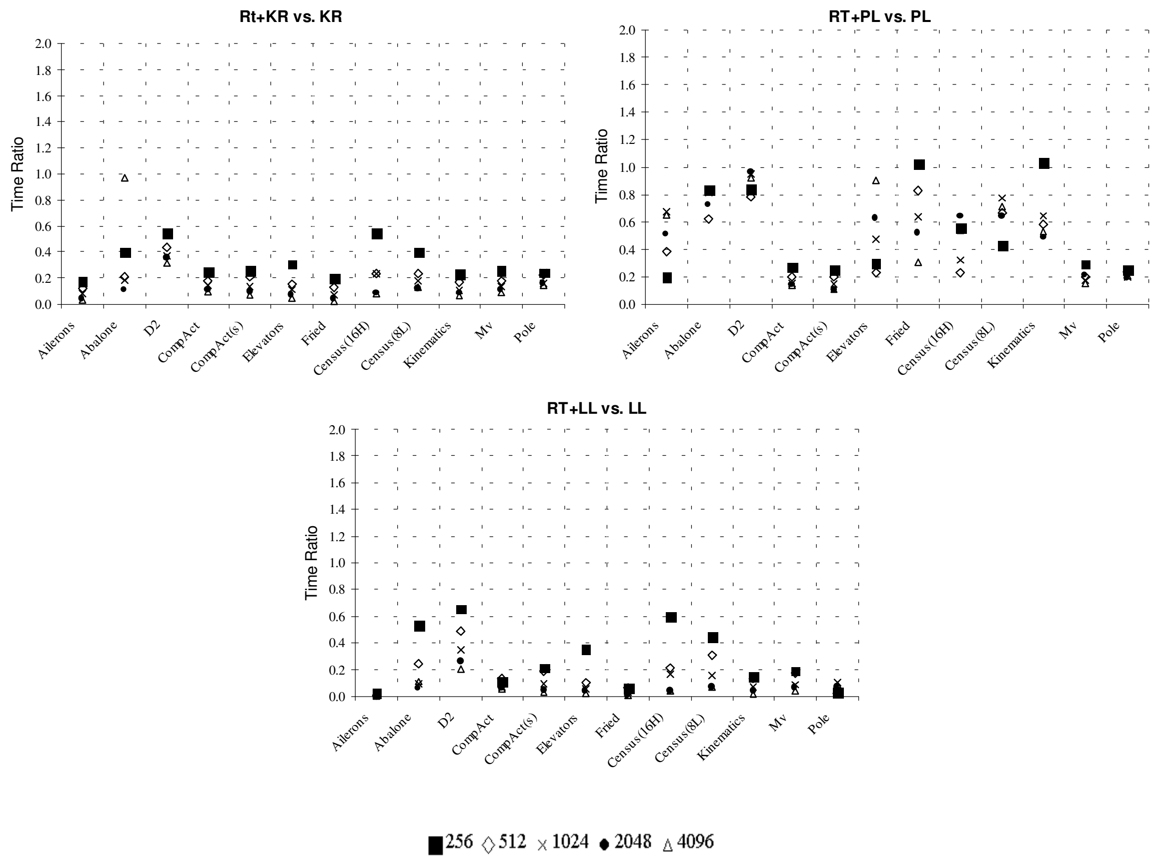


Figure 5.9 - Computation time ratio of Local Regression Trees and Local Models.

As it can be seen local regression trees successfully overcome the computational time limitations of local modelling. In effect, their advantage in terms of computation time is overwhelming in the domains we have examined.

Finally, we remark that local regression trees have another advantage over local models. In effect, local regression trees provide a symbolic representation of the regression surface, while local models do not obtain any compact model of the data. This is an important advantage despite the fact that this representation can only be regarded as a "rough" description of the true surface as it does not include a compact description of the local models used in the leaves.

5.4.3 Local Regression Trees vs. Linear Regression Trees

Other authors have tried to improve the accuracy of standard regression trees by using more complex models in their leaves. Namely, Quinlan (1992) and Karalic (1992) have used least squares linear polynomials in the leaves of regression trees. In this section we compare our local regression trees with system M5[61] (Quinlan, 1992) that obtains regression trees with linear polynomials in the leaves.

Figure 5.10 shows the accuracy comparison between our three alternative local regression trees and M5.

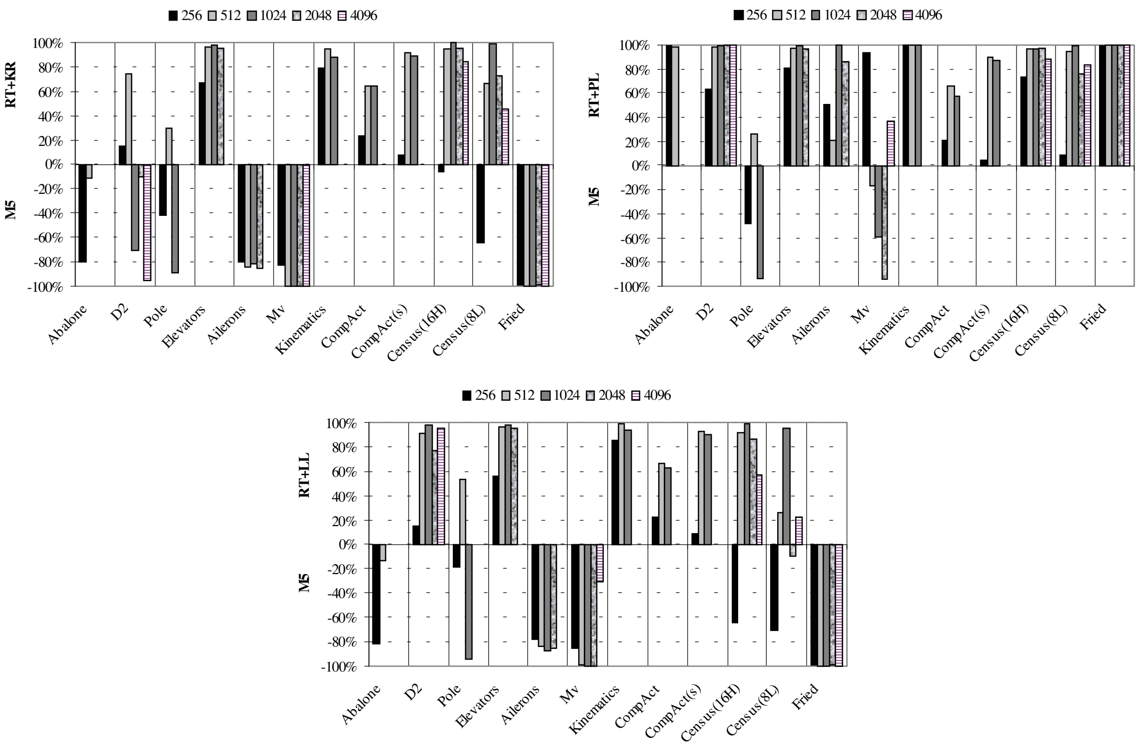


Figure 5.10 – Comparison between Local Regression Trees and M5 in terms of Significance Level of the MSE difference.

---

[61] M5 is also able to combine the predictions of the generated trees with kNN models. This feature was not used in this comparison, as the goal was to compare the use of local models in the leaves with other alternatives (in this case linear polynomials).

Figure 5.10 shows that partial linear trees[62] (RT+PL) clearly outperform M5 in most domains. Partial linear trees are similar to the models generated by M5. Both use linear polynomials in the leaves of a tree. The only difference is that partial linear models add a correction factor to the linear polynomial predictions based on a local estimation of its error (*c.f.* Section 5.2.3). These results show the importance of such smoothed correction. Both kernel regression trees[63] (RT+KR) and local linear trees[64] (RT+LL) show advantage with respect to M5 in *Elevators*, *Kinematics*, *CompAct*, *CompAct*(*s*), and *Census*(*8L* and *16H*) domains. However, they achieve worst results in the *Mv*, *Ailerons* and *Fried* domains.

5.4.4 Local Regression Trees versus Existing Regression Methods

In this section we describe a series of experiments that could determine how the best of our local regression tree variants compares with several state-of-the-art regression methods. We have selected for this comparison the following regression methods: CART (Breiman *et al.*, 1984) as a classical representative of regression trees; MARS (Friedman, 1991) as a sophisticated statistical regression tool; and CUBIST (http://www.rulequest.com) as a representative of a recent rule-based regression system. All these systems were briefly described in Chapter 2. As for local regression trees, we have selected local trees with partial linear models in the leaves (RT+PL), as the most successful variant of this type of models.

Figure 5.11 shows the results of an accuracy comparison of partial linear regression trees with CART, MARS and CUBIST.

---

[62] Regression trees with partial linear models in the leaves.

[63] Regression trees with kernel models in the leaves.

[64] Regression trees with local linear polynomials in the leaves.

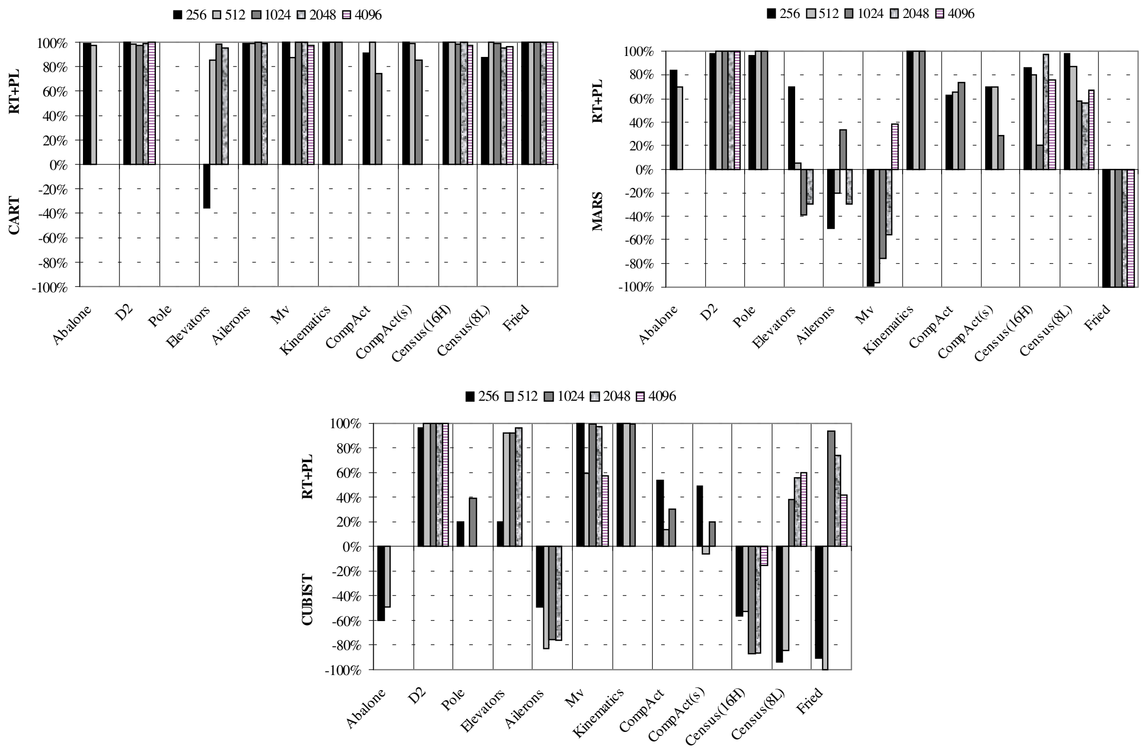


Figure 5.11 – Comparison between Partial Linear Trees and other regression techniques with respect to the Significance Level of the MSE difference.

Our system that generates partial linear trees clearly outperforms CART. In most domains the difference is statistically significant. With respect to MARS, our local trees proved to be superior in some data sets, particularly in *Pole*, *D2*, *Kinematics*, and the two *Census* domains. However, our local regression trees had difficulties in some domains like *Mv* and *Fried*[65].

Compared to CUBIST, partial linear models in the leaves of a tree proved particularly advantageous in the *D2*, *Elevators*, *Mv* and *Kinematics* domains. Less relevant advantages were achieved in the *Pole*, *CompAct* and *CompAct*(*s*) domains. CUBIST achieved some advantages in the *Abalone*, *Ailerons* and *Census*(*16H*) domains. Finally, there are two curious results in the *Census*(*8L*) and *Fried* domains. In both these two domains CUBIST

---

[65] We should remark that this artificial domain was obtained from the paper describing MARS (Friedman, 1991), so it is only natural that MARS behaves well on this data set.

shows advantage with small training sets, which is lost as more data becomes available. Still, the lack of statistical significance of most of these results does not allow any supported interpretation.

5.5 Conclusions

In this chapter we have presented a new type of tree-based regression models, referred to as local regression trees. We have described three variants of local trees that differ in the models used in their leaves: kernel models, partial linear models and local linear polynomials. Local regression trees are based on the assumption that using smoother models in the tree leaves can bring about gains in accuracy. We have confirmed this hypothesis by carrying out an extensive experimental comparison with "standard" regression trees. Among the three proposed variants, we have observed that the use of partial linear models in the leaves (RT+PL) led to the best overall accuracy. A comparison with the alternative of using linear models in the leaves (as in M5, for instance) revealed that local regression trees achieve better results in several domains. As partial linear models integrate both a parametric and a non-parametric component, this is a plausible explanation for the fact that partial linear trees proved to be the most competitive variant when compared to other variants of local regression trees.

Other motivations for considering the use of local models in the leaves of regression trees are the well-known drawbacks of local modelling. In effect, not only these modelling techniques are computationally demanding but they also do not provide a comprehensible model of the data. By integrating these models with regression trees we were able to overcome these two difficulties, as our experiments have shown. Moreover, we have also observed that the use of smoother models within the context of a partition-based representation can be advantageous in terms of accuracy, in domains where there are strong discontinuities on the regression surface.

Finally, accuracy comparisons with state-of-the-art regression techniques demonstrate that our partial linear trees achieve quite competitive overall results.

5.5.1 Open Research Issues

One of the drawbacks of local modelling techniques (or instance-based learning in general) is their requirement regarding the storage of all training data. This is also a drawback of our system based on local regression trees. It remains an open question whether we can adapt some of the existing techniques to improve this aspect in our system, without compromising too much its accuracy advantages.

Local modelling has many tuneable parameters. In our experiments we did not try out too many variants. It would be desirable to have some form of automatic tuning of these parameters to facilitate the task of the user.

In our study we have considered three kinds of local regression trees. All of them have shown some advantages in some domains. It is an open question whether it is useful and practical to have different local models in different leaves of the same regression tree. This could prove advantageous in domains where the regression surface is rather heterogeneous.

Chapter 6

Conclusions

In this dissertation we have explored different aspects of tree-based regression. We have studied two methods of growing regression trees using different error criteria. We then addressed the question of avoiding overfitting of the training data. Finally, we have described an enhancement of standard regression trees through the integration of local models in their leaves. The following section presents a brief summary of the main conclusions of the work carried out in this thesis. Finally, in Section 6.2 we present some possible directions for future research on these topics.

6.1 Summary

In this thesis we have focused our attention on regression, which is an important data analysis task. We have briefly described the main approaches to this task in the fields of machine learning, statistics and neural networks. Our contributions are within one particular type of techniques usually known as regression trees. We have described and improved several aspects related to the use of this type of regression models. Namely, we have studied the following main issues of tree-based regression:

- Generation of regression trees.
- Pruning of regression trees.
- Local regression trees that combine different regression methodologies.

6.1.1 Growing Regression Trees

As for generating regression trees we have considered the two alternatives of obtaining trees that minimise either the mean squared error or the mean absolute deviation. Regarding the former, we have analysed the main computational bottlenecks and derived a simplification of the criterion used to select the best split. Least squares (LS) regression trees generated with this simplification are very efficient in computational terms. These techniques can easily handle data sets with hundreds of thousands of cases, in few seconds. In effect, our simulation studies confirmed a clearly linear dependence of the computation time on the number of cases. This can be regarded as a crucial property when facing large regression problems, which was the main motivation behind our study of LS trees. With respect to least absolute deviation (LAD) trees we have presented a theoretical analysis of this methodology, leading to a series of algorithms that ensure high computational efficiency in the task of finding the best split for each tree node. We have also attempted to prove that a theorem by Breiman *et al.* (1984) concerning the issue of finding the best split for discrete variables was also applicable to LAD trees. Although we were not able to obtain a proof of its validity we have encountered a counter-example proving its falsity. However, we have experimentally observed that the use of a heuristic process of finding the best discrete split based on the results of that "theorem" does not bring any significant accuracy loss, while largely improving the efficiency of this task, as confirmed by our simulation studies.

We have also carried out a comparative study of LS and LAD regression trees. This study revealed that both models have different preference biases that can be considered useful depending on the application. LAD trees tend to make predictions that, on average, are more accurate. However, these trees do commit large errors more often than LS trees.

These latter models are less prone to extreme prediction errors, which can be crucial in some applications.

6.1.2 Pruning Regression Trees

This thesis addresses the problem of avoiding overfitting of the training data by pruning the initially grown trees. Pruning can be regarded as a preference bias for smaller trees that has advantages in terms of comprehensibility of the models and also may improve their predictive accuracy in many domains. In this dissertation we have followed a particular type of pruning methodology consisting of a two-stage process that begins with the generation of a set of alternative pruned trees and is followed by the selection of one of such trees.

We have described two new algorithms (*LSS* and *MCV*) for the pruned tree generation phase. These algorithms are based on the idea of progressively eliminating the nodes for which we predict that the error estimates are least reliable. This feature distinguishes them from existing methods. We have experimentally observed that our two algorithms generate sequences of pruned trees that include more accurate trees than other existing approaches (*e.g.* Error-Complexity of Breiman *et al.*,1984). In effect, our experiments have shown that the more accurate tree in the sequence generated by our methods is usually more accurate than the corresponding tree obtained using other algorithms. This empirical observation indicates that the method based on progressively eliminating the nodes that are potentially unreliable is preferable to existing heuristics to generate sequences of nested pruned trees.

We have also described a series of additions to existing tree evaluation methods, and presented a new strategy for evaluating candidate pruned trees. Regarding the additions we have extended the method based on error estimates obtained through cross validation, by presenting a new tree-matching procedure. Our method of tree-matching extends the use of this evaluation method to other than the Error-Complexity sequence of pruned trees, as opposed to the existing α-based tree matching method used in CART (Breiman *et al.*,1984). This is a relevant issue because we have empirically confirmed that cross validation is a good method of selecting a tree from a set of alternatives. This lead us to combine cross validation error estimates with one of our two new methods of generating sequences of trees, which appeared advantageous when compared to other alternatives. Our *LSS* generation algorithm together with cross validation selection achieved one of the best results in terms of predictive accuracy in the experimental comparisons carried out with existing pruning algorithms.

We have extended evaluation using *m* estimates by deriving an expression for the standard error of these estimates, which allows the use of the 1-SE rule (Breiman *et al.*,1984) that is known to provide useful bias towards simpler models. Moreover, we have extended *m* estimates to LAD trees by deriving the *m* estimate of the mean absolute deviation and its respective standard error.

We have also described a new method of evaluating the candidate pruned trees (*ChiEst*) based on a heuristic formulation of the estimated mean squared error. This heuristic relies on the sampling distribution properties of the mean squared error. Our experiments showed that pruning using our *ChiEst* method, is very competitive with other existing alternatives (resampling-based estimates, *m* estimates and MDL selection). The *ChiEst* method provides not only top predictive accuracy but also trees that are usually smaller than those selected on the basis of cross validation estimates, which was among the most accurate selection methods. Moreover, this evaluation method proved to be quite robust with respect to the best setting of its pruning parameter in a large set of domains. This feature avoids the need for resampling-based tuning contrary to other methods like *m* estimates or MDL selection. Because of this *ChiEst* was the fastest method of pruning by tree selection from the methods we have considered, which makes it a good choice for pruning regression trees obtained with large data sets.

The main conclusions from the large set of comparisons carried out with methods of pruning by tree selection are as follows. The best way to proceed to obtain better predictive accuracy is to use either our *LSS* or *MCV* algorithms together with selection based on *ChiEst* or Cross Validation error estimates. However, if we require low computational time we definitely would recommend *ChiEst* estimates, which also lead to smaller trees and hence more comprehensible than those generated by Cross Validation selection. However, if our aim is to obtain small trees the best way to proceed is to use selection based on the MDL principle following to the coding proposed by Robnik-Sikonja & Kononenko (1998). Unfortunately, we have observed that this strategy entails significant accuracy losses in several domains when compared to the above mentioned *ChiEst* method.

On another large set of experiments we have compared our most promising pruning by tree selection proposals (*LSS+ChiEst*(95%) and *LSS*+5-*fold CV*) with three state-of-the-art pruning algorithms used respectively in CART (Breiman *et al.*,1984), RETIS (Karalic & Cestnik, 1991) and CORE (Robnik-Sikonja ,1997). On several of the data sets that were considered we have observed a clear superiority of our pruning algorithms in terms of predictive accuracy. Moreover, this superiority is, in the case of *LSS+ChiEst*(*95%*), accompanied by lower computation time. However, we have also observed that our proposals usually generate somewhat larger trees than CART, and particularly, CORE.

6.1.3 Local Regression Trees

Local regression trees result from the integration of local modelling with regression trees. This new type of trees is motivated by the hypothesis that it is possible to improve the accuracy of regression trees through the use of smoother models in their leaves. We have presented three variants of local regression trees: kernel trees; partial linear trees; and local linear trees. Through a large set of experimental comparisons we have concluded that our hypothesis holds, *i.e.* local regression trees are significantly more accurate than the "standard" regression trees. However, we have also observed that our new regression models are computationally more demanding and less comprehensible than standard regression trees.

We have also conjectured that local regression trees could overcome some of the limitations of local modelling techniques, particularly their lack of comprehensibility and rather high processing time. We have carried out a large set of experiments to provide empirical evidence regarding these conjectures. We have concluded that local regression trees are significantly faster than local modelling techniques. Moreover, through the integration within a tree-based structure we obtain a comprehensible insight of the approximation of these models. We have also observed that the modelling bias resulting from combining local models and partition-based approaches improves accuracy in domains where there are strong discontinuities in the regression surface.

Finally, we have concluded that one of our local regression tree variants (partial linear trees) is among the most competitive regression methods in terms of predictive accuracy, as demonstrated by an empirical comparison with three state-of-the-art regression methods: CART (Breiman *et al.*,1984); MARS (Friedman, 1991); and CUBIST (http://www.rulequest.com).

6.2 Future Research Directions

Regarding LAD regression trees it would be beneficial to study in detail the situations under which the theorem proved by Breiman *et al.* (1984) for LS trees does not hold. This could provide useful guidance regarding the usage of the heuristic we have proposed that is based on the “theorem”.

We think it would be desirable to analyse the reasons why the *LSS* and *MCV* methods generate more accurate trees. Moreover, we would like to explore alternative ways of estimating the unreliability of node estimates. Another future topic of research is to extend the ideas behind the *ChiEst* method to LAD regression trees.

Regarding local trees there is plenty of space for further studies. In effect, it would be useful to examine what are the computational requirements of a system that integrates local models with regression trees during the pruning stage. We also think that the performance of local regression trees could be largely improved through further tuning of local models. Finally, we would like to find some solution for the memory requirements of local regression trees (*e.g.* by eliminating some of the cases) that would not imply costs in terms of predictive accuracy.

Annexes

Annex A

Materials and Methods

This annex provides a detailed description of the materials and methods used throughout this thesis. We describe the experimental methodology followed in the experiments carried out and the benchmark data sets used in these empirical tests. We also provide details regarding the learning systems that are used in our comparative studies.

A.1. The Experimental Methodology

Within current Machine Learning research the most used experimental methodology for comparing learning methods is k-fold Cross Validation (*e.g.* Michie *et al.*,1994). This methodology can be used to estimate the predictive accuracy of learning methods. Moreover, paired t tests can be used to get an assessment of the significance of the observed differences. A k-fold cross validation (CV) experiment starts with a random division of the given training sample in k disjoint subsets D_1, ..., D_k, each containing approximately the same number of observations. For each fold D_i a model is constructed using as learning sample $D \backslash D_i$, obtaining the model $r_i(\beta, \mathbf{x}_i)$. This model is then tested in the fold D_i. The same process is repeated for all k folds. The average test set performance over the k folds is obtained for each algorithm and usually a paired t test is carried out to assert the significance of the observed differences on performance between any two alternative methods (*e.g.* Salzberg, 1997). As pointed out by Dietterich (1997) and Rasmussen (1996) this latter procedure violates the assumptions of this statistical test. In effect, the strong overlap of the training sets on the k iterations does not allow one to assume that the scores obtained on each iteration are independent. Now, this is one of the assumptions of the paired t tests, which makes this procedure unsafe to use as pointed out by the authors above cited. This does not means that we should not use the Cross Validation method to carry out comparative experiments. It only means that performing paired t tests with the resulting fold scores may underestimate the variance resulting from the use of different training samples. This is particularly serious for the type of regression models being studied in this thesis, as tree-based models are known to be quite sensible to variations on the training data (Breiman, 1996).

Rasmussen (1996) proposes an experimental methodology based on non-overlapping train and test sets that does not have the problems mentioned above. This methodology was implemented on a general-purpose environment for performance assessment called DELVE[66] (Rasmussen *et al.*, 1996). In this thesis we have used these tools to assert the statistical significance of observed accuracy differences between different methods for different training set sizes. The main drawback of DELVE is the large amount of data necessary to allow multiple non-overlapping train and test samples. This fact has limited the number of data sets used in the experiments we have carried out. Still, DELVE allows fair comparisons of different learning methods for different data sets and different training sample sizes. In this thesis we have asserted the performance of the learning methods on samples with 256, 512, 1024, 2048 and 4096 cases.

For the sake of completeness we present a brief description of the analysis methodology used in DELVE. Further details can be found in Rasmussen (1996). DELVE provides two alternative experimental designs. In the *Hierarchical ANOVA design* each learning algorithm is trained on I disjoint training sets. Associated to each of these training

[66] Publicly available at http://www.cs.utoronto.ca/~delve/.

sets there is a test set with J cases. The test sets are disjoint one from one another and from the training sets. DELVE assumes that the losses of the learner can be modelled by,

$$y_{ij} = \mu + a_i + \varepsilon_{ij} \tag{A.1}$$

where,

y_{ij} is the loss on the test case j from the test set i when the method is trained with the training set i;
μ is the expected loss of the method;
a_i are called the effects due to the training set, and model the variability in the losses due to changes in the training set;
and the ε_{ij}'s model the residuals that include the effects of the test cases, interactions between train and test cases and stochastic elements in the prediction procedure.

DELVE assumes that both the a_i 's and the ε_{ij}'s are normally distributed. According to Rasmussen (1996) these assumptions may not be appropriate for some loss functions like the 0/1 loss function used in classification problems. However, for the squared loss function used throughout this thesis, extensive simulation experiments conducted by this author revealed no serious effects on the conclusions of the paired comparisons to assert significant differences in accuracy. DELVE provides a set of tools to obtain the estimated expected loss (*i.e.* $\hat{\mu} = \bar{y}$) and the respective standard error of the estimate. Moreover, the same model can be applied to estimate differences between losses of any two methods.

DELVE includes an alternative experimental method called the *2-way ANOVA design*. This method is more efficient in terms of the use of the available data, thus making it more adequate for smaller data sets. However, its analysis is much more complicated than the one of the hierarchical model. In this thesis we only had necessity of using this design for the *Abalone* data set. All other results are obtained with the hierarchical ANOVA model.

In this thesis we show the predictive accuracy results in the form of graphs of paired comparisons. These graphs show the statistical significance level of the observed differences in terms of predictive accuracy between any two methods. The following figure shows an example of such graph:

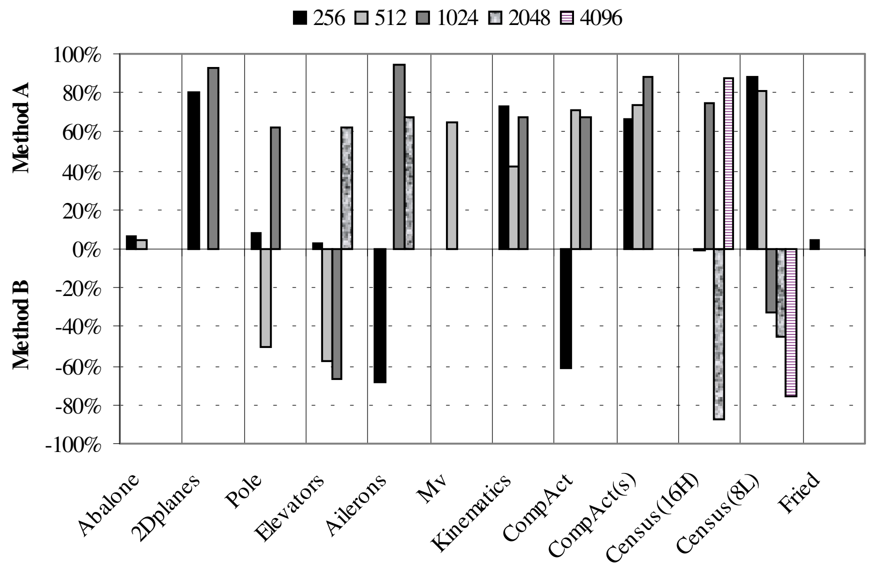


Figure A.1 – An example of a paired accuracy comparison.

This graph shows values that are the levels of significance of the accuracy differences between methods *A* and *B*. Positive values indicate that the observed difference is favourable to method *A*. Negative values indicate the opposite. The results are collected for 12 different data sets and for the different training sample sizes we have considered. A value of 80%, for instance, means that there is 80% probability that method *A* outperforms method *B* in other samples of the same size from the same domain. Throughout the thesis we consider values higher than 95% (or lower than –95%) statistically significant. Values above 99% (or below –99%) are considered highly significant.

In Annex C we present tables that include the full results of the paired accuracy comparisons mentioned before. These tables show the estimated difference in prediction error (we use the Squared Error loss function) between any two learning methods. For each of the combinations of data set and sample size, we show three results. The estimated difference in error, the statistical confidence level on this difference (the *p* value)[67], and some signs stressing highly significant differences. As an example let us suppose we want

[67] The values pictured in the graphs mentioned before are 1-*p*.

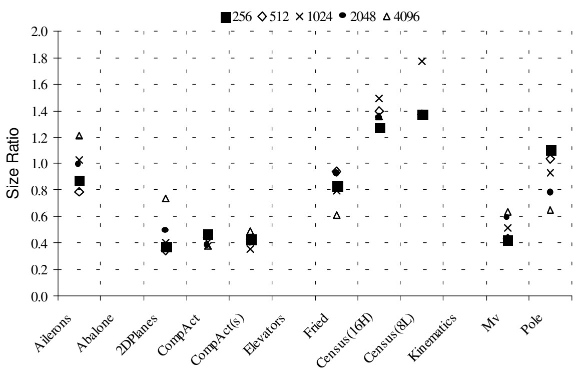
to compare method *A* with method *B*. The following table shows an example of the type of results we provide for this kind of comparisons:

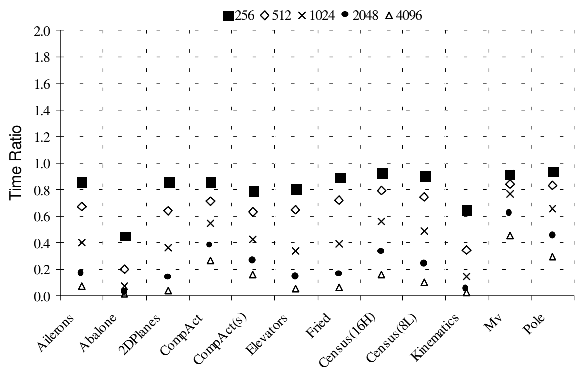
Table A.1. Difference in error (MSE) between method *A* and method *B*.

	256			**512**			**1024**			**2048**			**4096**		
Abalone	0.126	0.500		-0.280	0.543										
2Dplanes	0.429	0.043	+	-0.078	0.194		-0.042	0.216		-0.240	0.000	--	-0.099	0.000	--
Pole	-47.913	0.272		-17.195	0.143		-14.882	0.171							
Elevators	-1.459	0.489		-0.232	0.891		3.686	0.478		1.079	0.564				
Ailerons	0.013	0.101		0.012	0.152		0.010	0.770		-0.022	0.328				
Mv	0.750	0.098		0.154	0.077		-0.166	0.004	--	-0.094	0.001	--	-0.084	0.002	--
Kinematics	-0.013	0.623		0.014	0.435		0.003	0.564							
CompAct	5.709	0.012	+	1.282	0.103		1.845	0.201							
CompAct(s)	4.810	0.009	++	2.940	0.011	+	2.955	0.134							
Census(16H)	-1.45E+7	0.824		-3.55E+7	0.446		-2.48E+7	0.239		2.17E+7	0.641		-9.28E+6	0.820	
Census(8L)	-1.53E+8	0.461		8.46E+7	0.168		-2.74E+7	0.659		-1.81E+7	0.419		9.05E+6	0.518	
Fried	0.099	0.354		0.400	0.212		0.373	0.122		-0.001	0.915		-0.069	0.480	

The first column for each sample size indicates the estimated difference in MSE between the two methods. Negative numbers indicate that method *A* is expected to have less MSE than method *B*. The second column shown the confidence level of this estimated difference. Finally, the third column can have one sign if the difference is significant with at least 95% confidence, and two signs for more than 99% confidence. Minus signs indicate advantages of the first method (in this case the method *A*), while plus sings indicate the opposite. In some experimental set-ups we do not present results because there was not enough data to carry out the experiment according to DELVE requirements.

Apart from predictive accuracy this thesis emphasises model interpretability and computation efficiency. We have asserted model interpretability by the size of the trees, namely the number of leaves. With respect to computation efficiency we have collected information on the CPU seconds taken to carry out a full learn and test cycle. As we have mentioned DELVE needs large amounts of data to allow non-overlapping train and test sizes. Because of this, DELVE does not run many iterations for each data set. Thus averaging tree sizes and CPU seconds over these few iterations does not make much sense. Being so, whenever we wanted to compare the expected tree size or computation efficiency of several alternative systems we have used a different experimental method. Instead of using DELVE, we have obtained a set of 50 random samples for each of the sample sizes we have studied and obtained the average scores on these two issues. When presenting results for these two factors we show graphs of the ratios between the methods being compared. For instance if we wish to compare the method *A* with the method *B*, we ran the two methods on the same 50 random samples and calculate the average number of leaves and CPU time over the 50 runs. We repeat this process for each of the sample sizes being studied and show graphs with the ratios between these averages. The following figure shows an example of such graphs:





Ratios above 1 indicate that the first method has larger average value on the item being evaluated, while scores below 1 indicate the opposite. As there is possible overlap between these 50 random samples we avoid any statements regarding the statistical significance of the differences. Thus these results should be taken as merely indicative of the expected score of the methods being compared. However, as we have also calculated the standard deviations, these numbers can provide further indications on the expected variability of these scores.

A.2. The Used Benchmark Data Sets

The choice of the data sets used in this thesis was mainly conditioned by both the available domains in the community repositories and the necessity of large amounts of cases to allow the use of DELVE. We now describe the main characteristics of these data sets as well as a brief description of the task involved.

- *Abalone*

  This data set can be used to obtain a model to predict the age of abalone from physical measurements. The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope -- a boring and time-consuming task. Other measurements, which are easier to obtain, are used to predict the number of rings which determines the age.

  Source: UCI machine learning repository.

  Characteristics: 4177 cases, 8 attributes (1 nominal, 7 continuous).

- *Ailerons*

  This data set addresses a control problem, namely flying a F16 aircraft. The attributes describe the status of the aeroplane, while the goal is to predict the control action on the ailerons of the aircraft.

  Source: Experiments of Rui Camacho (rcamacho@garfield.fe.up.pt).

  Characteristics: 13750 cases, 40 attributes (0 nominal, 40 continuous).

- *Elevators*

  This data set is also obtained from the task of controlling a F16 aircraft, although the target variable and attributes are different. In this case the goal variable is related to an action taken on the elevators of the aircraft.

  Source: Experiments of Rui Camacho (rcamacho@garfield.fe.up.pt).

  Characteristics: 16559 cases, 18 attributes (0 nominal, 18 continuous).

- *2Dplanes*

  This is an artificial data set described in Breiman *et al.* (1984, p.238). The cases are generated using the following method:

  Generate the values of the 10 attributes independently using the following probabilities:

  $P(X_1 = -1) = P(X_1 = 1) = \frac{1}{2}$

  $P(X_m = -1) = P(X_m = 0) = P(X_m = 1) = \frac{1}{3}$, $m = 2,\ldots,10.$

  Obtain the value of the target variable Y using the rule:

  if $X_1 = 1$ set $Y = 3 + 3X_2 + 2X_3 + X_4 + \sigma(0,2)$

  if $X_1 = -1$ set $Y = -3 + 3X_5 + 2X_6 + X_7 + \sigma(0,2)$

Source: Breiman *et al.* (1984, p.238). The actual cases used in our experiments can be obtained at `http://www.ncc.up.pt/~ltorgo/Regression/DataSets.html`.

Characteristics: 40768 cases, 10 attributes (0 nominal, 10 continuous).

- *Pole*

  This is a commercial application described in Weiss & Indurkhya (1995). The data describes a telecommunication problem. No further information is available.

  Source: The data can be obtained in http://www.cs.su.oz.au/~nitin.

  Characteristics: 9065 cases, 48 attributes (0 nominal, 48 continuous).

- *Fried*

  This is an artificial data set used in Friedman (1991) and also described in Breiman (1996,p.139). The cases are generated using the following method:

  Generate the values of 10 attributes, X_1, …, X_{10} independently each of which uniformly distributed over [0,1]. Obtain the value of the target variable Y using the equation:

  $$Y = 10\sin(\pi X_1 X_2) + 20(X_3 - 0.5)^2 + 10X_4 + 5X_5 + \sigma(0,1)$$

  Source: Breiman (1996, p.139). The actual cases used in our experiments can be obtained at `http://www.ncc.up.pt/~ltorgo/Regression/DataSets.html`.

  Characteristics: 40768 cases, 10 attributes (0 nominal, 10 continuous).

- *D2*

  This is a highly non-linear artificial data set with a strong discontinuity.

  The values of the two attributes X_1 and X_2 are uniformly distributed reals over the interval [0..15]. The value of the target variable is obtained using the following generating function :

  $$\text{if } X_1 > 7.5 \text{ then } Y = 5 \times e^{X_2/3} - 10 \times \sin\left(X_1/2 \times X_2/4\right) + 5 \times X_1/X_2$$
  $$\text{else } Y = \sin(X_1)$$

  Source: The actual cases used in our experiments can be obtained at `http://www.ncc.up.pt/~ltorgo/Regression/DataSets.html`.

  Characteristics: 40768 cases, 2 attributes (0 nominal, 2 continuous).

- *Mv*

  This is an artificial data set with dependencies between the attribute values. The cases are generated using the following method:

  X_1 : uniformly distributed over [-5,5]
  X_2 : uniformly distributed over [-15,-10]

X_3 : IF (X_1 > 0) THEN X_3 = green
ELSE X_3 = red with probability 0.4 and X_4=brown with prob. 0.6
X_4 : IF (X_3=green) THEN X_4=X_1+2X_2
ELSE X_4=X_1/2 with prob. 0.3, and X_4=X_2/2 with prob. 0.7
X_5 : uniformly distributed over [-1,1]
X_6 : X_6=X_4×ε, where ε is uniformly distribute over [0,5]
X_7 : X_7=yes with prob. 0.3 and X_7=no with prob. 0.7
X_8 : IF (X_5 < 0.5) THEN X_8 = normal ELSE X_8 = large
X_9 : uniformly distributed over [100,500]
X_{10} : uniformly distributed integer over the interval [1000,1200]

Obtain the value of the target variable *Y* using the rules:
IF (X_2 > 2) THEN Y = 35 – 0.5 X_4
ELSE IF ($-2 \leq X_4 \leq 2$) THEN Y = 10 – 2 X_1
ELSE IF (X_7 = yes) THEN Y = 3 –X_1/X_4
ELSE IF (X_8 = normal) THEN Y = X_6 + X_1
ELSE Y = X_1/2

Source: `http://www.ncc.up.pt/~ltorgo/Regression/DataSets.html.`

Characteristics: 40768 cases, 10 attributes (3 nominal, 7 continuous).

- *Kinematics*

  This data set is concerned with the forward kinematics of an 8 link robot arm. Among the existing variants of this data set we have used the variant *8nm*, which is known to be highly non-linear and medium noisy.

  Source: DELVE repository of data.

  Characteristics: 8192 cases, 8 attributes (0 nominal, 8 continuous).

- *CompAct* and *CompAct(s)*

  The Computer Activity databases are a collection of computer systems activity measures. The data was collected from a Sun Sparcstation 20/712 with 128 Mbytes of memory running in a multi-user university department. Users would typically be doing a large variety of tasks ranging from accessing the internet, editing files or running very cpu-bound programs. The data was collected continuously on two separate occasions. On both occasions, system activity was gathered every 5 seconds. The final dataset is taken from both occasions with equal numbers of observations coming from each collection epoch.

  System measures used:

  1. lread - Reads (transfers per second) between system memory and user memory.
  2. lwrite - writes (transfers per second) between system memory and user memory.

3. scall - Number of system calls of all types per second.
4. sread - Number of system read calls per second.
5. swrite - Number of system write calls per second .
6. fork - Number of system fork calls per second.
7. exec - Number of system exec calls per second.
8. rchar - Number of characters transferred per second by system read calls.
9. wchar - Number of characters transfreed per second by system write calls.
10. pgout - Number of page out requests per second.
11. ppgout - Number of pages, paged out per second.
12. pgfree - Number of pages per second placed on the free list.
13. pgscan - Number of pages checked if they can be freed per second.
14. atch - Number of page attaches (satisfying a page fault by reclaiming a page in memory) per second.
15. pgin - Number of page-in requests per second.
16. ppgin - Number of pages paged in per second.
17. pflt - Number of page faults caused by protection errors (copy-on-writes).
18. vflt - Number of page faults caused by address translation.
19. runqsz - Process run queue size.
20. freemem - Number of memory pages available to user processes.
21. freeswap - Number of disk blocks available for page swapping.
22. usr - Portion of time (%) that cpus run in user mode.
23. sys - Portion of time (%) that cpus run in system mode.
24. wio - Portion of time (%) that cpus are idle waiting for block IO.
25. idle - Portion of time (%) that cpus are otherwise idle.

The two different regression tasks obtained from these databases are:
CompAct
Predict usr, the portion of time that cpus run in user mode from all attributes 1-21.

CompAct(s)
Predict usr using a restricted number of attributes (excluding the paging information (10-18)).

Source: DELVE repository of data.

Characteristics: Both data sets contain 8192 cases. For *ComptAct* we have 22 continuous attributes, while for *CompAct(s)* the cases are described by 8 continuous variables.

- *Census(16H)* and *Census(8L)*

  This database was designed on the basis of data provided by US Census Bureau [http://www.census.gov] (under Lookup Access [http://www.census.gov/cdrom/lookup]: Summary Tape File 1). The data were collected as part of the 1990 US census. These are mostly counts cumulated at different survey levels. For the purpose of this data set a level *State-Place* was used. Data from all states was obtained. Most of the counts were changed into appropriate proportions.

There are 4 different data sets obtained from this database:
House(8H)
House(8L)
House(16H)
House(16L)

These are all concerned with predicting the median price of the house in the region based on demographic composition and a state of housing market in the region. The number in the name of the data sets signifies the number of attributes of the problem. The following letter denotes a very rough approximation of the difficulty of the task. For Low task difficulty, more correlated attributes were chosen as signified by univariate smooth fit of that input on the target. Tasks with High difficulty have had their attributes chosen to make the modelling more difficult due to higher variance or lower correlation of the inputs to the target.

Source: DELVE repository of data.

Characteristics: Both data sets contain 22784 cases. For *House(16H)* we have 16 continuous attributes, while for *House(8L)* the cases are described by 8 continuous variables.

- *Algae*

  This is a group of small data sets that are used in some examples throughout the thesis. This data comes from a real world problem that was used in the 3rd International Competition organised by ERUTDIT (`http://www.erudit.de/erudit/activities/ic-99/`). The data actually can be regarded as 7 different regression problems as each case (described by 11 variables) as seven associated target values. As such we have divided the original data into seven problems: *Alga 1*, *Alga 2*, …, *Alga 7*. All have exactly the same descriptive attributes the single difference being the goal variable value of each case.

  The data analysis task concerns the environmental problem of determining the state of rivers and streams by monitoring and analysing certain measurable chemical concentrations with the goal of inferring the biological state of the river, namely the density of algae communities. This study is motivated by the increasing concern with the impact human activities are having on the environment. Identifying the key chemical control variables that influence the biological process associated with these algae has become a crucial task in order to reduce the impact of man activities. Water quality samples were collected from various European rivers during one year. These samples were analysed for various chemical substances. At the same time, algae samples were collected to determine the distributions of the algae populations. The dynamics of algae communities is strongly influenced by the external chemical environment. Determining which chemical factors are influencing more this dynamics is important knowledge that can be used to control these populations. At the same time there is an economical factor motivating even more this analysis. In effect, the chemical analysis is cheap and easily automated. On the contrary, the biological part involves microscopic examination, requires trained manpower and is therefore both expensive and slow. The competition task consisted of predicting the frequency distribution of seven different algae on the basis of eight measured concentrations of chemical substances plus some additional information characterising the environment from which the sample was taken (season, river size and flow velocity).

The used input attributes for all seven problems were:

season – The season when the sample was taken (winter, spring, autumn, summer).
riversz – The size of the river (small, medium, large).
flowvl – The flow velocity of the river (low, medium, high).
maxPH - Maximum Ph during the periods of 3 month different measurements where taken (continuous).
minO2 – Minimum of oxygen (continuous).
MeanCL – Mean of CL (continuous).
meanNO3N – Mean of NO3N (continuous).
meanNH4N – Mean of NH4N (continuous).
MeanOrthophosphat - Mean of Orthophosphate (continuous).
meanTotalPO4 - Mean of total PO4 (continuous).
MeanCHLORO – Mean of Chlorophyll (continuous).

Source: 3rd International Competition organised by ERUDIT; can obtained at `http://www.erudit.de/erudit/activities/ic-99/`.

Characteristics: 200 train cases + 140 test cases, 11 attributes (3 nominal, 8 continuous).

A.3. The Learning Systems Used in the Comparisons

In this thesis we have compared our RT system with other regression analysis algorithms. In this section we describe very briefly these systems particularly the used versions and where to obtain them.

- *RT*
  This is the system that implements the ideas described in this thesis. The version used in the experiments described in the thesis is version 4.0. RT is

written in C and we have binaries for Ultrix 4.2, SUN OS and Linux. We intend to make RT available for research purposes. More information can be obtained in the Web page http://www.ncc.up.pt/~ltorgo/RT or by contacting the author, ltorgo@ncc.up.pt.

- *CART*

  CART is a system that learns regression trees. We have access to Unix version 1.309 of CART. CART is a commercial product distributed by Salford Systems (http://www.salford-systems.com).

- *M5*

  M5 is able to induce model trees. Model trees consist basically of regression trees with linear least squares polynomials in the leaves. We have access to a Unix version of M5 gently provided by Prof. Ross Quinlan. The version used in the experiments of this thesis is M5.1 [release 1].

- *MARS*

  MARS is able to obtain adaptive regression splines. We have used in our experiments Mars version 3.6 with bagging. We have obtained the source from the DELVE repository.

- *CUBIST*

  CUBIST is able to learn regression rules with linear least squares polynomials in the consequent. In our experiments we have used release 1.01. CUBIST is a commercial system distributed by RuleQuest (http://www.rulequest.com).

Annex B

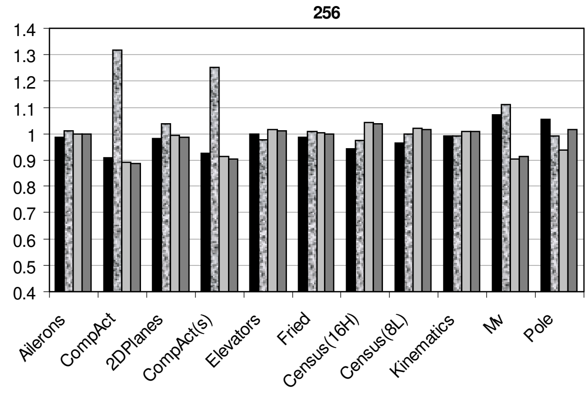
Experimental Results

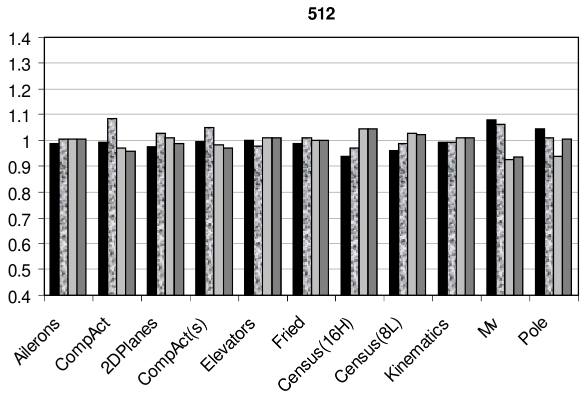
This annex provides the full tables of results and further graphs of the experiments carried out throughout the thesis.

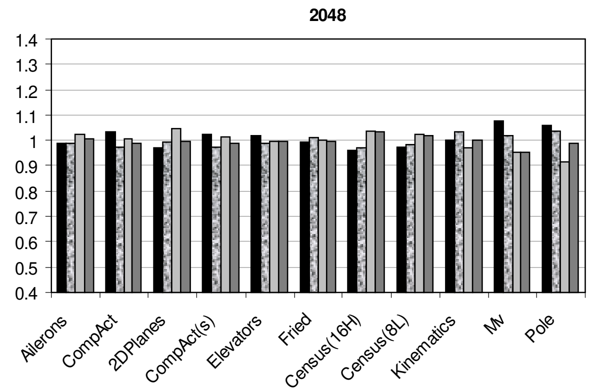
B.4. Experiments with Tree Generation Methods

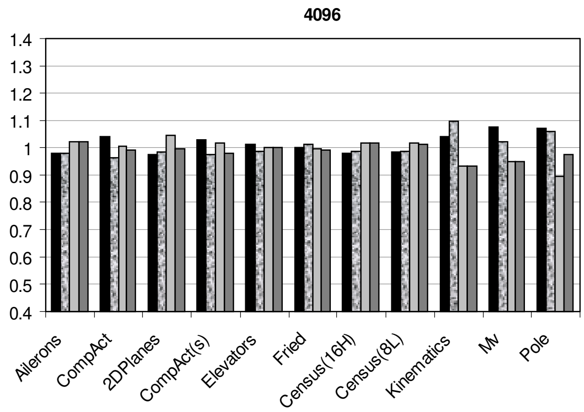
In this section we present further details concerning the experiments that compare methods of generating sequences of pruned trees in the context of pruning by tree selection.

The following figure shows the average “true” error of the sub-trees obtained by each method considered in our experiments. This experiment is described in Section 4.3.1.









The following tables show the complete results (average and best error) of all methods, for each training set size.

256 cases	***LSS***		***MCV***		***ErrCpx***		***MEL***	
	Average	*Best*	*Average*	*Best*	*Average*	*Best*	*Average*	*Best*
Ailerons	0.080	0.070	0.082	0.070	0.081	0.072	0.081	0.072
CompAct	50.324	18.205	72.996	18.343	49.373	18.309	49.125	18.304
2DPlanes	3.215	1.716	3.401	1.731	3.265	1.733	3.228	1.732
CompAct(s)	47.597	20.831	64.386	20.796	46.857	20.984	46.432	20.980
Elevators	33.963	30.153	33.241	29.617	34.553	29.556	34.488	29.323
Fried	11.780	10.332	12.015	10.283	11.965	10.470	11.920	10.463
Census(16H)	2.67E+9	2.37E+9	2.76E+9	2.46E+9	2.95E+9	2.52E+9	2.94E+9	2.50E+9
Census(8L)	2.00E+9	1.79E+9	2.06E+9	1.84E+9	2.11E+9	1.90E+9	2.10E+9	1.89E+9
Kinematics	0.057	0.050	0.057	0.050	0.058	0.050	0.058	0.050
Mv	20.132	4.271	20.914	4.262	17.026	4.261	17.163	4.261
Pole	559.222	299.807	523.845	303.654	496.340	309.796	539.218	307.199

512 cases	***LSS***		***MCV***		***ErrCpx***		***MEL***	
	Average	*Best*	*Average*	*Best*	*Average*	*Best*	*Average*	*Best*
Ailerons	0.067	0.059	0.068	0.060	0.068	0.061	0.068	0.061
CompAct	37.161	14.759	40.686	14.722	36.286	14.782	35.805	14.781
2DPlanes	2.237	1.318	2.360	1.379	2.313	1.339	2.261	1.341
CompAct(s)	35.521	17.521	37.401	17.525	35.121	17.723	34.582	17.701
Elevators	29.896	27.218	29.126	26.225	30.161	26.761	30.091	26.554
Fried	9.852	8.837	10.048	8.845	9.991	8.986	9.953	8.980
Census(16H)	2.38E+9	2.13E+9	2.46E+9	2.19E+9	2.65E+9	2.35E+9	2.64E+9	2.33E+9
Census(8L)	1.83E+9	1.63E+9	1.88E+9	1.68E+9	1.95E+9	1.76E+9	1.94E+9	1.76E+9
Kinematics	0.051	0.047	0.051	0.047	0.052	0.047	0.052	0.047
Mv	15.709	3.224	15.456	3.222	13.486	3.226	13.612	3.226
Pole	411.722	220.457	398.819	214.135	371.378	222.494	396.033	221.853

2048 cases	***LSS***		***MCV***		***ErrCpx***		***MEL***	
	Average	*Best*	*Average*	*Best*	*Average*	*Best*	*Average*	*Best*
Ailerons	0.053	0.047	0.053	0.047	0.055	0.048	0.054	0.048
CompAct	21.961	10.313	20.674	10.296	21.404	10.297	21.02	10.295
2DPlanes	1.42	1.037	1.448	1.065	1.528	1.055	1.454	1.059
CompAct(s)	20.043	11.344	19.086	11.361	19.811	11.355	19.283	11.353
Elevators	22.555	20.886	21.831	20.116	22.074	20.032	22.045	20.062
Fried	6.687	6.067	6.797	6.056	6.724	6.125	6.704	6.122
Census(16H)	2.00E+9	1.81E+9	2.02E+9	1.85E+9	2.16E+9	1.98E+9	2.15E+9	1.98E+9
Census(8L)	1.43E+9	1.31E+9	1.44E+9	1.32E+9	1.50E+9	1.39E+9	1.49E+9	1.39E+9
Kinematics	0.033	0.03	0.034	0.03	0.032	0.03	0.033	0.03
Mv	12.305	2.171	11.661	2.171	10.867	2.171	10.867	2.171
Pole	194.617	85.68	190.025	85.113	168.457	86.006	181.14	86.027

4096 cases	***LSS***		***MCV***		***ErrCpx***		***MEL***	
	Average	*Best*	*Average*	*Best*	*Average*	*Best*	*Average*	*Best*
Ailerons	0.048	0.042	0.048	0.043	0.05	0.044	0.05	0.044
CompAct	17.096	7.727	15.79	7.729	16.505	7.726	16.264	7.726
2Dplanes	1.252	1.015	1.263	1.02	1.347	1.016	1.281	1.016
CompAct(s)	14.716	7.759	13.971	7.764	14.515	7.763	14.023	7.763
Elevators	19.767	18.323	19.258	17.857	19.52	17.833	19.498	17.874
Fried	5.416	4.906	5.493	4.882	5.394	4.917	5.382	4.916
Census(16H)	1.71E+9	1.60E+9	1.72E+9	1.63E+9	1.78E+9	1.72E+9	1.78E+9	1.72E+9
Census(8L)	1.27E+9	1.18E+9	1.27E+9	1.20E+9	1.31E+9	1.24E+9	1.31E+9	1.25E+9
Kinematics	0.019	0.01	0.02	0.01	0.017	0.01	0.017	0.01
Mv	11.651	1.99	11.066	1.99	10.318	1.99	10.318	1.99
Pole	128.619	44.118	126.741	44.071	107.312	44.131	116.94	44.131

B.5. CART tree-matching vs. Our proposal

This section presents the full results of the comparison of our tree-matching method with the strategy used in CART (*c.f.* Section 4.3.2.1). The table presents the estimated difference in MSE between the tree selected using CART matching and the tree selected using our method. Positive values indicate that the MSE associated with the choice of CART method is larger than ours. Details concerning the information in these type of tables can be found in Appendix A.1.

	256		**512**		**1024**		**2048**		**4096**	
Abalone	0.042	0.938	0.038	0.958						
2Dplanes	0.035	0.198	0.000	0.829	0.029	0.075	0.000	0.727	0.000	0.338
Pole	0.240	0.922	-4.026	0.495	5.306	0.377				
Elevators	0.034	0.971	-4.263	0.425	-5.272	0.331	2.831	0.377		
Ailerons	-0.027	0.313	0.000	0.952	0.017	0.054	0.007	0.328		
Mv	0.000	0.351	0.021	0.351	0.000	1.000	0.000	1.000	0.000	1.000
Kinematics	0.012	0.271	0.006	0.576	0.007	0.323				
CompAct	-0.501	0.387	0.159	0.285	0.095	0.327				
CompAct(s)	0.464	0.330	0.326	0.265	0.410	0.121				
Census(16H)	4.57E+4	0.997	-6.37E+5	0.995	9.78E+7	0.256	-4.00E+7	0.129	5.00E+7	0.124
Census(8L)	1.50E+8	0.119	5.33E+7	0.193	-1.72E+7	0.673	-5.82E+7	0.553	-3.00E+6	0.243
Fried	0.012	0.951	0.000	0.746	0.000	0.905	0.000	0.828	0.000	0.813

B.6. Comparison of Methods for Generating Pruned Trees

In this section we present the tables with the full results concerning the comparison of different methods of generating sequences of sub-trees (*c.f.* Section 4.3.4.1).

Table B.1. Estimated MSE difference between *MEL* and *LSS* using *ChiEst*(*95%*).

	256			512			1024			2048			4096		
Abalone	0.139	0.684		-0.069	0.803										
2Dplanes	0.117	0.087		0.025	0.234		0.064	0.002	++	-0.011	0.386		0.008	0.102	
Pole	1.914	0.841		-2.256	0.746		-2.477	0.786							
Elevators	2.910	0.357		-0.760	0.539		-0.300	0.877		-0.961	0.278				
Ailerons	0.001	0.481		0.002	0.323		0.002	0.573		0.001	0.610				
Mv	0.146	0.623		0.063	0.200		0.000	0.000	++	0.006	0.351		0.000	0.000	++
Kinematics	0.000	0.785		0.000	0.871		0.000	0.772							
CompAct	-0.635	0.336		0.027	0.886		0.112	0.642							
CompAct(s)	-1.254	0.258		-0.150	0.820		-1.053	0.013	-						
Census(16H)	1.14E+8	0.068		2.82E+8	0.030	+	2.52E+8	0.054		1.40E+8	0.067		1.72E+8	0.205	
Census(8L)	-1.06E+7	0.874		1.35E+8	0.113		2.48E+8	0.138		9.38E+7	0.094		3.72E+7	0.807	
Fried	0.186	0.095		0.083	0.656		-0.039	0.723		0.016	0.891		0.106	0.333	

Table B.2. Estimated MSE difference between *ErrCpx* and *LSS* using *ChiEst*(*95%*).

	256			512			1024			2048			4096		
Abalone	0.077	0.960		-0.033	0.981										
2Dplanes	0.129	0.074		0.025	0.234		0.059	0.003	++	-0.008	0.530		0.016	0.031	+
Pole	1.941	0.839		-3.883	0.596		0.834	0.942							
Elevators	2.903	0.359		-0.539	0.627		-0.271	0.890		-1.269	0.220				
Ailerons	0.002	0.255		0.002	0.337		0.001	0.588		0.001	0.610				
Mv	0.146	0.623		0.063	0.200		0.000	0.000	++	0.006	0.351		0.000	0.000	++
Kinematics	0.000	0.977		0.000	0.833		0.000	0.514							
CompAct	-0.635	0.336		-0.029	0.870		0.112	0.642							
CompAct(s)	-1.263	0.253		-0.216	0.735		-1.044	0.013	-						
Census(16H)	1.14E+8	0.068		2.82E+8	0.030	+	2.49E+8	0.057		1.85E+8	0.021	+	1.77E+8	0.211	
Census(8L)	-1.51E+7	0.820		1.45E+8	0.091		2.45E+8	0.145		8.73E+7	0.136		8.14E+6	0.948	
Fried	0.172	0.104		0.083	0.656		-0.039	0.725		0.032	0.766		0.104	0.327	

Table B.3. Estimated MSE difference between *MEL* and *MCV* using *ChiEst*(*95%*).

	256			512			1024			2048			4096		
Abalone	0.320	0.116		0.137	0.748										
2Dplanes	-0.022	0.648		-0.034	0.095		0.005	0.730		0.012	0.015	+	0.002	0.675	
Pole	-37.508	0.271		-0.955	0.894		6.538	0.420							
Elevators	3.950	0.321		0.160	0.865		-0.985	0.524		-0.442	0.474				
Ailerons	0.002	0.259		0.002	0.213		0.002	0.403		0.000	0.898				
Mv	0.043	0.844		-0.263	0.327		0.000	0.000	++	0.006	0.351		0.000	0.000	++
Kinematics	-0.001	0.397		0.001	0.295		0.000	0.767							
CompAct	-0.696	0.285		0.195	0.101		0.330	0.407							
CompAct(s)	0.309	0.386		0.109	0.817		0.079	0.721							
Census(16H)	1.92E+7	0.702		1.18E+8	0.002	++	1.05E+8	0.186		1.89E+8	0.005	++	9.01E+7	0.292	
Census(8L)	-3.00E+7	0.576		1.46E+8	0.084		4.33E+7	0.601		1.35E+8	0.169		9.30E+7	0.161	
Fried	0.271	0.055		-0.012	0.945		-0.047	0.495		0.124	0.312		0.159	0.126	

Table B.4. Difference in error (MSE) between *ErrCpx* and *MCV* using *ChiEst*(*95%*).

	256			**512**			**1024**			**2048**			**4096**		
Abalone	0.258	0.313		0.173	0.626										
2Dplanes	-0.010	0.844		-0.034	0.095		0.000	0.993		0.016	0.020	+	0.010	0.130	
Pole	-37.481	0.271		-2.581	0.743		9.848	0.164							
Elevators	3.944	0.322		0.381	0.727		-0.956	0.538		-0.750	0.198				
Ailerons	0.003	0.071		0.002	0.223		0.002	0.415		0.000	0.898				
Mv	0.043	0.844		-0.263	0.327		0.000	0.000	++	0.006	0.351		0.000	0.000	++
Kinematics	-0.001	0.315		0.001	0.299		0.000	0.829							
CompAct	-0.696	0.285		0.138	0.246		0.330	0.407							
CompAct(s)	0.301	0.390		0.042	0.928		0.088	0.685							
Census(16H)	1.92E+7	0.702		1.19E+8	0.002	++	1.02E+8	0.200		2.35E+8	0.000	++	9.58E+7	0.259	
Census(8L)	-3.45E+7	0.517		1.56E+8	0.059		4.08E+7	0.625		1.29E+8	0.194		6.40E+7	0.223	
Fried	0.257	0.078		-0.012	0.945		-0.048	0.508		0.141	0.237		0.157	0.118	

Table B.5. Difference in Prediction error (MSE) between *MEL* and *LSS* using 5-fold CV.

	256			**512**			**1024**			**2048**			**4096**		
Abalone	-0.099	0.689		0.251	0.280										
2Dplanes	0.047	0.276		0.023	0.167		0.041	0.023	+	0.025	0.013	+	0.005	0.080	
Pole	-3.353	0.388		3.117	0.792		26.696	0.365							
Elevators	-3.680	0.496		0.411	0.694		-0.650	0.867		-1.806	0.175				
Ailerons	-0.002	0.729		0.004	0.255		0.000	0.902		0.000	0.968				
Mv	-0.088	0.351		-0.009	0.673		0.000	1.000		0.000	1.000		0.000	1.000	
Kinematics	0.001	0.133		0.000	0.541		0.000	0.849							
CompAct	-0.317	0.517		-0.268	0.234		0.217	0.298							
CompAct(s)	0.221	0.781		0.181	0.588		-0.514	0.398							
Census(16H)	2.16E+8	0.073		3.43E+8	0.036	+	1.12E+8	0.142		1.64E+8	0.143		7.61E+7	0.493	
Census(8L)	1.29E+8	0.431		4.34E+7	0.380		2.85E+8	0.088		1.73E+8	0.083		2.17E+8	0.106	
Fried	0.008	0.968		0.164	0.176		-0.018	0.904		0.004	0.951		0.093	0.261	

Table B.6. Difference in error (MSE) between *ErrCpx* and *LSS* using 5-fold CV.

	256			**512**			**1024**			**2048**			**4096**		
Abalone	-0.099	0.689		0.251	0.280										
2Dplanes	0.047	0.276		0.023	0.167		0.041	0.023	+	0.025	0.013	+	0.005	0.080	
Pole	-3.353	0.388		3.117	0.792		26.696	0.365							
Elevators	-3.680	0.496		0.411	0.694		-0.650	0.867		-1.806	0.175				
Ailerons	-0.002	0.729		0.004	0.255		0.000	0.902		0.000	0.968				
Mv	-0.088	0.351		-0.009	0.673		0.000	1.000		0.000	1.000		0.000	1.000	
Kinematics	0.001	0.133		0.000	0.541		0.000	0.849							
CompAct	-0.317	0.517		-0.268	0.234		0.217	0.298							
CompAct(s)	0.221	0.781		0.181	0.588		-0.514	0.398							
Census(16H)	2.16E+8	0.073		3.43E+8	0.036	+	1.12E+8	0.142		1.64E+8	0.143		7.61E+7	0.493	
Census(8L)	1.29E+8	0.431		4.34E+7	0.380		2.85E+8	0.088		1.73E+8	0.083		2.17E+8	0.106	
Fried	0.008	0.968		0.164	0.176		-0.018	0.904		0.004	0.951		0.093	0.261	

Table B.7. Difference in Prediction error (MSE) between *MEL* and *MCV* using 5-fold CV.

	256			**512**			**1024**			**2048**			**4096**		
Abalone	0.179	0.410		0.017	0.999										
2Dplanes	0.060	0.174		-0.041	0.278		-0.025	0.153		0.007	0.426		0.001	0.803	
Pole	12.210	0.082		6.010	0.564		24.916	0.372							
Elevators	1.059	0.178		-0.924	0.229		1.600	0.608		1.340	0.627				
Ailerons	0.008	0.004	++	0.003	0.403		0.002	0.187		0.000	0.892				
Mv	-0.042	0.109		-0.321	0.306		-0.004	0.351		0.001	0.351		0.000	1.000	
Kinematics	0.000	0.923		0.000	0.950		0.002	0.118							
CompAct	0.235	0.748		0.227	0.345		0.378	0.469							
CompAct(s)	0.970	0.286		0.082	0.888		-0.658	0.373							
Census(16H)	6.82E+7	0.492		1.67E+8	0.138		1.47E+8	0.045	+	2.53E+8	0.004	++	1.37E+8	0.162	
Census(8L)	1.95E+8	0.296		2.15E+8	0.329		7.45E+7	0.322		8.09E+7	0.420		1.75E+8	0.128	
Fried	0.275	0.134		0.367	0.022	+	-0.008	0.952		-0.042	0.588		0.022	0.821	

Table B.8. Difference in Prediction error (MSE) between *ErrCpx* and *MCV* using 5-fold CV.

	256			**512**			**1024**			**2048**			**4096**		
Abalone	0.211	0.288		-0.078	0.868										
2Dplanes	0.058	0.181		-0.037	0.180		-0.023	0.215		0.002	0.755		-0.001	0.712	
Pole	11.836	0.097		2.745	0.679		9.179	0.389							
Elevators	4.708	0.300		-0.729	0.368		2.331	0.479		1.097	0.478				
Ailerons	0.005	0.019	+	0.002	0.510		0.002	0.438		-0.001	0.549				
Mv	-0.042	0.109		-0.321	0.306		-0.004	0.351		0.001	0.351		0.000	1.000	
Kinematics	0.000	0.558		0.000	0.932		0.000	0.956							
CompAct	0.603	0.377		0.056	0.854		0.386	0.439							
CompAct(s)	0.221	0.743		-0.053	0.933		-0.669	0.362							
Census(16H)	7.20E+7	0.443		1.83E+8	0.087		1.46E+8	0.055		2.54E+8	0.002	++	1.55E+8	0.085	
Census(8L)	1.85E+8	0.312		2.18E+8	0.322		5.49E+7	0.402		1.05E+8	0.315		1.60E+8	0.128	
Fried	0.427	0.015	+	0.444	0.009	++	0.001	0.996		-0.009	0.910		0.002	0.980	

B.7. Tuning of Selection Methods

This section present the tables of results concerning the comparisons of different approaches to tuning the pruning parameters of the selection methods we have considered in Section 4.3.4.2.

Table B.9. Difference in MSE between *ChiEst*(95%) and *ChiEst* tuned by 5-fold CV.

	256			512			1024			2048			4096		
Abalone	0.000	1.000		0.333	0.392										
2Dplanes	0.031	0.367		0.005	0.130		-0.009	0.390		-0.008	0.289		-0.001	0.351	
Pole	15.421	0.088		14.748	0.128		5.123	0.364							
Elevators	-4.528	0.305		-0.774	0.325		-2.905	0.339		0.518	0.201				
Ailerons	0.001	0.675		-0.003	0.334		-0.001	0.747		0.000	1.000				
Mv	0.055	0.181		-0.002	0.936		0.000	1.000		0.002	0.351		0.000	1.000	
Kinematics	0.001	0.194		0.000	1.000		0.000	1.000							
CompAct	-0.501	0.175		-0.284	0.324		0.026	0.391							
CompAct(s)	0.603	0.151		-0.187	0.127		0.631	0.472							
Census(16H)	-2.94E+7	0.195		-6.51E+7	0.192		-4.40E+7	0.351		-2.06E+6	0.435		-1.70E+7	0.391	
Census(8L)	6.09E+7	0.174		-1.79E+8	0.265		-9.25E+7	0.544		3.84E+6	0.206		1.18E+6	0.391	
Fried	-0.179	0.347		-0.052	0.687		-0.043	0.514		0.008	0.859		-0.002	0.757	

Table B.10. Difference in MSE between *m*(2) and *m* tuned by 5-fold CV.

	256			512			1024			2048			4096		
Abalone	-0.029	1.000		-0.176	0.611										
2Dplanes	1.224	0.000	++	0.874	0.000	++	0.715	0.000	++	0.450	0.000	++	0.166	0.000	++
Pole	-9.573	0.198		-14.776	0.537		0.005	0.997							
Elevators	-4.426	0.316		0.181	0.791		-0.180	0.830		-0.988	0.216				
Ailerons	-0.024	0.557		0.021	0.321		-0.001	0.743		0.007	0.716				
Mv	5.414	0.000	++	4.273	0.000	++	2.333	0.003	++	1.271	0.000	++	1.096	0.000	++
Kinematics	0.006	0.000	++	0.065	0.007	++	0.055	0.063							
CompAct	5.328	0.004	++	1.373	0.034	+	3.699	0.002	++						
CompAct(s)	6.706	0.007	++	2.174	0.022	+	3.907	0.005	++						
Census(16H)	2.07E+8	0.047	+	1.79E+8	0.100		3.12E+8	0.080		1.61E+8	0.060		1.56E+8	0.336	
Census(8L)	-1.42E+7	0.127		-5.05E+7	0.270		2.68E+8	0.042	+	2.11E+8	0.190		2.41E+8	0.193	
Fried	-0.220	0.202		-0.142	0.418		0.470	0.001	++	0.432	0.000	++	0.500	0.002	++

Table B.11. Difference in MSE between *MDL*(0.1,0.5) and *MDL* tuned by 5-fold CV.

	256			512			1024			2048			4096		
Abalone	0.006	0.910		0.154	0.198										
2Dplanes	1.375	0.010	+	0.621	0.004	++	0.153	0.018	+	0.116	0.005	++	0.000	0.004	++
Pole	-1.483	0.790		-7.520	0.297		16.409	0.210							
Elevators	-0.085	0.809		-0.161	0.841		-2.544	0.408		4.793	0.084				
Ailerons	-0.026	0.315		0.020	0.149		0.053	0.131		0.097	0.236				
Mv	5.811	0.001	++	1.407	0.051		0.731	0.003	++	0.226	0.184		0.000	0.153	
Kinematics	-0.006	0.572		-0.004	0.810		0.000	0.765							
CompAct	10.139	0.006	++	4.963	0.023	+	1.502	0.302							
CompAct(s)	4.205	0.044	+	2.228	0.139		2.256	0.299							
Census(16H)	7.05E+7	0.052		7.18E+6	0.923		3.72E+7	0.147		2.44E+5	0.988		1.85E+7	0.653	
Census(8L)	4.51E+7	0.067		3.60E+7	0.258		8.58E+7	0.256		8.62E+7	0.158		9.51E+7	0.070	
Fried	1.937	0.066		1.165	0.004	++	1.470	0.001	++	1.558	0.000	++	0.988	0.000	++

B.8. Comparison of Methods of Evaluating a Tree

Table B.12. Difference in error (MSE) between 5-fold CV and *m* estimates tuned by 5-CV.

	256			**512**			**1024**			**2048**			**4096**		
Abalone	0.174	0.084		-0.361	0.209										
2Dplanes	-0.045	0.407		0.008	0.844		0.005	0.667		0.011	0.074		0.005	0.168	
Pole	-0.485	0.897		-13.641	0.414		-0.942	0.819							
Elevators	-0.729	0.173		2.133	0.648		4.230	0.051		2.370	0.511				
Ailerons	0.006	0.498		-0.002	0.795		-0.001	0.436		0.000	0.884				
Mv	-0.208	0.144		-0.179	0.043	-	-0.054	0.036	-	-0.025	0.017	-	-0.006	0.236	
Kinematics	-0.010	0.608		0.005	0.725		0.000	0.363							
CompAct	0.782	0.108		0.136	0.209		0.045	0.626							
CompAct(s)	0.005	0.991		-0.153	0.396		-0.129	0.764							
Census(16H)	-4.99E+7	0.234		-9.80E+7	0.218		9.34E+7	0.417		-4.97E+7	0.631		-5.71E+7	0.624	
Census(8L)	-1.24E+8	0.466		-7.23E+6	0.971		2.58E+7	0.577		-4.14E+7	0.453		-3.29E+6	0.778	
Fried	-0.279	0.243		-0.153	0.394		0.122	0.133		0.044	0.106		0.036	0.341	

Table B.13. Difference in error (MSE) between 5-fold CV and χ^2 estimates (CL=95%).

	256			**512**			**1024**			**2048**			**4096**		
Abalone	0.124	0.562		-0.320	0.518										
2Dplanes	-0.006	0.755		0.023	0.424		0.012	0.160		-0.055	0.000	--	0.008	0.075	
Pole	-5.129	0.689		-3.137	0.785		-10.243	0.121							
Elevators	3.624	0.417		3.014	0.518		7.164	0.023	+	2.858	0.430				
Ailerons	0.013	0.198		0.001	0.577		-0.001	0.782		-0.002	0.198				
Mv	-0.071	0.098		0.000	1.000		0.000	0.351		0.000	0.974		0.000	1.000	
Kinematics	-0.001	0.597		0.000	0.933		0.000	0.824							
CompAct	0.393	0.513		0.449	0.167		-0.123	0.546							
CompAct(s)	-0.782	0.574		-0.410	0.498		-0.876	0.349							
Census(16H)	-9.11E+7	0.216		1.14E+8	0.381		1.20E+8	0.257		5.29E+7	0.322		-1.77E+5	0.998	
Census(8L)	-1.68E+8	0.349		2.24E+8	0.172		3.16E+7	0.763		-7.65E+7	0.301		-1.16E+8	0.333	
Fried	0.195	0.228		0.148	0.281		0.013	0.809		-0.110	0.177		-0.051	0.174	

Table B.14. Difference in error (MSE) between 5-fold CV and MDL tuned by 5-fold CV.

	256			512			1024			2048			4096		
Abalone	0.025	1.000		-0.227	0.529										
2Dplanes	-0.744	0.001	--	-0.234	0.009	--	-0.145	0.000	--	-0.049	0.040	-	0.004	0.455	
Pole	9.924	0.159		2.745	0.757		-3.668	0.675							
Elevators	5.247	0.415		3.169	0.605		1.862	0.700		0.212	0.920				
Ailerons	-0.002	0.788		-0.032	0.152		-0.031	0.331		-0.012	0.241				
Mv	-1.192	0.010	--	-0.495	0.001	--	-0.044	0.173		-0.011	0.233		0.000	0.188	
Kinematics	0.000	0.938		-0.002	0.864		0.001	0.473							
CompAct	-5.562	0.009	--	-1.196	0.056		-2.409	0.083							
CompAct(s)	-5.106	0.018	-	-3.405	0.002	--	-3.900	0.033	-						
Census(16H)	-6.89E+7	0.157		1.01E+8	0.413		1.04E+8	0.290		-3.00E+6	0.944		-2.36E+7	0.177	
Census(8L)	-3.69E+7	0.819		1.14E+8	0.488		-7.09E+7	0.488		-1.22E+8	0.129		-1.80E+8	0.129	
Fried	-0.149	0.320		-0.354	0.319		-0.535	0.084		-0.399	0.015	-	-0.312	0.001	--

B.9. Comparisons with Other Pruning Algorithms

This section presents the results of comparing our pruning proposals with existing pruning algorithms.

Table B.15. Difference in error (MSE) between *LSS*+5CV and CART pruning.

	256			512			1024			2048			4096		
Abalone	0.025	1.000		-0.194	0.627										
2Dplanes	-0.080	0.149		-0.029	0.224		-0.072	0.004	--	-0.024	0.072		0.002	0.581	
Pole	3.486	0.352		4.175	0.736		-16.266	0.369							
Elevators	-0.002	0.997		3.657	0.503		5.191	0.037	+	-0.781	0.480				
Ailerons	0.009	0.374		-0.003	0.130		-0.003	0.306		0.000	0.982				
Mv	0.079	0.351		-0.012	0.754		0.000	1.000		0.000	1.000		0.000	1.000	
Kinematics	-0.002	0.121		0.000	0.966		0.001	0.331							
CompAct	0.449	0.265		0.281	0.423		-0.320	0.108							
CompAct(s)	0.064	0.885		-0.374	0.222		0.115	0.817							
Census(16H)	-2.20E+8	0.064		-3.58E+8	0.015	-	-2.09E+8	0.119		-1.25E+8	0.285		-1.44E+8	0.159	
Census(8L)	-2.68E+8	0.163		-9.93E+7	0.246		-2.48E+8	0.108		-1.40E+8	0.008	--	-2.00E+8	0.103	
Fried	-0.172	0.187		-0.188	0.482		0.000	0.998		-0.044	0.571		-0.082	0.250	

Table B.16. Difference in error (MSE) between *LSS*+5CV and RETIS pruning.

	256			512			1024			2048			4096		
Abalone	-0.296	0.188		-0.628	0.034	-									
2Dplanes	-0.001	0.975		-0.077	0.004	--	-0.090	0.004	--	-0.069	0.000	--	-0.040	0.006	--
Pole	-0.513	0.892		-6.591	0.597		-28.346	0.361							
Elevators	0.467	0.699		4.160	0.460		4.909	0.029	+	1.507	0.705				
Ailerons	0.011	0.319		-0.001	0.749		-0.007	0.237		-0.001	0.557				
Mv	-0.091	0.311		-0.164	0.028	-	-0.055	0.008	--	-0.020	0.039	-	-0.006	0.236	
Kinematics	-0.003	0.095		0.000	0.868		-0.003	0.082							
CompAct	0.446	0.334		0.229	0.415		-0.179	0.192							
CompAct(s)	-0.319	0.645		-0.080	0.754		-0.119	0.876							
Census(16H)	-3.54E+8	0.028	-	-2.98E+8	0.020	-	-2.84E+8	0.088		-2.44E+8	0.061		-2.45E+8	0.239	
Census(8L)	-1.66E+8	0.335		-1.68E+7	0.933		-2.84E+8	0.059		-2.61E+8	0.085		-2.32E+8	0.110	
Fried	-0.104	0.640		-0.246	0.016	-	-0.076	0.555		-0.060	0.480		-0.111	0.157	

Table B.17. Difference in error (MSE) between *LSS*+5CV and CORE pruning.

	256			512			1024			2048			4096		
Abalone	0.521	0.282		0.182	0.479										
2Dplanes	-0.468	0.000	--	-0.206	0.003	--	-0.110	0.004	--	-0.056	0.001	--	-0.022	0.022	-
Pole	-4.364	0.509		1.775	0.841		-25.617	0.187							
Elevators	0.558	0.505		2.600	0.613		4.039	0.175		4.544	0.278				
Ailerons	-0.013	0.225		-0.003	0.259		-0.005	0.158		-0.002	0.264				
Mv	-1.056	0.014	-	-0.677	0.066		-0.299	0.035	-	-0.099	0.017	-	-0.072	0.024	-
Kinematics	0.000	0.628		-0.003	0.336		0.000	0.570							
CompAct	-2.746	0.014	-	-1.189	0.041	-	-1.259	0.029	-						
CompAct(s)	-2.362	0.152		-0.801	0.234		-0.598	0.349							
Census(16H)	-5.27E+7	0.296		4.84E+7	0.695		2.85E+6	0.923		-9.61E+6	0.806		-1.58E+6	0.975	
Census(8L)	-1.85E+6	0.991		1.79E+8	0.294		-7.08E+7	0.504		-8.80E+7	0.151		-1.51E+8	0.191	
Fried	-0.096	0.523		-0.200	0.389		-0.239	0.157		-0.203	0.060		-0.188	0.002	--

Table B.18. Difference in error (MSE) between *LSS*+*ChiEst(95%)* and CART pruning.

	256			512			1024			2048			4096		
Abalone	-0.098	0.905		0.126	0.907										
2Dplanes	-0.074	0.218		-0.051	0.028	-	-0.084	0.001	--	-0.031	0.015	-	-0.005	0.089	
Pole	8.615	0.509		7.311	0.495		-6.023	0.710							
Elevators	-3.626	0.423		0.643	0.644		-1.972	0.234		-3.639	0.365				
Ailerons	-0.004	0.154		-0.004	0.046	-	-0.002	0.539		0.002	0.540				
Mv	0.151	0.155		-0.012	0.754		0.000	0.351		0.000	0.974		0.000	1.000	
Kinematics	0.000	0.969		0.000	0.961		0.001	0.494							
CompAct	0.056	0.938		-0.168	0.572		-0.197	0.571							
CompAct(s)	0.846	0.608		0.036	0.956		0.991	0.223							
Census(16H)	-1.29E+8	0.268		-4.72E+8	0.067		-3.29E+8	0.016	-	-1.78E+8	0.078		-1.44E+8	0.007	--
Census(8L)	-9.99E+7	0.284		-3.23E+8	0.056		-2.79E+8	0.104		-6.31E+7	0.326		-8.39E+7	0.569	
Fried	-0.367	0.049	-	-0.336	0.196		-0.013	0.916		0.066	0.344		-0.030	0.664	

Table B.19. Difference in error (MSE) between *LSS+ChiEst(95%)* and RETIS pruning.

	256			512			1024			2048			4096		
Abalone	-0.419	0.181		-0.308	0.654										
2Dplanes	0.005	0.890		-0.099	0.002	--	-0.102	0.001	--	-0.014	0.113		-0.047	0.002	--
Pole	4.615	0.752		-3.454	0.796		-18.103	0.525							
Elevators	-3.157	0.387		1.146	0.480		-2.255	0.353		-1.350	0.413				
Ailerons	-0.001	0.732		-0.002	0.562		-0.006	0.338		0.001	0.814				
Mv	-0.019	0.852		-0.164	0.028	-	-0.055	0.008	--	-0.020	0.018	-	-0.006	0.236	
Kinematics	-0.002	0.465		0.000	0.939		-0.002	0.205							
CompAct	0.053	0.859		-0.221	0.145		-0.056	0.819							
CompAct(s)	0.463	0.784		0.330	0.594		0.757	0.122							
Census(16H)	-2.63E+8	0.061		-4.12E+8	0.063		-4.04E+8	0.042	-	-2.97E+8	0.020	-	-2.45E+8	0.167	
Census(8L)	2.36E+6	0.973		-2.41E+8	0.027	-	-3.15E+8	0.078		-1.85E+8	0.272		-1.16E+8	0.524	
Fried	-0.299	0.065		-0.394	0.058		-0.089	0.402		0.050	0.463		-0.060	0.488	

Table B.20. Difference in error (MSE) between *LSS+ChiEst(95%)* and CORE pruning.

	256			512			1024			2048			4096		
Abalone	0.397	0.541		0.503	0.304										
2Dplanes	-0.462	0.000	--	-0.229	0.003	--	-0.122	0.002	--	-0.001	0.904		-0.030	0.005	--
Pole	0.765	0.953		4.912	0.557		-15.374	0.302							
Elevators	-3.066	0.409		-0.414	0.753		-3.124	0.326		1.686	0.061				
Ailerons	-0.026	0.059		-0.004	0.118		-0.004	0.042	-	0.000	0.847				
Mv	-0.984	0.018	-	-0.677	0.066		-0.299	0.035	-	-0.099	0.018	-	-0.072	0.024	-
Kinematics	0.001	0.713		-0.003	0.424		0.000	0.566							
CompAct	-3.139	0.008	--	-1.638	0.004	--	-1.136	0.038	-						
CompAct(s)	-1.579	0.141		-0.391	0.679		0.278	0.522							
Census(16H)	3.84E+7	0.599		-6.58E+7	0.313		-1.17E+8	0.262		-6.25E+7	0.089		-1.41E+6	0.974	
Census(8L)	1.67E+8	0.434		-4.49E+7	0.196		-1.02E+8	0.307		-1.15E+7	0.887		-3.53E+7	0.632	
Fried	-0.291	0.165		-0.348	0.188		-0.251	0.061		-0.093	0.403		-0.137	0.019	-

Table B.21. Difference in error between *LSS+ChiEst(95%,0.5-SE)* and CORE pruning.

	256			512			1024			2048			4096		
Abalone	0.377	0.442		0.590	0.149										
2Dplanes	-0.223	0.036	-	-0.038	0.552		-0.012	0.720		0.122	0.000	++	0.033	0.020	+
Pole	29.914	0.477		15.313	0.165		-7.067	0.622							
Elevators	-3.306	0.370		-0.362	0.795		-0.740	0.865		2.864	0.143				
Ailerons	-0.023	0.093		-0.002	0.329		-0.002	0.382		0.004	0.286				
Mv	-0.789	0.030	-	-0.566	0.127		-0.219	0.108		-0.057	0.159		-0.048	0.112	
Kinematics	0.000	0.984		-0.003	0.410		0.000	0.323							
CompAct	-3.039	0.007	--	-1.203	0.018	-	-1.401	0.024	-						
CompAct(s)	-1.455	0.180		-0.405	0.686		0.270	0.805							
Census(16H)	4.62E+7	0.472		-1.39E+7	0.814		-9.07E+7	0.380		-5.14E+7	0.245		1.60E+7	0.648	
Census(8L)	1.60E+8	0.441		-1.98E+7	0.688		-5.68E+7	0.431		1.72E+7	0.827		-6.79E+6	0.911	
Fried	-0.121	0.534		-0.270	0.214		-0.021	0.888		0.087	0.621		0.300	0.027	+

Table B.22. Difference in MSE between *LSS+ChiEst(95%,1-SE)* and CORE pruning.

	256			512			1024			2048			4096		
Abalone	0.416	0.281		0.792	0.028	+									
2Dplanes	-0.119	0.288		0.121	0.028	+	0.082	0.089		0.251	0.000	++	0.073	0.001	++
Pole	35.683	0.400		28.502	0.022	+	4.190	0.754							
Elevators	-3.294	0.373		-0.514	0.716		-1.390	0.756		3.314	0.118				
Ailerons	-0.021	0.127		0.001	0.776		-0.002	0.391		0.006	0.274				
Mv	-0.547	0.087		-0.330	0.336		-0.090	0.369		0.006	0.902		0.019	0.522	
Kinematics	0.001	0.794		-0.004	0.300		-0.001	0.593							
CompAct	-1.481	0.100		-1.160	0.032	-	-0.373	0.672							
CompAct(s)	-0.591	0.564		0.162	0.858		0.438	0.667							
Census(16H)	5.61E+7	0.377		1.48E+7	0.812		-6.55E+7	0.537		-2.03E+7	0.587		3.13E+7	0.193	
Census(8L)	2.01E+8	0.339		-7.48E+6	0.872		7.44E+6	0.896		7.57E+7	0.351		1.95E+7	0.574	
Fried	-0.037	0.853		-0.284	0.149		0.258	0.247		0.409	0.105		0.595	0.002	++

References

Agresti,A: (1990) : *Categorical data analysis.* John Wiley & Sons.

Aha, D. (1990) : A study of instance-based learning algorithms for supervised learning tasks: Mathematical, empirical, and psychological evaluations. PhD Thesis. Technical Report 90-42. University of California at Irvine, Department of Information and Computer Science.

Aha, D. (1992) : Generalizing from case studies : A case study. In *Proceedings of the 9th International Conference on Machine Learning*. Sleeman,D. & Edwards,P. (eds.). Morgan Kaufmann.

Aha, D. (1997) : *Lazy Learning*, edited by D. Aha. Kluwer Academic Publishers.

Aha,D., Kibler,D., Albert,M. (1991) : Instance-based learning algorithms. *Machine Learning*, **6** (1), 37-66.

Almuallin,H. (1996) : An efficient algorithm for optimal pruning of decision trees. *Artificial Intelligence*, **83** (2), 347-362. Elsevier.

Almuallin,H., Akiba,Y., Kaneda,S. (1995) : On handling Tree-Structured Attributes in Decision Tree Learning. In *Proceedings of the International Conference on Machine Learning (ICLM-95).* Morgan Kaufmann.

Atkeson,C.G., Moore,A.W., Schaal,S. (1997) : Locally Weighted Learning. *Artificial Intelligence Review*, **11**, 11-73. Special issue on lazy learning, Aha, D. (Ed.).

Banerji,R. (1964) : A language for the description of concepts. *General Systems*, **9**, 135-141.

Bellman,R. (1961) : *Adaptive Control Processes.* Princeton University Press.

Bentley,J.L. (1975) : Multidimensional binary search trees used for associative searching. *Communications of the ACM*, **18**(9), 509-517.

Bhattacharyya,G., Johnson,R. (1977) : *Statistical Concepts and Methods*. John Wiley & Sons.

Bohanec, M., Bratko.I. (1994) : Trading Accuracy for Simplicity in Decision Trees. *Machine Learning*, **15**-3, 223-250. Kluwer Academic Publishers.

Brazdil,P. , Gama,J., Henery,B. (1994) : Characterizing the applicability of classification algorithms using meta-level learning. In *Proceedings of the European Conference on Machine Learning* (ECML-94). Bergadano,F. & De Raedt,L. (eds.). Lecture Notes in Artificial Intelligence, 784. Springer-Verlag.

Brazdil,P. Torgo,L. (1990) : Knowledge Acquisition via Knowledge Integration. In *Current Trends in Knowledge Acquisition.* Wielinga,B. et al. (eds.). IOS Press.

Breiman,L. (1996) : Bagging Predictors. *Machine Learning*, **24**, (p.123-140). Kluwer Academic Publishers.

Breiman,L. Friedman,J. (1995) : Predicting Multivariate Responses in Multiple Linear Regression. Technical Report. Available at ftp://ftp.stat.berkley.edu/pub/users/breiman/curds-whey-all.ps.Z.

Breiman,L. , Friedman,J.H., Olshen,R.A. & Stone,C.J. (1984): *Classification and Regression Trees*. Wadsworth Int. Group, Belmont, California, USA, 1984.

Breiman,L., Meisel,W. (1976) : General estimates of the intrinsic variability of data in nonlinear regression models. *Journal of American Statistics Association*, **71**, 301-307.

Broadley, C. E. (1995) : Recursive automatic bias selection for classifier construction. *Machine Learning*, **20**, 63-94. Kluwer Academic Publishers.

Broadley,C., Utgoff,P. (1995) : Multivariate decision trees. *Machine Learning*, **19**, 45-77. Kluwer Academic Publishers.

Buntine, W. (1990) : A theory of learning classification rules. Ph.D. Thesis. University of Technology, School of Computing Science, Sydney.

Carbonell,J. (1986) : Derivational Analogy : A theory of reconstructive problem solving and expertise acquisition. In *Machine Learning, an artificial intelligence approach*, vol. II, Michalski *et al.* (eds.). Morgan Kaufmann Publishers.

Cattlet,J. (1991) : Megainduction : a test flight. In *Proceedings of the 8th International Conference on Machine Learning*. Birnbaum,L. & Collins,G. (eds.). Morgan Kaufmann.

Chatfield, C. (1983) : *Statistics for technology* (third edition). Chapman and Hall, Ltd.

Chou, P.A. (1991) : Optimal partitioning for classification and regression trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **13** (4), 340-354.

Cestnik,B. (1990) : Estimating probabilities : a crucial task in machine learning. In *Proceedings of the European Conference on Artificial Intelligence (ECAI-90).*

Cestnik,B., Bratko,I. (1988) : Learning redundant rules in noisy domains. In *Proceedings of the European Conference on Artificial Intelligence (ECAI-88).*

Cestnik,B., Kononenko,I., Bratko,I. (1987) : ASSISTANT-86 : A knowledge elicitation tool for sophisticated users. In *Progress in machine learning*, Bratko & Lavrac (eds.). Sigma Press.

Clark,P., Nibblet,T. (1989) : The CN2 induction algorithm. *Machine Learning*, **3**, 261-283. Kluwer Academic Publishers.

Cleveland, W. (1979) : Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, **74**, 829-836.

Cleveland,W., Loader,C. (1995) : Smoothing by Local Regression: Principles and Methods (with discussion). *Computational Statistics.*

Connel,M. Utgoff,P. (1987) : Learning to control a dynamical physical system. In *Proceedings of the 6th National Conference on Artificial Intelligence*. Morgan Kaufmann.

Costa, J.P. (1996) : Coefficients d'association et binarization par la classificiation hiérarchique dans les arbes de decision. Application à l'indentification de la structure secondaire des protéines. Ph.D. Thesis, Université de Rennes I, Rennes, France.

Cover,T., Hart,P. (1967) : Nearest Neighbor pattern classification. *IEEE Transactions on Information Theory*, **13**, 21-7.

Dattatreya,G., Kanal,L. (1985) : Decision trees in pattern recognition. *Progress in pattern recognition*, vol. 2, Kanal and Rosenfeld (eds.), 189-239. Elsevier Science.

DeJong,G., Mooney,R. (1986) : Explanation-based learning: An alternative view. *Machine Learning*, **1** (2), 145-176. Kluwer Academic Publishers.

Deng,K., Moore,A.W. (1995) : Multiresolution Instance-based Learning. In *Proceedings of IJCAI'95*.

De Raedt,L., Lavrac,N., Dzeroski,S. (1993) : Multiple Predicate Learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Bajcsy,R. (ed.). Morgan Kaufmann Publishers.

Dietterich,T. (1996) : Proper statistical tests for comparing supervised classification learning algorithms. Technical Report. Dept. of Computer Science, Oregon State University.

Dietterich,T. (1998) : Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. *Neural Computation*, **10** (7) 1895-1924.

Domingos,P. (1994) : Design and evaluation of the RISE 1.0 learning system. Technical Report 94-34, Department of Information and Computer Science, University of California, Irvine.

Domingos,P. (1996) : Unifying Instance-based and Rule-based Induction. *Machine Learning*, **24**-2, 141-168. Kluwer Academic Publishers.

Domingos,P. (1997) : Knowledge acquisition from examples via multiple models. In *Proceedings of the 14th International Conference on Machine Learning*, Fisher,D. (ed.). Morgan Kaufmann.

Dougherty,J., Kohavi,R., Sahami,M. (1995) : Supervised and unsupervised discretisation of continuous features. In *Proceedings of the 12th International Conference on Machine Learning*, Prieditis,A. & Russel,S. (eds.). Morgan Kaufmann.

Drapper,N., Smith,H. (1981) : *Applied Regression Analysis* (2nd edition). Wiley Series in Probability and Mathematical Statistics.

Efron,B. (1979) : Bootstrap methods : another look at the jacknife. *Annals of Statistics*, **7**(1), 1-26.

Efron,B, Tibshirani,R. (1993) : *An introduction to the bootstrap*. Chapman & Hall.

Esposito,F., Malerba,D., Semeraro,G. (1993) : Decision Tree Pruning as a Search in the State Space. In *Proceedings of the European Conference on Machine Learning (ECML-93)*, Brazdil,P. (ed.). LNAI-667, Springer Verlag.

Esposito,F., Malerba,D., Semeraro,G. (1997) : A Comparative Analysis of Methods for Pruning Decision Trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **19** – 5.

Fan,J. (1995) : Local Modelling. In *Encyclopidea of Statistical Science*.

Fan,J., Marron,J. (1993) : Comment on Hastie & Loader (1993). *Statistical Science,* **8**, 120-143.

Fisher,D. (1987) : Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, **2**, 139-172. Kluwer Academic Publishers.

Fisher,D. (1992) : Pessimistic and Optimistic Induction. Tech. Report CS-92-12. Dept. of Computer Science, Vanderbilt University.

Fisher, R.A. (1936) : The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, **7**, 179-188.

Fisher, W.D. (1958) : On grouping for maximum hogeneity. *Journal of American Statistical Association*, **53**, 789-798.

Fix, E., Hodges, J.L. (1951) : Discriminatory analysis, nonparametric discrimination consistency properties. Technical Report 4, Randolph Filed, TX: US Air Force, School of Aviation Medicine.

Freund,Y. (1995) : Boosting a weak learning algorithm by majority. *Information and Computation*, **121** (2), 256-285.

Freund,Y., Schapire,R.E. (1995) : A decision-theoretic generalization of on-line learning and an application to boosting. Technical Report . AT & T Bell Laboratories.

Friedman,J. (1979) : A tree-structured approach to nonparametric multiple regression. In *Smoothing Techniques for Curve Estimation*, Gasser & Rosenblatt (eds.), 5-22. Springer. New York.

Friedman, J. (1991) : Multivariate Adaptative Regression Splines. *Annals of Statistics*, **19**:1, 1-141.

Friedman,J.H., Stuetzle,W. (1981) : Projection pursuit regression. *Journal of the American Statistical Association*, **76** (376), 817-823.

Gama,J. (1997) : Probabilistic linear tree. In *Proceedings of the 14th International Conference on Machine Learning*, Fisher,D. (ed.). Morgan Kaufmann.

Gama,J. (1998) : Combining classifiers using constructive induction. In *Proceedings of the 10th European Conference on Machine Learning*, Nedellec,C. and Rouveirol,C. (eds.). LNAI - 1398, Springer Verlag.

Gama,J. (1998) : Local cascade generalization. In *Proceedings of the 15th International Conference on Machine Learning*, Shavlik,J. (ed.). Morgan Kaufmann.

Gama,J., Brazdil,P. (1995) : Characterization of Classification Algorithms. In *Proc. of the 7th Portuguese Conference on Artificial Intelligence (EPIA-95).* Lecture Notes in AI. Springer Verlag.

Gams,M. (1989) : New measurements that highlight the importance of redundant knowledge. In *Proceedings of the 4th European Working Session on Learning*, Morik,K. (ed.). Pitman - Morgan Kaufmann.

Gelfand,S., Delp,E. (1991) : On tree structured classifiers. In Sethi & Jain (1991), p. 51-70.

Gelfand,S., Ravishankar,C., Delp,E. (1991) : An iterative growing and pruning algorithm for classification tree design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **13**, p. 163-174.

Good, I.J. (1965) : *The Estimation of Probabilities.* MIT press.

Hardle,W. (1990) : *Applied Nonparametric Regression*. Cambridge University Press.

Harrison,D., Rubinfeld,D. (1978) : Hedonic prices and the demand for clean air. *Journal of Environment Econ. and Management*, **5** , 81-102.

Hastie,T., Loader,C. (1993) : Local Regression: Automatic Kernel Carpentry. *Statistical Science,* **8**, 120-143.

Hastie,T., Tibshirani,R. (1990) : *Generalized Additive Models*. Chapman & Hall.

Highleyman,W.H. (1962) : The design and analysis of pattern recognition experiments. *Bell Systems Technical Journal*, **41**, 723-744.

Holte,R., Acker,L., Porter,B. (1989) : Concept Learning and the accuracy of small disjuncts. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 813-818. Morgan Kaufmann.

Hong,S.J. (1994) : Use of contextual information for feature ranking and discretization. Technical Report RC19664, IBM. To appear in IEEE Trans. on Knowledge and Data Engineering.

Hunt,E., Marin,J., Stone,P. (1966) : *Experiments in Induction*. Academic Press.

John,G. (1997) : Enhancements to the data mining process. Ph.D. Thesis. Department of Computer Science, University of Stanford.

Karalic,A. (1992) : Employing Linear Regression in Regression Tree Leaves. In *Proceedings of ECAI-92*. Wiley & Sons.

Karalic,A. (1995) : First Order Regression. Ph.D. Thesis. Faculty of Electrical Engineering and Computer Science, University of Ljubljana.

Karalic,A., Cestnik,B. (1991) : The bayesian approach to tree-structured regression. In *Proceedings of ITI-91*.

Karmarkar,N. (1984) : A new polynomial time algorithm for linear programming. *Combinatorica*, **4** (4), 373-381.

Katkovnik,V. (1979) : Linear an nonlinear methods of nonparametric regression analysis. *Soviet Automatic Control*, **5**, 25-34.

Kendall,M., Stuart,A. (1969) : *The Advanced Theory of Statistics* (vol. 1-3). Charles Griffin & Company Limited.

Kibler,D., Aha,D., Albert,M. (1989) : Instance-based prediction of real-valued attributes. *Computational Intelligence*, **5**, 51-57.

Kira,K., Rendell,L.A. (1992) : The feature selection problem : traditional methods and new algorithm. In *Proceedings of AAAI'92*.

Kohavi. R. (1985) : Wrappers for performance enhancement and oblivious decision graphs. Ph.D. thesis. Stanford University.

Kononenko,I. (1989) : Interpretation of neural networks decisions. In *Proceedings of the IASTED International Conference on Expert Systems and Applications*, Zurich.

Kononenko,I. (1991) : An experiment in machine learning of redundant knowledge. In *Proceedings of the International Conference MELECON 1991*, Ljubljana.

Kononenko,I. (1994) : Estimating attributes : analysis and extensions of Relief. In *Proceedings of the European Conference on Machine Learning (ECML-94).* Bergadano,F. & De Raedt,L. (eds.). Lecture Notes in Artificial Intelligence, 784. Springer-Verlag.

Kononenko,I. Bratko,I., Roskar,E. (1984) : Experiments in automatic learning of medical diagnostic rules. Technical Report. Jozef Stefan Institute, Slovenia.

Kramer,S. (1996) : Structural regression trees. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96).* MIT Press.

Kubat,M. (1996) : Second tier for decision trees. In *Proceedings of the 13th International Conference on Machine Learning*. Saitta,L. (ed.). Morgan Kaufmann.

Langley,P. (1996) : *Elements of Machine Learning*. Morgan Kaufmann Publishers.

Langley,P., Zytkow,J., Simon,H. (1986) : The search for regularity : four aspects of scientific discovery. In *Machine Learning, an artificial intelligence approach*, vol. II, Michalski *et al.* (eds.). Morgan Kaufmann Publishers.

Lavrac,N., Dzeroski,S. (1994) : *Inductive Logic Programming : Techniques and applications*. Ellis Horwood.

Lenat, D.B. (1970) : The ubiquity of discovery. *Artificial Intelligence*, **9**, 257-285. Elsevier Science.

Loader,C. (1995) : Old Faithful Erupts: Bandwidth Selection Reviewed. Technical Report 95.9. AT&T Bell Laboratories, Statistics Department.

Lubinsky,D. (1995) : Tree Structured Interpretable Regression. In *Proceedings of the 5th International Workshop on AI and Statistics*. Ft. Lauderdale, Florida.

Martin,J. (1997) : An exact probability metric for decision tree splitting and stopping. *MachineLearning*, **28** (2/3). Kluwer Academic Publishers.

Mansfield,A. (1991) : Comparison of perceptron training by linear programming and by the perceptron convergence procedure. In *IEEE International Joint Conference on Neural Networks*.

McClelland,J.L., Rumelhart,D.E. (1981) : An interactive activation model of context effects in letter perception : Part 1. An accoutn an basic findings. *Psychological Review*, **88**, 375-407.

McCulloch,W. Pitts,W. (1943) : A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5**, 115-133.

Michalski, R. (1983) : A theory and methodology of inductive learning. In *Machine Learning, an artificial intelligence approach*, Michalski *et al.* (eds.). Tioga Publishing Company.

Michalski,R. (1990) : Learning Flexible Concepts : Fundamental ideas and a method based on two-tiered representation. In *Machine Learning : an artificial intelligence approach*, vol. 3, Kodratoff,Y. & Michalski,R. (eds.). Morgan Kaufmann.

Michalski,R. (1994) : Inferential Theory of Learning : Developing Foundations for Multistrategy Learning. In *Machine Learning : a Multistrategy Approach*, vol. 4, Michalski,R. & Tecuci,G. (eds.). Morgan Kaufmann.

Michalski,R. Chilausky,R. (1980) : Learning by being told and learning from examples : An experimental comparison of two methods of knowledge acquaisition in the context of developing an expert system for soybean disease diagnosis. *International Journal of Policy Analysis and Information Systems*, **4**.

Michalski,R., Mozetic,I., Hong,J., Lavrac,N. (1986) : The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, AAAI Press.

Michalski,R., Stepp,R. (1983) : Learning from observation : conceptual clustering. In *Machine Learning, an artificial intelligence approach*, Michalski *et al.* (eds.). Tioga Publishing Company.

Michie,D., Spiegelhalter;D.J. & Taylor,C.C. (1994): *Machine Learning, Neural and Statistical Classification*. Ellis Horwood Series in Artificial Intelligence.

Mingers,J. (1989) : An Empirical Comparison of Pruning Methods for Decision Tree Induction. *Machine Learning*, **4**-2, p.227-243. Kluwer Academic Publishers.

Minsky,M., Papert,S. (1969) : *Perceptrons : an introduction to computational geometry*. MIT Press.

Mitchell, T. (1982) : Generalization as search. *Artificial Intelligence*, **18**, 203-226. Elsevier Science.

Mitchell, T. (1997) : *Machine Learning*. McGraw-Hill.

Mitchell,T., Utgoff,P., Banerji,R. (1983) : Learning by experimentation: Acquiring and refining problem-solving heuristics. In *Machine Learning, an artificial intelligence approach*, Michalski *et al.* (eds.). Tioga Publishing Company.

Moore,A., Lee,M. (1998) : Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, **8**, 67-91.

Moore,A. Schneider,J., Deng,K. (1997) : Efficient Locally Weighted Polynomial Regression Predictions. In *Proceedings of the 14th International Conference on Machine Learning (ICML-97)*, Fisher,D. (ed.). Morgan Kaufmann Publishers.

Morgan,J., Sonquist,J. (1963) : Problems in the analysis of survey data, and a proposal. *Journal of American Statistics Society*, **58**, 415-434.

Mosteller,F., Wallace,D.L. (1963) : Inference in an authorship problem. *Journal of the American Statistical Association*, **58**, 275-309.

Muggleton,S. (1992) : *Inductive Logic Programming*. Academic Press.

Muggleton,S., De Raedt,L. (1994) : Inductive Logic Programming : Theory and Methods. *The Journal of Logic Programming*, vols. **19-20**. Elsevier Science.

Muggleton,S., Feng,C. (1990) : Efficient induction of logic programs. In *Proceedings of the first Conference on Algorithmic Learning Theory*.

Muller,W., Wysotzki,F. (1994) : A splitting algorithm, based on a statistical approach in the decision tree algorithm CAL5. In *Proceedings of the ECML-94 Workshop on Machine Learning and Statistics*. Nakhaeizadeh,G. & Taylor,C. (eds.).

Murthy,S. (1996) : On growing better decision trees from data. Ph.D. thesis. Johns Hopkins University.

Murthy,S., Kasif,S., Salzberg,S. (1994) : A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, **2**, 1-33.

Nadaraya, E.A. (1964) : On estimating regression. *Theory of Probability and its Applications*, **9**:141-142.

Natarajan,B. (1991) : *Machine Learning - a theoretical approach*. Morgan Kaufmann.

Niblett,T., Bratko,I. (1986) : Learning decision rules in noisy domain. *Expert Systems*, **86**. Cambridge University Press.

Oliveira,A.L., Sangionvanni-Vincentelli,A. (1996) : Using the minimum description length principle to infer reduced order decision graphs. *Machine Learning*, **26**, 23-50, Kluwer Academic Publishers.

Oliver,J.J. (1993) : Decision graphs – an extension of decision trees. In *Proceedings of the fourth International workshop on Artificial Intelligence and Statistics*, 343-350.

Parzen,E. (1962) : On estimation of a probability density function and mode. *Annals Mathematical Statistics*, **33**, 1065-1076.

Plotkin,G. (1970) : A note on Inductive Generalization. In *Machine Intelligence 5*, Meltzer & Michie (eds.). Edinburgh University Press.

Plotkin,G. (1971) : A further note on inductive generalization. In *Machine Intelligence 6*, Meltzer & Michie (eds.). Edinburgh University Press.

Press,W.H., Teukolsky,S.A., Vetterling,W.T., Flannery,B.P. (1992) : *Numerical Recipes in C*. Cambridge University Press.

Quinlan,J. (1979) : Discovering rules by induction from large collections of examples. In *Experts systems in the micro electronic age*, Michie,D. (ed.). Edinburgh University Press.

Quinlan, J. (1986) : Induction of Decision Trees. *Machine Learning*, **1**, 81-106. Kluwer Academic Publishers.

Quinlan,J. (1990) : Learning Logical Definitions from Relations. *Machine Learning*, **5**, 239-266. Kluwer Academic Publishers.

Quinlan, J. (1992): Learning with Continuous Classes. In *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*. Singapore: World Scientific, 1992.

Quinlan, J. (1993a) : *C4.5 : programs for machine learning*. Morgan Kaufmann Publishers.

Quinlan,J. (1993b) : Combining Instance-based and Model-based Learning. In *Proceedings of the 10th ICML*. Morgan Kaufmann.

Quinlan,J. (1996) : Learning First-Order Definitions of Functions. *Journal of Artificial Intelligence Research*, **5**, 139-161. (http://www.jair.org/home.html).

Quinlan,J. Rivest,R. (1989) : Inferring decision trees using the minimum description length principle. *Information & Computation*, **80**, 227-248.

Rasmussen,C. (1996) : Evaluation of Gaussian Processes and other Methods for Non-linear Regression. Ph.D. Thesis, Department of Computer Science, University of Toronto.

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

Rissanen,J. (1982) : A universal prior for integers and estimation by the minimum description length. *Annals of Statistics*, **11**, 416-431.

Robnik-Sikonja,M. (1997) : CORE – a system that predicts continuous variables. In *Proceedings of ERK'97*, Portoroz, Slovenia.

Robnik-Sikonja,M., Kononenko,I. (1996) : Context-sensitive attribute estimation in regression. In *Proceedings of the ICML-96 Workshop on Learning in Context-Sensitive Domains*.

Robnik-Sikonja,M., Kononenko,I. (1997) : An adaptation of Relief for attribute estimation in regression. In *Proceedings of ICML-97*, Fisher,D.(ed.). Morgan Kaufmann Publishers.

Robnik-Sikonja,M., Kononenko,I. (1998) : Pruning Regression Trees with MDL. In *Proceedings of ECAI-98*.

Rojas,R. (1996) : *Neural Networks*. Springer-Verlag.

Rosenblatt,F. (1958) : The perceptron : a probabilistic model for information storage and organization in the brain. *Psychological Review*, **65**, 386-408.

Rosenblatt,M. (1956) : Remarks on some nonparametric estimates of a density function. *Annals Mathematical Statistics*, **27**, 832-837.

Rumelhart,D., Hinton,G., Williams,R. (1986) : Learning internal representations by error propagation. In *Parallel distributed processing*, vol.1, Rumelhart *et al.*(eds.). MIT Press.

Rumelhart,D., Widrow,B., Lehr,M. (1994) : The basic ideas in neural networks. *Communications of the ACM*, **37** (3), 87-92.

Salzberg,S. (1988) : Examplar-based learning : theory and implementation. Technical Report TR-10-88. Cambridge,MA, Harvard University.

Salzberg,S. (1991) : A nearest hyperrectangle learning method. *Machine Learning*, **6**-3, 251-276. Kluwer Academic Publishers.

Salzberg.S. (1997) : On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach. *Data Mining and Knowledge Discovery*, **1**, 317-327. Kluwer Academic Publishers.

Sammut,C. (1993) : The origins of Inductive Logic Programming : A Prehistoric Tale. In *Proceedings of the Third International Workshop on Inductive Logic Programming*, Muggleton,S. (ed.). Josef Stefan Institute, Technical report IJS-DP-6707.

Schaffer,C. (1992) : Deconstructing the Digit Recognition Problem. In *Proceedings of the 9th International Machine Learning Conference (ICML99)*, Sleeman,D. and Edwards,P. (eds.). Morgan Kaufmann.

Schaffer,C. (1993a) : Overfitting Avoidance as Bias. *Machine Learning*, **10** (2). Kluwer Academic Publishers.

Schaffer,C. (1993b) : Selecting a Classification Method by Cross-validation. *Machine Learning*, **13** (1). Kluwer Academic Publishers.

Schaffer,C. (1994) : A conservation law for generalization performance. In *Proceedings of the 11th International Conference on Machine Learning*. Morgan Kaufmann.

Schlimmer ,J., Fisher,D. (1986) : A case study if incremental concept induction. In *Proceedings of the 5th AAAI Conference*. Morgan Kaufmann.

Sethi,I., Jain,A. (eds) (1991) : *Artificial Neural Networks and Statistical Pattern Recognition: Old and New Connections*. North Holland.

Silverman,B. (1986) : *Density Estimation for Statistics and Data Analysis*. Chapman and Hall.

Simon,H. (1983) : Why Should Machines Learn ? In *Machine Learning, an Artificial Intelligence approach*, Michalski *et al.* (eds.). Tioga Publishing Company.

Smyth,P., Gray,A., Fayyad,U.M. (1995) : Retrofitting Decision Tree Classifiers using Kernel Density Estimation. In *Proceedings of the 12th International Conference Machine Learning*. Prieditis,A., Russel,S. (Eds.). Morgan Kaufmann.

Spiegelman,C. (1976) : Two techniques for estimating treatment effects in the presence of hidden variables: adaptive regression and a solution to Reiersol's problem. Ph.D. Thesis. Dept. of Mathematics, Northwestern University.

Stone, C.J. (1977) : Consistent nonparametric regression. *The Annals of Statistics*, **5**, 595-645.

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

Swain,P., Hauska,H. (1977) : The decision tree classifier design and potential. *IEEE Transactions on Geoscience and Electronics*, **GE-15**, 142-147.

Torgo,L. (1992) : YAILS, an incremental learning program. Technical Report 92-1, LIACC, University of Porto. Postscript and HTML available in http://www.ncc.up.pt/~ltorgo/Papers/list_reports.html.

Torgo, L. (1993a) : Controlled Redundancy in Incremental Rule Learning. In *Proceedings of the European Conference on Machine Learning (ECML-93)*, Brazdil.P. (ed.). LNAI-667, Springer-Verlag. Also available in http://www.ncc.up.pt/~ltorgo/Papers/list_pub.html.

Torgo,L. (1993b) : Rule combination in inductive learning. In *Proceedings of the European Conference on Machine Learning (ECML-93)*, Brazdil.P. (ed.). LNAI-667, Springer-Verlag. Also available in http://www.ncc.up.pt/~ltorgo/Papers/list_pub.html.

Torgo, L. (1995) : Data Fitting with Rule-based Regression. In *Proceedings of the 2nd international workshop on Artificial Intelligence Techniques (AIT'95)*, Zizka,J. and Brazdil,P. (eds.). Brno, Czech Republic. Also available in http://www.ncc.up.pt/~ltorgo/Papers/list_pub.html.

Torgo,L. (1997) : Functional models for Regression Tree Leaves. In *Proceedings of the International Conference on Machine Learning (ICML-97)*, Fisher,D. (ed.). Morgan Kaufmann Publishers. Also available in http://www.ncc.up.pt/~ltorgo/Papers/list_pub.html.

Torgo,L. (1998) : Sequence-based methods for Pruning Regression Trees. Technical Report 98-1, LIACC, University of Porto. Postscript available in http://www.ncc.up.pt/~ltorgo/Papers/list_reports.html.

Torgo,L. Gama,J. (1996) : Regression by Classification. In *Proceedings of SBIA'96*, Borges *et al.* (eds.). Springer-Verlag. Also available in http://www.ncc.up.pt/~ltorgo/Papers/list_pub.html.

Torgo,L., Gama,J. (1997) : Regression using classification algorithms. *Intelligent Data Analysis*, **1** (4). Elsevier Science. http://www.elsevier.com/locate/ida.

Utgoff,P. (1989) : Incremental induction of decision trees. *Machine Learning*, **4** (2), 161-186. Kluwer Academic Publishers.

Wallace,C., Patrick,J. (1993) : Coding decision trees. *Machine Learning*, **11** (1), 7-22. Kluwer Academic Publishers.

Watson, G.S. (1964) : Smooth Regression Analysis. *Sankhya: The Indian Journal of Statistics*, Series A, **26** : 359-372.

Weiss, S. and Indurkhya, N. (1993) : Rule-base Regression. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pp. 1072-1078.

Weiss,S., Indurkhya,N. (1994) : Decision Tree Pruning: Biased or Optimal ? In *Proceedings of AAAI-94*.

Weiss, S. and Indurkhya, N. (1995) : Rule-based Machine Learning Methods for Functional Prediction. *Journal of Artificial Intelligence Research* (JAIR), **3**, pp.383-403.

Werbos,P. (1974) : Beyond regression – New tools for prediction and analysis in the behavioral sciences. Ph.D. thesis, Harvard University.

Werbos,P. (1996) : *The roots of backpropagation – from observed derivatives to neural networks and political forecasting*. J. Wiley & Sons.

Wettschereck,D. (1994) : A study of distance-based machine learning algorithms. PhD thesis. Oregon State University.

Wettschereck,D., Aha,D.W., Mohri,T. (1997) : A review and empirical evaluation of feature weighting methods for a class of lazzy learning algorithms. *Artificial Intelligence Review*, **11**, 11-73. Special issue on lazy learning, Aha, D. (Ed.).

Winston,P. (1970) : Learning structural descriptons from examples. Ph.D. thesis. MIT Technical Report AI-TR-231.

Winston, P. (1980) : Learning an reasoning by analogy. *Communications of the ACM*, **23**, 689-703.

Wolpert,D.H. (1992) : Stacked Generalization. *Neural Networks*, **5**, (p.241-259).