Pruning for monotone classification trees

Ad Feelders and Martijn Pardoel

Utrecht University, Institute of Information and Computing Sciences,
P.O. Box 80089, 3508TB Utrecht, The Netherlands,
ad@cs.uu.nl, mapardoe@cs.uu.nl

Abstract. For classification problems with ordinal attributes very often the class attribute should increase with each or some of the explanatory attributes. These are called classification problems with monotonicity constraints. Standard classification tree algorithms such as CART or C4.5 are not guaranteed to produce monotone trees, even if the data set is completely monotone. We look at pruning based methods to build monotone classification trees from monotone as well as nonmonotone data sets. We develop a number of fixing methods, that make a non-monotone tree monotone by additional pruning steps. These fixing methods can be combined with existing pruning techniques to obtain a sequence of monotone trees. The performance of the new algorithms is evaluated through experimental studies on artificial as well as real life data sets. We conclude that the monotone trees have a slightly better predictive performance and are considerably smaller than trees constructed by the standard algorithm.

1 Introduction

A common form of prior knowledge in data analysis concerns the monotonicity of relations between the dependent and explanatory variables. For example, economic theory would state that, all else equal, people tend to buy less of a product if its price increases. The precise functional form of this relationship is however not specified by theory.

Monotonicity may also be an important model requirement with a view toward explaining and justifying decisions, such as acceptance/rejection decisions. Consider for example a university admission procedure where candidate a scores at least as good on all admission criteria as candidate b, but a is refused whereas b is admitted. Such a nonmonotone admission rule would clearly be unacceptable. Similar considerations apply to selection procedures for applicants for e.g. a job or a loan.

Because the monotonicity constraint is quite common in practice, many data analysis techniques have been adapted to be able to handle such constraints. In this paper we look at pruning based methods to build monotone classification trees from monotone as well as nonmonotone data sets. We develop a number of fixing methods, that make a non-monotone tree monotone by further pruning steps. These fixing methods can be combined with existing pruning techniques to obtain a sequence of monotone trees. The performance of the new algorithms is evaluated through experimental studies on artificial as well as real life data sets.

This paper is organised as follows. In section 2 we define monotone classification and other important concepts that are used throughout the paper. In section 3 we shortly review previous work in the area of monotone classification trees. Then we discuss a number of methods to make a non-monotone tree monotone by additional pruning steps in section 4. These methods are evaluated experimentally in section 5 on real life and artificial data sets. Finally, in section 6 we end with a summary, and some ideas for further research.

2 Monotone Classification

In this section we define the problem of monotone classification and introduce some notation that will be used throughout the paper. Notation is largely based on [Pot99].

Let $\mathcal{X}$ be a *feature space* $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_p$ consisting of vectors $\mathbf{x} = (x_1, x_2, \ldots, x_p)$ of values on p features or attributes. We assume that each feature takes values x_i in a linearly ordered set $\mathcal{X}_i$. The partial ordering $\leq$ on $\mathcal{X}$ will be the ordering induced by the order relations of its coordinates $\mathcal{X}_i$: $\mathbf{x} = (x_1, x_2, \ldots, x_p) \leq \mathbf{x}' = (x'_1, x'_2, \ldots, x'_p)$ if and only if $x_i \leq x'_i$ for all i. Furthermore, let $\mathcal{C}$ be a finite linearly ordered set of *classes*.

A *monotone* classification rule is a function $f : \mathcal{X} \to \mathcal{C}$ for which

$$\mathbf{x} \leq \mathbf{x}' \Rightarrow f(\mathbf{x}) \leq f(\mathbf{x}')$$

for all instances $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$. It is easy to see that a classification rule on a feature space is monotone if and only if it is non-decreasing in each of its features, when the remaining features are held fixed.

A *data set* is a series $(\mathbf{x}_1, c_1), (\mathbf{x}_2, c_2), \ldots, (\mathbf{x}_n, c_n)$ of n examples $(\mathbf{x}_i, c_i)$ where each $\mathbf{x}_i$ is an element of the instance space $\mathcal{X}$ and c_i is a class label from $\mathcal{C}$. We call a dataset *consistent* if for all i, j we have $\mathbf{x}_i = \mathbf{x}_j \Rightarrow c_i = c_j$. That is, each instance in the data set has a unique associated class. A data set is *monotone* if for all i, j we have $\mathbf{x}_i \leq \mathbf{x}_j \Rightarrow c_i \leq c_j$. It is easy to see that a monotone data set is necessarily consistent.

The classification rules we consider are univariate binary classification trees. For such trees, at each node a split is made using a test of the form $X_i < d$ for some $d \in \mathcal{X}_i, 1 \leq i \leq p$. Thus, for a binary tree, in each node the associated set $T \subset \mathcal{X}$ is split into the two subsets $T_\ell = \{\mathbf{x} \in T : x_i < d\}$ and $T_r = \{\mathbf{x} \in T : x_i \geq d\}$. The classification rule that is induced by a decision tree $\mathcal{T}$ will be denoted by $f_{\mathcal{T}}$.

For any node or leaf T of $\mathcal{T}$, the subset of the instance space associated with that node can be written

$$T = \{\mathbf{x} \in \mathcal{X} : \mathbf{a} \leq \mathbf{x} < \mathbf{b}\} = [\mathbf{a}, \mathbf{b}) \tag{1}$$

for some $\mathbf{a}, \mathbf{b} \in \overline{\mathcal{X}}$ with $\mathbf{a} \leq \mathbf{b}$. Here $\overline{\mathcal{X}}$ denotes the extension of $\mathcal{X}$ with infinity-elements $-\infty$ and ∞. In some cases we need the infinity elements so we can specify a node as in equation (1).

Below we will call $\min(T) = \mathbf{a}$ the *minimal element* and $\max(T) = \mathbf{b}$ the *maximal element* of T. Together, we call these the *corner* elements of the node T. There is a straightforward manner to test the monotonicity using the maximal and minimal elements of the leaves of the classification tree [Pot99]:

```
for all pairs of leaves $T, T'$:
    if $\Big(f_{\mathcal{T}}(T) > f_{\mathcal{T}}(T')$ and $\min(T) < \max(T')\Big)$ or
        $\Big(f_{\mathcal{T}}(T) < f_{\mathcal{T}}(T')$ and $\max(T) > \min(T')\Big)$
    then stop: $\mathcal{T}$ not monotone
```

Fig. 1. Algorithm to test the monotonicity of a classification tree

3 Previous Research

In this section we briefly describe previous work on monotone classification trees. Ben-David [BD95] was the first to incorporate the monotonicity constraint in classification tree construction. He proposed a measure for the quality of a candidate split that takes into account both the impurity reduction that the split achieves, as well as the degree of non-monotonicity of the resulting tree. The degree of non-monotonicity is derived from the number of nonmonotonic leaf pairs of the tree that is obtained after the candidate split is performed. The relative weight of impurity reduction and degree of non-monotonicity in computing the quality of a candidate split can be determined by the user. Ben-David shows that his algorithm results in trees with a lower degree of nonmonotonicity, without a significant deterioration of the inductive accuracy.

Makino et al. [MSOI99] were the first to develop an algorithm that guarantees a monotone classification tree. However, their method only works in the two class setting, and requires monotone training data. Potharst and Bioch [PB99,PB00] extended this method to the k-class problem, also accommodating continuous attributes. Both Makino as well as Potharst and Bioch enforce monotonicity by adding the corner elements of a node with an appropriate class label to the existing data whenever necessary. While their results are encouraging, applicability of the algorithms is limited by the requirement of monotone training data. In actual practice this is hardly ever the case.

Bioch and Popova [BP02] extend the work of Potharst to nonmonotone data sets by relabeling data if necessary. While their approach is certainly interesting, a potential disadvantage is that the algorithm may have to add very many data points to the initial training sample. This may result in highly complex trees that have to be pruned back afterwards. Bioch and Popova, in another recent paper [BP03], perform some experiments with an alternative splitting criterion that was proposed by Cao-Van and De Baets [CVB02]. This *number of conflicts* measure chooses the split with the least number of inconsistencies or conflicts, i.e. the number of non-monotone pairs of points in the resulting branches. An important conclusion of Bioch and Popova [BP03] is that with the increase of monotonicity noise in the data the number of conflicts criterion generates larger trees with a lower misclassification rate than the entropy splitting criterion.

Feelders [Fee00] argues that the use of a measure of monotonicity in determining the best splits has certain drawbacks, for example that a nonmonotone tree may become monotone after additional splits. He proposes not to enforce monotonicity during tree construction, but to use resampling to generate many different trees and select the ones that are monotone. This allows the use of a slightly modified standard tree algorithm. Feelders concludes that the predictive performance of the monotone trees found with this method is comparable to the performance of the nonmonotone trees. Furthermore, the monotone trees were much simpler and proved to be more stable.

Daniels and Velikova [DV03] present an algorithm to make the training sample monotone by making minimal adjustments to the class labels. They combine this idea with the resampling approach described above. Their results show that the predictive performance of the trees constructed on the cleaned data is superior to the predictive performance of the trees constructed on the raw data.

Finally, Potharst and Feelders [PF02] give a short survey of monotone classification trees.

4 Pruning towards monotone trees

The objective of our work is to develop an algorithm that produces a monotone tree with good predictive performance from both monotone and nonmonotone training data. In order to construct a monotone tree, we initially grow a large overfitted tree, and then prune towards monotone subtrees. For growing the initial tree we implemented an algorithm that is very similar to CART [BFOS84]; the only major difference is that our implementation records the corner elements of each node during tree construction.

Central to our approach are a number of so-called fixing methods. The basic idea of all our fixing methods is to make minimal adjustments to a nonmonotone tree in order to obtain a monotone subtree. To quantify the degree of nonmonotonicity, we count the number of leaf pairs that are nonmonotone with respect to each other. If a leaf participates in one or more nonmonotone leaf pairs, we call it a nonmonotone leaf. Hence the goal of all fixing methods is to obtain a tree without any nonmonotone leaves. They do so by pruning in a parent node that has at least one nonmonotone leaf for a child; this is repeated until the resulting tree is monotone. The fixing methods differ in the heuristic that is used to select the parent node to be pruned.

In figure 2 we have depicted a nonmonotone tree that we use to illustrate the different heuristics. Each node is numbered (the italicized numbers to the left of each node); the associated split is to the right of each node; and inside each node the classlabel is given. A node is always allocated to the class with the highest relative frequency in that node.

Below each leaf node we have listed the number of observations in that leaf. As the reader can verify, the nonmonotone leaf pairs of this tree are [9,10], [9,12], [9,14], [11,12], [11,14] and [13,14]. For example, the leaf pair [9,10] is nonmonotone because $f_{\mathcal{T}}(9) > f_{\mathcal{T}}(10)$ and $\min(9) < \max(10)$, that is

$$(-\infty, -\infty, 0, -\infty, -\infty) < (0, \infty, \infty, 0, \infty).$$

The leaf pair [8,13] is monotone however, since $f_{\mathcal{T}}(13) > f_{\mathcal{T}}(8)$ but $\min(13) \not< \max(8)$.

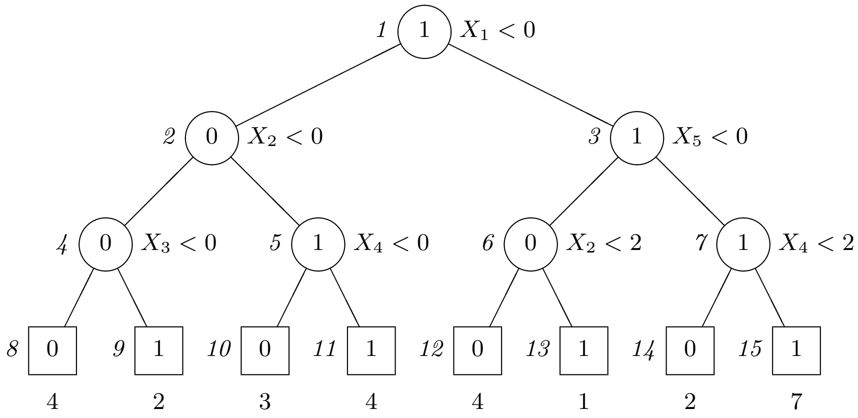

Fig. 2. Example nonmonotone decision tree

The first method prunes in the parent whose children participate in the most nonmonotone leaf pairs. Note that it may be the case that only one of the two children is a leaf node. In that case we only count the number of nonmonotone leaf pairs in which this single child participates. This way of counting aims to discourage pruning in nodes higher up in the tree. We call this heuristic the most nonmonotone parent (MNP) fixmethod. In our example we first count for each parent of a leaf the number of nonmonotonic leaf pairs its children participate in. For node 4 this number is 3 ([9,10],[9,12],[9,14]), node 5 also gives 3 ([9,10], [11,12], [11,14]), and for node 6 and 7 this number is 3 as well. We now face the problem what to do in case of a tie. Apart from choosing one of the equally good fixes at random, we will also consider the alternative of choosing the node with the least number of observations. Henceforth, this rule will be referred to as the LNO tie breaker. The rationale behind this rule is that if the data generating process is indeed monotone, then nonmonotonicity is likely to be due to small erratic leaf nodes. Application of this rule leads to pruning in node 6 since it contains the least observations (5) of the candidates.

A potential disadvantage of the MNP heuristic is that it ignores the fact that after pruning in a node, new nonmonotone leaf pairs, involving the newly created leaf, may appear. Therefore we tested a second method that prunes in the parent of a nonmonotone leaf that gives the biggest reduction in the number of nonmonotone leaf pairs. To prevent pruning too high in the tree too early, we do this levelwise, i.e. we first consider the set of parent nodes with minimal number of descendants for an improvement before we consider parent nodes with higher numbers of descendants. This fixmethod is called the best fix (BF). In our example we first note that all parent nodes have the same number of descendants and consequently will all be considered at the same time. Now, we need to know how many nonmonotone leaf pairs remain after each possible prune action, thus how big the improvement is. To determine this, we need to know with which leaves the candidate leaf nodes 4, 5, 6, and 7 are nonmonotone. These are the pairs [5,12], [5,14], [6,9] and [6,11]; nodes 4 and 7 are monotone with all other leaves, as the reader can verify. We can now compute the number of nonmonotone leaf pairs left after each possible prune. If we prune in node 4 we lose [9,10], [9,12] and [9,14], and are left with [11,12], [11,14] and [13,14], i.e. 3 pairs. If we prune in node 5 we lose [9,10], [11,12] and [11,14], but get [5,12] and [5,14] and still have [9,12], [9,14] and [13,14], i.e. 5 pairs. If we prune in node 6 we are left with 5 pairs and if we prune in node 7 only 3 pairs remain. Thus, we have two best fixes, namely nodes 4 and 7, and node 4 is selected by the LNO tie breaker.

The fixing methods discussed can be combined in different ways with existing pruning techniques. We study how they can be combined with cost-complexity pruning ([BFOS84]). An elementary cost-complexity pruning step works as follows. Let $R(\mathcal{T})$ denote the fraction of cases in the training sample that are misclassified by tree $\mathcal{T}$. The total cost $C_\alpha(\mathcal{T})$ of tree $\mathcal{T}$ is defined as

$$C_\alpha(\mathcal{T}) = R(\mathcal{T}) + \alpha|\tilde{\mathcal{T}}|. \qquad (2)$$

The total cost of tree $\mathcal{T}$ consists of two components: resubstitution error $R(\mathcal{T})$, and a penalty for the complexity of the tree $\alpha|\tilde{\mathcal{T}}|$. In this expression $\tilde{\mathcal{T}}$ denotes the set of leaf nodes of $\mathcal{T}$, and α is the parameter that determines the complexity penalty. Depending on the value of α (≥ 0) a complex tree that makes no errors may now have a higher total cost than a small tree that makes a number of errors. When considering a tree $\mathcal{T}$, for each node T we can compute the value of α for which the tree obtained by pruning $\mathcal{T}$ in T has equal cost to $\mathcal{T}$ itself. By pruning in the node for which this α-value is minimal, we obtain the tree that is the first to become better than $\mathcal{T}$, when α is increased.

Now, the first way to combine pruning and fixing is to *alternate* between cost-complexity pruning steps and fixing steps. This proceeds as follows. If the initial tree is monotone we add it as the first tree in the sequence; otherwise we first fix it, i.e. make it monotone. Once we have a monotone tree, a cost-complexity pruning step is performed. Subsequently, we check again if the resulting tree is monotone, add it to the sequence if it is, or first fix it otherwise. We continue alternating the pruning and fixing steps until we eventually reach the root node.

A second possibility is to simply take the sequence of trees produced by cost-complexity pruning, and apply the fixing methods to all trees in that sequence. Finally, one could simply take the best tree (in terms of predictive accuracy on a test sample) from the cost-complexity sequence and fix that tree. In the experiments we discuss in the next section we only consider the strategy that alternates between pruning and fixing.

5 Experimental Evaluation

In this section we discuss the experimental results we obtained on real life data sets and artificially generated data. For the real life data (see table 1) we selected a number of data sets from domains where monotonicity is either a plausible assumption or requirement. The *bankruptcy* data set was used by Pompe in [Pom01]. The class label indicates whether or not a company has gone bankrupt within one year of the annual report from which the attributes were taken. These attributes are 11 financial ratios that are indicative for the profitability, activity, liquidity and solvency of the company. The *Windsor housing* data set was used as an example by Koop [Koo00]. It contains data on 546 houses sold in Windsor, Canada. The dependent variable is the sales price of the house in Canadian dollars. This was converted into a class label by taking the quartiles of the sales price.

All other data sets, *adult income, Australian credit approval, Boston housing* and *cpu performance* were taken from the UCI machine learning repository [BM98]. Whenever necessary, attributes with no monotonicity relation with respect to the class label were removed from the data sets. Also, some negative monotone attributes were inverted to make them positive monotone. For example, in the cpu performance data set the machine cycle time in nanoseconds was converted to clock speed in kilohertz. The class labels of Boston housing and cpu performance were created in the same way as for Windsor housing.

Table 1 gives a number of important characteristics of the data sets. The column *Error* gives the misclassification error of the rule that assigns every observation to the majority class. With regard to this study, the degree of nonmonotonicity of the data sets is of special interest. This can be quantified in various ways. In the column *NmDeg*, for degree of nonmonotonicity, we give the ratio of the number of nonmonotone pairs of datapoints to the total number of pairs. However, we require a measure that allows the meaningful comparison of the degree of nonmonotonicity for data sets with highly different characteristics. Important characteristics in this regard are: the number of attributes and class labels, and the fraction of incomparable pairs of points (*FracInc* in table 1). To this end we take the ratio of the observed number of nonmonotone pairs to the number of nonmonotone pairs that would be expected if the class labels were assigned randomly to the observed attribute vectors. This last number is approximated by taking fifty permutations of the observed class labels and computing the average number of nonmonotone pairs of the resulting fifty data sets. The resulting number is given in table 1 in the column *NmRat*. If the ratio is close to zero, then the observed number of nonmonotone pairs is much lower than would be expected by chance.

Table 1. Overview of real life data sets and their characteristics

Description	Classes	Size	Attr.	Error	FracInc	NmDeg	NmRat
Adult Income	2	1000	4	0.500	0.610	0.020	0.21
Aus. Credit Approval	2	690	4	0.445	0.284	0.034	0.19
Boston Housing	4	506	6	0.745	0.678	0.006	0.05
CPU Performance	4	209	6	0.732	0.505	0.007	0.04
Windsor Housing	4	546	11	0.747	0.726	0.009	0.08
Bankruptcy	2	1090	11	0.500	0.877	0.003	0.08

Table 2. Misclassification rate averaged over twenty trials; MNP = Most Nonmonotone Parent, BF = Best Fix, BF-LNO = Best Fix with Least Number of Observations tie breaker

Dataset	Standard	Default	MNP	BF	BF-LNO
AdInc	0.263	0.263	0.260	0.261	0.260
AusCr	0.146	0.146	0.147	0.146	0.146
BosHou	0.331	0.332	0.335	0.336	0.330
CpuPerf	0.394	0.406	0.391	0.388	0.388
DenBHou	0.181	0.191	0.190	0.181	0.181
WindHou	0.531	0.534	0.534	0.533	0.536
Bankr	0.225	0.224	0.223	0.223	0.223
Total	2.071	2.096	2.080	2.068	2.064

For all data sets we follow the same procedure. The data sets are randomly partitioned into a training set (half of the data), a test set (a quarter of the data) and a validation set (also a quarter of the data). This is done twenty times. Each of those times the training set is used to build a tree sequence with the different algorithms considered, and for each algorithm the tree with the lowest misclassification rate on the test set is selected. Subsequently, we test for statistically significant differences in predictive accuracy between the methods using the validation set. The results of these experiments are reported in table 2 and table 3.

The column labeled *Standard* contains the results of a standard CART-like algorithm that we implemented. It uses cost-complexity pruning to create a sequence of trees from which the one with the smallest error on the test set is selected. The resulting trees may be either monotone or nonmonotone. The

Table 3. Tree size averaged over twenty trials; MNP = Most Nonmonotone Parent, BF = Best Fix, BF-LNO = Best Fix with Least Number of Observations tie breaker

Dataset	Standard	Default	MNP	BF	BF-LNO
AdInc	16.40	10.00	13.00	13.95	14.15
AusCr	4.05	3.55	3.95	4.6	4.60
BosHou	13.55	7.65	9.60	10.85	10.65
CpuPerf	15.80	7.40	10.35	10.50	10.50
DenBHou	4.80	2.55	3.75	3.65	3.65
WindHou	44.00	13.15	18.85	19.70	20.10
Bankr	9.15	7.10	8.45	6.65	6.75
Total	107.75	51.40	67.95	69.90	70.40

column *Default* was computed by taking the best monotone tree from the tree sequence produced by the standard algorithm. The last three columns contain the results of the different fixing methods discussed in section 4 combined with the alternating pruning strategy.

From table 2 we conclude that the BF-LNO method performs best, even better than the standard method. However, the differences are not statistically significant: in a total of 140 experiments (20 per dataset) the standard method was significantly better 2 times and significantly worse 1 time, at the 5% level. The default method performs worst, but, again, this is not statistically significant: the standard method is significantly better only once. The slightly worse performance of the default method can be explained by looking at table 3, which contains the tree sizes measured in number of leafs. We see that the trees produced by the default method are somewhat smaller than the trees produced by the MNP, BF and BF-LNO methods: it seems that the default method slightly overprunes. More importantly, we see that the trees produced by the standard method are much larger than the trees produced by the other methods. To take an extreme example: on the *Windsor housing* data set the standard algorithm produces trees with an average of 44 leaf nodes, against an average of 13.15 for the default method, and an average of 18.85 for the MNP heuristic.

Table 4. Average misclassification rate and tree size of all artificial data sets per ratio of nonmonotonicity. The column *Frac. Mon.* gives the fraction of monotone trees generated by the standard algorithm.

NmRat	Standard	Frac. Mon.	Default	MNP	BF	BF-LNO
0	0.096 / 6.0	0.775	0.103 / 5.3	0.102 / 5.6	0.103 / 5.5	0.101 / 5.6
0.05	0.163 / 6.1	0.825	0.164 / 5.5	0.165 / 5.6	0.168 / 5.4	0.167 / 5.4
0.20	0.324 / 5.5	0.850	0.321 / 4.6	0.320 / 4.7	0.324 / 4.6	0.324 / 4.6

Table 4 summarizes the results of our experiments with the artificial data sets. Each artificial data set was generated by making random draws from a prespecified tree model. The results were averaged along the dimensions *number of class labels* (2 and 4), *number of observations* (100 and 500), and twenty random partitions into training and test data. We generated data sets with differing degrees of nonmonotonicity (0, 0.05 and 0.2 respectively). As with the real life data, the differences between the predictive accuracy of the different fixing heuristics are not significant. Furthermore, we conclude that the predictive accuracy of the standard trees and the monotone trees is comparable.

Also note that the performance of the different methods is comparable for *all* levels of nonmonotonicity considered. More detailed results (not presented here), show that the relative performance is not affected by the other characteristics (the size of the data set and the number of class labels) either. Hence, we conclude that the proposed methods are quite robust.

Another interesting, perhaps somewhat counterintuitive observation is that the fraction of monotone trees produced by the standard algorithm seems to *increase* with the nonmonotonicity ratio. An attempt to explain this in terms of average tree size is not entirely satisfactory, even though the trees at NmRat = 0.20 are somewhat smaller than the trees constructed from completely monotone data. A final observation is that on all real life and artificial data sets, the fixing methods at most had to be applied once, namely to the initial tree. This means that, although we described the algorithm as an alternation of fixing and pruning steps, it turns out that in practice no fixing steps were required once a monotone tree had been constructed.

6 Conclusions and future research

We have presented a number of fixing methods, that make a non-monotone tree monotone by pruning towards a monotone subtree. We have shown that these fixing methods can be combined with cost-complexity pruning to obtain a sequence of monotone trees. The performance of the new algorithms has been evaluated through experimental studies on artificial as well as real life data sets. We conclude that in terms of predictive performance the monotone trees are comparable to the trees produced by the standard algorithm; however, they are considerably smaller.

We summarize the strong points of our pruning based methods as we see them. First of all, the methods proposed are *guaranteed* to produce a monotone tree. Secondly, the algorithm works on nonmonotone as well as monotone data sets and therefore has no problems with noisy data. Thirdly, for the problems considered the monotone trees predict slightly, though not significantly, better than the standard trees. Finally, the monotone trees are much smaller than those produced by the standard algorithm.

We see a number of issues for further research. In this study we only applied the the strategy that alternates between pruning and fixing, as presented in section 4. We intend to investigate the other options that we described there.

Preliminary experiments suggest however, that the differences in performance are negligible. Another issue for further research is to extend our work in a different direction. Currently, our algorithm can only be applied to problems for which there is a monotonicity constraint on all variables. For many problems this requirement is too strong: usually there are one or more variables for which the constraint does not apply. We intend to extend the algorithm to handle such cases. Yet another interesting extension would be to consider monotone *regression* trees. In fact, some data sets we used in the experiments are more naturally analysed with regression trees, for example the housing data.

Finally, it seems worthwhile to further investigate the relation between the degree of nonmonotonicity of the data and the fraction of monotone trees generated by a standard algorithm.

Acknowledgements

The authors would like to thank Paul Pompe for the kind donation of the bankruptcy data set, and Eelko Penninkx for helpful suggestions on earlier drafts of this paper.

References

[BD95] Arie Ben-David. Monotonicity maintenance in information-theoretic machine learning algorithms. *Machine Learning*, 19:29–43, 1995.

[BFOS84] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees (CART).* Wadsworth, 1984.

[BM98] C.L. Blake and C.J. Merz. UCI repository of machine learning databases [http://www.ics.uci.edu/∼mlearn/mlrepository.html], 1998.

[BP02] Jan C. Bioch and Viara Popova. Monotone decision trees and noisy data. *ERIM Report Series Research in Management*, ERS-2002-53-LIS, 2002.

[BP03] Jan C. Bioch and Viara Popova. Induction of ordinal decision trees: an MCDA approach. *ERIM Report Series Research in Management*, ERS-2003-008-LIS, 2003.

[CVB02] Kim Cao-Van and Bernard De Beats. Growing decision trees in an ordinal setting. Submitted to International Journal of Intelligent Systems, 2002.

[DV03] H.A.M Daniels and M. Velikova. Derivation of monotone decision models from non-monotone data. Technical Report 2003-30, Center, Tilburg University, 2003.

[Fee00] A.J. Feelders. Prior knowledge in economic applications of data mining. In D.A. Zighed, J. Komorowski, and J. Zytkow, editors, *Principles of Data Mining and Knowledge Discovery*, Lecture Notes in Artificial Intelligence 1910, pages 395–400. Springer, 2000.

[Koo00] Gary Koop. *Analysis of Economic Data.* John Wiley and Sons, 2000.

[MSOI99] Kazuhisa Makino, Takashi Susa, Hirotaka Ono, and Toshihide Ibaraki. Data analysis by positive decision trees. *IEICE Transactions on Information and Systems*, E82-D(1), 1999.

[PB99] 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.

[PB00] R. Potharst and J.C. Bioch. Decision trees for ordinal classification. *Intelligent Data Analysis*, 4(2):97–112, 2000.

[PF02] Rob Potharst and Ad Feelders. Classification trees for problems with monotonicity constraints. *SIGKDD Explorations*, 4:1–10, 2002.

[Pom01] Paul P.M. Pompe. *New developments in bankruptcy prediction*. PhD thesis, University of Twente, 2001.

[Pot99] Rob Potharst. *Classification using Decision Trees and Neural Nets*. PhD thesis, Erasmus University Rotterdam, 1999.