Chapter 3

Tree-based Regression

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

3.1 Tree-based Models

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

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

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

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

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

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

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

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

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

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

where,

k_i are constants;

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

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

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

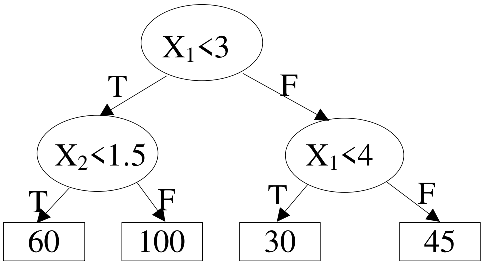
Example 3.1 provides a better illustration of this type of models through a small example of a regression tree:

---

[20] All nodes except the leaves.

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

<u>EXAMPLE 3.1</u>



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

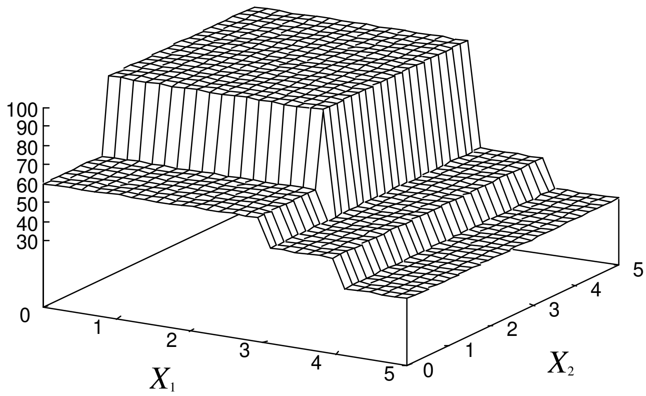
$P_2 \equiv X_1 < 3 \wedge X_2 \geq 1.5$, with $k_2 = 100$

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

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

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

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



Using the more concise representation of Equation 3.1 we obtain:

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

♦

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

```
Algorithm 3.1 - Recursive Partitioning Algorithm.

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

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

The algorithm has three main components:

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

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

3.2 Least Squares Regression Trees

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

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

where,

n is the sample size;

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

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

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

The following theorem holds for the LS minimisation criterion:

THEOREM 3.1

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

♦

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

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

where,

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

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

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

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

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

where, k_t is defined by Equation 3.3.

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

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

where,

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

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

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

where,

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

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

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

DEFINITION 3.1

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

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

♦

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

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

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

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

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

3.2.1 Efficient Growth of LS Regression Trees

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

Algorithm 3.2 – Growing a LS Regression Tree.

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

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

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

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

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

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

---

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

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

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

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

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

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

♦

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

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

DEFINITION 3.2

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

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

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

♦

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

3.2.2 Splits on Continuous Variables

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

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

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

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

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

3.2.3 Splits on Discrete Variables

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

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

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

$$\bar{y}(b_1) \le \bar{y}(b_2) \le \; ... \; \le \bar{y}(b_{\#B})$$

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

DEFINITION 3.3 (BREIMAN ET AL., 1984)[27]

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

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

♦

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

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

---

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

EXAMPLE 3.2

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

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

leading to the averages

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

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

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

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

♦

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

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

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

Obtain the average Y value associated to each value of X_v
Sort the values of X_v according to the average Y associated to each value
S_R = S_t ; S_L = 0
n_R = n_t ; n_L = 0
BestTillNow = 0
FOR each value b of the obtained ordered set of values DO
       YB = sum of the Y values of the cases with X_v = b
       NB = number of the cases with X_v = b
       S_L = S_L + YB; S_R = S_R - YB
       n_L = n_L + NB   ; n_R = n_R - NB
       NewSplitValue = (S_L^2 / n_L)  + (S_R^2 / n_R)
       IF ( NewSplitValue > BestTillNow ) THEN
              BestTillNow = NewSplitValue
              BestPosition = position of b in set of ordered values
       ENDIF
ENDFOR
```

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

3.2.4 Some Practical Considerations

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

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

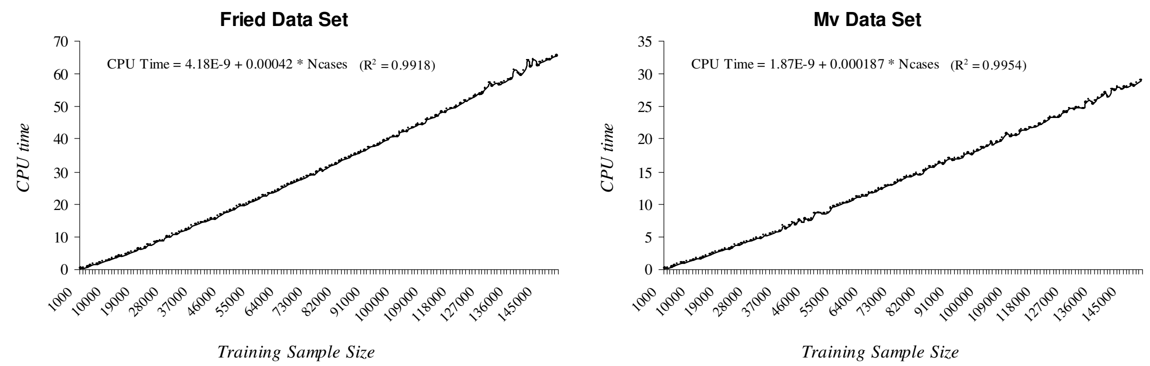


Figure 3.1 - Computation time to generate a LS tree for different sample sizes.

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

---

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

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

3.3 Least Absolute Deviation Regression Trees

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

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

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

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

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

<u>THEOREM 3.2</u>

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

♦

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

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

DEFINITION 3.4

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

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

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

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

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

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

♦

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

3.3.1 Splits on Continuous Variables

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

$X_i < 145$ $X_i \geq 145$

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

$\nu_L = 53$
$SAD_L = 407$

$\nu_R = 67$
$SAD_R = 708$

Figure 3.2 - A split on X_i < 145.

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

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

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

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

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

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

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

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

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

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

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

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

or equivalently,

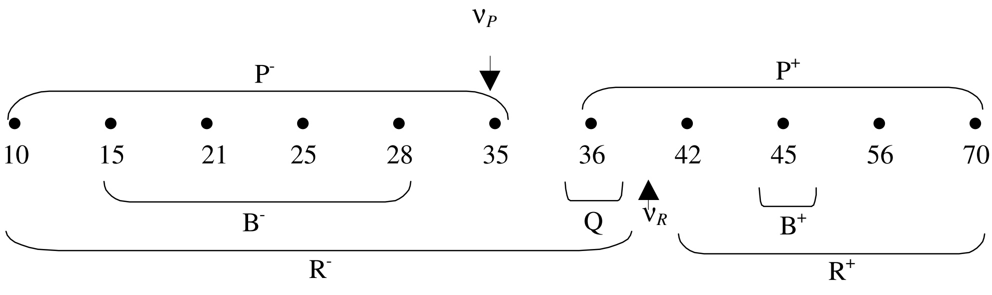
$$SAD_L = (200 + 230) - (10 + 13 + 53) + 53 \times (3 - 2) = 430 - 76 + 53 = 407$$

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

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

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

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



where,

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

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

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

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

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

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

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

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

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

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

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

This leads to,

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

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

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

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

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

leading to,

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

Integrating all three cases into one single formula we get,

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Algorithm 3.6 – Best LAD split of a continuous variable.

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

Sort the cases according to their value in Xv

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

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

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

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

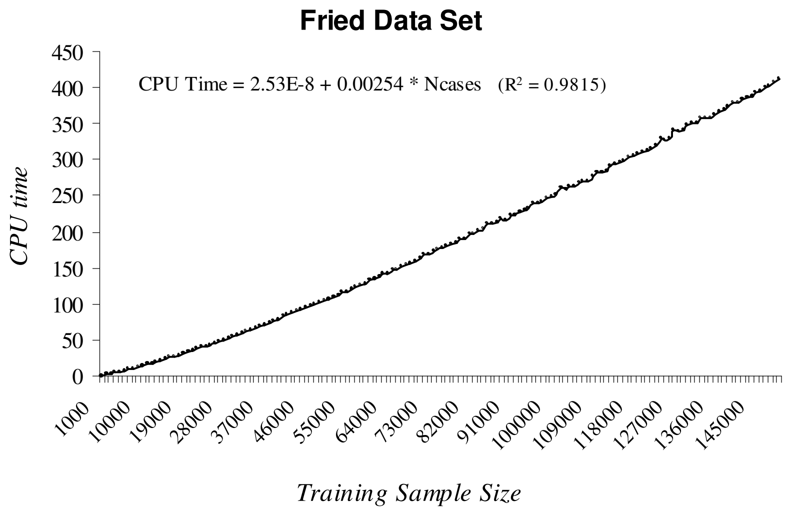


Figure 3.3 - Computation time to grow a LAD tree for different sample sizes of the Fried domain.

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

3.3.2 Splits on Discrete Variables

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

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

HYPOTHESIS 3.1

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

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

♦

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

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

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

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

---

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

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

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

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

---

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

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

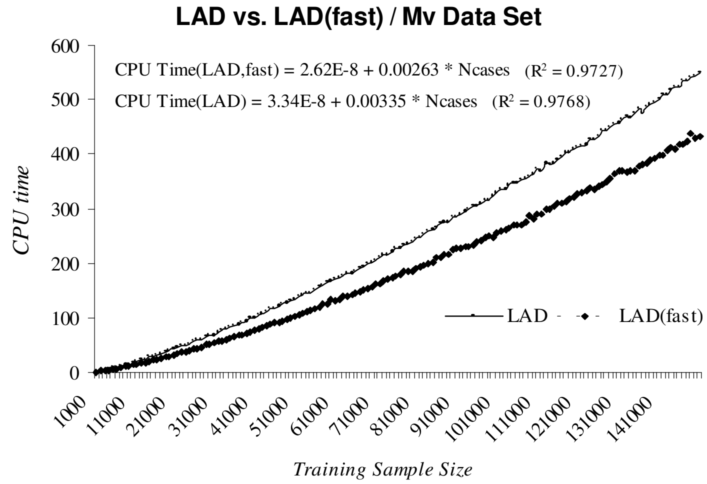


Figure 3.4 - Computation Time of LAD and LAD(fast) trees for different sample sizes.

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

3.4 LAD vs. LS Regression Trees

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

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

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

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

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

---

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

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

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

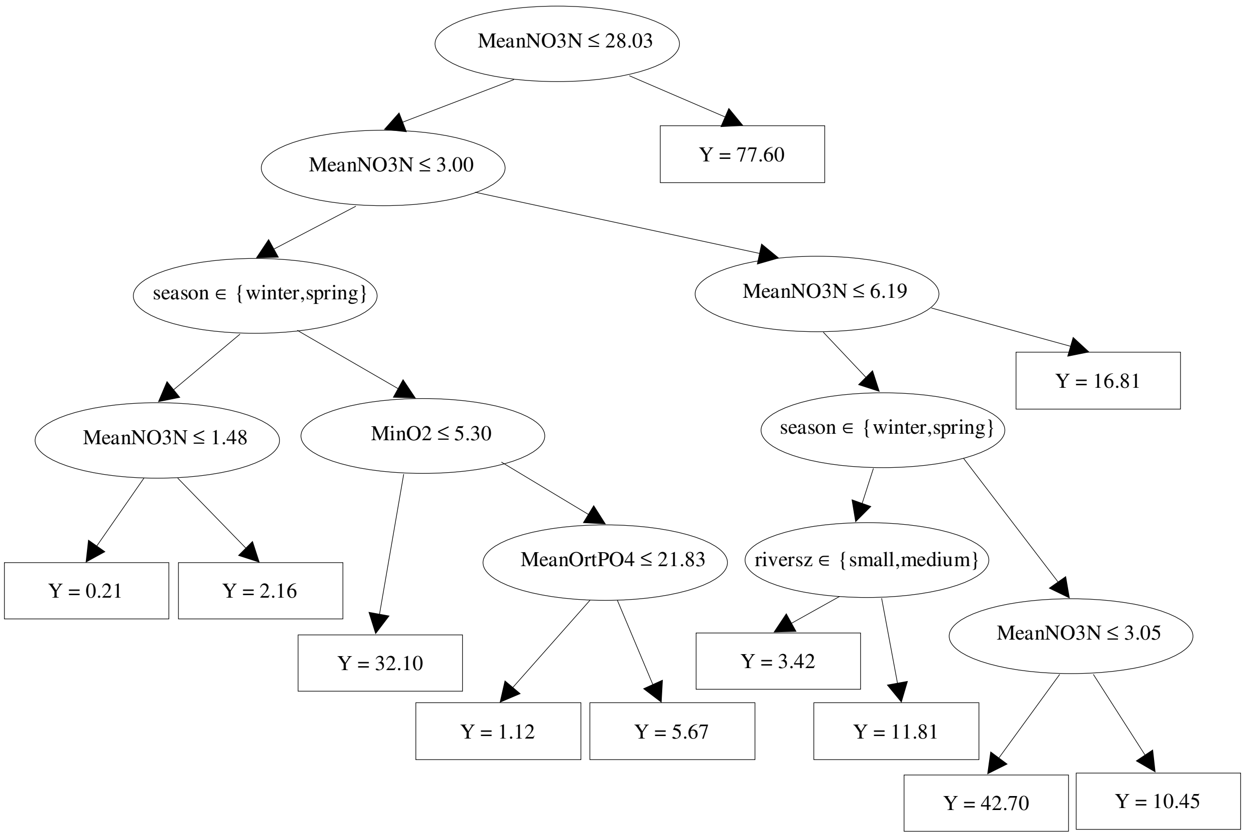


Figure 3.5 - A LS regression tree for the Alga 6 problem[35]*.*
(MSE = 162.286; MAD = 7.328)

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

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

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

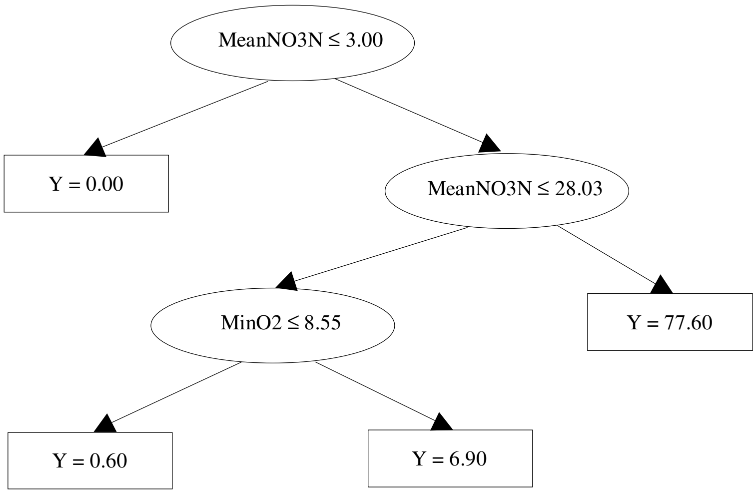


Figure 3.6 - A LAD regression tree for the Alga 6 problem.
(MSE = 179.244; MAD = 6.146)

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

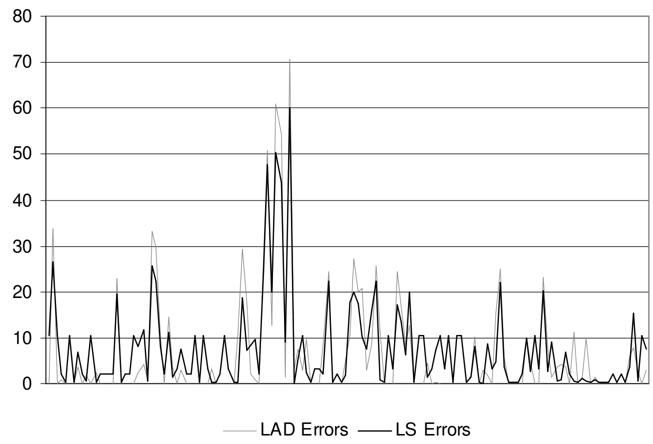


Figure 3.7 - The absolute difference of the errors committed by the LAD and LS trees.

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

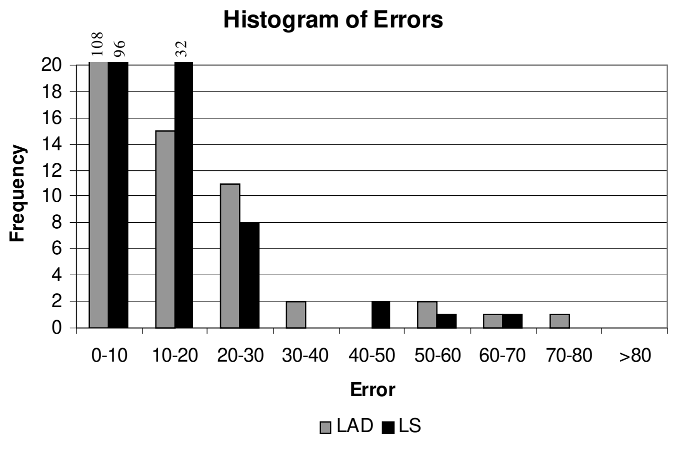


Figure 3.8 - The histogram of the errors of the LAD and LS trees.

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

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

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

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

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

3.5 Conclusions

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

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

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

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

3.5.1 Open Research Issues

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

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

APPENDIX.

PROOF OF THEOREM 3.1.

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

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

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

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

Minimising with respect to k we have,

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

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

♦

PROOF OF THEOREM 3.2.

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

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

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

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

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

So, we have

$$\begin{aligned}\phi(k) &= k\int_{-\infty}^{k} f(y)dy \;-\; k \;+\; k\int_{-\infty}^{k} f(y)dy \;-\; \int_{-\infty}^{k} y\,f(y)dy \;+\; \int_{k}^{+\infty} y\,f(y)dy \\ &= 2k\int_{-\infty}^{k} f(y)dy \;-\; k \;-\; \int_{-\infty}^{k} y\,f(y)dy \;+\; \int_{k}^{+\infty} y\,f(y)dy \\ &= 2k\,F(k) \;-\; k \;-\; \int_{-\infty}^{k} y\,f(y)dy \;+\; \int_{k}^{+\infty} y\,f(y)dy\end{aligned}$$

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

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

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

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

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

♦

TENTATIVE PROOF OF HYPOTHESIS 3.1.

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

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

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

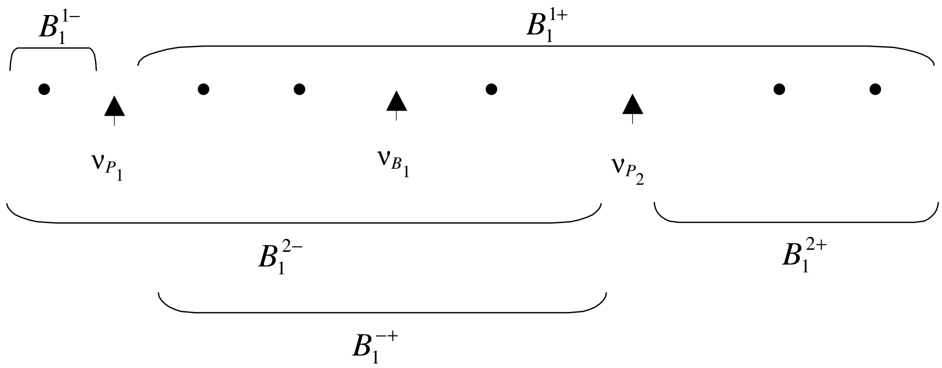
$$\psi_{N_1,N_2} = SAD_{P_1} + K_1 + SAD_{P_2} + K_2$$

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

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

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

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



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

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

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

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

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

which leads to,

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

We know that[36]

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

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

where,

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

---

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

So, finally we get

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

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

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

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

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

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

This set of cases leads to the following ordering of the values, based on their respective medians,

$\{d, a, c, e, b\}$ as ν_d (-281.5) < ν_a (-128) < ν_c (-94) < ν_e (-24.5) < ν_b (185.5)

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