Monotone Decision Trees and Noisy Data

Jan C. Bioch, Viara Popova

ERIM REPORT SERIES *RESEARCH IN MANAGEMENT*	
ERIM Report Series reference number	ERS-2002-53-LIS
Publication	June 2002
Number of pages	23
Email address corresponding author	bioch@few.eur.nl
Address	Erasmus Research Institute of Management (ERIM) Rotterdam School of Management / Faculteit Bedrijfskunde Erasmus Universiteit Rotterdam P.O. Box 1738 3000 DR Rotterdam, The Netherlands Phone: +31 10 408 1182 Fax: +31 10 408 9640 Email: info@erim.eur.nl Internet: www.erim.eur.nl

ERASMUS **R**ESEARCH **I**NSTITUTE OF **M**ANAGEMENT

REPORT SERIES

RESEARCH IN MANAGEMENT

BIBLIOGRAPHIC DATA AND CLASSIFICATIONS		
Abstract	The decision tree algorithm for monotone classification presented in [4, 10] requires strictly monotone data sets. This paper addresses the problem of noise due to violation of the monotonicity constraints and proposes a modification of the algorithm to handle noisy data. It also presents methods for controlling the size of the resulting trees while keeping the monotonicity property whether the data set is monotone or not.	
Library of Congress Classification (LCC)	5001-6182	Business
	5201-5982	Business Science
	HB 143	Mathematical Programming
Journal of Economic Literature (JEL)	M	Business Administration and Business Economics
	M 11 R 4	Production Management Transportation Systems
	C 6	Mathematical Methods and Programming
European Business Schools Library Group (EBSLG)	85 A	Business General
	260 K 240 B	Logistics Information Systems Management
	5 C	Logic
Gemeenschappelijke Onderwerpsontsluiting (GOO)		
Classification GOO	85.00	Bedrijfskunde, Organisatiekunde: algemeen
	85.34 85.20	Logistiek management Bestuurlijke informatie, informatieverzorging
	31.10	Logica
Keywords GOO	Bedrijfskunde / Bedrijfseconomie	
	Bedrijfsprocessen, logistiek, management informatiesystemen	
	monotone, monotonicity constraint, classification, classification tree, decision tree, ordinal data	
Free keywords	ordinal classification, monotone decision trees, noise, pruning	

Monotone Decision Trees and Noisy Data

Jan C. Bioch and Viara Popova
Dept. of Computer Science, Erasmus University Rotterdam,
P.O. Box 1738, 3000 DR Rotterdam.
email: {bioch,popova}@few.eur.nl

Abstract

The decision tree algorithm for monotone classification presented in [4, 10] requires strictly monotone data sets. This paper addresses the problem of noise due to violation of the monotonicity constraints and proposes a modification of the algorithm to handle noisy data. It also presents methods for controlling the size of the resulting trees while keeping the monotonicity property whether the data set is monotone or not.

Keywords: ordinal classification, monotone decision trees, noise, pruning

1 Introduction

Ordinal classification refers to the category of problems, in which the attributes of the objects to be classified are ordered. Ordinal classification has been studied by a number of authors, e.g. [1, 2, 3, 4, 7, 8, 9, 10] in the context of decision trees, decision lists, logical analysis of data, rough sets theory, etc. However a number of problems require further research in order to successfully apply ordinal classification in practice.

Noise in the data is a problem that often occurs in practical applications of the classification algorithms and is extensively studied by many authors. The traditional definition of noise considers data points which do not agree with the underlying function because of wrong classification, incorrect/imprecise measurements, typing mistakes, etc. Such points can mislead the classification algorithm and cause the generation of an overly complicated and/or inaccurate classifier.

The ordinal classification methods, however, might suffer from a specific type of noise that is not relevant for the general methods. The restriction of monotonicity of the data might be violated and data points can be inconsistent with each other, i.e. one point might dominate another on all attribute values but be classified in a lower class. This paper is an attempt to solve the problem in the context of monotone binary decision trees. It also addresses the problem of pruning the generated tree so that the monotonicity property is preserved. A number of approaches for that are presented.

2 Monotone decision trees

A classifier *class* is called monotone if each pair of data points (x,y) satisfies the constraint: $x \leq y \Rightarrow class(x) \leq class(y)$. The traditional decision tree algorithms such as C4.5 cannot guarantee the generation of a monotone classifier even when they are given a fully-monotone data set. An extension for dealing with ordinal data was proposed in [9] for 2-class problems. A more general approach applicable to k-class problems is proposed in [4, 10]. However, in this approach a fully monotone data set is required so that noise with respect to ordinality cannot be handled . In this paper we extend the method for dealing with noise.

The decision tree algorithms are characterized by three main rules: a splitting rule, a stopping rule and a labeling rule. The splitting rule defines how to split the current set of data points in two disjoint subsets - for this often the entropy criterion is used. The stopping rule defines when a subset cannot be split anymore and, whenever it fires, the labeling rule is checked which defines how to label the new leaf.

split(node T):
    update(T);
    if T is homogeneous
      label T;
    else
      split T into disjoint T_L and T_R;
      split(T_L);
      split(T_R);
	update (node T):
	    if $a \notin D$
	      $\lambda(a) = \lambda_{max}(a)$;
	      add a to D;
	    if $b \notin D$
	      $\lambda(b) = \lambda_{min}(b)$;
	      add b to D;

Figure 1: The monotone decision tree algorithm

The monotone decision tree (MDT) algorithm uses one more rule, the update rule, in order to preserve the monotonicity. It is executed for every node that we consider for splitting. As a stopping rule, the homogeneity of the node is checked. The whole algorithm can be represented by a procedure given in figure 1. Here D denotes the data set and T denotes the current node.

The update rule adds at most two new data points to D - the minimal and the maximal possible points in T, also called *corners*, labeled with respectively the maximal and minimal value allowed given D. More precisely, if $\mathcal{X}$ is the input space and T is defined as $T = \{x \in \mathcal{X} : a(T) \leq x \leq b(T)\}$ then $a(T)$ and $b(T)$ are considered for adding to the data set $D \subseteq \mathcal{X}$. If they are not present, then their labels are chosen to be the corresponding values of λ_{min} and λ_{max} which are defined in the following.

Let $\lambda(x)$ be the label of a data point $x \in D$, and c_{min} respectively c_{max} the minimal and the maximal possible label in D. The *downset* and the *upset* generated by x are defined as:

$$\downarrow x = \{y \in \mathcal{X} : y \leq x\}, \qquad \uparrow x = \{y \in \mathcal{X} : y \geq x\}$$

and the *downset* and the *upset* generated by D are defined as:

$$\downarrow D = \bigcup_{x \in D} \downarrow x, \qquad \uparrow D = \bigcup_{x \in D} \uparrow x$$

Then λ_{min} and λ_{max} are defined as follows:

$$\lambda_{\min}(x) = \begin{cases} \max\{\lambda(y) : \ y \in D \cap \downarrow x\} & \text{if } x \in \uparrow D \\ c_{min} & \text{otherwise} \end{cases} \tag{1}$$

$$\lambda_{\max}(x) = \begin{cases} \min\{\lambda(y) : \ y \in D \cap \uparrow x\} & \text{if } x \in \downarrow D \\ c_{max} & \text{otherwise .} \end{cases} \tag{2}$$

As a running example we use the monotone data set, given in table 1, which consists of 15 data points described by 6 condition attributes ($a1$ to $a6$) and one decision/class attribute (λ). Figure 2 shows the monotone decision tree generated by the algorithm from this data set.

Note, that a simple criterion for checking the monotonicity of a tree (see [4, 10]) can be defined as follows. Let $\mathcal{L}$ be the set of leaves of a tree $\mathcal{T}$ and $\mathcal{N}$ be the set of nodes of $\mathcal{T}$. We define a relation on $\mathcal{N}$ - for $T, T' \in \mathcal{N}$:

$$T \leq T' \Leftrightarrow a(T) \leq b(T').$$

Let $T, T' \in \mathcal{L}$ with labels $\lambda(T)$, $\lambda(T')$ where $T = \{x \in \mathcal{X} : a(T) \leq x \leq b(T)\}$ and $T' = \{x \in \mathcal{X} : a(T') \leq x \leq b(T')\}$ Then the tree is monotone if for any choice of T and T':

$$T \leq T' \Rightarrow \lambda(T) \leq \lambda(T').$$

x	$a1$	$a2$	$a3$	$a4$	$a5$	$a6$	λ
1	0	0	1	1	0	1	0
2	1	1	2	1	3	1	0
3	0	1	1	0	0	1	0
4	2	3	1	3	3	1	1
5	1	0	2	2	3	1	1
6	0	0	0	3	2	2	1
7	2	2	1	1	1	2	1
8	2	4	2	2	2	3	2
9	1	1	2	1	3	2	2
10	3	2	1	0	0	1	2
11	3	2	2	1	2	2	3
12	3	3	4	1	2	2	3
13	4	2	3	3	3	3	3
14	3	3	3	4	1	3	3
15	4	4	2	3	0	1	3

Table 1: The example data set

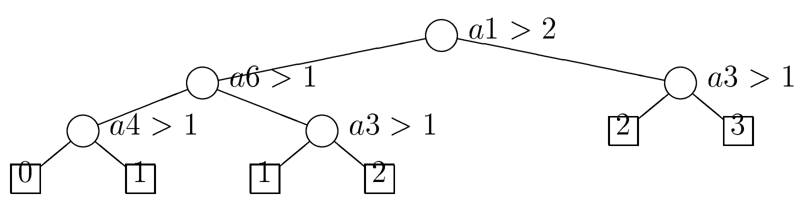

Figure 2: The MDT generated for the example data set

3 Monotone decision trees from noisy data

When monotonicity noise occurs in the data it appears as pairs of data points that are inconsistent with respect to monotonicity. For the MDT algorithm this results in tree nodes for which the lower left corner is assigned a higher label than the upper right corner. More precisely, let $T = \{x \in \mathcal{X} : a(T) \leq x \leq b(T)\}$ be the set (node) considered for splitting and let $\lambda(a)$ and $\lambda(b)$ be the labels of $a(T)$ and $b(T)$ respectively. Then it might occur that $\lambda(a) > \lambda(b)$.

In order to solve the problem we propose a simple extension of the update rule that not only grows the data set but also tries to repair the inconsistencies. The new update rule is given in figure 3. The procedure always relabels the corners with the consistent labels that are calculated from the rest of the data. This algorithm always generates a monotone tree.

```
update D for T:
    l1 = λmax(a); l2 = λmin(b);
    if a ∈ D
        relabel a: λ(a) = l1;
    else
        label a: λ(a) = l1;
        add a to D;
    if b ∈ D
        relabel b: λ(b) = l2;
    else
        label b: λ(b) = l2;
        add b to D;
```

Figure 3: The new update rule

Theorem 1 *The MDT algorithm of figure 1 with the update rule of figure 3 always generates a monotone tree.*

Proof. Let us assume that the generated tree is not monotone:

$$\exists T, T' \in \mathcal{L} : T \leq T' \text{ and } \lambda(T) > \lambda(T').$$

By assumption T and T' are homogeneous. Therefore $\lambda(T) = \lambda(a(T)) = \lambda(b(T))$ and $\lambda(T') = \lambda(a(T')) = \lambda(b(T'))$. This implies

$$\lambda(a(T)) > \lambda(b(T').) \tag{3}$$

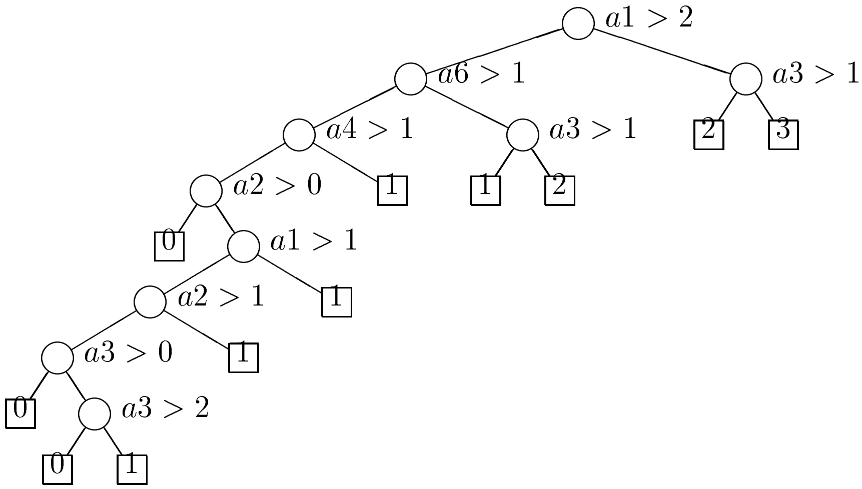

Figure 4: MDT on the non-monotone data set

The labels of the leaves are assigned as follows:

$$\text{at a moment } t \text{ we assign } \lambda(a(T)) = \lambda_{max}(a)$$

$$\text{at a moment } t' \text{ we assign } \lambda(b(T')) = \lambda_{min}(b).$$

Let $t < t'$. Since $T \leq T'$ then $a(T) \in \downarrow b(T') \cap D \neq \emptyset$

$$\Rightarrow \lambda_{min}(b(T')) \geq \lambda(a(T))$$

$$\Rightarrow \lambda(a(T)) \leq \lambda(b(T')),$$

which is a contradiction with condition (3). The case $t' < t$ is analogous. Ξ

An interesting observation is that after a leaf is created all the points belonging to the leaf except the corners can be deleted from the data set since they will not be used further in the tree generation. This remains true also for the algorithms presented in the rest of the paper.

To illustrate the algorithm we introduce monotone inconsistency in the example data set of table 1 - we change the label of data point $x3$ from 0 to 1. Thus we introduce an inconsistent pair of data points ($x2$,$x3$). The output of the algorithm on the new data set is given in figure 4.

In our implementation we use depth-first strategy for generating the tree. The same strategy is used in the algorithms presented in section 4.

4 Pruning a monotone tree

As it was noted before, the update rule of the MDT algorithm tries to add new points to the data set. Thus the number of points to be split grows and that in general causes the generation of bigger trees. When noise is present in the data set this creates difficulties for the classification algorithms, i.e. by creating areas in the data that are difficult to describe, and thus also causes (sometimes substantial) increase in the size of the generated tree. The same effect is present for the special case of monotonicity noise. This can be illustrated by the following example. We introduce inconsistency in the data set from table 1 by changing the label of $x8$ from 2 to 0 and that results in one pair of inconsistent points ($x7$,$x8$). The monotone tree generated by the algorithm has 148 leaves and 288 data points in the updated data set.

Therefore we need methods for pruning the monotone tree in such a way that we keep the monotonicity property of the tree and do not increase the misclassification rate more than a predefined threshold. While the area of decision tree pruning attracts a lot of attention and a number of methods are developed (see [6] for an extensive survey), these methods do not take into account the monotonicity property and cannot guarantee that the pruned tree will still be monotone.

This paper proposes a number of methods for pruning within two main approaches - pre-pruning and post-pruning. Pre-pruning is a general approach for pruning while generating the tree by not growing branches which fail to satisfy a predefined criterion and turning them to leaves. Therefore pre-pruning modifies the stopping and the labeling rule of the algorithm. Post-pruning on the other hand first grows the full tree and then tries to cut branches from it while a predefined criterion is satisfied. It is therefore a post-processing step which requires two separate rules - for choosing a branch to cut and for labeling the new leaves.

Post-pruning requires the full tree to be generated which if the tree is very large takes a lot of resources for generating and storing the tree as well as the updated data set. Pre-pruning stops the generation of the tree prematurely and therefore takes less resources since the tree and the updated data set remain smaller. It is however more difficult in general to decide when to stop and what label to assign to the leaf since not much information about the tree is available yet.

4.1 Pre-pruning

One criterion often used in traditional pruning techniques for prematurely stopping the generation of a branch is a predefined threshold for the minimal number of points in a leaf. Splitting is not allowed if the number of points in any of the new leaves drops below the threshold, the current node is turned to a leaf and assigned an appropriate label. Different methods are used to choose a good label for the new leaf - one method that often works well in practice is to assign the label of the majority class among the points in the leaf. The traditional methods however do not guarantee the monotonicity property of the resulting tree.

As mentioned before, we use the depth-first strategy for the tree generation. First we note an observation that holds for this strategy.

Lemma 1 *Let $T, T' \in \mathcal{L}$ in the monotone tree $\mathcal{T}$ generated with the depth-first strategy. Let $T \leq T'$. Then leaf T is generated before leaf T'.*

Proof. Let N be a node in $\mathcal{T}$ such that N is the least common ancestor of T and T'. Therefore $\exists i$ such that exactly one of the following is true:

$$\forall x \in T, \forall y \in T' : x(i) \leq y(i) \tag{4}$$

$$\forall x \in T, \forall y \in T' : x(i) > y(i) \tag{5}$$

Condition 5 contradicts the requirement $T \leq T'$. Therefore condition (4) is true and T belongs to the left branch of N while T' belongs to the right branch of N. Therefore using the depth-first strategy T will be generated before T'. Ξ

Using this result, we propose, *in the case of the depth first strategy* the labeling rule given in figure 5 for a newly generated leaf. The resulting tree remains monotone.

Theorem 2 *Let $\mathcal{T}$ be a tree generated with the depth first strategy using the update rule from figure 3 and the labeling rule of figure 5 with a threshold of at least m points in a leaf, $m > 1$. Then $\mathcal{T}$ is monotone.*

label leaf T:

$\lambda(T) = \lambda(b(T))$;

$\lambda(a(T)) = \lambda(b(T))$;

Figure 5: The new labeling rule

Proof. Let $\exists T, T' \in \mathcal{L}$ such that $T \leq T'$ and $\lambda(T) > \lambda(T')$. According to lemma (1), T should be generated before T'.

The update rule guarantees that $\lambda(b(T')) \geq \lambda(a(T))$. The labeling rule guarantees that $\lambda(T) = \lambda(b(T)) = \lambda(a(T))$

$$\Rightarrow \lambda(T') = \lambda(b(T')) \geq \lambda(T)$$

which is a contradiction with the assumption. Ξ

To illustrate the algorithm we use the example from table 1 with the change described in section 4. Figure 6 shows the tree generated using pre-pruning with a threshold of at least 4 points in a leaf. The tree misclassifies 2 points from the original data set. The new algorithm is an extension of the algorithm proposed in [4, 10] in the sense that it generates the same tree if the data set is fully monotone data set. Moreover, the pruning algorithm can also be used with the traditional MDT algorithm in order to reduce the size of the generated tree.

In some cases both children-leaves of a node might be assigned the same label. In that case the node can be pruned without further increase in the misclassification rate.

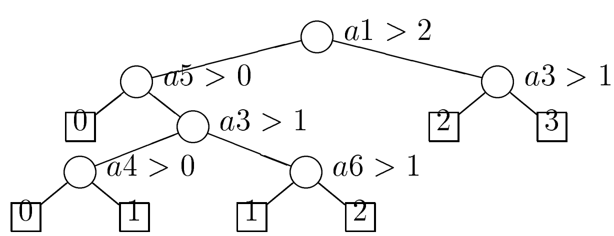

Figure 6: MDT generated with pre-pruning

4.2 Post-pruning

The general approach of post-pruning the already generated tree defines two additional rules - for choosing a branch to prune and for choosing a label for the new leaf. First we address the second problem taking into account the monotonicity property of the tree.

Let $\mathcal{T}$ be a monotone decision tree. For a node $T = \{x \in \mathcal{X} : a(T) \leq x \leq b(T)\}$ we define a *consistency interval* $L(T)$ where:

$$L(T) = [l_{min}(T), l_{max}(T)]$$

$$l_{min}(T) = max\{\lambda(T') : T' \in \mathcal{L}, T' \leq T\}$$

$$l_{max}(T) = min\{\lambda(T') : T' \in \mathcal{L}, T' \geq T\}.$$

If $L(T) \neq \emptyset$ then any value in $L(T)$ is a possible consistent label for T preserving the monotonicity property of the tree.

Theorem 3 *Given a monotone tree $\mathcal{T}$ and an arbitrary node T of $\mathcal{T}$. Suppose that the children of T are pruned and that T is turned to a leaf. Let $L(T) \neq \emptyset$. Then for any $l \in L(T)$, l can be assigned as a label of T and the resulting tree remains monotone.*

Proof. Let us assume that the new tree is not monotone. Then there exists $T' \in \mathcal{L}$ such that one of the following occurs:

$$T' \leq T \text{ and } \lambda(T') > \lambda(T) \text{ or} \tag{6}$$

$$T \leq T' \text{ and } \lambda(T) > \lambda(T') \tag{7}$$

.

Let condition (6) be the case.

Since $T' \leq T$ we have $\lambda(T') \leq l_{min}(T)$. But $\lambda(T) = l \geq l_{min}(T)$. Therefore:

$$\lambda(T') \leq l_{min} \leq \lambda(T) < \lambda(T')$$

which is a contradiction.

The case of condition (7) is analogous. Ξ

When the consistency interval contains only one point $l = l_{min} = l_{max}$, then there is only one possibility for a consistent label of the pruned node. However, if $l_{min} < l_{max}$, then a choice has to be made which point from the interval to assign. This choice is often domain dependent and reflects i.e. how optimistic or pessimistic the prediction is required to be.

The second open question with monotone pruning is the choice of a node to prune. It includes the order of visiting the nodes and the criterion for approval or rejection of the current node for pruning. We consider two search strategies for visiting the nodes which are shown in figures 7 and 8. The first follows the depth-first order of visiting the nodes and tries to prune the current node if both its children are leaves. The second strategy iteratively tries to prune the frontier of the tree in depth-first order. On each iteration it tries to prune all nodes whose both children are leaves none of which has just be pruned. The loop terminates when the tree is traversed without pruning any node. Our experiments point out that the second strategy produces more balanced trees while the size of the trees is comparable to the size of the trees produced by the first strategy.

```
search-tree(Tree-root);
———
search-tree(node T):
  if (leaf(T.left-child) && leaf(T.right-child))
    if (good-for-pruning(T))
      prune(T);
  else
    if (! leaf(T.left-child)
      search-tree(T.left-child);
    search-tree(T.right-child);
```

Figure 7: Depth-first strategy for choosing candidates for pruning

Once a candidate for pruning is reached it has to be decided whether to prune it or not. One logical criterion is the misclassification rate. The algorithm computes the new label and then checks whether the misclassification rate of the tree with the new leaf is below a predefined threshold for the percentage of misclassified data points. It is a general approach to use a separate pruning set for checking the accuracy of the tree.

To illustrate the post-pruning algorithm we use the same example. The full tree contains 148 leaves. Figure 9 shows the pruned tree at misclassification threshold 25% and assigning label l_{max}. For simplicity we don't use a separate pruning set but check the

```
search-tree():
  do:
      pruned=0;
      pruning-iter(Tree-root);
  while(pruned);
-----
pruning-iter(node T):
  if (! leaf(T.left-child)
     pruning-iter(T.left-child);
  if (! leaf(T.right-child))
     pruning-iter(T.right-child);
  if ((both-children-leaves(T)) &&(! child-just-pruned(T)))
     if (good-for-pruning(T))
        prune(T);
        pruned++;
```

Figure 8: Frontier strategy for for choosing candidates for pruning

misclassification on the original data set. The pruned tree misclassifies 3 points from the original data set. Figure 10 shows the tree pruned at threshold 30% and 4 misclassified points.

Again as with pre-pruning it might happen that both children of a node are assigned the same label - then again we can prune the node without increasing the misclassification rate.

Figure 11 illustrates the same algorithm with choosing l_{min} as the label of the new leaf. The tree is pruned at misclassification threshold 25% and 3 misclassified points. Figure 12 shows the tree at threshold 30% and 4 misclassified points.

The post-pruning algorithm can be used separately from the rest of the algorithms presented in the paper. It can be applied as a post-processing step on any monotone tree generated with another algorithm as soon as the information about the leaf corners is available. It can also be used on a monotone tree generated with the pre-pruning algorithm for further simplification of the tree.

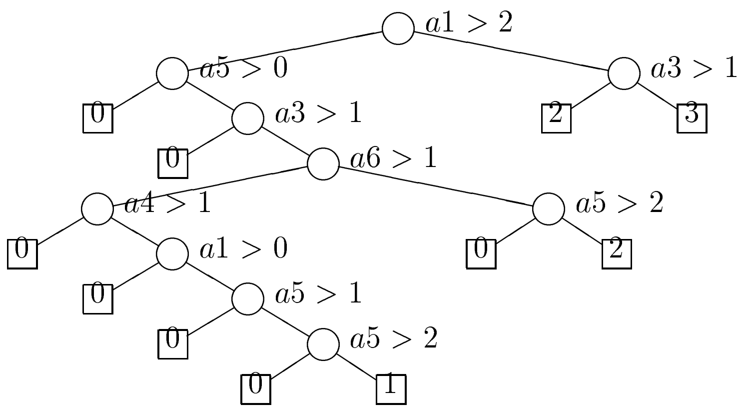

Figure 9: MDT generated with post-pruning

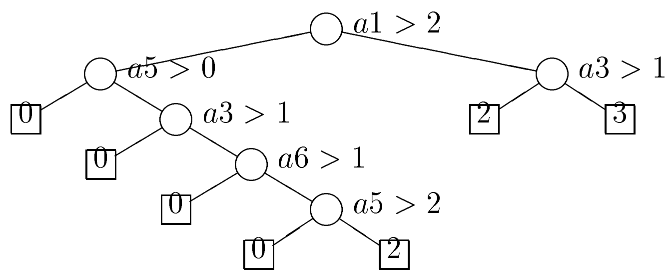

Figure 10: MDT generated with post-pruning

5 Experiments

In order to compare and study the specifics of the algorithms presented in the paper, experiments were conducted using three data sets - one artificial and two real-world data sets. The original data for all of them is monotone. Further, some monotonicity noise is introduced in the following way: among all pairs of comparable data points, one pair is selected and, if the labels differ, they are switched. This results in one or more non-monotone(inconsistent) pairs. The same procedure can be performed on the new data set. For each of the original data sets, 3 noisy sets are generated by switching the labels of respectively 1, 2 and 3 pairs. The new data sets are used to build the full MDT, the pre-pruned MDTs with varied threshold of 2 to 5 points in a node and the post-pruned trees with varied misclassification rate threshold of 5% to 20%. Tables 2 to 4 represent

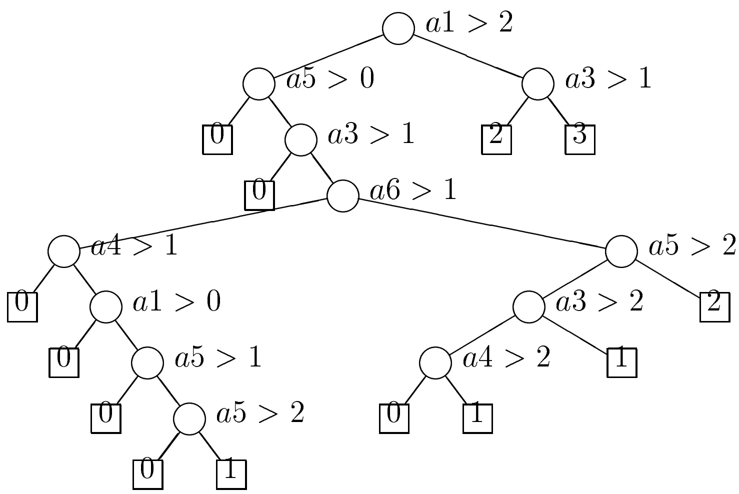

Figure 11: MDT generated with post-pruning

the results by the following indicators: number of points in the updated data set, number of nodes in the tree, number of leaves, average depth the tree, maximal depth, number of misclassified points on the original (not updated) data, on the updated data (without the newly added points) and on the separate test set.

The artificial data set is generated in the following way. First a monotone model is assumed to be the underlying model. A set of random data points is generated and the points are classified according to the model. The resulting set is monotone with 15 comparable pairs of data points. The size of the data is 50 points described by 10 attributes taking values from 0 to 5 and a decision attribute taking values from 0 to 2. Using the same procedure a separate test set of the same size is generated. The features of the generated MDT are given in table 2, column 2. Further 3 noisy data sets are generated by the above described procedure resulting in 2, 4 and 5 inconsistent pairs of points. Their features are given in the rest of tables 2.

The second data set used in the experiments is discussed in [7, 11]. The sample consists of 39 objects representing firms that are described by 12 financial parameters. To each company a decision value is assigned - the expert evaluation of its category of risk. The condition attributes take integer values from 0 to 4 and the decision attribute is in the range of 0 to 2 where: 0 means *unacceptable*, 1 means *uncertainty* and 2 means *acceptable*.

The problem is monotone - if one company outperforms another on all condition attributes then it should not have a lower value of the decision attribute, nevertheless,

	full-m	full	pre2	pre3	pre4	pre5	po5	po6	po7	po8	po9	po10	po15	po20
updated	114	637	208	110	106	104	637	637	637	637	637	637	637	637
num-nodes	63	587	163	59	55	53	49	49	47	47	41	41	13	35
num-leaves	32	249	82	30	28	27	25	25	24	24	21	21	7	18
av-depth	6	25	12	9	8	8	6	6	6	6	6	6	3	6
max-depth	9	35	26	19	18	18	12	12	12	12	12	12	4	12
miscl-org	0	1	5	7	7	7	2	2	3	3	4	4	7	9
miscl-upd	0	0	0	7	7	7	1	1	2	2	3	3	6	8
miscl-test	10	12	15	17	17	17	12	12	14	14	16	16	18	21
updated		9644	209	113	109	1000	9644	9644	9644	9644	9644	9644	9644	9644
num-nodes		9597	167	63	59	49	9597	9597	47	47	47	47	37	21
num-leaves		4799	84	32	30	25	4799	4799	24	24	24	24	19	11
av-depth		22	11	9	8	7	22	22	6	6	6	6	6	4
max-depth		35	29	20	19	15	35	35	12	12	12	12	12	8
miscl-org		3	7	8	8	8	3	3	3	3	4	4	7	9
miscl-upd		0	0	7	7	8	0	0	2	2	3	3	6	8
miscl-test		15	12	11	11	12	15	15	16	16	17	17	16	14
updated		9680	245	148	113	104	9680	9680	9680	9680	9680	9680	9680	9680
num-nodes		9635	205	99	63	53	9635	9635	9635	9635	51	51	43	31
num-leaves		4818	103	50	32	27	4818	4818	4818	4818	26	26	22	16
av-depth		22	12	11	8	7	22	22	22	22	6	6	6	4
max-depth		35	29	22	19	15	35	35	35	35	12	12	12	8
miscl-org		4	8	9	9	9	4	4	4	4	4	4	7	9
miscl-upd		0	0	8	8	9	0	0	0	0	2	2	5	7
miscl-test		15	12	11	12	13	15	15	15	15	16	16	19	15

Table 2: Experimental results for the artificial data set

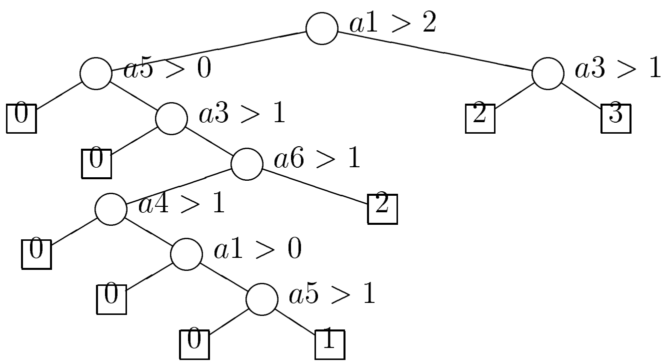

Figure 12: MDT generated with post-pruning

one noisy/inconsistent pair is present. By deleting one of the inconsistent points we get a monotone data set. The tree generated from it is described in table 3, column 2. The rest of table 3 is based on the original sample having 1 inconsistent pair (out of 199 comparable pairs) and 2 more data sets generated by adding noise with resp. 3 and 6 inconsistent pairs of points.

Since the original data set is very small, no separate test data is used. The last data set took too much time to generate the full tree because of exponential growing of the updated data set. Therefore the data on the full and the post-pruned trees is not available. However, the pre-pruning algorithm generates manageable trees (even for a threshold of 2 points) which are represented in the table.

The third data set was obtained from UCI Machine Learning Repository [5]. It represents applications for a nursery school which are classified based on their situation in 5 groups ranging from *not recommended* to *special priority*. The problem is monotone since the objective is to give more priority to children with worse situation on every indicator. The size of the data set is 8 attributes taking between 2 and 5 ordered values, one decision attribute and 12960 data points covering the whole input space.

For the experiments a random sample of 200 points was drawn and 3 noisy data sets were constructed having respectively 2, 6, and 27 inconsistent pairs of points. The features of the generated trees are presented in table 4. A separate random sample of the same size is used as a test set. From the experimental results the following observations can be deduced. The relation between the number of inconsistent pairs of data points and the size of the tree is not straightforward - more important is the type of inconsistency that can confuse the tree generation. When the noise disturbs severely the tree generation it is possible for the data set and the tree size to grow exponentially. This is a known result

	full-m	full	pre2	pre3	pre4	pre5	po5	po6	po7	po8	po9	po10	po15	po20
updated	50	91	91	90	75	73	91	91	91	91	91	91	91	91
num-nodes	11	53	53	51	35	33	11	9	9	7	7	7	7	7
num-leaves	6	27	27	26	18	17	6	5	5	4	4	4	4	4
av-depth	2	13	13	12	8	8	2	2	2	2	2	2	2	2
max-depth	4	25	25	24	16	15	4	3	3	2	2	2	2	2
miscl-org	0	1	1	1	1	2	1	2	2	3	3	3	3	3
miscl-upd	0	0	0	1	1	2	0	1	1	2	2	2	2	2
updated		123	123	121	104	83	123	123	123	123	123	123	123	123
num-nodes		87	87	83	65	43	87	45	45	21	21	21	17	17
num-leaves		44	44	42	33	22	44	23	23	11	11	11	9	9
av-depth		12	12	12	9	7	12	9	9	4	4	4	4	3
max-depth		25	25	24	17	15	25	19	19	7	7	7	7	6
miscl-org		2	2	2	2	3	2	2	2	3	3	3	5	7
miscl-upd		0	0	1	1	3	0	0	0	3	3	3	5	7
updated		*	195	180	121	100	*	*	*	*	*	*	*	*
num-nodes		*	163	145	83	61	*	*	*	*	*	*	*	*
num-leaves		*	82	73	42	31	*	*	*	*	*	*	*	*
av-depth		*	14	14	9	8	*	*	*	*	*	*	*	*
max-depth		*	31	30	16	15	*	*	*	*	*	*	*	*
miscl-org		*	4	5	5	6	*	*	*	*	*	*	*	*
miscl-upd		*	0	4	4	6	*	*	*	*	*	*	*	*

Table 3: Experimental data for the bankruptcy data set

	full-m	full	pre2	pre3	pre4	pre5	po5	po6	po7	po8	po9	po10	po15	po20
updated	482	598	459	353	311	283	598	598	598	598	598	598	598	598
num-nodes	321	471	297	161	111	83	143	131	75	27	27	73	25	17
num-leaves	161	236	149	81	56	42	72	66	38	14	14	37	13	9
av-depth	10	10	10	8	7	7	9	9	7	4	4	7	4	4
max-depth	16	15	14	13	12	12	14	14	11	7	7	11	6	6
miscl-org	0	1	35	38	39	40	9	11	13	15	15	19	29	36
miscl-upd	0	0	0	30	38	39	8	10	12	14	14	18	28	36
miscl-test	17	16	41	44	45	44	24	26	27	30	30	31	36	47
updated		592	410	341	295	275	592	592	592	592	592	592	592	592
num-nodes		493	239	149	95	75	35	41	31	55	39	29	37	21
num-leaves		247	120	75	48	38	18	21	16	28	20	15	19	11
av-depth		11	9	8	7	6	5	5	5	6	5	5	5	5
max-depth		16	13	11	10	10	8	7	7	10	7	7	8	8
miscl-org		1	27	34	36	39	9	11	13	15	17	19	29	37
miscl-upd		0	0	27	35	38	8	10	12	14	16	18	28	37
miscl-test		26	38	43	43	44	24	28	34	32	32	38	46	48
updated		2795	576	440	377	328	2795	2795	2795	2795	2795	2795	2795	2795
num-nodes		3449	419	255	179	129	3449	3449	3449	3449	3449	3449	45	13
num-leaves		1725	210	128	90	65	1725	1725	1725	1725	1725	1725	23	7
av-depth		14	10	9	8	7	14	14	14	14	14	14	6	3
max-depth		18	16	15	13	13	18	18	18	18	18	18	10	5
miscl-org		24	16	28	41	47	24	24	24	24	24	24	28	37
miscl-upd		0	0	20	39	45	0	0	0	0	0	0	37	39
miscl-test		40	29	37	43	52	40	40	40	40	40	40	39	37

Table 4: Experimental data for the nursery data set

for the original MDT algorithm for monotone data. The problem occurs more often with noisy data since it is originally inconsistent and can more easily confuse the generation process. However, using pre-pruning the problem can be easily overcome and manageable trees can be generated from any noisy data set. It can also be used together with the original algorithm on monotone data.

Pre-pruning generates smaller data sets and therefore consumes less resources than growing the whole tree and pruning it afterwards. On the other hand, for some of the data sets post-pruning seems to produce better results by pruning a large part of the tree with no change in the misclassification rate. As it was expected, for several data sets the results generated with pre- or post-pruning using smaller thresholds improves the accuracy of the tree by giving lower misclassification rate than the full tree.

6 Conclusions and further research

This paper presents a method for generating MDTs from noisy data by modifying the update rule. One possible direction for further research is to study the effect of noise on the choice of a good attribute for splitting. That might result in a modification of the criterion to ignore the noisy points and base the splitting decision only on the monotone data.

The paper also presents methods for controlling the size of the trees by means of pre- and post-pruning while the tree is guaranteed to remain monotone. These methods can be applied both with the original MDT algorithm and with the modified algorithm for generating MDTs from noisy data.

References

[1] Ben-David, A.: Monotonicity Maintenance in Information-Theoretic Machine Learning Algorithms. Machine Learning 19 (1995) 29–43

[2] Bioch, J.C.: Dualization, Decision Lists and Identification of Monotone Discrete Functions. Annals of Mathematics and Artificial Intelligence 24 (1998) 69-91

[3] Bioch, J.C., Popova, V.: Rough Sets and Ordinal Classification. Lecture Notes in Artificial Intelligence 1968, (2000) 291–305 Springer-Verlag.

[4] Bioch, J.C., Potharst, R.: Decision Trees for Monotone Classification. In: K. van Marcke and W. Daelmans (eds), Proceedings of the Dutch Artificial Conference on Artificial Intelligence (NAIC'97), Antwerp (1997) 361–369

[5] Blake, C.L., Merz, C.J.: UCI Repository of machine learning databases [http://www.ics.uci.edu/ mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science (1998).

[6] Breslow, L., Aha, D.W.: Simplifying Decision Trees: A Survey. Knowledge Engineering Review 12 (1997) 1–40. http://citeseer.nj.nec.com/14866.html

[7] Greco, S., Matarazzo, B., Slowinski, R.: A New Rough Set Approach to Evaluation of Bankruptcy Risk. in: C. Zopounidis (ed.), Operational Tools in the Management of Financial Risks, Kluwer, Dordrecht (1998) 121–136

[8] Crama, Y., Hammer, P.L., Ibaraki, T.: Cause-Effect Relationships and Partially Defined Boolean Functions. Annals of Operations Research 16 (1988) 299–326

[9] Makino, K., Suda, T., Yano, K., Ibaraki, T.: Data Analysis by Positive Decision Trees. In: Proceedings International symposium on cooperative database systems for advanced applications (CODAS), Kyoto (1996) 282–289

[10] Potharst, R., Bioch, J.C.: Decision Trees for Ordinal Classification. Intelligent Data Analysis 4 (2000) 1–15

[11] Slowinski, R., Zopounidis, C.: Application of the Rough Set Approach to the Evaluation of Bankruptcy Prediction. Intelligent Systems in Accounting, Finance and Management 4 (1995) 27–41

Publications in the Report Series Research* in Management

ERIM Research Program: "Business Processes, Logistics and Information Systems"

2002

The importance of sociality for understanding knowledge sharing processes in organizational contexts
Niels-Ingvar Boer, Peter J. van Baalen & Kuldeep Kumar
ERS-2002-05-LIS

Crew Rostering for the High Speed Train
Ramon M. Lentink, Michiel A. Odijk & Erwin van Rijn
ERS-2002-07-LIS

Equivalent Results in Minimax Theory
J.B.G. Frenk, G. Kassay & J. Kolumbán
ERS-2002-08-LIS

An Introduction to Paradigm
Saskia C. van der Made-Potuijt & Arie de Bruin
ERS-2002-09-LIS

Airline Revenue Management: An Overview of OR Techniques 1982-2001
Kevin Pak & Nanda Piersma
ERS-2002-12-LIS

Quick Response Practices at the Warehouse of Ankor
R. Dekker, M.B.M. de Koster, H. Van Kalleveen & K.J. Roodbergen
ERS-2002-19-LIS

Harnessing Intellectual Resources in a Collaborative Context to create value
Sajda Qureshi, Vlatka Hlupic, Gert-Jan de Vreede, Robert O. Briggs & Jay Nunamaker
ERS-2002-28-LIS

Version Spaces and Generalized Monotone Boolean Functions
Jan C. Bioch & Toshihide Ibaraki
ERS-2002-34-LIS

Periodic Review, Push Inventory Policies for Remanufacturing
B. Mahadevan, David F. Pyke, Moritz Fleischman
ERS-2002-35-LIS

Modular Decomposition of Boolean Functions
Jan C. Bioch
ERS-2002-37-LIS

Classification Trees for Problems with Monotonicity Constraints
R. Potharst & A.J. Feelders
ERS-2002-45-LIS

---

* A complete overview of the ERIM Report Series Research in Management:
http://www.ers.erim.eur.nl

ERIM Research Programs:
LIS Business Processes, Logistics and Information Systems
ORG Organizing for Performance
MKT Marketing
F&A Finance and Accounting
STR Strategy and Entrepreneurship

Allocation of Railway Rolling Stock for Passenger Trains
Erwin Abbink, Bianca van den Berg, Leo Kroon & Marc Salomon
ERS-2002-47-LIS

Monotone Decision Trees and Noisy Data
Jan C. Bioch and Viara Popova
ERS-2002-53-LIS