Universiteit Utrecht

Master's Thesis

Isotonic Classification Trees

Author:
R.P. van de Kamp

Supervisors:
Dr. A.J. Feelders
Prof. Dr. A.P.J.M. Siebes

February 2010
INF/SCR-09-34

Abstract

In many data mining applications it is plausible that the value of the response variable increases (or decreases) as a function of the values of one or more attributes. This kind of relation is known as *monotonicity*. This thesis introduces a new algorithm called ICT for enforcing monotonicity on classification trees through post-processing. Any classification tree algorithm can be used to grow classification trees to be processed, but in this thesis we restrict ourselves to a CART-like algorithm for demonstration purposes. The algorithm makes classification trees monotone through relabelling the leaf nodes. This relabelling is done in such a way that the L_1 error on the training sample is minimised. We also propose a way to handle partially monotone problems, where some of the attributes are assumed to have a monotone relation with the response variable, but not all. The performance of ICT is experimentally compared to the standard CART algorithm. All results are evaluated on the basis of L_1 prediction error.

Acknowledgements

Writing a Master's Thesis is not an easy task, and would not have been possible without the help of other people. First of all I would like to thank dr. A.J. Feelders and N. Barile, MSc. for their invaluable guidance during this Master's project. I would also like to thank dr. A.J. Feelders, T. Mioch and M.N. Kous for reviewing this thesis and giving valuable feedback. I will be forever grateful for all of your help.

Contents

Chapter 1

Introduction

Decisions are part of every day life. Some decisions are fairly easy and can be made without a lot of effort, or sometimes even without the person being aware that he is making a decision. However, other decisions are rather complex because a lot of variables are involved or because a wrong decision can have negative consequences.

Take for example a bank which has to decide whether to accept a loan application. Since it is possible that the applicant is not able to pay back the loan, this decision should not be taken lightly. An informed decision should be made, based on known information about the applicant and the bank's knowledge of past loan applications. In practice a lot of information is asked from the applicant when applying for a loan, but for this example we assume that the age, income, and the severity of a possible criminal record of the applicant are the only criteria asked.

Usually an applicant will be accepted for a loan if more than a set percentage of past loan applicants with similar information as the new applicant were able to pay back their loans. In order to decide whether to accept a new loan application the bank could assign an employee to make a comparison between all involved information manually, but this is a lot of work and might become infeasible if a lot of loan applications are made. It would be preferable if this decision process could be automated, or at least supported.

This is where *data mining* can help. Data mining is the field of information sciences concerned with discovering and studying patterns within data. For the example of the bank, the information on past loan applications and whether the corresponding loan was paid back successfully could be learnt by an algorithm, after which this algorithm will be able to predict for new applicants whether they will be able to pay back their requested loans. An algorithm such as this, that gives a prediction on a certain outcome by comparing new information to learnt information is called a *classifier*.

Consider loan applications from two persons who both have the same age, no criminal record, but a different income: applicant A has twice as much income as applicant B. Given this information it would be unfair, and unprofitable for the bank, if applicant B is accepted for the loan but applicant A is not. This unfairness stems from the intuition that a person with a better income should be at least equally likely to get a loan as a person who has less income, because he has a better chance of paying back the loan. This intuition is formally known as *monotonicity*, which states that the order of the input must be preserved in the output. Monotonicity can be split into *increasing monotonicity* and *decreasing monotonicity*. Increasing monotonicity states that the greater an input is, the greater the output must be, as is the case with the income example above. Decreasing monotonicity states that the greater the input is, the smaller the output must be, which is the case with the criminal record in the example above; having a more severe criminal record makes it less likely to be accepted for a loan.

Suppose the bank knows that people with a certain eye colour are always having trouble paying back their loan. The bank would therefore like a classifier that takes the eye colour of applicants into account. The concept of monotonicity does not apply to eye colour, since there is no natural order defined on colors. The classifier should therefore handle this criterion as being non-monotone. In this case the classifier could use *partial monotonicity*, where only a part of the information (age, income and severity of a possible criminal record) is considered to have a monotone relation with the response variable, while other information is not considered to have this relation (eye colour).

Beside the bank example above, monotonicity is a common phenomenon in various domains and is, among others, used in medicine [26, 6], finance [15] and law [16]. Because of the frequent occurrence of monotonicity it is important that we understand it and know how we can use it. For example in classifiers, such as the classifier for loan applications, we would like to add a constraint that does not allow the classifier to make several predictions that are non-monotone with respect to each other. This constraint is called the *monotonicity constraint*.

In the usual process of data mining [12], data mining experts acquire background information on the domain at hand from one or more domain experts in order to understand the domain. Among this background information might be an indication that there are one or more monotone relations within the domain. Using the acquired background information the data mining experts construct a classifier for the problem at hand and subsequently show this classifier to the domain experts. If the domain experts indicated that one or more monotone relations exist within the domain, but this is not reflected in the classifier, the domain experts are likely to reject the classifier [22]. For this reason it is important that all monotone relations within the domain are reflected in the classifier.

Furthermore, if the domain experts can not understand the classifier they are also likely to reject it. For example neural networks are regarded as being a "black box", since it is nearly impossible to understand and interpret the internal process of this classifier. Another type of classifiers, namely classification trees, are among the classifiers that do not have this "black box" property, but are easy to understand and interpret by domain experts.

Algorithms for classification trees by default do not incorporate the monotonicity constraint. Auxiliary algorithms are needed that either change the way the algorithm grows classification trees, or post-process existing classification trees in order to fix violations of the monotonicity constraint.

In this thesis an algorithm is introduced that belongs to the group of algorithms that fixes violations of the monotonicity constraint in classification trees through post-processing.

1.1 Research Question

The introduction posed two requirements on classification problems. The first requirement is that monotone relations within the domain as indicated by domain experts must be reflected in the classifier. The other requirement is that domain experts have to understand the classifier. These requirements lead to the following research question:

Is it possible to create an algorithm that can post-process classification trees in order to make them monotone in a justifiable way?

With *a justifiable way* we mean that the way classification trees are post-processed needs a solid theoretical foundation. For example post-processing a classification tree to prune all nodes except for the root node also makes it monotone, but this is a quite simplistic and naive approach.

Predictions in classification trees are made at the leaf nodes, therefore we want to make the trees monotone by adjusting the class labels at those leaf nodes. To make the classification trees monotone we take into account that the class labels are ordered by using properties of ordered sets.

Beside the main research question we would also like to answer the following questions:

- Can relabelling a dataset in order to fix violations on the monotonicity constraint within that dataset improve the predictive performance of our proposed algorithm?
- Is it possible to efficiently make a classification tree monotone in case of partial monotonicity instead of complete monotonicity?

1.2 Research Method

In order to answer the research question we first explored how to adjust the class labels of leaf nodes in a way that results in a monotone tree. We studied literature on the subject in order to answer that question. After deciding how to adjust the class labels of the leaf nodes we implemented our new algorithm to post-process classification trees. The performance of the algorithm was experimentally evaluated through the application of the algorithm on different real-life datasets. This evaluation was split into three different tests where each test was designed to answer one of the questions posed in the previous section.

1.3 Structure of the Thesis

This thesis is structured as follows. First, the basic concepts and notation used throughout this thesis are introduced in chapter 2. Next, chapter 3 discusses previous research on monotone classification trees. We discuss existing monotone classification tree algorithms and their performance and applicability. This is followed by the introduction of the Isotonic Classification Tree (ICT) algorithm in chapter 4, where we describe how it post-processes classification trees in order to make them monotone and explain the special pruning step used in our algorithm. Thereafter the performed experiments are discussed in chapter 5, where we discuss the datasets used, the test set-up and the statistical evaluation of the tests. A discussion on decisions made throughout this thesis and the impact of these decisions on the final results is given in chapter 6. Chapter 7 concludes this thesis with a synopsis.

Chapter 2

Preliminaries

This chapter lays the groundwork for the basic concepts and notation used throughout this thesis.

2.1 Notation for Datasets

Let $\mathcal{X}$ be a *feature space* $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \ldots \mathcal{X}_p$ consisting of vectors $\mathbf{x} = (x_1,$ $x_2, \ldots, x_p)$ of values on p *attributes* or *features*. Assume that each attribute takes values x_i in a linearly ordered set $\mathcal{X}_i$. The partial ordering $\preceq$ on $\mathcal{X}$ is the ordering induced by the linear ordering defined on each $\mathcal{X}_i$, that is

$$\mathbf{x} = (x_1, x_2, \ldots, x_p) \preceq \mathbf{x}' = (x'_1, x'_2, \ldots, x'_p) \Leftrightarrow \forall i : x_i \leq x'_i$$

Furthermore let $\mathcal{Y}$ be a finite linearly ordered set of k *classes*. Without loss of generality, assume that $\mathcal{Y} = \{1, 2, \ldots, k\}$.

A *dataset* $\mathcal{D}$ is a finite subset of $\mathcal{X} \times \mathcal{Y}$, consisting of a series $(\mathbf{x}_1, y_1)$, $(\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$, of n *observations* $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$.

If for all pairs of observations in the dataset, pairs with the same values $\mathbf{x}$ have the same class label y the dataset is said to be *consistent*, i.e., a dataset is consistent if and only if

$$\forall i, j : \mathbf{x}_i = \mathbf{x}_j \Rightarrow y_i = y_j$$

Furthermore, a dataset is said to be *monotone* if for all pairs in the dataset it holds that higher values of $\mathbf{x}$ do not lead to a lower class label y

$$\forall i, j : \mathbf{x}_i \preceq \mathbf{x}_j \Rightarrow y_i \leq y_j \tag{2.1}$$

It can be easily seen that a monotone dataset is necessarily consistent.

Note that monotonicity only makes sense for datasets with *ordered class labels*, i.e., datasets for which holds that for every two consecutive class labels the higher class label indicates a better (or worse) result than the lower class label.

2.2 Notation for Classification Trees

Let T denote a classification tree with nodes t, root node $\mathbf{t}$ and a collection of leaf nodes denoted by $\tilde{T}$. Throughout this thesis we restrict ourselves to univariate, binary classification trees. For each internal node in a univariate, binary tree a split is defined on an attribute X_i for a single value s in the form $x_i \leq s$ that splits the observations that fall into that node into two child nodes. As a result, the observations that fall into a node $t \subset \mathcal{X}$ are split into two disjoint subsets $t_\ell = \{\mathbf{x} \in t : x_i \leq s\}$ and $t_r = \{\mathbf{x} \in t : x_i > s\}$.

For each node $t \in T$, the subset of the feature space $\mathcal{X}$ corresponding to that node can be written

$$t = \{\mathbf{x} \in \mathcal{X} : \mathbf{a} \prec \mathbf{x} \preceq \mathbf{b}\} = (\mathbf{a}, \mathbf{b}] \tag{2.2}$$

For some $\mathbf{a}, \mathbf{b} \in \bar{\mathcal{X}}$ with $\mathbf{a} \preceq \mathbf{b}$. $\bar{\mathcal{X}}$ denotes the extension of $\mathcal{X}$ with infinity elements $-\infty$ and ∞. In some cases these infinity elements are needed in order to specify a node as in equation (2.2). The element $\min(t) = \mathbf{a}$ is called the *minimal element* of t and the element $\max(t) = \mathbf{b}$ is called the *maximal element* of t. Together, $\mathbf{a}$ and $\mathbf{b}$ are called the *corner elements* of t.

An allocation rule c is a function that maps a point from the input space $\mathcal{X}$ to a class label from $\mathcal{Y}$

$$c : \mathcal{X} \rightarrow \mathcal{Y}$$

An allocation rule is said to be *monotone* if the following holds

$$\mathbf{x} \preceq \mathbf{x}' \Rightarrow c(\mathbf{x}) \leq c(\mathbf{x}'), \qquad \mathbf{x}, \mathbf{x}' \in \mathcal{X} \tag{2.3}$$

For a pair $t, t' \in \tilde{T}$ with $\min(t) \prec \max(t')$, t contains some elements that are smaller than some elements in t'. The monotonicity constraint on T

therefore demands that the class label assigned to t by an allocation rule c cannot be higher than the class label assigned to t'.

A pair of leaf nodes $t, t' \in \tilde{T}$ is *non-monotone* if $\min(t) \prec \max(t')$ and $c(t) > c(t')$. A tree T is non-monotone if it contains at least one non-monotone pair of leaf nodes, otherwise it is *monotone*.

2.3 Evaluation of Classifiers

In data mining a popular measure to evaluate the performance of classifiers is their error rate or 0-1 loss:

$$L_{0-1}(i,j) = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{if } i \neq j \end{cases}, \qquad i,j = 1,\ldots,k$$

where i is the true class label and j is the predicted class label.

For classification problems with ordered class labels using 0-1 loss is less obvious; we are not only interested in how many errors the classification algorithm makes, but also in the magnitude of those errors. A more obvious choice to measure the error of classification algorithms for ordered class labels is therefore a loss function that takes misclassification distance into account, i.e., a loss function that gives a higher penalty for misclassifications that are "far" from the true class label than for misclassifications that are "close" to the true class label. Several loss functions exist that adhere to this property, but because of its ease of interpretation and implementation we will confine our attention L_1 loss:

$$\mathrm{L}_1(i,j) = |i-j|, \qquad i,j = 1,\ldots,k \tag{2.4}$$

Chapter 3

Previous Research

This chapter discusses previous research that has been done on monotone classification trees. First we discuss algorithms that modify the way classification trees are grown in order to take the monotonicity constraint into account. This is followed by a discussion of algorithms that post-process classification trees in order to fix violations of the monotonicity constraint.

3.1 Growing Monotone Classification Trees

As classification tree algorithms in general are not able to take monotonicity constraints into account, several modifications have been proposed for this purpose.

The first algorithm for creating monotone classification trees was developed by Ben-David [4], who suggested extending the impurity measure for the ID3 classification tree algorithm in such a way that it takes into account the degree of non-monotonicity of the classification tree after a split has been made. The resulting impurity measure is called the *total ambiguity score* and is a trade-off between entropy E, the impurity measure normally used in ID3, and the degree of non-monotonicity resulting after the split. In order to express the degree of non-monotonicity after a split for a tree with b leaf nodes, first a $b \times b$ symmetric matrix M is constructed. Element m_{pq} is 1 if leaf node p is non-monotone with respect to leaf node q, 0 otherwise.

Let W denote the sum over all elements in M

$$W = \sum_{p=1}^{b} \sum_{q=1}^{b} m_{pq}$$

Since a leaf node can not be non-monotone with respect to itself there can be at most $b^2 - b$ non-monotone leaf nodes. In order to normalise the non-monotonicity of the tree, the *non-monotonicity index* is defined by

$$I = \frac{W}{b^2 - b}$$

which is then used in the *order ambiguity score*

$$A = \begin{cases} 0 & \text{if } I = 0 \\ -(\log_2 I)^{-1} & \text{otherwise} \end{cases}$$

The entropy E and the order ambiguity score are combined in the *total ambiguity score*

$$TA = E + \rho A$$

where ρ is a parameter to set the trade-off between the entropy and the order ambiguity score. Setting this parameter to a high value makes the impurity measure focus more on monotonicity, while setting it to a lower value makes it focus more on entropy. The drawback of this algorithm is that there is no way to determine the optimal value for ρ given a dataset, thus for every dataset this value has to be sought empirically in order to obtain a classification tree that has the right balance between monotonicity and entropy. An advantage of this algorithm is that it works on monotone as well as non-monotone datasets.

Makino et al. [19] created the first algorithm that *enforces* monotonicity on classification trees for data with two classes; a positive class and a negative class. They propose an adjustment in the splitting rule of classification tree algorithms such that it favours not only splits that balance the number of positive and negative points, but also those that have the right child-node larger than the left child-node. The algorithm enforces monotonicity by adding the corner element of a node with the appropriate class label to the data whenever necessary. The disadvantage of this algorithm is that it only works on monotone datasets with discrete attributes and two classes.

Potharst and Bioch [24] extended the work of Makino et al. in a non-trivial way to support more than two classes and handle continuous attributes. This algorithm also adds data points to the original data whenever necessary. This algorithm, like the algorithm of Makino et al., has the disadvantage that it requires monotone datasets.

Makino et al. and Potharst and Bioch have both shown that their algorithms are able to get encouraging results for data on which it is applicable. However, since real life datasets are almost never monotone the algorithms do not have a wide applicability.

Popova [23] extended the algorithm of Potharst and Bioch to be able to handle non-monotone data through changing the update rule in such a way that it fixes inconsistencies in the data. The update rule is a rule that is fired just before splitting a node so changes can be made to the data before the splitting of the node occurs. The new update rule always relabels corner elements of the tree with consistent labels that are calculated from the rest of the data. This extended algorithm, like the original algorithm by Potharst and Bioch, enforces monotone trees. It has the disadvantage that it can change the original data substantially, resulting in a classification tree that might not entirely reflect the original data. However, it overcomes the limitation of the need for monotone datasets.

3.2 Post-processing Classification Trees Toward Monotonicity

Feelders [14] argues that instead of enforcing (or promoting) monotonicity while growing a tree, it would be better to first grow an overfitted tree and post-process this tree in order to make it monotone. The rationale behind this is that while a certain split made while growing the classification tree can make the tree non-monotone, additional splits might cause it to become monotone again. Also, when enforcing the monotonicity constraint during the growing of the tree, the order in which nodes are expanded needs to be considered. He proposes instead a method that first grows a classification tree, than applies cost-complexity pruning [7] to that tree, and check for each of the trees in the cost-complexity sequence if they adhere to the monotonicity constraint. The ones that do not are dropped, resulting in a sequence of monotone trees. As Velikova [27] points out, this method has the drawback that it can only guarantee one monotone tree, namely the tree consisting of only the root node. Since this is a very simplistic model it can never serve as a real classification tree. Advantages of this algorithm are that it enforces monotonicity on classification trees, and does not need to alter the data to achieve this.

Another algorithm, by Feelders and Pardoel [13], is an algorithm that uses different *fixing methods* to create a series of monotone trees out of an overfitted non-monotone tree. A fixing method is a method that prunes in the parent node of a child node participating in a non-monotone leaf pair. They implemented two fixing methods, that use different criteria to decide in which parent node is to be pruned. The first fixing method is the *Most Non-monotone Parent*, which prunes in parent nodes whose children participate in the most non-monotone leaf node pairs. This method is not recursive, i.e., it does not take into account children of child nodes, in order to discourage pruning higher up the tree. The second fixing method is *Best Fix*, which prunes in the parent of a leaf node participating in a non-monotone leaf node pair that gives the biggest reduction of non-monotone leaf node pairs. This is done level wise, again to avoid pruning higher up the tree. Both fixing methods can be combined with a *Least Number of Observations* tie-breaker, which causes the algorithm to prune in the parent with the least number of observations in case of a tie, instead of choosing one at random. The algorithm interleaves fixing steps within the different steps of cost-complexity pruning. For every tree in the cost-complexity sequence it checks if it is monotone and is it added to the sequence of monotone trees if it is. If it is however not monotone it is first fixed and then added to the sequence of monotone trees. The disadvantages of this algorithm are firstly that it is rather ad-hoc, i.e., has no solid theoretical foundation, and secondly that a lot of pruning might be needed to make a tree monotone, resulting in small classification trees with less predictive accuracy than the original trees. On the other hand, the resulting classification trees from this algorithm are guaranteed to be monotone and no data from the dataset has to be altered to achieve this.

Chapter 4

The ICT Algorithm

This chapter introduces the main contribution of this thesis; an algorithm that enforces monotonicity through post-processing classification trees. This algorithm we call Isotonic Classification Trees, or ICT for short. It differs from previous algorithms in that it relabels the leaf nodes of classification trees. ICT also has the ability to handle partially monotone problems. To our knowledge no other algorithm currently has this ability.

Section 4.1 starts with the explanation of several modifications made to the CART algorithm in order to make it perform better on datasets with ordered class labels by using L_1 prediction error instead of misclassification error. Because the isotonic regression plays a central role in the ICT algorithm, it is discussed in section 4.2. Hereafter the ICT algorithm itself is explained, starting with how the algorithm makes classification trees monotone in section 4.3 and followed by the ICT pruning method in section 4.4. Section 4.5 shows the outline of the ICT algorithm. To illustrate the algorithm, section 4.6 gives an example application of the algorithm on a small tree. The way the ICT algorithm handles partial monotonicity is discussed in section 4.7. Finally, the computational complexity of the algorithm is shown in section 4.8.

4.1 Growing Classification Trees with Ordered Class Labels

The ICT algorithm can in principle work with any classification tree algorithm. However, since it works on ordered class labels we used some properties of ordered sets to increase the performance of the ICT algorithm by making several modifications to the CART algorithm [7]. This section describes these modifications. First, section 4.1.1 describes the statistics that are recorded during the growing of classification trees. Section 4.1.2 describes the allocation rule used for the modified CART algorithm, which ensures minimum L_1 prediction error on the training sample. Next, section 4.1.3 discusses the used resubstitution error, which for each node gives an indication of the probability that a data point is misclassified by that node. Lastly, section 4.1.4 discusses two impurity measures that are candidate to be used in the modified CART algorithm.

4.1.1 Recorded Statistics

The ICT algorithm requires several statistics about a classification tree in order to post-process it. In our modified algorithm we record these statistics while growing the classification tree (on-line), but they can also be calculated after the tree has been constructed (off-line). For each node the observations that fall into the node need to be known, as well as the corner elements of the node.

Let $n(t)$ denote the number of observations in node t, and let $n(t, j)$ denote the number of observation in node t with class label j. The relative frequency of class label j in node t can be obtained by dividing the number of observations in node t having class label j by the total number of observations in node t

$$\hat{P}_j(t) = \frac{n(t,j)}{n(t)}, \qquad j = 1, \ldots, k; t \in T$$

This can than be used to obtain the empirical cumulative distribution

$$\hat{F}_i(t) = \sum_{j \leq i} \hat{P}_j(t), \qquad j, i = 1, \ldots, k; t \in \tilde{T}$$

which is the unconstrained maximum likelihood estimate of

$$F_i(t) = P(y \leq i \mid \mathbf{x} \in t), \qquad i = 1, \ldots, k; t \in \tilde{T}$$

4.1.2 Allocation Rule

In classification tree algorithms it is customary to let each leaf node allocate the majority class label of all observations that fall into that node, in order to minimise misclassification errors. However, since we do not want to minimise the number of misclassification errors, but also want to take into account the magnitude of those errors, we will minimise L_1 prediction error. In order to minimise L_1 prediction error each leaf node is allocated the median of the class labels that fall into that node, since the median is known to minimise L_1 prediction error [10]. In case the median is not unique, we (arbitrarily) choose the smallest median. In order to make this assignment, the empirical cumulative distribution $\hat{F}$ of class labels for the nodes is used

$$c(t) = \min_i : \hat{F}_i(t) \geq 0.5 \tag{4.1}$$

4.1.3 Resubstitution Error

For each node in a classification tree the classification tree algorithm calculates the *resubstitution error*, which gives an indication of the probability that a data point is misclassified by that node. The resubstitution error consist of two factors, namely the *risk*, which indicates the severity of the misclassification, and the *probability* that a data point falls into the node. In standard CART the risk for a node t is given by

$$r(t) = 1 - \max_j \hat{P}_j(t)$$

and the probability that a data point falls into node t is given by

$$p(t) = \frac{n(t)}{n}$$

The *risk* and *probability* are combined into the resubstitution error through multiplication

$$R(t) = r(t) \cdot p(t)$$

Since the risk given above aims to minimise the number of misclassification errors, but we want to minimise L_1 prediction error, we replaced the risk as given above by the Mean Absolute Error (MAE)

$$r(t) = \sum_{i:\mathbf{x}_i \in t} \frac{|y_i - c(t)|}{n(t)}$$

thus taking the magnitude of the misclassification errors into account.

4.1.4 Impurity Measures

In classification tree algorithms, an impurity measure is used to calculate the quality of splits. At each point during the growing of the tree when a split has to be made several candidate splits are compared using the impurity measure. Both the standard CART algorithm and our modified CART algorithm select that split from the candidate splits that has the highest impurity reduction

$$\Delta m(s,t) = m(t) - p_r m(t_r) - p_\ell m(t_\ell)$$

where $m(\cdot)$ is an impurity measure, t is the node to be split using split s, and p_r and p_ℓ indicate the fraction of the data in t that go to the right child node (t_r) and the left child node (t_ℓ) respectively.

Two different impurity measures were considered for the modified CART algorithm. The first is resubstition error based on L_1 loss, as given in equation (2.4). The second impurity measure considered is the gini index combined with the L_1 loss matrix, a $k \times k$ matrix L with elements $l_{ij} = |i - j|$.

The gini index is a commonly used impurity measure that tends to favour splits that split the data into a small, pure node and a large, non-pure node [7]. Combined with the L_1 loss matrix it achieves the desired result that a candidate split gets a higher penalty for classification errors that are “far” from the true class label while classification errors that are “close” to the true class label get a lower penalty. Throughout the rest of this thesis the gini index combined with the L_1 loss matrix will be referred to as $\mathrm{Gini}_{\mathrm{L}_1}$ and the resubstition error based on L_1 loss will –in a slight abuse of notation– be referred to as the L_1 impurity measure.

For $\mathrm{Gini}_{\mathrm{L}_1}$ we used the following formula, as proposed by Breiman et al. [7]:

$$i(t) = \sum_{i \neq j} |i - j| P_i(t) P_j(t) \tag{4.2}$$

4.2 The Isotonic Regression

As the isotonic regression plays an important role in the ICT algorithm, it is described here. Section 4.3 explains how the isotonic regression is used in the ICT algorithm.

Let $Z = z_1, z_2, \ldots, z_n$ be a finite set of constants and let $\preceq$ be a partial order on Z. Any real-valued function f on Z is *isotonic* with respect to $\preceq$ if

$$z \preceq z' \Rightarrow f(z) \leq f(z')$$

Assume that each element $z_i \in Z$ is associated with a real value $g(z_i)$ which typically is an estimate of an unknown isotonic function on Z. Furthermore, assume that each element $z_i \in Z$ is associated with a positive weight $w(z_i)$ which typically indicates the precision of the estimate $g(z_i)$.

An isotonic function g^* on Z now is an *isotonic regression function* of g with respect to the weights w and partial order $\preceq$ if and only if it minimises the sum

$$\sum_{i=1}^{n} w(z_i)[f(z_i) - g(z_i)]^2 \tag{4.3}$$

within the class of isotonic functions f on Z. In other words, the isotonic regression takes an arbitrary function g and returns the isotonic function g^* for which the distance is the smallest to g when compared to all other isotonic functions on Z. Brunk [8] proved the existence of a unique g^*.

Any real-valued function f on Z is *antitonic* with respect to $\preceq$ if

$$z \preceq z' \Rightarrow f(z) \geq f(z')$$

The *antitonic regression function* g^* of g with respect to the weights w and partial order $\preceq$ is defined completely analogous to the isotonic regression function as the function that minimises equation (4.3) within the class of antitonic functions. The *antitonic regression* with respect to a partial order is equivalent to the *isotonic regression* with respect to the inverse of that order. The best known time complexity for an exact solution to the isotonic regression problem for an arbitrary partial order is $O(n^4)$ [20]. It is based on a divide-and-conquer strategy into at most n maximal flow problems.

4.3 Making Trees Monotone

In the following the workings of the ICT algorithm are explained. The algorithm consists of two steps. In the first step the classification tree is made monotone and the second step prunes the classification tree in case redundancies have been introduced in the first step. This pruning step will be explained in section 4.4.

Let T be a classification tree and let $\tilde{T}$ denote the set of leaf nodes of T. Let $(\tilde{T}, \preceq)$ be the partial order $\preceq$ on $\tilde{T}$ that dictates that t precedes t' if it contains points that are smaller than some points in t', i.e.,

$$t \preceq t' \Leftrightarrow \min(t) \prec \max(t'), \qquad t, t' \in \tilde{T} \tag{4.4}$$

We define

$$F_i^*(t), \qquad i = 1, \ldots, k; t \in \tilde{T}$$

as the antitonic regression of $\hat{F}_i(t)$ with weights $w(t) = n(t)$ and partial order $(\tilde{T}, \preceq)$, for $i = 1, \ldots, k$. By construction, these estimates satisfy the stochastic order constraint

$$t \preceq t' \Rightarrow F_i^*(t) \geq F_i^*(t'), \qquad i = 1, \ldots, k \tag{4.5}$$

Subsequently, in accordance with the allocation rule defined in equation (4.1), t is allocated to the smallest median of $F^*(t)$:

$$c^*(t) = \min_i : F_i^*(t) \geq 0.5 \tag{4.6}$$

Because F^* satisfies the stochastic order constraint as specified in equation (4.5), c^* satisfies the monotonicity constraint

$$t \preceq t' \Rightarrow c^*(t) \leq c^*(t'), \qquad t, t' \in \tilde{T}$$

It follows that c^* satisfies the monotonicity constraint specified in equation (2.3) and repeated here for convenience:

$$\mathbf{x} \preceq \mathbf{x}' \Rightarrow c^*(\mathbf{x}) \leq c^*(\mathbf{x}'), \qquad \mathbf{x}, \mathbf{x}' \in \mathcal{X}$$

Furthermore, it has been shown [11, 2] that $c^*(t)$ minimises L$_1$ loss

$$\sum_{i=1}^{n} |y_i - c(t(\mathbf{x}_i))|$$

within the class of monotone integer-valued functions $c(\cdot)$ on $\tilde{T}$, where $t(\mathbf{x}_i) = t \in \tilde{T}$: $\mathbf{x}_i \in t$. In other words, of all monotone classifiers on $\tilde{T}$, c^* is among the ones (there may be more than one) that minimise L$_1$ loss on the training sample.

4.4 ICT Pruning

After a classification tree has been post-processed by the ICT algorithm, and thus has been made monotone, it may be that two leaf nodes t_ℓ and t_r sharing a parent t have been assigned the same class label. In this case the split in t is redundant and the algorithm prunes in t. However, instead of using the customary pruning rule that would assign the class label of both t_ℓ and t_r to t, we assign the median of $\hat{F}$ to t, to ensure the smallest L_1 loss on the training sample. It is possible that after the tree has been pruned in t it becomes non-monotone, in which case the leaf nodes need to be relabelled again. Note that pruning may create two new leaf nodes t'_r, t'_ℓ with the same class label and parent t', in which case the algorithm will prune in t'.

4.5 Algorithm Outline

Algorithm 4.1 The ICT algorithm

```
1: if $T$ is monotone then
2:     return $T$
3: else
4:     for $i \in \{1, \dots, k-1\}$ do
5:         $F_i^* \leftarrow$ AntitonicRegression($\tilde{T}$, $\preceq$, $n(t) : t \in \tilde{T}$, $\hat{F}_i(t) : t \in \tilde{T}$)
6:     end for
7:     for all $t \in \tilde{T}$ do
8:         $F_k^*(t) \leftarrow 1$
9:         $c_T(t) \leftarrow \min_i F_i^*(t) \geq 0.5$
10:    end for
11:    while there are $t_\ell$,$t_r \in \tilde{T}$ with common parent $t$ and $c_T(t_\ell) = c_T(t_r)$
       do
12:        $T \leftarrow$ prune $T$ in $t$
13:        $c_T(t) \leftarrow \min_i \hat{F}_i \geq 0.5$
14:    end while
15:    $T' \leftarrow$ ICT($T$)
16:    return $T'$
17: end if
```

The algorithm takes a binary, univariate classification tree T as input and returns a monotone version of that classification tree. Due to ICT pruning the returned classification tree might be smaller than the classification tree that was given to the algorithm. The algorithm starts on line 1 by checking whether the tree is monotone. If this is the case the tree is returned unchanged. If the tree is not monotone however, the algorithm performs the antitonic regression on the empirical cumulative distribution on all but the last class label assigned to the leaf nodes in line 4-6. Subsequently, the leaf nodes are assigned new class labels according to the results of the antitonic regression on line 7-10, after which the tree is guaranteed to be monotone. Line 11-14 perform the ICT pruning, which might cause the tree to become non-monotone, which is why the algorithm recurses on line 15. Lastly, on line 16 the final monotone tree is returned.

4.6 An Example

This section demonstrates the ICT algorithm by means of an example for a small tree. For this example we assume an input space $\mathcal{X} = [0, 1]^2$ and $\mathcal{Y} = \{1, 2, 3\}$. The tree is given in figure 4.1 on the left and the corresponding input space partitioning is given in figure 4.1 on the right.

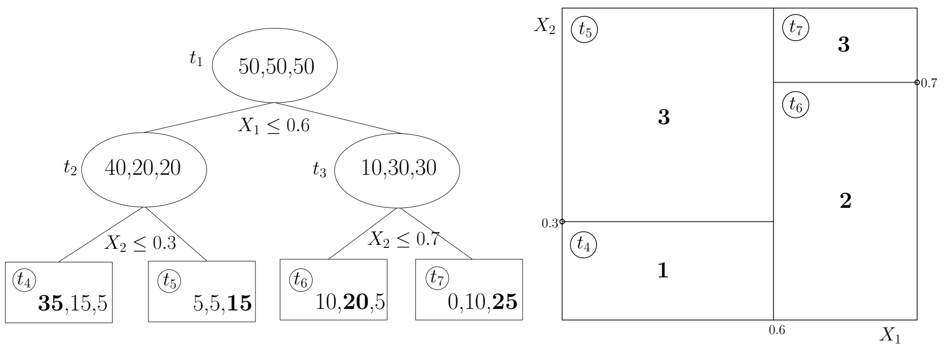

Figure 4.1: Left: Classification tree for three classes. The numbers in the leaf nodes indicate the count for class labels 1, 2 and 3 respectively. In the leaf nodes, the counts of the median class are shown in boldface. Right: The partitioning of the input space induced by the tree on the left. The class labels assigned to the different regions are shown in boldface. The circled labels in the regions correspond to the leaf nodes shown in the tree on the left.

As can be seen in figure 4.1, there is one non-monotone pair of leaf nodes in this tree, namely the pair of leaf nodes t_5 and t_6. These leaf nodes are non-monotone with respect to each other because t_5 has a higher class label than t_6 but contains points that are smaller than some points in t_6: the minimum element of t_5 is smaller than the maximum element of t_6. The ICT algorithm fixes this monotonicity violation through performing the antitonic regression on $\hat{F}_1(t)$ and $\hat{F}_2(t)$, $t \in \{t_4, t_5, t_6, t_7\}$. Note that performing the antitonic regression on $\hat{F}_3(t)$ is not needed, since 3 is the highest class label and by definition $\hat{F}_3(t) = 1$ for every t. The corner elements of the nodes are known, so we can calculate the partial order over the leaf nodes. For this particular tree we have the total order $t_4 \preceq t_5 \preceq t_6 \preceq t_7$. The class probability estimates $\hat{P}$ and the empirical cumulative distribution $\hat{F}$ for the tree given in figure 4.1 are shown in table 4.1, as well as the corner elements of all leaf nodes.

	$\hat{P}_1$	$\hat{P}_2$	$\hat{P}_3$	$\hat{F}_1$	$\hat{F}_2$	$\hat{F}_3$	min(t)	max(t)
t_4	7/11	3/11	1/11	7/11	10/11	1	0 , 0	0.6 , 0.3
t_5	1/5	1/5	3/5	1/5	2/5	1	0 , 0.3	0.6 , 1
t_6	2/7	4/7	1/7	2/7	6/7	1	0.6 , 0	1 , 0.7
t_7	0	2/7	5/7	0	2/7	1	0.6 , 0.7	1 , 1

Table 4.1: $\hat{P}$ and $\hat{F}$ for nodes t_4 through t_7 as shown in the tree in figure 4.1.

Now the antitonic regression is used to fix the violation of the monotonicity constraint through averaging $\hat{F}_1(t_5)$ with $\hat{F}_1(t_6)$ and $\hat{F}_2(t_5)$ with $\hat{F}_2(t_6)$, using weights $n(t_5)$ and $n(t_6)$.

$$F_1^*(t_5) = F_1^*(t_6) = \frac{5+10}{25+35} = \frac{1}{4} \qquad F_2^*(t_5) = F_2^*(t_6) = \frac{10+30}{25+35} = \frac{2}{3}$$

After assigning to the smallest median according to equation (4.6), both t_5 and t_6 are assigned class label 2. The L_1 error of the original non-monotone tree is 25 + 15 + 15 + 10 = 65 and after relabelling both t_5 and t_6 to class label 2 the L_1 error increases to 25 + 20 + 15 + 10 = 70. This is in fact the relabelling that minimises L_1 error. For example, assigning class label 3 to both t_5 and t_6 would lead to an L_1 error of 25 + 15 + 40 + 10 = 90 and assigning class label 1 to both t_5 and t_6 would lead to an L_1 error of 25 + 35 + 30 + 10 = 100.

4.7 Partial Monotonicity

This section shows how to make trees monotone using the ICT algorithm, taking into account *partial monotonicity*, where only a subset of attributes is considered to have a monotone relation with the response variable, while the other attributes do not. The algorithm is largely the same for partial monotonicity as it is for complete monotonicity; only the partial order used for the antitonic regression needs to be changed.

Let **X** be the subset of attributes that are considered to have a monotone relation with the response variable, subject to the definition of $\mathcal{X}$ as defined in section 2.1, and let $\mathbf{Z} = \times \mathcal{Z}_i$, $i = 1, \ldots, q$ denote the subset of attributes that are not considered to have a monotone relation with the response variable. The values in $\mathcal{Z}$ may either be ordered or non-ordered.

In order for a classification rule c to be partially monotone it has to be monotone in **X** for each value $\mathbf{z} \in \mathcal{Z}$. That is, a classification rule $c : \mathcal{X} \times \mathcal{Z} \to \mathcal{Y}$ is monotone in **X** if and only if

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \forall \mathbf{z} \in \mathcal{Z} : \mathbf{x} \preceq \mathbf{x} \Rightarrow c(\mathbf{x}, \mathbf{z}) \leq c(\mathbf{x}', \mathbf{z}) \tag{4.7}$$

For classification trees this condition only holds for two leaf nodes t and t' if t can contain points with the same values for the attributes in **Z** as t'. In other words, the condition only holds if t and t' have *overlap* in the attributes of **Z**. This leads to the partial order $(\tilde{T}, \preceq)'$ over $\tilde{T}$

$$t \preceq t' \Leftrightarrow \min(t_{\mathbf{x}}) \prec \max(t_{\mathbf{x}}) \wedge t_{\mathbf{z}} \cap t'_{\mathbf{z}} \neq \emptyset, \qquad t, t' \in \tilde{T} \tag{4.8}$$

Where $t_{\mathbf{x}}$ and $t_{\mathbf{z}}$ denote the projection of t onto the attributes **X** and the attributes **Z** respectively.

For partially monotone problems the ICT algorithm as described in section 4.3 can be used, with the substitution of the partial order $(\tilde{T}, \preceq)$ as defined in equation (4.4) by the partial order $(\tilde{T}, \preceq)'$ as defined in equation (4.8).

4.8 Computational Complexity

Since the ICT algorithm works in two stages, the time complexity is the combined time complexity of these stages. The stages with corresponding time complexities are:

1. Antitonic regression
   The antitonic regression as described in section 4.2 has a time complexity of $\mathrm{O}(|\tilde{T}|^4)$ and needs to be calculated $k - 1$ times, giving this stage a total time complexity of $\mathrm{O}(k \cdot |\tilde{T}|^4)$.

2. Relabelling leaf nodes
   To relabel the leaf nodes of a classification tree, equation (4.6) has to be applied to each of them. Since equation (4.6) has time complexity $\mathrm{O}(1)$, this stage has a total time complexity of $\mathrm{O}(|\tilde{T}|)$.

Here $|\tilde{T}|$ denotes the number of leaf nodes in a classification tree T.

The combined time complexity of the algorithm consists of $\mathrm{O}(|\tilde{T}|)$ plus $\mathrm{O}(k \cdot |\tilde{T}|^4)$, which gives the algorithm as a whole a time complexity of $\mathrm{O}(k \cdot |\tilde{T}|^4)$. Note that the time complexity is a function of $|\tilde{T}|$, which is worst case the same as the number of observations in the dataset on which the classification tree was constructed, but will generally be much smaller.

Chapter 5

Experiments

This chapter describes the experiments performed to experimentally evaluate the performance of the ICT algorithm. Section 5.1 describes the used datasets and the pre-processing performed on those datasets. The test set-up for different assumptions of monotonicity is described in section 5.2, after which the results for these tests are discussed in section 5.3. Every aspect of the ICT algorithm described in this thesis has been implemented and tested in R [25]. This program was chosen because it has clear and efficient handling of arrays and matrices, as well as a large collection of built-in functions for statistical analysis.

5.1 Datasets

The datasets used to evaluate the ICT algorithm were the *KC4*, *PC3*, *PC4* and *PC5* datasets from the NASA Metrics Data Program [21], the Employee Acception/Rejection (*ERA*), Employee Selection (*ESL*), Lecturers Evaluation (*LEV*) and Social Workers Decisions (*SWD*) from Ben-David [3], the Windsor Housing (*whousing*) dataset [1], the Den Bosch Housing (*dbhousing*) dataset [9], as well as several datasets from the UCI Machine Learning Repository [5], i.e., Australian Credit Approval (*aus*), Auto MPG (*auto*), Car Evaluation (*car*), Haberman's Survival (*haberman*), Computer Hardware (*computer*), and Pima Indians (*pima*). Most of these datasets have been used in similar studies and thus make good candidates. The datasets from the NASA Metrics Data Program however have not yet been used in a similar study. They can be used to predict the number of errors in programming code using several complexity measures as attributes. Since this relation should intuitively be monotone, these datasets are also good candidates to test the ICT algorithm. The short names between parentheses printed in italics indicate the abbreviated names for the datasets used throughout the rest of this thesis.

5.1.1 Dataset Pre-processing

For all datasets the response variable intended by the author of the dataset was used, except for the datasets from the NASA Metrics Data Program where no response was explicitly indicated. For these datasets we used the attribute `ERROR_COUNT` as the response variable.

The ICT algorithm makes the harmless assumption that for every dataset all attributes are positively correlated with the response variable, in accordance with the monotonicity constraint as defined in equation (2.1). Since for some of the used datasets this assumption does not hold, attributes that had a negative correlation with the response variable were inverted to make this correlation positive. For the Computer Hardware dataset this means the cycle time in nanoseconds was converted to the frequency in Khz. For the *auto* dataset we inverted the attributes `cylinders`, `displacement`, `horsepower` and `weight` since these attributes intuitively have a negative correlation with the response variable; for example a heavier car can drive less miles per gallon than a lighter car. For the *bhousing* dataset attributes `CRIM`, `INDUS`, `NOX`, `AGE`, `RAD`, `TAX`, `PRATIO`, `LSTAT` were inverted because they intuitively have a negative correlation with the response variable; for example a higher crime rate tends to lead to a lower popularity of a city area.

We used the following formula to invert all attributes that had a negative correlation with the response variable

$$x_i = x_{max} - x_i + x_{min}, \qquad i = 1, \dots, n$$

where x is an attribute of $\mathcal{D}$, $x_{max} = \max(x)$, $x_{min} = \min(x)$

The datasets Auto MPG, Boston Housing, Computer Hardware, Windsor Housing and Den Bosch housing had a numeric response. In order to be able to use these datasets for classification trees, we discretised the response into four intervals of roughly equal size.

For the datasets from the NASA Metrics Data Program all attributes with missing values were dropped because our CART implementation cannot cope with missing data[1]. In order to reduce the number of attributes for these datasets we used the function `stepAIC` with backward elimination in `R` to fit a linear model to each dataset; attributes that did not occur in the final model were dropped from the datasets. Since the distribution of the response variable was highly skewed (most modules have zero errors), the modules with zero errors were sampled to create a more balanced distribution. Furthermore, since high error counts are less frequent than low error counts, high error counts were merged into a single class. For example for the KC4 dataset, all class labels greater than five were set to five.

Non-predictive attributes were dropped from the datasets, i.e., the `MODULE` attribute from the NASA Metrics Data Program datasets, the `Vendor Name` and `Model Name` attributes from the Computer Hardware dataset and lastly the `y-coordinate` and `x-coordinate` attributes from the Den Bosch Housing dataset.

Finally we dropped the `ERP` attribute from the Computer Hardware dataset and the `ERROR_DENSITY` attribute from all NASA Data Metric Program datasets because these attributes are functions of the response variable of the respective datasets.

5.1.2 Relabelling Toward Monotonicity

The results presented in [28, 17] suggest that classification trees constructed using training sets for which the class labels y are relabelled in such a way that they adhere to equation (2.1), perform better on average than classification trees that are constructed on training sets with the original class labels. Therefore, we also tested the ICT algorithm on relabelled trainings sets, which we computed as the relabelling of the observations that minimises

$$\sum_{i=1}^{n} |y_i - y'_i| \tag{5.1}$$

within the class of monotone relabellings y'. Note that this relabelling is not unique and possibly not the most optimal relabelling; there may be other monotone relabellings in the class y' that also minimise equation 5.1 and possibly need to relabel less data points than y^*. Throughout the rest of this

[1]Our implementation of CART cannot cope with missing data because we did not implement this due to time constraints, not because our modifications to the algorithm make it impossible to cope with missing data.

thesis the relabelled version of y for a complete dataset will be referred to as y^* and relabelled training sets will be referred to as y^*_{train}.

5.1.3 Final Datasets

Table 5.1 lists the characteristics of the pre-processed datasets. For all datasets the cardinality, the number of attributes remaining after pre-processing, the number of distinct class labels and the L_1 distance between y and y^* are shown. For example for the Haberman survival dataset, the L_1 distance between y and y^* is 55. Since this dataset has two class labels this means that 55 data points have been relabelled in order to make it completely monotone.

Dataset	cardinality	#attributes	#labels	$\sum \lvert y - y^* \rvert$
aus	690	14	2	14
auto	392	7	4	23
bhousing	506	13	4	28
car	1728	6	4	21
computer	209	6	4	26
dbhousing	119	8	4	10
ERA	1000	4	9	1161
ESL	488	4	9	104
haberman	306	3	2	55
KC4	122	4	6	80
LEV	1000	4	5	364
PC3	320	15	5	1
PC4	356	16	6	6
PC5	1032	21	6	141
pima	768	8	2	53
SWD	1000	10	4	375
whousing	546	11	4	134

Table 5.1: Dataset characteristics and L_1 distance between y and y^*

5.2 Test Set-up

We tested the ICT algorithm experimentally on the datasets described in section 5.1. Three similar tests were set up in accordance with the questions posed in section 1.1. In the first test we tested the original pre-processed data. This test was set up to establish that post-processing a classification tree in order to make it monotone does not significantly decrease the performance when compared to the original, unprocessed classification trees.

The second test used relabelled training sets y^*_{train} instead of y. The comparison of the first two tests enabled us to establish whether relabelling training sets to make them completely monotone improves the performance of the classification trees compared to classification trees constructed using training sets with the original class labels.

Finally the third test assumed partial monotonicity, where some attributes of a datasets were assumed to have a monotone relation with the response variable, but not all of them. The aim of this test was to establish whether the performance of classification trees post-processed by the ICT algorithm under the assumption of partial monotonicity was better than the performance of classification trees constructed using the original class labels.

Since for each of these tests the set-up is mainly the same, section 5.2.1 shows the general outline of the test set-up, after which the following sections go into more detail about the set-up for the individual tests.

5.2.1 General Outline

Each dataset was first randomly divided into a training set and a test set. The training set consisted of roughly four fifth of all data points while the test set consisted of the remaining one fifth of the data points. Using the training set, a classification tree was constructed by the modified CART algorithm and subjected to cost-complexity pruning [7] to obtain a sequence of trees $T_1 \succ T_2 \succ \ldots \succ \{\mathbf{t}\}$. Each tree T_i in the sequence is obtained by pruning T_{i-1} in one or more leaf nodes, and the sequence ends with the root node $\{\mathbf{t}\}$. For each tree in the sequence the performance was tested once directly and once after the application of the ICT algorithm. The tree in the sequence with the lowest error[2] on the test set was selected as the best tree of that sequence. In case two or more trees within the sequence had the same error the smallest of those trees was selected. This procedure was repeated one hundred times, each time with a different training and test set randomly selected from the dataset, resulting in one hundred best trees. For each test the performance reported in this thesis is the average performance over these best trees.

All tests were performed twice; once using Gini_{L_1} as defined in equation (4.2) and once using the L_1 impurity measure as defined in equation (2.4).

[2] Note that we are only interested in the errors to compare the performance of the different trees in the sequence, not for error estimation.

In order to get a large sequence of trees, we grew overfitted trees by using a stopping criterion that dictates that the tree growing must stop at a node only if the node is pure, or if there is exactly one observation in the node.

5.2.2 Using Original Class Labels

For the tests using the original class labels we used the pre-processed datasets as described in 5.1.1, where we assumed that all attributes were monotone with respect to the response variable for each dataset. Since the preprocessing ensured for each of the datasets that the correlation between each attribute and the response variable was positive, this is a reasonable assumption.

5.2.3 Using Relabelled Data

For the tests using relabelled data we also used the pre-processed datasets as described in 5.1.1, but instead of using the original class labels y for the training sets, we relabelled each training set before growing a tree to be completely monotone. As with the tests using the original class labels, these tests also assume that all attributes are monotone with respect to the response variable for each dataset. These tests were evaluated using the original class labels y for reasons of fair comparison with the other tests.

5.2.4 Assuming Partial Monotonicity

The tests assuming partial monotonicity also use the pre-processed datasets as described in 5.1.1. For these tests some of the attributes were considered not to have a monotone relation with the response variable. Since at the time of writing there is no sound way to determine which attributes can be considered to have this property we used an heuristic algorithm proposed by Velikova [27]. This algorithm first calculates the fraction of monotone pairs of all comparable pairs in the dataset, called the Degree of Monotonicity (DgrMon). Once the DgrMon has been established for the complete dataset it is then calculated k more times, where each time one of the attributes is left out of the calculation. In the original algorithm, if the DgrMon for the dataset with an attribute left out is lower than or equal to the DgrMon of the complete dataset, the attribute that was left out is considered not to have a monotone relation with the response variable. However, since it is arguable whether it is acceptable to consider an attribute not to have a monotone relation with the response variable when it has a DgrMon that is equal to the DgrMon of the complete dataset, we modified the algorithm so that the difference in DgrMon values has to exceed a threshold α. As there is no statistical sound way to determine the value of α we tried several different values empirically, namely 0.001, 0.0005 and 0.00001. The value 0.001 was empirically established to be the highest α-value to yield workable results, 0.0005 is half of 0.001 and was used because our expectation was that slightly more attributes would be found not to have a monotone relation with the response variable than for 0.001. Lastly 0.00001 was used because it is a value close to zero, but sufficiently large to prevent false results due to rounding errors in floating point calculations. As an alternative to this algorithm we also created our own empirical algorithm to find attributes that can be considered not to have a monotone relation with the response variable, but this algorithm performed worse than the algorithm outlined above. More information on our algorithm can be found in the discussion in chapter 6.

5.3 Statistical Evaluation

This section discusses the results of the tests described in section 5.2, starting with the results of the tests using the original data. This is followed by the results of the tests using relabelled training sets and this section concludes with the results of the tests assuming partial monotonicity.

5.3.1 Using Original Class Labels

The results for the tests using the original data are shown in table 5.2 for $\mathrm{Gini}_{\mathrm{L}_1}$ and in table 5.3 for the L_1 impurity measure. The column `Error CART` indicates the average L_1 error made by the classification trees constructed by the modified CART algorithm without any post-processing, while the column indicated by `Error ICT` indicates the average L_1 error made by the classification trees constructed by the modified CART algorithm, and post-processed by the ICT algorithm. The columns for the number of leaf nodes in the classification trees are indicated analogously. For each column the mean error is indicated together with the standard deviation of that mean error, separated by a $\pm$ sign. The lowest error and the lowest number of leaf nodes are printed in boldface.

Dataset	Error ICT	Error CART	Leafs ICT	Leafs CART
aus	**0.1426**± 0.0070	0.1431± 0.0068	3.1700± 1.8589	**2.8800**± 1.8602
auto	**0.2982**± 0.0292	0.3045± 0.0282	**8.8400**± 2.7587	10.3900± 5.2395
bhousing	**0.3966**± 0.0370	0.4050± 0.0376	7.9700± 2.9965	**7.4000**± 4.4789
car	0.0871± 0.0181	**0.0836**± 0.0164	**27.2200**± 5.4985	31.6900± 8.4384
computer	**0.4556**± 0.0540	0.4773± 0.0554	**8.2000**± 2.4495	8.5600± 4.2790
dbhousing	**0.4931**± 0.0830	0.5165± 0.0852	5.4300± 1.5059	**5.2900**± 1.7014
ERA	**1.2764**± 0.0407	1.2926± 0.0415	10.1800± 3.9604	**8.0900**± 3.3001
ESL	**0.4348**± 0.0369	0.4590± 0.0395	**23.3200**± 4.4989	26.0000± 8.7640
haberman	**0.2585**± 0.0146	0.2605± 0.0139	2.1700± 1.8205	**1.8200**± 1.5134
KC4	**1.1871**± 0.1349	1.2358± 0.1474	**4.2100**± 2.1334	4.3000± 3.1734
LEV	**0.4764**± 0.0267	0.4903± 0.0267	19.8800± 5.2054	**19.0800**± 8.8839
PC3	**0.5357**± 0.0440	0.5363± 0.0394	2.4600± 1.2667	**2.3600**± 1.5408
PC4	**0.5735**± 0.0492	0.5835± 0.0543	4.8700± 2.9667	**4.2000**± 2.6247
PC5	0.4960± 0.0188	**0.4948**± 0.0242	5.8400± 4.0245	**5.7100**± 4.5755
pima	**0.2586**± 0.0145	0.2619± 0.0144	5.1800± 3.0990	**4.3700**± 3.1226
SWD	**0.4707**± 0.0209	0.4772± 0.0181	8.9500± 4.1252	**7.4500**± 4.5179
whousing	**0.6244**± 0.0328	0.6619± 0.0364	16.4000± 5.3522	**14.7700**± 12.2638

Table 5.2: Results for the tests using the original data and Gini_{L_1}

Dataset	Error ICT	Error CART	Leafs ICT	Leafs CART
aus	**0.1439**± 0.0062	0.1443± 0.0062	2.5400± 1.3738	**2.3600**± 1.3297
auto	**0.3094**± 0.0289	0.3213± 0.0329	**8.4200**± 3.2292	9.9500± 6.1977
bhousing	**0.4037**± 0.0442	0.4119± 0.0406	9.2900± 3.4824	**8.2700**± 5.4788
car	0.1255± 0.0452	**0.1187**± 0.0311	**29.9100**± 7.2378	40.6200± 12.8572
computer	**0.4644**± 0.0542	0.4815± 0.0528	8.1600± 2.6428	**8.1200**± 4.4637
dbhousing	**0.4938**± 0.0825	0.5153± 0.0871	5.5700± 1.6712	**5.5500**± 2.2082
ERA	**1.2796**± 0.0411	1.2987± 0.0469	10.0700± 3.3882	**8.0600**± 3.0578
ESL	**0.4516**± 0.0372	0.4730± 0.0434	**22.4100**± 5.1993	25.0200± 10.3816
haberman	**0.2569**± 0.0154	0.2591± 0.0141	2.0800± 1.4817	**1.9200**± 1.7155
KC4	**1.1983**± 0.1374	1.2400± 0.1460	4.3500± 1.9968	**4.1000**± 3.0067
LEV	**0.4682**± 0.0254	0.4811± 0.0272	**18.4000**± 5.2896	19.2300± 8.8499
PC3	0.5279± 0.0394	**0.5272**± 0.0345	2.4400± 1.3358	**2.1200**± 0.4090
PC4	**0.5788**± 0.0546	0.5843± 0.0565	4.8700± 3.2150	**4.2500**± 2.7280
PC5	**0.4949**± 0.0219	0.4973± 0.0217	6.5100± 4.8648	**5.4800**± 6.7262
pima	**0.2605**± 0.0104	0.2612± 0.0109	3.6100± 2.5816	**3.4000**± 2.7780
SWD	**0.4721**± 0.0175	0.4801± 0.0160	10.1800± 4.0110	**8.0500**± 5.5018
whousing	**0.6298**± 0.0338	0.6629± 0.0332	16.3300± 5.5342	**11.7900**± 9.4541

Table 5.3: Results for the tests using the original data and the L_1 impurity measure

When comparing classification trees constructed using Gini_{L_1} to classification trees constructed using the L_1 impurity measure it can be seen that the classification trees constructed using Gini_{L_1} almost always perform better than the classification trees constructed using the L_1 impurity measure for both the classification trees post-processed by the ICT algorithm as well as their unprocessed counterparts. For the tree size criterion the classification trees constructed using the L_1 impurity measure are on average slightly smaller than the classification trees constructed using Gini_{L_1}.

Overall, the classification trees post-processed by the ICT algorithm almost always perform better for both Gini_{L_1} and the L_1 impurity measure than their unprocessed counterparts, but are on average slightly larger. These results are however not statistically significant.

Readers interested in a side-by-side comparison of all results for these tests are referred to appendix A.

5.3.2 Using Relabelled Data

Tables 5.4 and 5.5 show the results of the tests using relabelled training sets for both Gini_{L_1} and the L_1 impurity measure. The columns are indicated the same way as for the tests using the original data and as before, the lowest error and the lowest number of leaf nodes are printed in boldface and for each column the mean error is indicated together with the standard deviation of that mean error, separated by a $\pm$ sign.

Dataset	Error ICT	Error CART	Leafs ICT	Leafs CART
aus	**0.1424**± 0.0074	0.1430± 0.0072	3.2400± 1.9286	**2.9600**± 1.9793
auto	**0.3008**± 0.0312	0.3040± 0.0339	**9.0000**± 2.7524	10.2100± 4.8872
bhousing	**0.3931**± 0.0366	0.3991± 0.0358	**8.1900**± 3.0375	8.9600± 5.3445
car	0.0874± 0.0178	**0.0838**± 0.0163	**27.9400**± 5.2220	31.1600± 8.0374
computer	**0.4479**± 0.0544	0.4556± 0.0549	**8.6100**± 2.1315	9.0100± 3.3075
dbhousing	**0.4919**± 0.0770	0.5053± 0.0788	5.2300± 1.4624	**5.1700**± 1.5509
ERA	**1.3009**± 0.0487	1.3182± 0.0421	11.7600± 4.1830	**11.6800**± 6.1626
ESL	**0.4346**± 0.0358	0.4375± 0.0368	**23.5500**± 4.2791	26.1900± 5.9198
haberman	0.2601± 0.0195	**0.2575**± 0.0150	2.7600± 2.0652	**2.3700**± 2.0776
KC4	**1.1534**± 0.1394	1.1569± 0.1306	4.7000± 1.6787	**4.5700**± 1.9553
LEV	0.4735± 0.0245	**0.4722**± 0.0245	**21.3100**± 6.3987	26.3100± 8.8565
PC3	**0.5351**± 0.0443	0.5359± 0.0399	2.4800± 1.3669	**2.3600**± 1.5408
PC4	**0.5866**± 0.0494	0.5951± 0.0540	5.4600± 3.2642	**5.0500**± 4.0009
PC5	0.4993± 0.0214	**0.4952**± 0.0241	6.2000± 4.5438	**6.0700**± 4.5644
pima	0.2589± 0.0158	**0.2583**± 0.0128	**3.5300**± 1.7318	3.9800± 2.7265
SWD	**0.4814**± 0.0219	0.4856± 0.0215	**9.5300**± 5.0920	11.0700± 8.5010
whousing	**0.6245**± 0.0413	0.6396± 0.0362	**16.9200**± 4.4624	22.5800± 11.9850

Table 5.4: Results for the tests using relabelled data and Gini_{L_1}

When comparing classification trees constructed using Gini_{L_1} to classification trees using the L_1 impurity measure we observe that, as with the tests using the original data, classification trees constructed using Gini_{L_1} on average have a better performance than classification trees constructed using the L_1 impurity measure. In contrast with the results of the tests using

Dataset	Error ICT	Error CART	Leafs ICT	Leafs CART
aus	**0.1437**± 0.0065	0.1438± 0.0065	**2.4500**± 1.0286	2.5100± 1.4249
auto	**0.3082**± 0.0292	0.3141± 0.0311	**8.7500**± 3.1055	10.9200± 6.1933
bhousing	**0.4104**± 0.0434	0.4126± 0.0386	9.3500± 3.3101	**8.8800**± 5.5601
car	0.1252± 0.0419	**0.1179**± 0.0291	**30.3600**± 7.2843	40.3600± 12.5839
computer	**0.4544**± 0.0528	0.4589± 0.0529	**8.2200**± 2.2811	9.6700± 3.9108
dbhousing	**0.4939**± 0.0764	0.5059± 0.0801	**5.6900**± 1.5935	5.8700± 2.1726
ERA	**1.3107**± 0.0486	1.3231± 0.0437	**12.3600**± 4.3520	12.7200± 6.9647
ESL	**0.4449**± 0.0393	0.4506± 0.0408	**23.5800**± 5.0516	28.3800± 7.4002
haberman	**0.2559**± 0.0178	0.2567± 0.0157	2.9600± 2.8530	**2.7900**± 2.9346
KC4	**1.1515**± 0.1357	1.1572± 0.1331	4.7700± 1.7971	**4.6400**± 2.1155
LEV	0.4717± 0.0264	**0.4705**± 0.0237	**21.6600**± 6.4514	28.5900± 9.9961
PC3	0.5279± 0.0394	**0.5272**± 0.0345	2.4400± 1.3358	**2.1200**± 0.4090
PC4	**0.5899**± 0.0643	0.5936± 0.0652	4.9700± 3.1381	**4.3600**± 3.2490
PC5	0.4966± 0.0220	**0.4940**± 0.0235	**6.3700**± 4.9597	6.4900± 6.4346
pima	**0.2592**± 0.0147	0.2602± 0.0145	3.7300± 2.8985	**3.3900**± 3.5073
SWD	**0.4882**± 0.0266	0.4926± 0.0209	**9.6700**± 4.9422	11.9600± 11.1264
whousing	**0.6226**± 0.0325	0.6411± 0.0328	**16.4200**± 5.5817	19.4500± 12.5210

Table 5.5: Results for the tests using relabelled data and the L_1 impurity measure

the original data, classification trees that are constructed using Gini_{L_1} are on average slightly smaller than classification trees constructed using the L_1 impurity measure for these test.

Overall, the classification trees post-processed by the ICT algorithm perform better on average than their unprocessed counterparts, for both Gini_{L_1} and the L_1 impurity measure. In contrast to the results of the tests using the original data, for these results the classification trees that have been post-processed by the ICT algorithm are most of the time slightly smaller than their unprocessed counterparts for both Gini_{L_1} and the L_1 impurity measure. However, both the performance differences and the tree size differences are not statistically significant.

Readers interested in a side-by-side comparison of all results for these tests are referred to appendix B.

Compared to the previous test where complete monotonicity was also assumed, but the original class labels were used to grow classification trees, on average the performance of the classification for the tests using relabelled data was slightly better, but not statistically significant.

Readers interested in a side-by-side comparison of the results for these tests versus the results of the tests using original class labels are referred to appendix C.

5.3.3 Assuming Partial Monotonicity

When assuming partial monotonicity it is necessary that the classification trees post-processed by the ICT algorithm under this assumption actually use the attributes that are found to be non-monotone. If these attributes were never used, the results would be the same as for the tests assuming complete monotonicity and using the original class labels. For this reason tables 5.6, 5.7 and 5.8 show the attributes that were found by the algorithm described in 5.2.4 to have a non-monotone relation with the response variable for α-values 0.001, 0.0005 and 0.00001 respectively, along with the average number of internal nodes that split on those attributes, averaged over the best trees of 100 cost-complexity sequences as described in section 5.2. Datasets for which no attributes were found to have a non-monotone relation with the response variable are omitted from the tables.

As α is a threshold value it is possible that the same results are found for different α-values. When this is the case the results that are unchanged when compared to a higher α-value are printed in grey. For each column the average number of internal nodes that split on an attribute is indicated together with the standard deviation thereof, separated by a $\pm$ sign.

Dataset	Attribute	Usage Gini$_{L_1}$	Usage L_1
aus	1	0.0100±0.1000	0.0100±0.1000
	3	0.1800±0.5199	0.0700±0.2932
bhousing	8	0.4300±0.7000	0.4800±0.7032
	9	0.1200±0.3562	0.1400±0.4269
dbhousing	7	0.1400±0.3487	0.1100±0.3145
ERA	4	1.3400±1.4718	0.9200±1.1072
haberman	2	0.2700±0.6172	0.2800±0.5700
PC4	1	0.5400±1.0093	0.6200±1.1615
PC5	15	0.0500±0.2190	0.1000±0.3892
SWD	8	0.6600±0.8554	0.5300±0.7714

Table 5.6: Non-monotone attributes and the average number of nodes splitting on those attributes for $\alpha = 0.001$

The first observation is that, as one would expect due to the nature of the α parameter, more attributes are considered to have a non-monotone relation with the response variable for lower values of α. Another observation is that the attributes that are considered not to have a monotone relation with the response variable are, with one exception, used in the classification trees to split on.

Dataset	Attribute	Usage Gini_{L_1}	Usage L_1
aus	1	0.0100±0.1000	0.0100±0.1000
	3	0.1800±0.5199	0.0700±0.2932
bhousing	3	0.2700±0.5291	0.3700±0.5801
	8	0.4300±0.7000	0.4900±0.7316
	9	0.1200±0.3562	0.1300±0.4181
dbhousing	6	0.0400±0.1969	0.0500±0.2190
	7	0.1400±0.3487	0.1100±0.3145
ERA	4	1.3400±1.4718	0.9200±1.1072
haberman	2	0.2700±0.6172	0.2800±0.5700
PC4	1	0.5400±1.0093	0.6200±1.1615
PC5	15	0.0500±0.2190	0.1000±0.3892
SWD	8	0.6600±0.8554	0.5300±0.7714

Table 5.7: Non-monotone attributes and the average number of nodes splitting on those attributes for $\alpha = 0.0005$

Tables 5.9, 5.10 and 5.11 show the performance of the classification trees for the tests assuming partial monotonicity. Like the tables showing the non-monotone attributes and the number of internal nodes splitting on those attributes, unchanged results are printed in grey. The column `Error` Gini_{L_1} indicates the average L_1 error made on the test set by classification trees constructed by the modified CART algorithm using Gini_{L_1} and post-processed by the ICT algorithm. The `Error` L_1 column indicates the average L_1 error of the classification trees constructed by the modified CART algorithm using the L_1 impurity measure and post-processed by the ICT algorithm. The columns for the average number of leaf nodes in the trees are indicated analogously.

As before, for each column the mean error is indicated together with the standard deviation of that mean error, separated by a $\pm$ sign. The lowest error and the lowest number of leaf nodes are printed in boldface. Errors lower than the error of classification trees constructed using the same impurity measure, but using the original class labels and assuming complete monotonicity are printed in italics. Note that the two are not mutually exclusive; some results are printed in boldface italics.

First we observe that for $\alpha = 0.001$, classification trees constructed using Gini_{L_1} are almost always better than classification trees constructed using the L_1 impurity measure both in terms of performance as well as for the tree size criterion. When comparing these results to the results of the test using the original class labels (and assuming complete monotonicity) there is no clear winner in terms of performance, since Gini_{L_1} is better three out of eight

Dataset	Attribute	Usage Gini_{L_1}	Usage L_1
aus	1	0.0100±0.1000	0.0100±0.1000
	3	0.1800±0.5199	0.0700±0.2932
	11	0.0000±0.0000	0.0000±0.0000
bhousing	3	0.2700±0.5291	0.3700±0.5801
	8	0.4300±0.7000	0.4900±0.7316
	9	0.1200±0.3562	0.1300±0.4181
dbhousing	6	0.0400±0.1969	0.0500±0.2190
	7	0.1400±0.3487	0.1100±0.3145
computer	1	2.2500±1.8876	3.4900±3.0765
ERA	4	1.3400±1.4718	0.9200±1.1072
haberman	2	0.2700±0.6172	0.2800±0.5700
PC4	1	0.4400±0.8799	0.5400±1.0485
	11	0.0800±0.3387	0.0500±0.2190
	12	0.0100±0.1000	0.0300±0.1714
	14	0.0500±0.2611	0.0100±0.1000
PC5	14	0.2100±0.4984	0.1500±0.4352
	15	0.0600±0.2387	0.1200±0.4330
pima	3	0.2600±0.5245	0.2000±0.5860
	4	0.1600±0.5069	0.0600±0.2778
	5	0.1600±0.4197	0.1200±0.3835
SWD	6	0.2900±0.6243	0.3300±0.6039
	7	0.5700±0.7946	0.5300±0.6269
	8	0.6200±0.7886	0.5600±0.8327

Table 5.8: Non-monotone attributes and the average number of nodes splitting on those attributes for $\alpha = 0.00001$

times and the L_1 impurity measure is better four of eight times. For the tree size criterion, the classification trees from the tests using the original class labels are in most cases slightly smaller than the classification trees from the tests assuming partial monotonicity.

The results for $\alpha = 0.0005$ are mostly the same as the results for $\alpha = 0.001$ since the results for six out of eight datasets have not changed. For the two datasets where the results have changed, the performance of the classification trees constructed using Gini_{L_1} has improved slightly when compared to $\alpha = 0.001$, while the performance of the classification trees constructed using the L_1 impurity measure has decreased slightly. The classification trees for this α-value are slightly bigger than the classification trees for $\alpha = 0.001$ three out of four times. The fourth time the classification trees on average had an equal number of leaf nodes.

Lastly for $\alpha = 0.00001$, the performance for both Gini_{L_1} and the L_1 impurity measure was almost always equal or worse when compared to the results of $\alpha = 0.0005$, with one exception; the performance of the classification trees constructed using the L_1 impurity measure for the *SWD* data-

Dataset	Error Gini_{L_1}	Error L_1	Leafs Gini_{L_1}	Leafs L_1
aus	***0.1425***± 0.0072	*0.1439*± 0.0063	3.2700± 2.0639	***2.5400***± 1.4868
bhousing	**0.3972**± 0.0351	0.4061± 0.0427	**8.0700**± 3.1885	9.6500± 4.0136
dbhousing	***0.4921***± 0.0823	*0.4933*± 0.0805	**5.4600**± 1.5270	5.6200± 1.6804
ERA	***1.2756***± 0.0397	1.2814± 0.0416	10.8100± 4.8901	**10.3900**± 3.7923
haberman	0.2586± 0.0146	***0.2569***± 0.0152	**2.2200**± 2.0577	2.3400± 2.0510
PC4	**0.5754**± 0.0496	0.5803± 0.0560	*4.8700*± 2.9803	***4.6800***± 3.1136
PC5	0.4966± 0.0183	**0.4955**± 0.0215	***5.6800***± 4.0623	*6.4800*± 4.8772
SWD	**0.4708**± 0.0202	*0.4715*± 0.0176	**9.3300**± 4.3042	10.3100± 3.9713

Table 5.9: Results of the tests assuming partial monotonicity for $\alpha = 0.001$

Dataset	Error Gini_{L_1}	Error L_1	Leafs Gini_{L_1}	Leafs L_1
aus	***0.1425***± 0.0072	*0.1439*± 0.0063	3.2700± 2.0639	***2.5400***± 1.4868
bhousing	**0.3970**± 0.0345	0.4063± 0.0424	**8.2900**± 3.3674	9.6500± 4.1277
dbhousing	***0.4919***± 0.0816	*0.4934*± 0.0806	**5.5300**± 1.5338	5.7300± 1.7283
ERA	***1.2756***± 0.0397	1.2814± 0.0416	10.8100± 4.8901	**10.3900**± 3.7923
haberman	0.2586± 0.0146	***0.2569***± 0.0152	**2.2200**± 2.0577	2.3400± 2.0510
PC4	**0.5754**± 0.0496	0.5803± 0.0560	*4.8700*± 2.9803	***4.6800***± 3.1136
PC5	0.4966± 0.0183	**0.4955**± 0.0215	***5.6800***± 4.0623	*6.4800*± 4.8772
SWD	**0.4708**± 0.0202	*0.4715*± 0.0176	**9.3300**± 4.3042	10.3100± 3.9713

Table 5.10: Results of the tests assuming partial monotonicity for $\alpha = 0.0005$

set improved. The number of leaf nodes for this same comparison mostly increased, but there is no general trend. For this α-value the algorithm described in 5.2.4 found one attribute to have a non-monotone relation with the response variable for the *computer* dataset and three attributes for the *pima* dataset whereas no attributes were found to be have a non-monotone relation with the response variable for these datasets using higher α-values. The results for these two datasets show that the performance of classification trees constructed using Gini_{L_1} is better than the performance of the classification trees constructed using the L_1 impurity measure for these tests, but when comparing the results of the classification trees constructed using the L_1 impurity measure with the tests using the original class labels we see that the performance of the classification trees constructed using the L_1 impurity measure has increased, while for classification trees constructed using Gini_{L_1}this is not the case.

As with the previous two test, classification trees constructed using Gini_{L_1} for the tests assuming partial monotonicity perform better than classification trees constructed using the L_1 impurity measure. There is however no clear winner on the tree size criterion. When we compare these results to the

Dataset	Error Gini_{L_1}	Error L_1	Leafs Gini_{L_1}	Leafs L_1
aus	***0.1426***± 0.0071	*0.1439*± 0.0063	3.2100± 2.0116	***2.5400***± 1.4868
bhousing	**0.3970**± 0.0345	0.4063± 0.0424	**8.2900**± 3.3674	9.6500± 4.1277
computer	**0.4582**± 0.0532	*0.4638*± 0.0552	**9.0100**± 3.2983	10.2400± 4.5971
dbhousing	***0.4919***± 0.0816	*0.4934*± 0.0806	**5.5300**± 1.5338	5.7300± 1.7283
ERA	***1.2756***± 0.0397	1.2814± 0.0416	10.8100± 4.8901	**10.3900**± 3.7923
haberman	0.2586± 0.0146	***0.2569***± 0.0152	**2.2200**± 2.0577	2.3400± 2.0510
PC4	**0.5764**± 0.0487	0.5809± 0.0563	*4.6400*± 2.9112	***4.4600***± 2.9282
PC5	0.4968± 0.0191	**0.4956**± 0.0215	***5.8000***± 4.1779	*6.4000*± 5.0871
pima	**0.2589**± 0.0153	*0.2603*± 0.0108	5.3500± 3.3466	**4.1000**± 3.4480
SWD	0.4715± 0.0194	***0.4712***± 0.0182	**9.1500**± 4.5823	10.6500± 4.2979

Table 5.11: Results of the tests assuming partial monotonicity for $\alpha = 0.00001$

results of the tests using the original class labels we see that the performance of the classification trees constructed using the L_1 impurity measure has increased more than the performance of the classification trees constructed using Gini_{L_1}. However, the average performance of the classification trees post-processed by the ICT algorithm under the assumption of partial monotonicity is not better than the classification trees constructed using the original class labels and post-processed by the ICT algorithm under the assumption of complete monotonicity.

Chapter 6

Discussion

This chapter discusses some of the aspects of the ICT algorithm as outlined in this thesis. First the impurity measures used are discussed, followed by a discussion of the algorithm by Velikova that was used to find non-monotone relations within datasets. This is followed by a discussion on the use of loss matrices in problems where the distance between consecutive class labels is not constant. Finally the speed of the algorithm is discussed, which is still quite fast despite the $\mathrm{O}(k \cdot |\tilde{T}|^4)$ time complexity.

6.1 Impurity Measures

As can be seen from the results in chapter 5, classification trees constructed using $\mathrm{Gini}_{\mathrm{L}_1}$ perform slightly better than classification trees constructed using the L_1 impurity measure. There is no clear winner on the tree size criterion. Beside yielding worse results, growing classification trees using the L_1 impurity measure takes more time than growing trees using $\mathrm{Gini}_{\mathrm{L}_1}$. Tests using the L_1 impurity measure took around twice as much time as the tests using $\mathrm{Gini}_{\mathrm{L}_1}$. Most likely the cause of this is that the L_1 impurity measure does not favour splits that result in leaf nodes, while the gini index used in $\mathrm{Gini}_{\mathrm{L}_1}$ favours splits that split into a small pure node and a large non-pure node. As a result the trees grown using the L_1 impurity measure are initially bigger than the trees constructed using $\mathrm{Gini}_{\mathrm{L}_1}$, which makes it likely that for classification trees created using the L_1 impurity measures more rounds of the ICT algorithm need to be performed than for the trees constructed using

Gini_{L_1}. In conclusion we discourage the use of the L_1 impurity measure for use in classification trees in future works.

6.2 Finding Non-monotone Relations

The algorithm of Velikova [27] we used to detect which attributes could be considered not to have a monotone relation with the response variable is heuristic and highly dependent on the dataset at hand. As described in section 5.2.4, the way the algorithm decides which attributes can be considered not to have a monotone relation with the response variable is arguable, which is why we introduced the threshold variable α to indicate how much the DgrMon of the dataset with one attribute left out has to differ from the DgrMon of the complete dataset in order to consider it not to have a monotone relation with the response variable. This parameter however brought with it the problem that there is no statistical sound way to determine which α-value should be used, which is why we reverted to empirical values.

As an alternative to this algorithm we also tried an algorithm we designed ourselves. This algorithm first calculates the fraction of monotone observation pairs within all observation pairs in a dataset. After this, the fraction is calculated k more times where each time one of the attributes is discretised into d distinct values. A pair of observations was defined to be monotone if they were monotone in all non-discretised attributes and the value of the discritesed attribute was equal for both observations. Attributes for which the fraction of monotone observation pairs is bigger than the fraction of monotone observations pairs over the complete dataset are considered not to have monotone relation with the response variable. The intuition behind this algorithm lies in the definition of partial monotonicity as given in equation (4.7). We discretised the values for $\mathbf{z}$ to encourage more observation pairs to have the same value for $\mathbf{z}$ than there would be using the original values for $\mathbf{z}$. When testing this algorithm it appeared to yield worse results than Velikova's algorithm and has the disadvantage that there is no statistical sound way to the determine the value of d given a dataset.

We can safely conclude that further research is needed that either improves Velikova's algorithm or to create a new algorithm that can correctly identify the attributes that can be considered not to have a monotone relation with the response variable. When the non-monotone relations can be detected correctly, which also includes information from a domain expert indicating partial monotonicity, the performance of the ICT algorithm for partially monotone problems is likely to improve.

6.3 The Use of Loss Matrices

The use of the L_1 loss matrix in Gini_{L_1} and the L_1 impurity measure implies that the distance between consecutive class labels is constant, which is not necessarily the case in real life datasets. For example for a problem with class labels 1, 2 and 3 that represent bad, good and excellent respectively, the distance between class labels 1 (excellent) and 2 (good) is intuitively smaller than the distance between class labels 2 (good) and 3 (bad). In such cases the loss matrix can be adjusted to reflect the different distances between consecutive class labels. However, it needs to be taken into account that some loss matrices produce non-monotone results, thus making them useless for use with the ICT algorithm. Kotlowski [17] proved that for a probability distribution that adheres to the stochastic order constraint, a classifier that assigns to the label with minimum loss is monotone if and only if the loss function is convex., i.e. if it adheres to the following constraints:

$$\begin{aligned} l_{i,j+1} - l_{ij} &\geq l_{i+1,j+1} - l_{i+1,j} \quad \text{if } j > i \\ l_{i,j-1} - l_{ij} &\geq l_{i-1,j-1} - l_{i-1,j} \quad \text{if } j < i \end{aligned}$$

Since we have shown that the ICT algorithm adheres to the stochastic order constraint (see section 4.3), we can replace the L_1 matrix in the impurity measures with any other convex loss matrix to create monotone classification trees for problems where the distance between consecutive class labels is not constant. We can not guarantee however that the solution obtained minimises loss on the training sample subject to the monotonicity constraint.

6.4 Speed of the Algorithm

In section 4.8 we showed that the time complexity for the ICT algorithm is $O(k \cdot |\tilde{T}|^4)$. Although normally this would indicate that the algorithm is quite slow, since $|\tilde{T}|$ indicates the number of leaf nodes in a classification tree, which is typically quite small, it does not take a long time to grow a classification tree and post-process it using the ICT algorithm to make it monotone. For example, growing a monotone classification tree and post-process it using the ICT algorithm for the *car* dataset, which has 6 attributes, 4 different class labels and a cardinality of 1728 only took 28 seconds on our test machine[1]. However, for very large datasets the time complexity of $O(|\tilde{T}|^4)$ for the anititonic regression might become problematic. In that case the Generalized Pool Adjacent Violators (GPAV) algorithm can be used to calculate an approximation of the isotonic regression. This algorithm has a time complexity of $O(|\tilde{T}|^2)$ and only makes small errors when compared to an algorithm that calculates an exact solution [18]. Since the isotonic regression is the bottleneck of the ICT algorithm in terms of computational complexity, using the GPAV algorithm reduces the time complexity of the ICT from $O(k \cdot |\tilde{T}|^4)$ to $O(k \cdot |\tilde{T}|^2)$. It would be interesting, though outside the scope of this thesis, to research the impact on the performance of the ICT algorithm when the GPAV algorithm is used instead of an algorithm that calculates an exact solution to the isotonic regression.

[1] Our test machine had a AMD Athlon™ 64 X2 4200+ processor running at the factory speed of 2.22 Ghz with 2GB PC3200 internal memory, running R version 2.9.1 under Microsoft® Windows® 7 build 7100.

Chapter 7

Conclusion

This thesis proposes an algorithm that makes classification trees monotone through relabelling of the leaf nodes. This relabelling is not performed directly on the class labels, but on the class probability estimates of the leaf nodes. Furthermore, this algorithm is the first algorithm that proposes a way to handle partially monotone problems. In this chapter we revisit the main research question and the two sub questions, and give answers to these questions.

7.1 Synopsis

Below are the main research question and the two sub questions from section 1.1 together with the answers to these questions as they have been established in this thesis.

1. *Is it possible to create an algorithm that can post-process classification trees in order to make them monotone in a justifiable way?*
   In this thesis we have shown that it is possible to post-process classification trees, through relabelling the leaf nodes, in order to make them monotone. We have also shown that our algorithm has a solid theoretical foundation, making it justifiable.

2. *Can relabelling a dataset in order to fix violations of the monotonicity constraint within that dataset improve the predictive performance of our proposed algorithm?*
   The results show that through relabelling the training set before growing a tree increases the performance of trees post-processes by the ICT algorithm slightly most of the time, though not significantly. It appears that the performance can indeed be improved for classification trees constructed on several datasets, while for other datasets the performance is equal or worse, when compared to classification trees constructed on the original class labels.

3. *Is it possible to make a classification tree monotone assuming partial monotonicity instead of complete monotonicity?*
   As we have shown, this is possible through the use of an alternative partial ordering that takes into account that some of the attributes are not considered to have a monotone relation with the response variable.

Overall, we have not been able to show a significant increase in performance for classification trees that have been post-processed by the ICT algorithm in comparison with classification trees constructed by a standard algorithm without any post-processing. However, the classification trees that are post-processed by the ICT algorithm are guaranteed to be monotone, whereas for classification trees constructed by a standard algorithm without any post-processing this is not guaranteed, not even when a classification tree is constructed on a dataset that has no violations of the monotonicity constraint. Lastly, to our knowledge no classification tree algorithm other than ICT currently has the ability to deal with partial monotonicity, making the ICT algorithm the only candidate to produce classification trees where partial monotonicity needs to be taken into account.

Bibliography

[1] P.M. Anglin and R. Gençay. Semiparametric estimation of a hedonic price function. *Journal of Applied Econometrics*, 11(6):633–648, 1996.

[2] N. Barile and A. Feelders. Nonparametric monotone classification with MOCA. In F. Giannotti, editor, *Proceedings of the Eighth IEEE International Conference on Data Mining (ICDM 2008)*, pages 731–736. IEEE Computer Society, 2008.

[3] A. Ben-David, L. Sterling, and Y. Pao. Learning and classification of monotonic ordinal concepts. *Computational Intelligence*, 5:45–49, 1989.

[4] Arie Ben-David. Monotonicity maintenance in information-theoretic machine learning algorithms. *Machine Learning*, 19:29–43, 1995.

[5] C.L. Blake and C.J. Merz. UCI repository of machine learning databases [http://www.ics.uci.edu/∼mlearn/mlrepository.html], 1998.

[6] D.A. Bloch and B.W. Silverman. Monotone discriminant functions and their applications in rheumatology. *Journal of the American Statistical Association*, 92(437):144–153, 1997.

[7] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification And Regression Trees*. Chapman and Hall, 1984.

[8] H.D. Brunk. Conditional expectation given a σ-lattice and applications. *Annals of Mathematical Statistics*, 36:1339–1350, 1965.

[9] H.A.M. Daniels and B. Kamp. Application of MLP networks to bond rating and house pricing. *Neural Computing & Applications*, 8(3):226–234, 1999.

[10] M.H. DeGroot and M.J. Schervish. *Probability and Statistics (third edition).* Addison-Wesley, 2002.

[11] R. Dykstra, J. Hewett, and T. Robertson. Nonparametric, isotonic discriminant procedures. *Biometrika*, 86(2):429–438, 1999.

[12] A. Feelders, H. Daniels, and M. Holsheimer. Methodological and practical aspects of data mining. *Information and Management*, 37, 200.

[13] A. Feelders and M. Pardoel. Pruning for monotone classification trees. In M.R. Berthold, H-J. Lenz, E. Bradley, R. Kruse, and C. Borgelt, editors, *Advances in Intelligent Data Analysis V*, volume 2810 of *LNCS*, pages 1–12. Springer, 2003.

[14] A.J. Feelders. Prior knowledge in economic applications of data mining. In D.A. Zighed, J. Komorowski, and J. Zytkow, editors, *Proceedings of PKDD 2000*, volume 1910 of *LNAI*, pages 395–400. Springer, 2000.

[15] D. Gamarnik. Efficient learning of monotone concepts via quadratic optimization. In *Proceedings of the eleventh annual conference on computational learning theory*, pages 134–143. ACM Press, 1998.

[16] J. Karpf. Inductive modelling in law: example based expert systems in administrative law. In *Proceedings of the third international conference on artificial intelligence in law*, pages 297–306. ACM Press, 1991.

[17] W. Kotłowski and R. Słowiński. Statistical approach to ordinal classification with monotonicity constraints. In *Preference Learning ECML/PKDD 2008 Workshop*, 2008.

[18] J. Lijffijt. Fast approximation of isotonic regression and its application to classification. Technical report, Utrecht University, 2008.

[19] K Makino, T Susa, H Ono, and T Ibaraki. Data analysis by positive decision trees. *IEICE Transactions on Information and Systems*, D82-D(1), 1999.

[20] W.L. Maxwell and J.A. Muckstadt. Establishing consistent and realistic reorder intervals in production-distribution systems. *Operations Research*, 33(6):1316–1341, 1985.

[21] L. Montgomery. Nasa metrics data program [http://mdp.ivv.nasa.gov/repository.html], 2008.

[22] M.J. Pazzani, S. Mani, and W.R. Shankle. Acceptance of rules generated by machine learning among medical experts. *Methods of Information in Medicine*, 40:380–385, 2001.

[23] V.N. Popova. *Knowledge Discovery and Monotonicity.* PhD thesis, Erasmus University Rotterdam, 2004.

[24] R. Potharst and J.C. Bioch. A decision tree algorithm for ordinal classification. In D.J. Hand, J.N. Kok, and M.R. Berthold, editors, *Advances in Intelligent Data Analysis*, Lecture Notes in Computer Science 1642, pages 187–198. Springer, 1999.

[25] R Development Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[26] J. Sill. Monotonic networks. In *Advances in neural information processing systems*, NIPS (Vol. 10), pages 661–667, 1998.

[27] M. Velikova. *Monotone Models for Prediction in Data Mining.* PhD thesis, Tilburg University, 2006.

[28] M. Velikova and H. Daniels. Decision trees for monotone price models. *Computational Management Science*, 1(3-4):231–244, 2004.

Appendices

Appendix A

All Results for the Test Using Original Class Labels

Dataset	Error ICT, Gini_{L_1}	Error ICT, L_1	Error CART, Gini_{L_1}	Error CART, L_1
aus	**0.1426**± 0.0070	0.1439± 0.0062	0.1431± 0.0068	0.1443± 0.0062
auto	**0.2982**± 0.0292	0.3094± 0.0289	0.3045± 0.0282	0.3213± 0.0329
bhousing	**0.3966**± 0.0370	0.4037± 0.0442	0.4050± 0.0376	0.4119± 0.0406
car	0.0871± 0.0181	0.1255± 0.0452	**0.0836**± 0.0164	0.1187± 0.0311
computer	**0.4556**± 0.0540	0.4644± 0.0542	0.4773± 0.0554	0.4815± 0.0528
dbhousing	**0.4931**± 0.0830	0.4938± 0.0825	0.5165± 0.0852	0.5153± 0.0871
ERA	**1.2764**± 0.0407	1.2796± 0.0411	1.2926± 0.0415	1.2987± 0.0469
ESL	**0.4348**± 0.0369	0.4516± 0.0372	0.4590± 0.0395	0.4730± 0.0434
haberman	0.2585± 0.0146	**0.2569**± 0.0154	0.2605± 0.0139	0.2591± 0.0141
KC4	**1.1871**± 0.1349	1.1983± 0.1374	1.2358± 0.1474	1.2400± 0.1460
LEV	0.4764± 0.0267	**0.4682**± 0.0254	0.4903± 0.0267	0.4811± 0.0272
PC3	0.5357± 0.0440	0.5279± 0.0394	0.5363± 0.0394	**0.5272**± 0.0345
PC4	**0.5735**± 0.0492	0.5788± 0.0546	0.5835± 0.0543	0.5843± 0.0565
PC5	0.4960± 0.0188	0.4949± 0.0219	**0.4948**± 0.0242	0.4973± 0.0217
pima	**0.2586**± 0.0145	0.2605± 0.0104	0.2619± 0.0144	0.2612± 0.0109
SWD	**0.4707**± 0.0209	0.4721± 0.0175	0.4772± 0.0181	0.4801± 0.0160
whousing	**0.6244**± 0.0328	0.6298± 0.0338	0.6619± 0.0364	0.6629± 0.0332

Table A.1: L_1 errors for classification trees constructed using the original class labels.

Dataset	Leafs ICT, Gini_{L_1}	Leafs ICT, L_1	Leafs CART, Gini_{L_1}	Leafs CART, L_1
aus	3.1700± 1.8589	2.5400± 1.3738	2.8800± 1.8602	**2.3600**± 1.3297
auto	8.8400± 2.7587	**8.4200**± 3.2292	10.3900± 5.2395	9.9500± 6.1977
bhousing	7.9700± 2.9965	9.2900± 3.4824	**7.4000**± 4.4789	8.2700± 5.4788
car	**27.2200**± 5.4985	29.9100± 7.2378	31.6900± 8.4384	40.6200± 12.8572
computer	8.2000± 2.4495	8.1600± 2.6428	8.5600± 4.2790	**8.1200**± 4.4637
dbhousing	5.4300± 1.5059	5.5700± 1.6712	**5.2900**± 1.7014	5.5500± 2.2082
ERA	10.1800± 3.9604	10.0700± 3.3882	8.0900± 3.3001	**8.0600**± 3.0578
ESL	23.3200± 4.4989	**22.4100**± 5.1993	26.0000± 8.7640	25.0200± 10.3816
haberman	2.1700± 1.8205	2.0800± 1.4817	**1.8200**± 1.5134	1.9200± 1.7155
KC4	4.2100± 2.1334	4.3500± 1.9968	4.3000± 3.1734	**4.1000**± 3.0067
LEV	19.8800± 5.2054	**18.4000**± 5.2896	19.0800± 8.8839	19.2300± 8.8499
PC3	2.4600± 1.2667	2.4400± 1.3358	2.3600± 1.5408	**2.1200**± 0.4090
PC4	4.8700± 2.9667	4.8700± 3.2150	**4.2000**± 2.6247	4.2500± 2.7280
PC5	5.8400± 4.0245	6.5100± 4.8648	5.7100± 4.5755	**5.4800**± 6.7262
pima	5.1800± 3.0990	3.6100± 2.5816	4.3700± 3.1226	**3.4000**± 2.7780
SWD	8.9500± 4.1252	10.1800± 4.0110	**7.4500**± 4.5179	8.0500± 5.5018
whousing	16.4000± 5.3522	16.3300± 5.5342	14.7700± 12.2638	**11.7900**± 9.4541

Table A.2: Tree sizes expressed in the number of leaf nodes for classification trees constructed using the original class labels.

Appendix B

All Results for the Test Using Relabelled Data

Dataset	Error ICT, Gini_{L_1}	Error ICT, L_1	Error CART, Gini_{L_1}	Error CART, L_1
aus	**0.1424**± 0.0074	0.1437± 0.0065	0.1430± 0.0072	0.1438± 0.0065
auto	**0.3008**± 0.0312	0.3082± 0.0292	0.3040± 0.0339	0.3141± 0.0311
bhousing	**0.3931**± 0.0366	0.4104± 0.0434	0.3991± 0.0358	0.4126± 0.0386
car	0.0874± 0.0178	0.1252± 0.0419	**0.0838**± 0.0163	0.1179± 0.0291
computer	**0.4479**± 0.0544	0.4544± 0.0528	0.4556± 0.0549	0.4589± 0.0529
dbhousing	**0.4919**± 0.0770	0.4939± 0.0764	0.5053± 0.0788	0.5059± 0.0801
ERA	**1.3009**± 0.0487	1.3107± 0.0486	1.3182± 0.0421	1.3231± 0.0437
ESL	**0.4346**± 0.0358	0.4449± 0.0393	0.4375± 0.0368	0.4506± 0.0408
haberman	0.2601± 0.0195	**0.2559**± 0.0178	0.2575± 0.0150	0.2567± 0.0157
KC4	1.1534± 0.1394	**1.1515**± 0.1357	1.1569± 0.1306	1.1572± 0.1331
LEV	0.4735± 0.0245	0.4717± 0.0264	0.4722± 0.0245	**0.4705**± 0.0237
PC3	0.5351± 0.0443	0.5279± 0.0394	0.5359± 0.0399	**0.5272**± 0.0345
PC4	**0.5866**± 0.0494	0.5899± 0.0643	0.5951± 0.0540	0.5936± 0.0652
PC5	0.4993± 0.0214	0.4966± 0.0220	0.4952± 0.0241	**0.4940**± 0.0235
pima	0.2589± 0.0158	0.2592± 0.0147	**0.2583**± 0.0128	0.2602± 0.0145
SWD	**0.4814**± 0.0219	0.4882± 0.0266	0.4856± 0.0215	0.4926± 0.0209
whousing	0.6245± 0.0413	**0.6226**± 0.0325	0.6396± 0.0362	0.6411± 0.0328

Table B.1: L_1 errors for classification trees constructed using relabelled data.

Dataset	Leafs ICT, Gini_{L_1}	Leafs ICT, L_1	Leafs CART, Gini_{L_1}	Leafs CART, L_1
aus	3.2400± 1.9286	**2.4500**± 1.0286	2.9600± 1.9793	2.5100± 1.4249
auto	9.0000± 2.7524	**8.7500**± 3.1055	10.2100± 4.8872	10.9200± 6.1933
bhousing	**8.1900**± 3.0375	9.3500± 3.3101	8.9600± 5.3445	8.8800± 5.5601
car	**27.9400**± 5.2220	30.3600± 7.2843	31.1600± 8.0374	40.3600± 12.5839
computer	8.6100± 2.1315	**8.2200**± 2.2811	9.0100± 3.3075	9.6700± 3.9108
dbhousing	5.2300± 1.4624	5.6900± 1.5935	**5.1700**± 1.5509	5.8700± 2.1726
ERA	11.7600± 4.1830	12.3600± 4.3520	**11.6800**± 6.1626	12.7200± 6.9647
ESL	**23.5500**± 4.2791	23.5800± 5.0516	26.1900± 5.9198	28.3800± 7.4002
haberman	2.7600± 2.0652	2.9600± 2.8530	**2.3700**± 2.0776	2.7900± 2.9346
KC4	4.7000± 1.6787	4.7700± 1.7971	**4.5700**± 1.9553	4.6400± 2.1155
LEV	**21.3100**± 6.3987	21.6600± 6.4514	26.3100± 8.8565	28.5900± 9.9961
PC3	2.4800± 1.3669	2.4400± 1.3358	2.3600± 1.5408	**2.1200**± 0.4090
PC4	5.4600± 3.2642	4.9700± 3.1381	5.0500± 4.0009	**4.3600**± 3.2490
PC5	6.2000± 4.5438	6.3700± 4.9597	**6.0700**± 4.5644	6.4900± 6.4346
pima	3.5300± 1.7318	3.7300± 2.8985	3.9800± 2.7265	**3.3900**± 3.5073
SWD	**9.5300**± 5.0920	9.6700± 4.9422	11.0700± 8.5010	11.9600± 11.1264
whousing	16.9200± 4.4624	**16.4200**± 5.5817	22.5800± 11.9850	19.4500± 12.5210

Table B.2: Tree sizes expressed in the number of leaf nodes for classification trees constructed using relabelled data.

Appendix C

All Results Using Original Class Labels versus All Results Using Relabelled Data

Dataset	Error ICT	Error CART	Leafs ICT	Leafs CART
aus	0.1426± 0.0070	0.1431± 0.0068	3.1700± 1.8589	**2.8800**± 1.8602
	0.1424± 0.0074	0.1430± 0.0072	3.2400± 1.9286	2.9600± 1.9793
auto	**0.2982**± 0.0292	0.3045± 0.0282	**8.8400**± 2.7587	10.3900± 5.2395
	0.3008± 0.0312	0.3040± 0.0339	9.0000± 2.7524	10.2100± 4.8872
bhousing	0.3966± 0.0370	0.4050± 0.0376	7.9700± 2.9965	**7.4000**± 4.4789
	0.3931± 0.0366	0.3991± 0.0358	8.1900± 3.0375	8.9600± 5.3445
car	0.0871± 0.0181	**0.0836**± 0.0164	**27.2200**± 5.4985	31.6900± 8.4384
	0.0874± 0.0178	0.0838± 0.0163	27.9400± 5.2220	31.1600± 8.0374
computer	0.4556± 0.0540	0.4773± 0.0554	**8.2000**± 2.4495	8.5600± 4.2790
	0.4479± 0.0544	0.4556± 0.0549	8.6100± 2.1315	9.0100± 3.3075
dbhousing	0.4931± 0.0830	0.5165± 0.0852	5.4300± 1.5059	5.2900± 1.7014
	0.4919± 0.0770	0.5053± 0.0788	5.2300± 1.4624	**5.1700**± 1.5509
ERA	**1.2764**± 0.0407	1.2926± 0.0415	10.1800± 3.9604	**8.0900**± 3.3001
	1.3009± 0.0487	1.3182± 0.0421	11.7600± 4.1830	11.6800± 6.1626
ESL	0.4348± 0.0369	0.4590± 0.0395	**23.3200**± 4.4989	26.0000± 8.7640
	0.4346± 0.0358	0.4375± 0.0368	23.5500± 4.2791	26.1900± 5.9198
haberman	0.2585± 0.0146	0.2605± 0.0139	2.1700± 1.8205	**1.8200**± 1.5134
	0.2601± 0.0195	**0.2575**± 0.0150	2.7600± 2.0652	2.3700± 2.0776
KC4	1.1871± 0.1349	1.2358± 0.1474	**4.2100**± 2.1334	4.3000± 3.1734
	1.1534± 0.1394	1.1569± 0.1306	4.7000± 1.6787	4.5700± 1.9553
LEV	0.4764± 0.0267	0.4903± 0.0267	19.880± 5.2054	**19.0800**± 8.8839
	0.4735± 0.0245	**0.4722**± 0.0245	21.3100± 6.3987	26.3100± 8.8565
PC3	0.5357± 0.0440	0.5363± 0.0394	2.4600± 1.2667	**2.3600**± 1.5408
	0.5351± 0.0443	0.5359± 0.0399	2.4800± 1.3669	**2.3600**± 1.5408
PC4	**0.5735**± 0.0492	0.5835± 0.0543	4.8700± 2.9667	**4.2000**± 2.6247
	0.5866± 0.0494	0.5951± 0.0540	5.4600± 3.2642	5.0500± 4.0009
PC5	0.4960± 0.0188	**0.4948**± 0.0242	5.8400± 4.0245	**5.7100**± 4.5755
	0.4993± 0.0214	0.4952± 0.0241	6.2000± 4.5438	6.0700± 4.5644
pima	0.2586± 0.0145	0.2619± 0.0144	5.1800± 3.0990	4.3700± 3.1226
	0.2589± 0.0158	**0.2583**± 0.0128	**3.5300**± 1.7318	3.9800± 2.7265
SWD	**0.4707**± 0.0209	0.4772± 0.0181	8.9500± 4.1252	**7.4500**± 4.5179
	0.4814± 0.0219	0.4856± 0.0215	9.5300± 5.0920	11.0700± 8.5010
whousing	**0.6244**± 0.0328	0.6619± 0.0364	16.4000± 5.3522	**14.770**± 12.2638
	0.6245± 0.0413	0.6396± 0.0362	16.9200± 4.4624	22.5800± 11.9850

Table C.1: Results for the tests using original class labels and Gini_{L_1} versus the results of the tests using relabelled data and Gini_{L_1}. For each dataset the top row indicates the average L_1 error and average tree size of the classification trees constructed using the original class labels, while the bottom row shows the same statistics for the classification trees constructed using the relabelled data.

Dataset	Error ICT	Error CART	Leafs ICT	Leafs CART
aus	0.1439± 0.0062	0.1443± 0.0062	2.5400± 1.3738	**2.3600**± 1.3297
	0.1437± 0.0065	0.1438± 0.0065	2.4500± 1.0286	2.5100± 1.4249
auto	0.3094± 0.0289	0.3213± 0.0329	**8.4200**± 3.2292	9.9500± 6.1977
	0.3082± 0.0292	0.3141± 0.0311	8.7500± 3.1055	10.9200± 6.1933
bhousing	**0.4037**± 0.0442	0.4119± 0.0406	9.2900± 3.4824	**8.2700**± 5.4788
	0.4104± 0.0434	0.4126± 0.0386	9.3500± 3.3101	8.8800± 5.5601
car	0.1255± 0.0452	0.1187± 0.0311	**29.9100**± 7.2378	40.6200± 12.8572
	0.1252± 0.0419	**0.1179**± 0.0291	30.3600± 7.2843	40.3600± 12.5839
computer	0.4644± 0.0542	0.4815± 0.0528	8.1600± 2.6428	**8.1200**± 4.4637
	0.4544± 0.0528	0.4589± 0.0529	8.2200± 2.2811	9.6700± 3.9108
dbhousing	**0.4938**± 0.0825	0.5153± 0.0871	5.5700± 1.6712	**5.5500**± 2.2082
	0.4939± 0.0764	0.5059± 0.0801	5.6900± 1.5935	5.8700± 2.1726
ERA	**1.2796**± 0.0411	1.2987± 0.0469	10.0700± 3.3882	**8.0600**± 3.0578
	1.3107± 0.0486	1.3231± 0.0437	12.3600± 4.3520	12.7200± 6.9647
ESL	0.4516± 0.0372	0.4730± 0.0434	**22.4100**± 5.1993	25.0200± 10.3816
	0.4449± 0.0393	0.4506± 0.0408	23.5800± 5.0516	28.3800± 7.4002
haberman	0.2569± 0.0154	0.2591± 0.0141	2.0800± 1.4817	**1.9200**± 1.7155
	0.2559± 0.0178	0.2567± 0.0157	2.9600± 2.8530	2.7900± 2.9346
KC4	1.1983± 0.1374	1.2400± 0.1460	4.3500± 1.9968	**4.1000**± 3.0067
	1.1515± 0.1357	1.1572± 0.1331	4.7700± 1.7971	4.6400± 2.1155
LEV	0.4682± 0.0254	0.4811± 0.0272	**18.4000**± 5.2896	19.2300± 8.8499
	0.4717± 0.0264	**0.4705**± 0.0237	21.6600± 6.4514	28.5900± 9.9961
PC3	0.5279± 0.0394	**0.5272**± 0.0345	2.4400± 1.3358	**2.1200**± 0.4090
	0.5279± 0.0394	**0.5272**± 0.0345	2.4400± 1.3358	**2.1200**± 0.4090
PC4	**0.5788**± 0.0546	0.5843± 0.0565	4.8700± 3.2150	**4.2500**± 2.7280
	0.5899± 0.0643	0.5936± 0.0652	4.9700± 3.1381	4.3600± 3.2490
PC5	0.4949± 0.0219	0.4973± 0.0217	6.5100± 4.8648	**5.4800**± 6.7262
	0.4966± 0.0220	**0.4940**± 0.0235	6.3700± 4.9597	6.4900± 6.4346
pima	0.2605± 0.0104	0.2612± 0.0109	3.6100± 2.5816	3.4000± 2.7780
	0.2592± 0.0147	0.2602± 0.0145	3.7300± 2.8985	**3.3900**± 3.5073
SWD	**0.4721**± 0.0175	0.4801± 0.0160	10.1800± 4.0110	**8.0500**± 5.5018
	0.4882± 0.0266	0.4926± 0.0209	9.6700± 4.9422	11.9600± 11.1264
whousing	0.6298± 0.0338	0.6629± 0.0332	16.3300± 5.5342	**11.7900**± 9.4541
	0.6226± 0.0325	0.6411± 0.0328	16.4200± 5.5817	19.4500± 12.5210

Table C.2: Results for the tests using original class labels and the L_1 impurity measure versus the results of the tests using relabelled data and the L_1 impurity measure. For each dataset the top row indicates the average L_1 error and average tree size of the classification trees constructed using the original class labels, while the bottom row shows the same statistics for the classification trees constructed using the relabelled data.