# 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 $\alpha$ is called the complexity parameter and defines the cost of each leaf.

Depending on the cost of each additional leaf (*i.e.* the $\alpha$ value) different sub-trees of $T_{\max}$ minimise the error-complexity measure. Breiman and his colleagues proved that although $\alpha$ can run through a continuum of values there is a sequence of pruned trees such that each element is optimal for a range of $\alpha$, and so there is only a finite number of "interesting" $\alpha$ 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 $\alpha_i$. 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 $[\alpha_i .. \alpha_{i+1})$. 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" $\alpha$ 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 ($\alpha$), 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 $T^v_{\max}$ 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 $\alpha$ 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 $\alpha$ values. The main danger of this tree-matching process results from the fact that these trees with similar $\alpha$ 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 $\alpha$ value most similar to $\alpha'_i$. 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,

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

and for the mean squared error,

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

♦

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,

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

where,

$T_t$ is the sub-tree of $T$ rooted at node $t$;
and $\#\tilde{T}_t$ 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 $O\left(\#\tilde{T} \log \#\tilde{T}\right)$, and a worst case complexity of $O\left(\#\tilde{T}^2\right)$ 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.

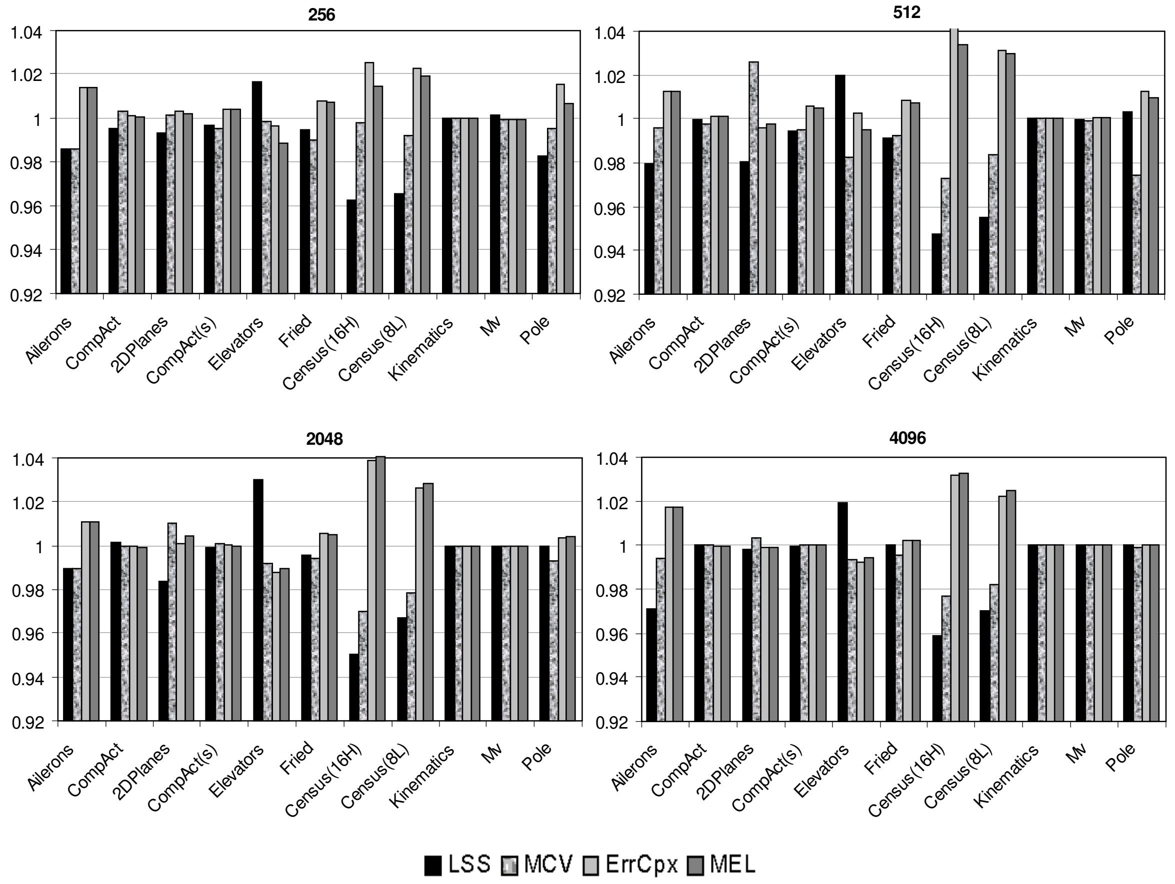


*Figure 4.1 - Comparison of the four methods for generating sequences of sub-Trees.*

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 $\theta$. 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 $\theta$, one being preferable to the other because its sampling distribution is more tightly spread around the true value of $\theta$. 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 $\alpha$ 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 $D_1,\ldots,D_k$, 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,

$Err_i = | \, y_i - r(\beta, \mathbf{x}_i) \, |$ or $Err_i = (y_i - r(\beta, \mathbf{x}_i))^2$, 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,

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

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 $\alpha$ 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 $\alpha$ 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 $\alpha$ value. Let us denote a tree belonging to this sequence as $T(\alpha_i)$ to reinforce this association between the trees and the respective $\alpha$ 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 $T^1\left(\sqrt{\alpha_i\alpha_{i+1}}\right),\ldots,T^k\left(\sqrt{\alpha_i\alpha_{i+1}}\right)$, where $T(\alpha_r)$ is the tree belonging to the sequence $T(\alpha)$ with $\alpha$ value most similar to $\alpha_r$. The assumption behind this procedure is that the trees $T^1\left(\sqrt{\alpha_i\alpha_{i+1}}\right),\ldots,T^k\left(\sqrt{\alpha_i\alpha_{i+1}}\right)$ 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 $T^1_{\max},\ldots,T^k_{\max}$ (*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 $\rho^2$ 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 $\rho_i^2$ values by $\rho_0^2$ (*i.e.* by the maximum value of $\rho_i^2$). 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 $\vartheta^2$ 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].

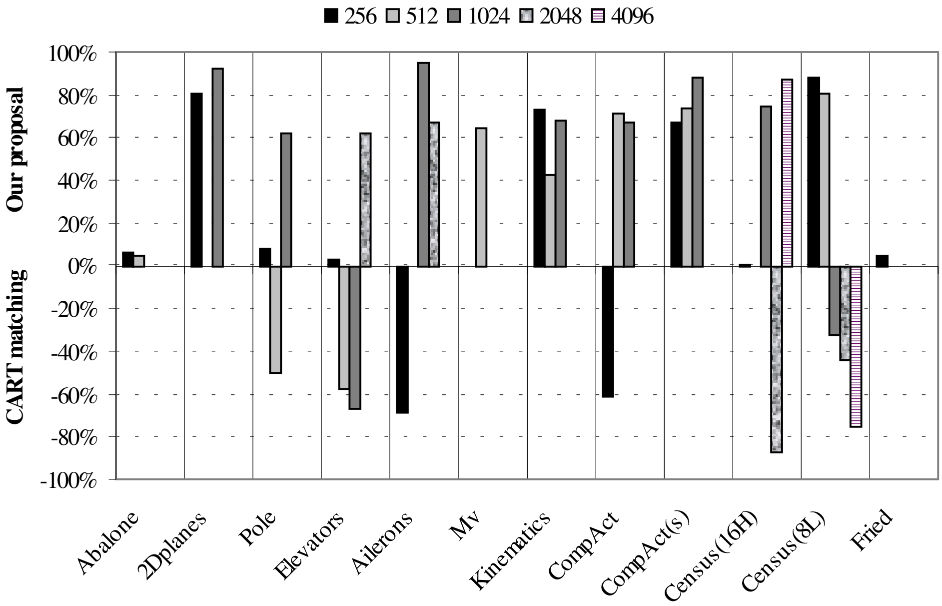


*Figure 4.2 - Our tree-matching proposal vs. CART method.*

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:

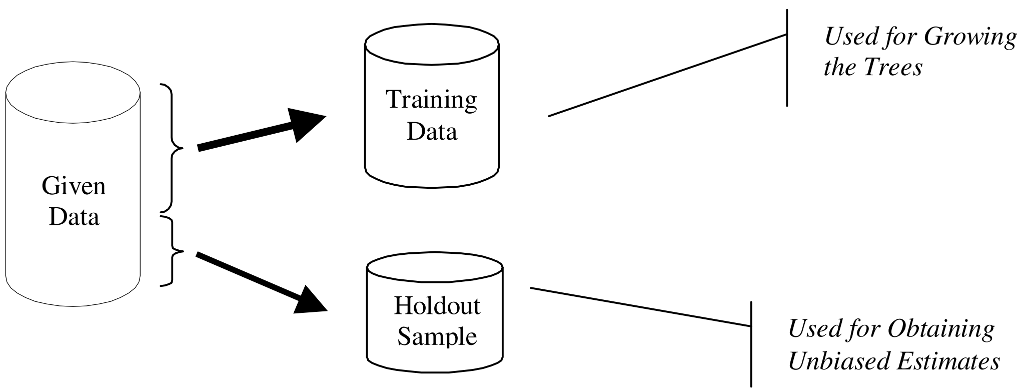


*Figure 4.3 - The Holdout Method.*

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

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

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 $\mu$ and variance $\sigma^2$ 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,

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

where,

$\sigma_E^2$ 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,

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

---

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

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,

$$m\text{Est}\left(S.E.\left(MSE_{\bar{y}}\right)\right)=$$

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

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

$$\sqrt{\frac{1}{n}\left(\frac{1}{n}\sum_{i=1}^{n}\left(y_i-m\text{Est}(\bar{y})\right)^4-\left(\frac{1}{n}\sum_{i=1}^{n}\left(y_i-m\text{Est}(\bar{y})\right)^2\right)^2\right)}= \frac{1}{n}\sqrt{s_4-4s_3m\text{Est}(\bar{y})+4s_2\left(m\text{Est}(\bar{y})\right)^2-\frac{1}{n}s_2^2+\frac{4}{n}s_2s_1m\text{Est}(\bar{y})-\frac{4}{n}s_1^2\left(m\text{Est}(\bar{y})\right)^2} \quad (4.24)$$

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,

$$m\text{Est}(\nu)=\frac{n_l}{n_l+m}\nu(D_l)+\frac{m}{n_l+m}\nu(D_n) \quad (4.25)$$

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,

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

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,

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

where,

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

Using the sample estimates of the $\sigma^2$ and $\delta_\nu$ 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 $\chi^2$ distribution (*e.g.* Bhattacharyya & Johnson, 1977), if the original variable follows a normal distribution. According to the properties of the $\chi^2$ 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 $\chi^2$ distribution for a given confidence level $\alpha$ 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 $\chi^2$ 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 $\chi^2$ 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 $\chi^2$ distribution.

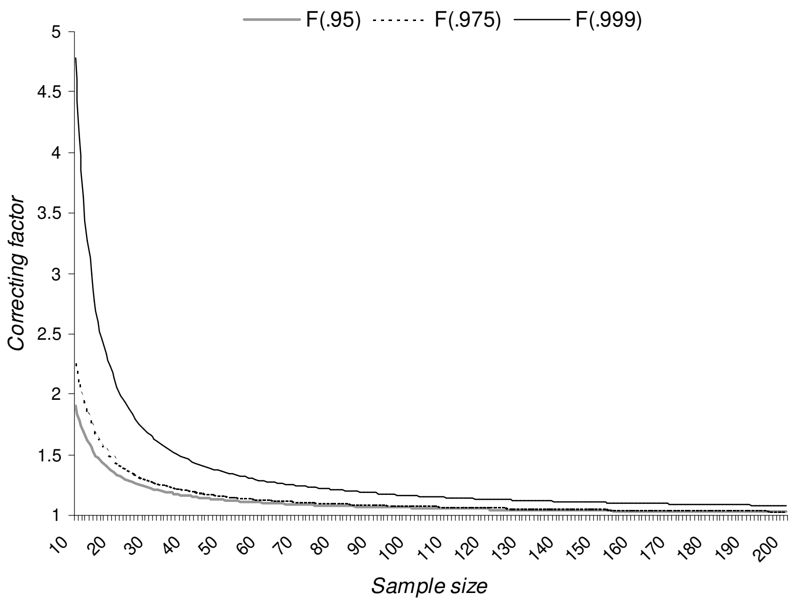


*Figure 4.4 –Different values of the "correcting factor" used in the ChiEst estimator.*

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 $\hat{E}rr_l + S.E.(\hat{E}rr_l)$, where $\hat{E}rr_l$ is the lowest error estimate and $S.E.(\hat{E}rr_l)$ 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 $T_{\max}$.

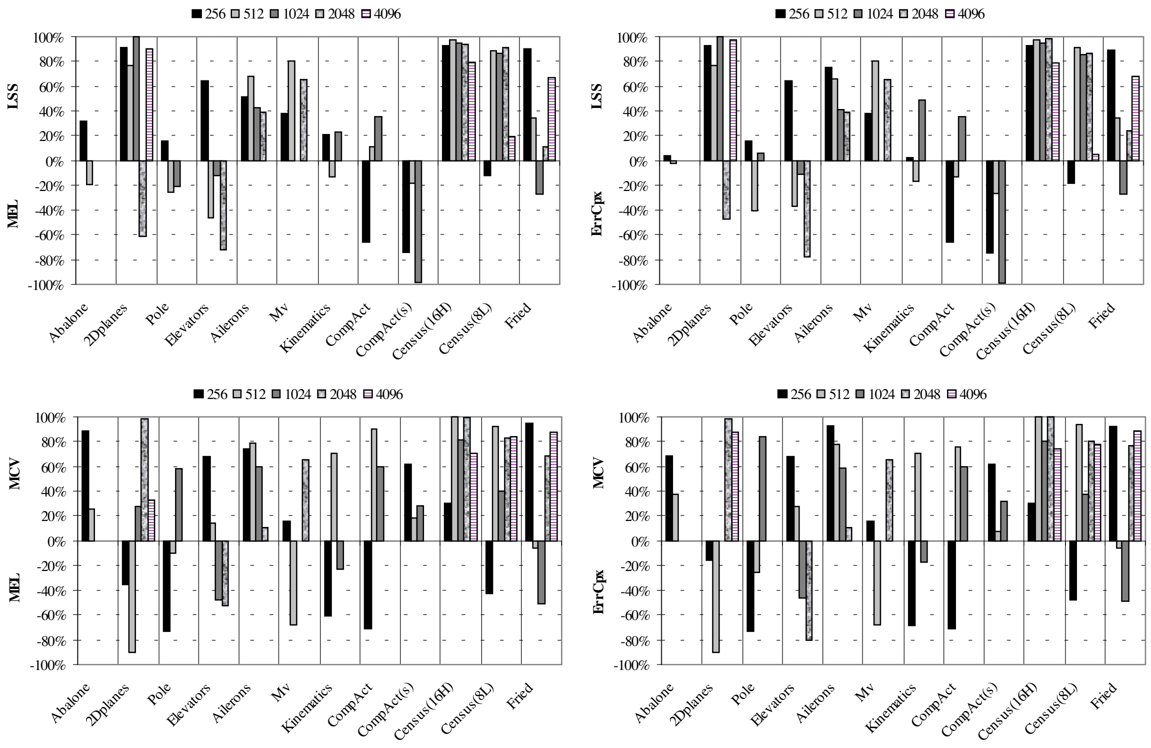


*Figure 4.5 – Comparison of LSS with other sequence generation methods using ChiEst(95%) as selection method.*

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 ($\alpha$ 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 $\chi^2$ 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.

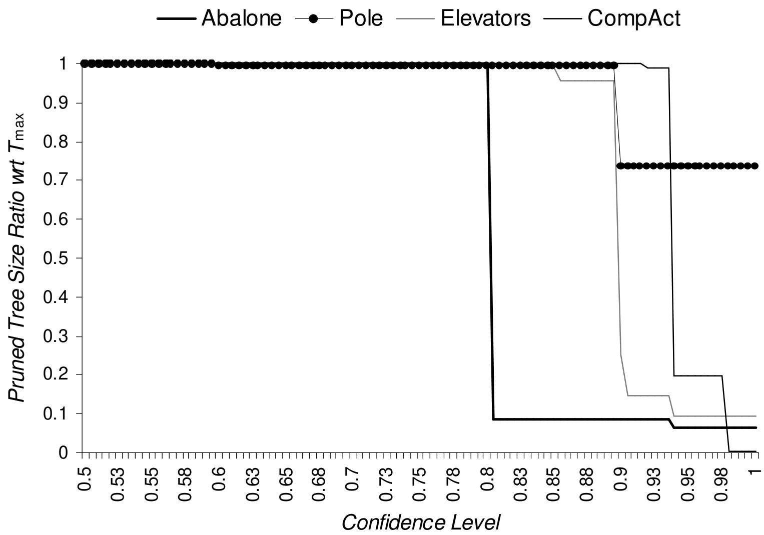


*Figure 4.6 - The effect of the value of the confidence level on the pruned tree size.*

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.

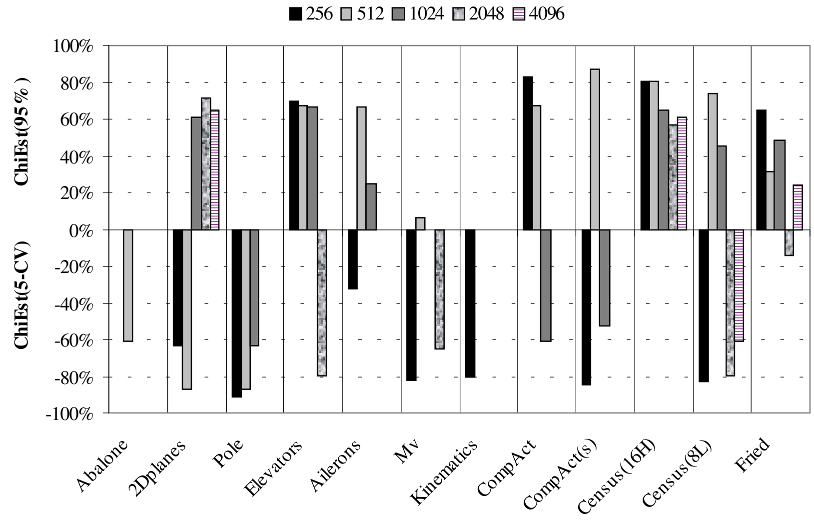


*Figure 4.7 - Significance of MSE difference between ChiEst(95%) and ChiEst(5-CV).*

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.

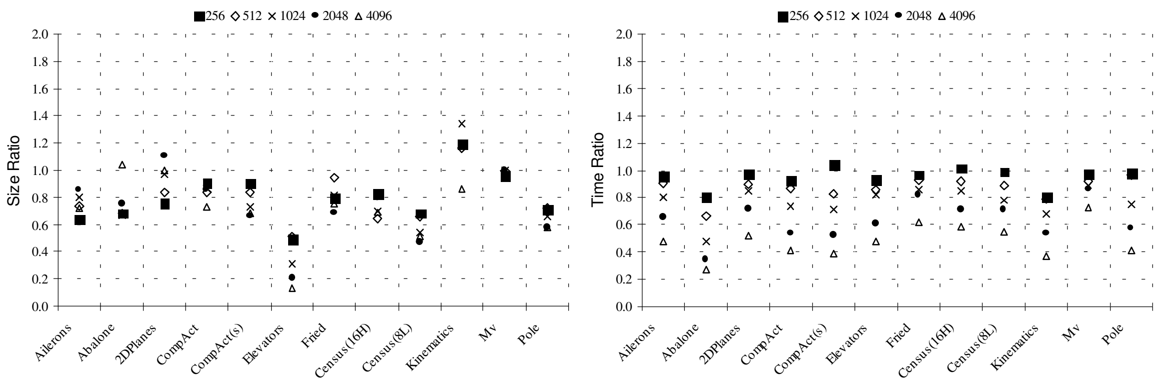


*Figure 4.8 - Tree size and Cpu time ratios between ChiEst(95%) and ChiEst(cv).*

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.

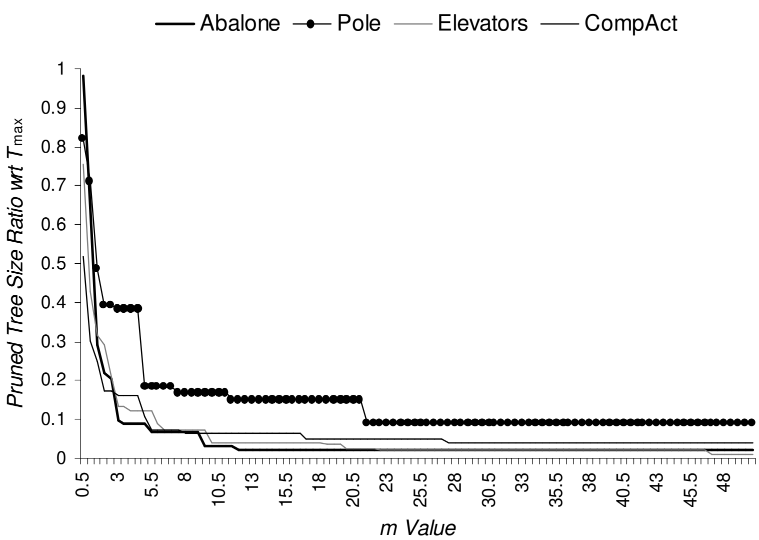


*Figure 4.9 - Variation of tree size for different m values.*

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 $m_i = 0.1 \times e^{i/5}$ , $i = 0..30$.

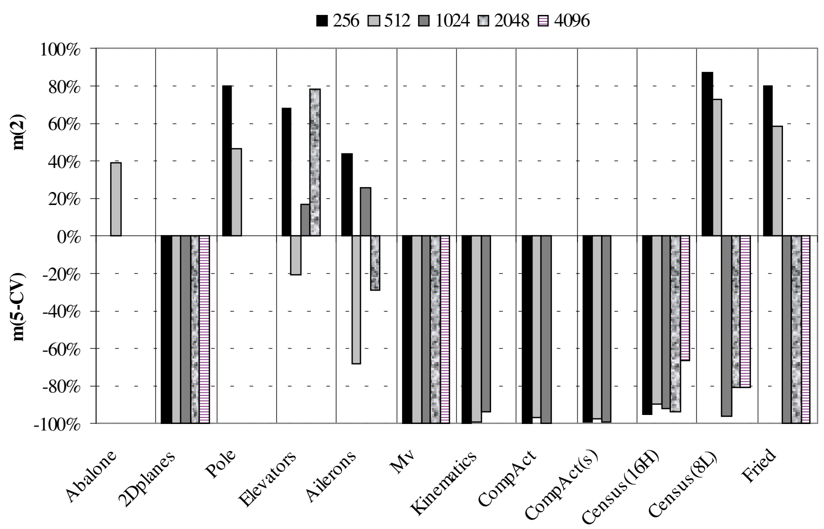


*Figure 4.10 - Significance of MSE difference between m(2) and m(5-CV).*

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).

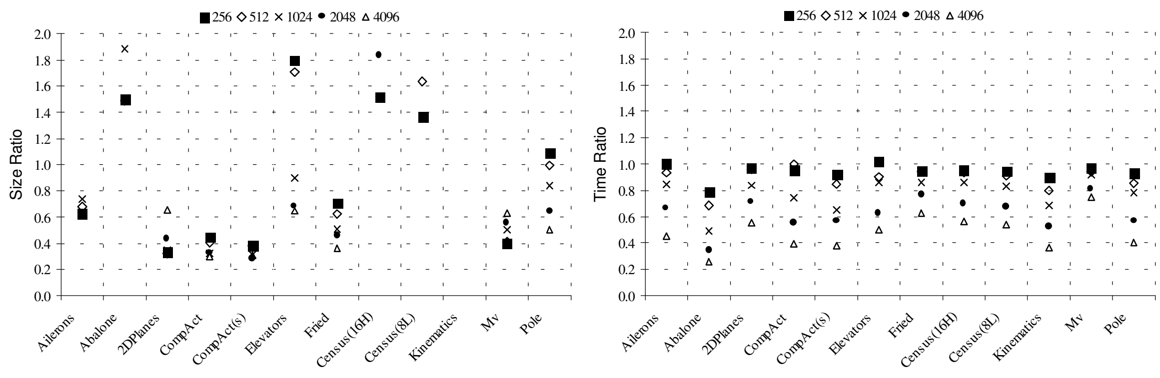


*Figure 4.11 - Tree size and Cpu time Ratios for m=2 and m(cv) selection.*

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.

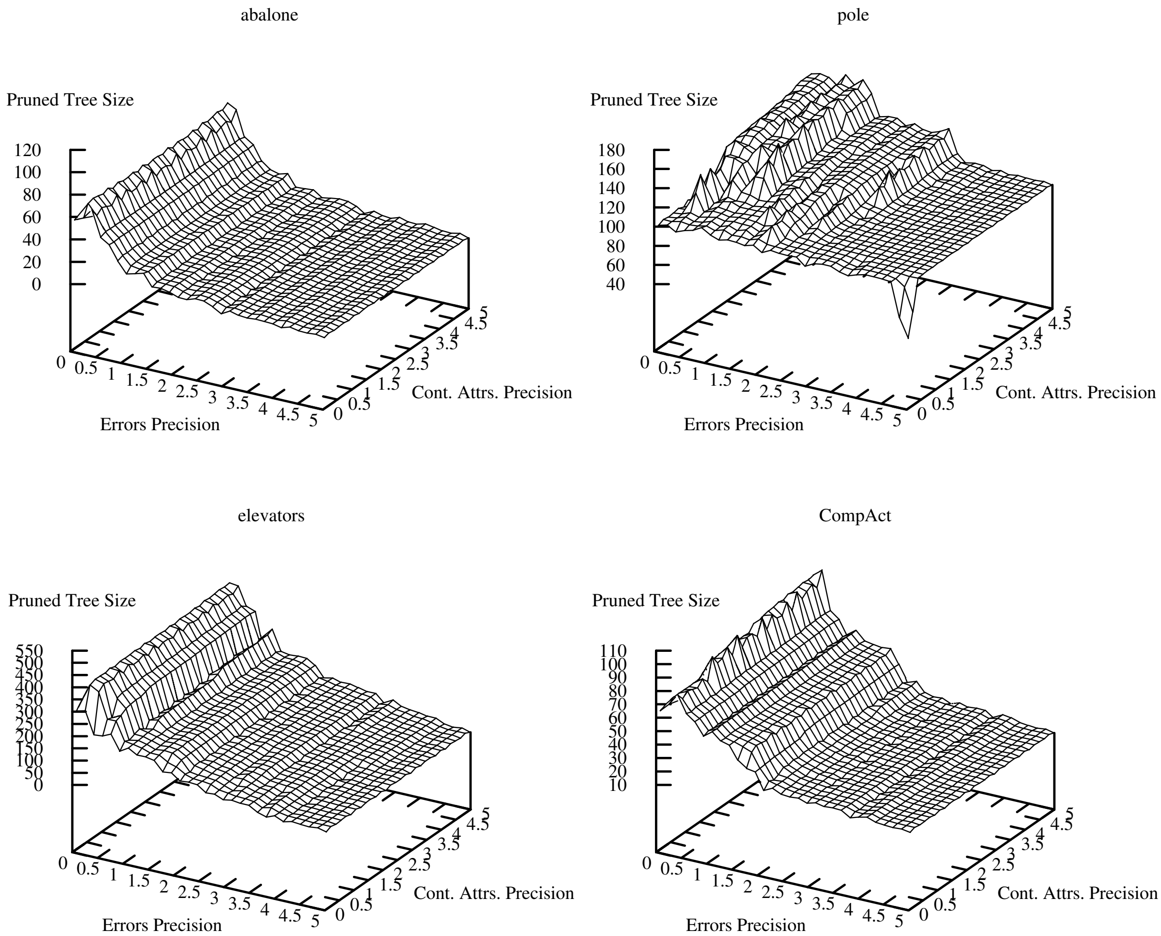


*Figure 4.12 - The effect of varying the MDL coding parameters on tree size.*

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.

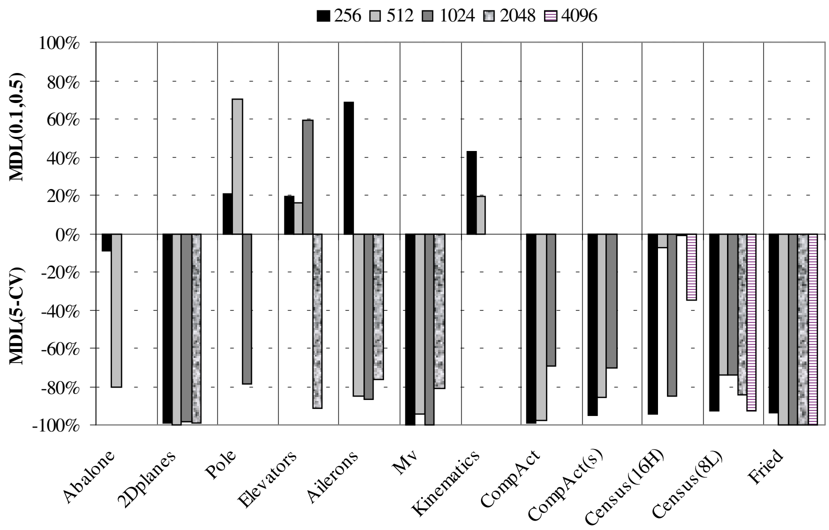


*Figure 4.13 - Significance of MSE difference between MDL(0.1,0.5) and MDL(5-CV).*

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.

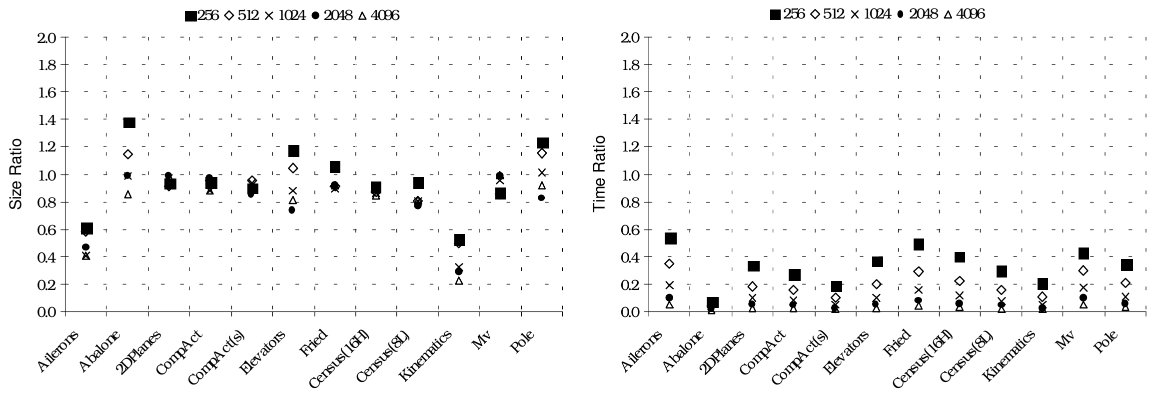


*Figure 4.14 - Tree size and Cpu time ratios between MDL(0.1,0.5) and MDL(cv5).*

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.

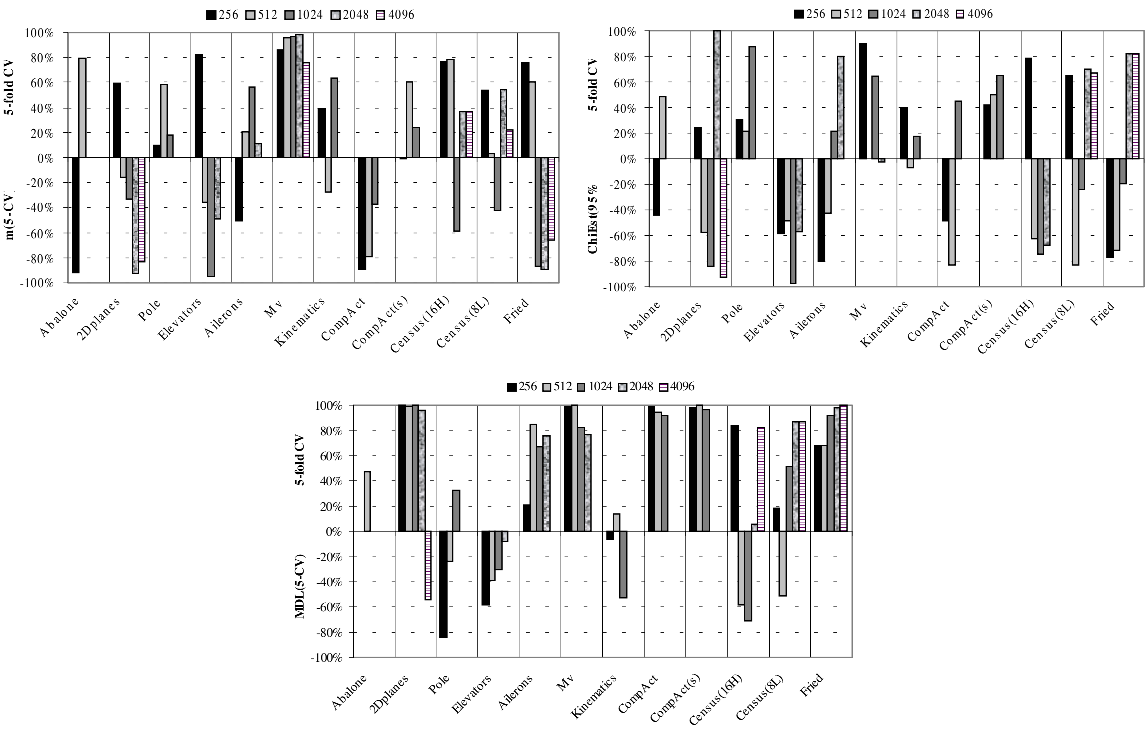


*Figure 4.15 - Significance of MSE difference between tree selection methods.*

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.

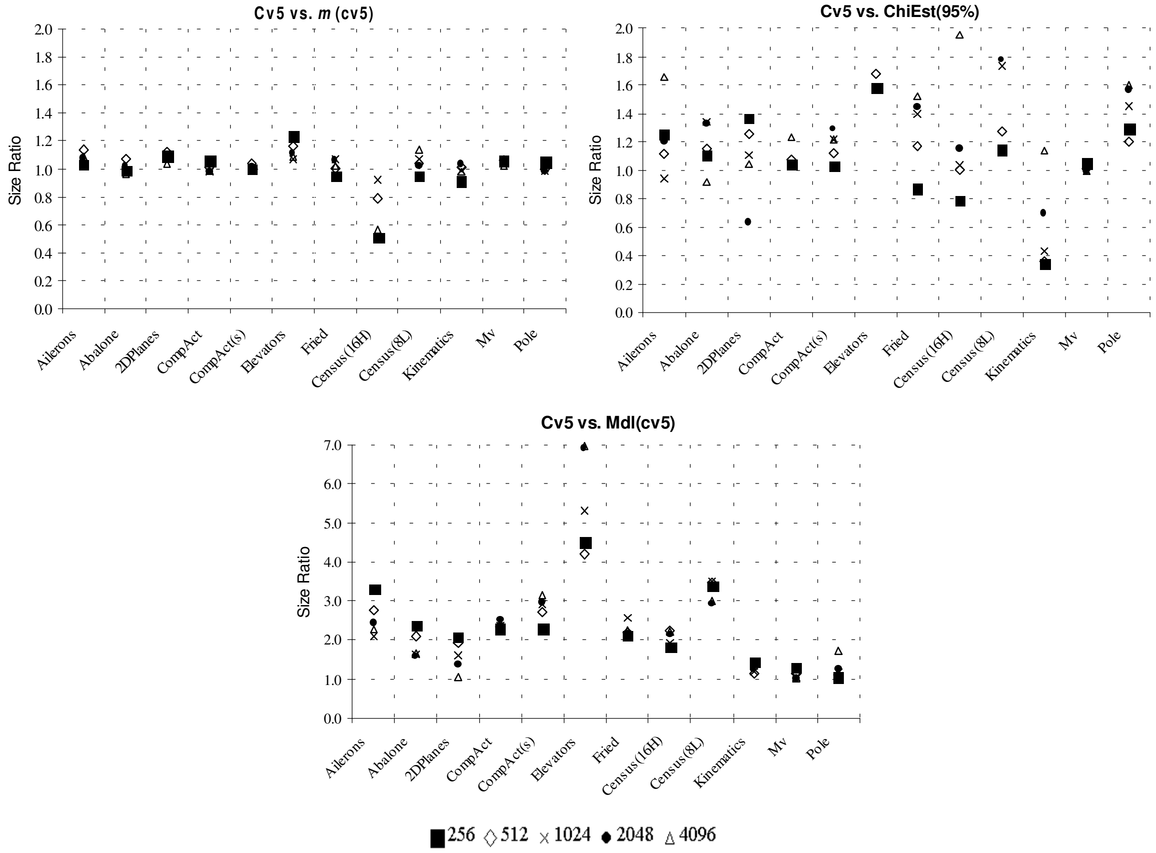


*Figure 4.16 - Tree Size Ratio comparison among Selection Methods.*

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.

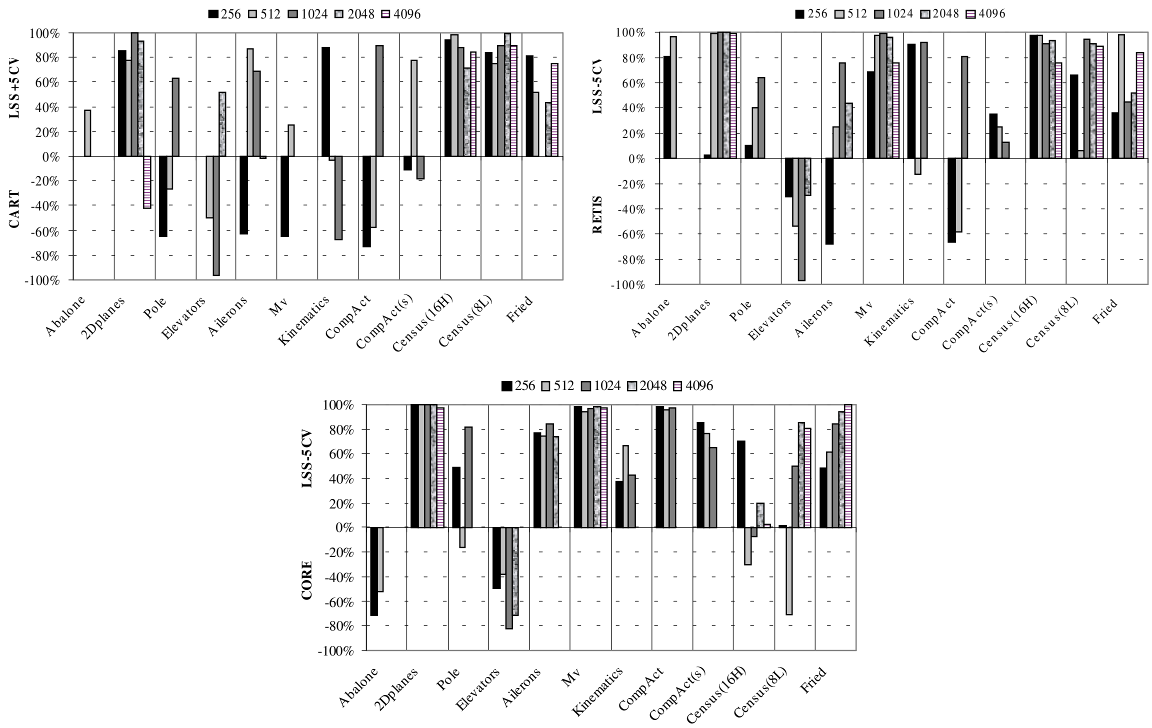


*Figure 4.17 - Significance of MSE difference between LSS+5CV and other pruning algorithms.*

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.

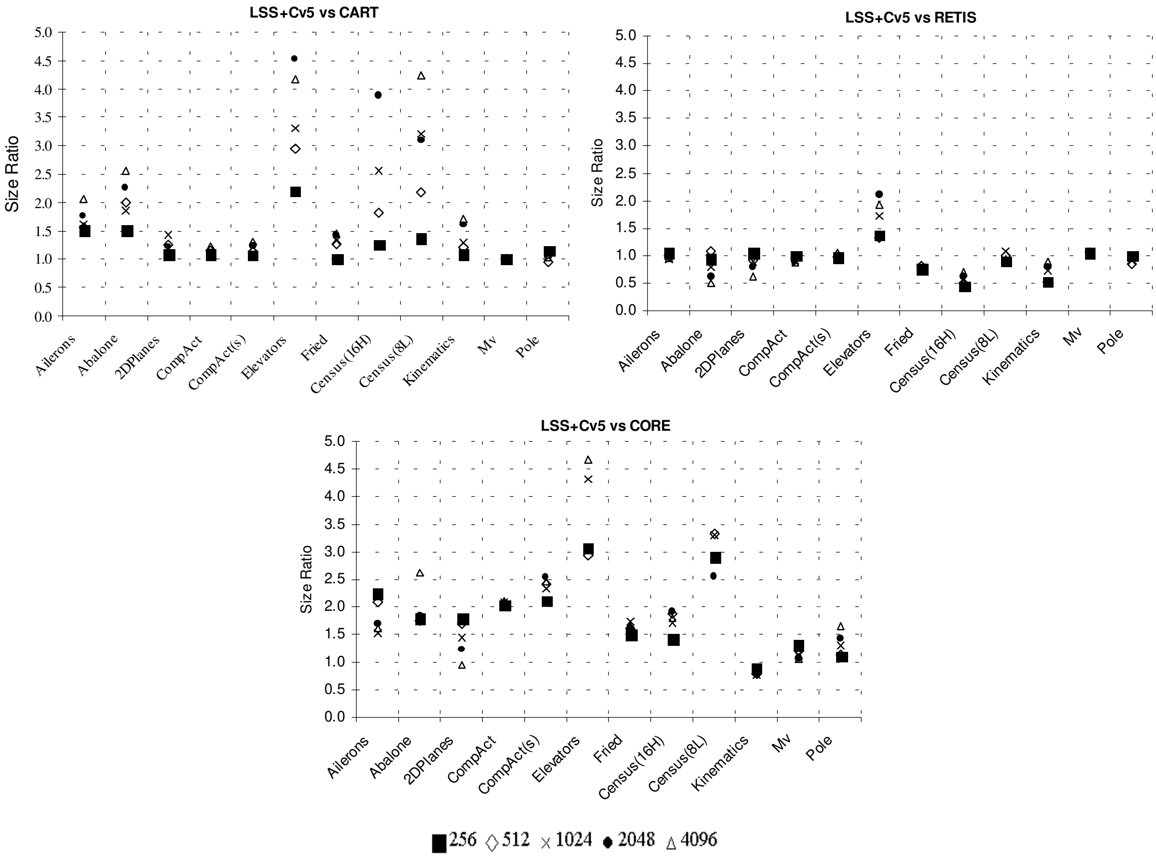


*Figure 4.18 - Tree size ratios between LSS+CV5 and other pruning algorithms.*

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.

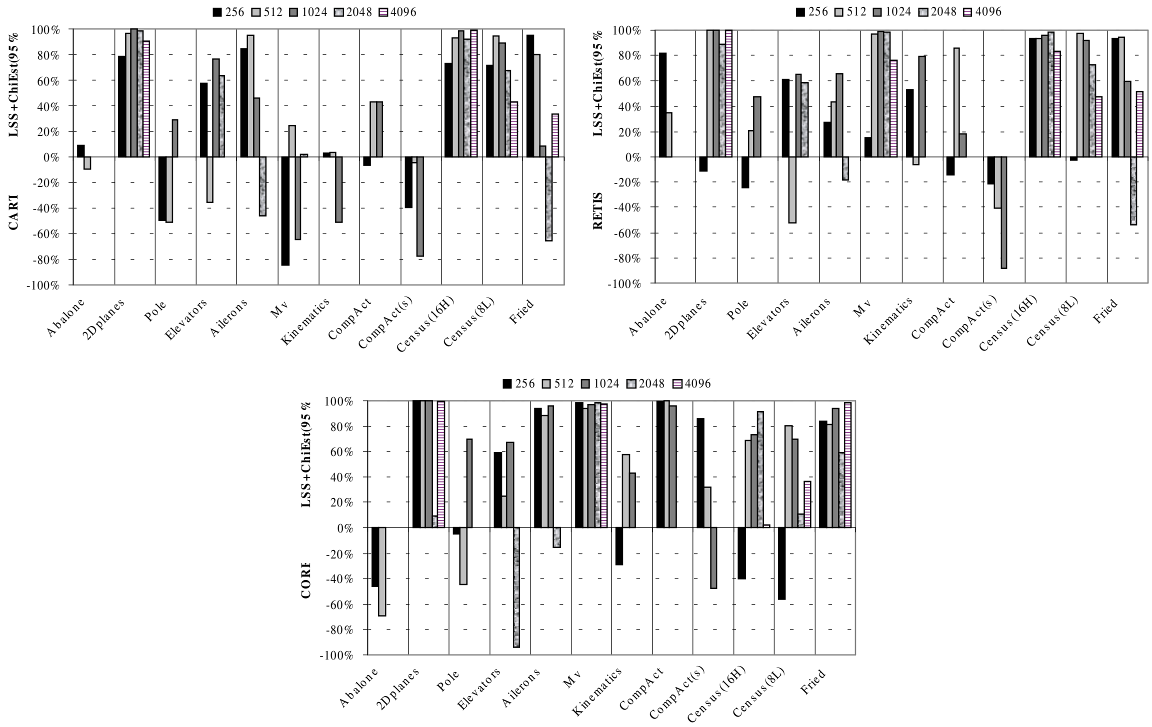


*Figure 4.19 - Significance of the MSE difference between LSS+ChiEst(95%) and other pruning algorithms.*

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.

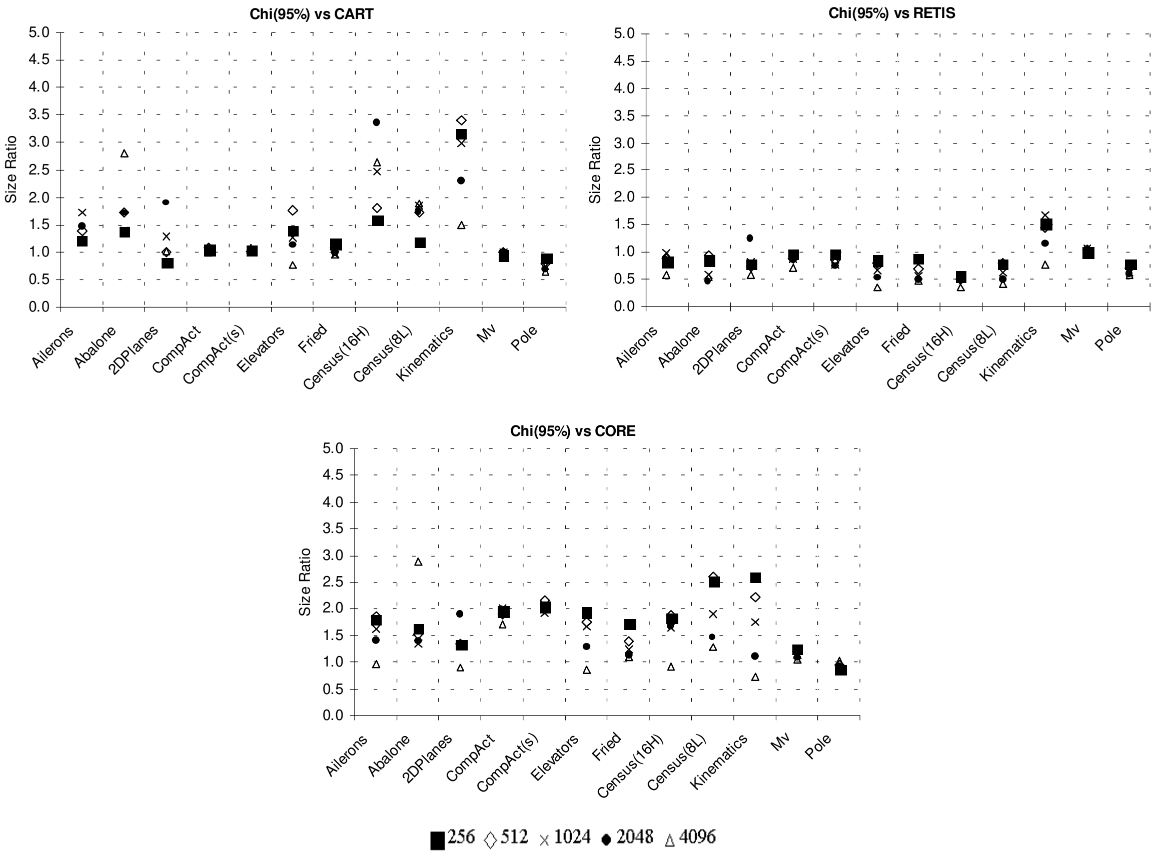


*Figure 4.20 - Tree size ratios between LSS+ChiEst(95%) and other pruning algorithms.*

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.

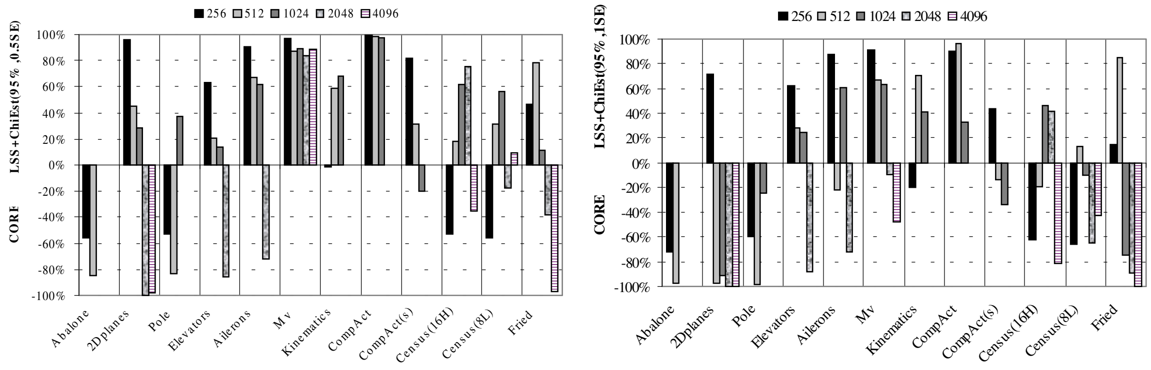


*Figure 4.21 - Significance of MSE difference between ChiEst with the k-SE rule and CORE.*

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.

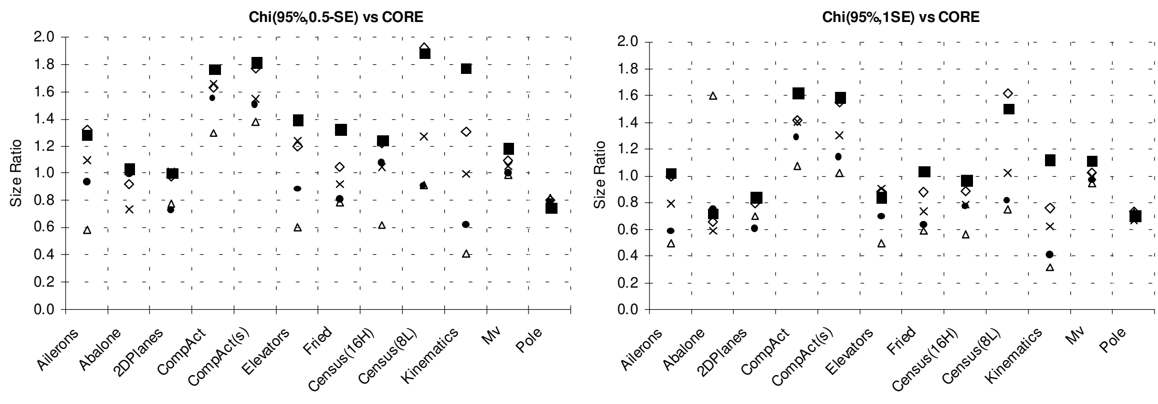


*Figure 4.22 - The effect on tree size of the k-SE rule when compared to CORE.*

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.

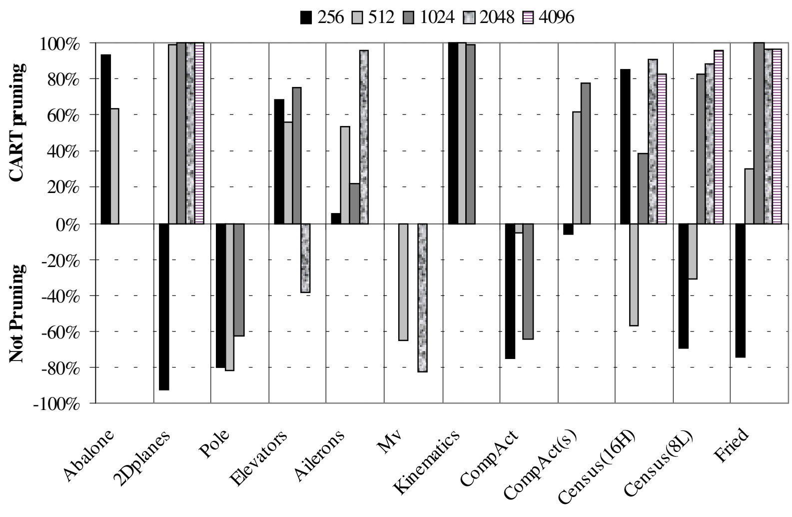


*Figure 4.23 - Significance of MSE difference between CART and not pruning.*

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 $\varepsilon$, 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 $\chi_i$, 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 $\varepsilon$ 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.