Generating Artificial Data with Monotonicity Constraints

Rob Potharst and Michiel van Wezel
Erasmus University, P.O.Box 1738, 3000 DR Rotterdam

Econometric Institute Report EI 2005-06

Abstract

The monotonicity constraint is a common side condition imposed on modeling problems as diverse as hedonic pricing, personnel selection and credit rating. Experience tells us that it is not trivial to generate artificial data for supervised learning problems when the monotonicity constraint holds. Two algorithms are presented in this paper for such learning problems. The first one can be used to generate random monotone data sets without an underlying model, and the second can be used to generate monotone decision tree models. If needed, noise can be added to the generated data. The second algorithm makes use of the first one. Both algorithms are illustrated with an example.

1 Introduction

When developing a new or revising an existing method for solving a data modeling problem, one usually tests this method extensively not only on real life data, but also on artificial data. One reason for the inclusion of artificial data in such experiments is that the parameters of the models underlying the data can be much better controlled for artificial data when compared with real data. For instance, if we want to test the robustness of a newly devised data modeling method against varying degrees of noise in the data, it is generally much easier to generate artificial data than to search for real data sets with varying degrees of noise. In many cases, artificial data sets can be constructed without much effort using simulation techniques [21], for instance by first generating random values for the parameters of our model, next generating random input values, calculating the corresponding outputs and adding random noise to the output values. However, artificial

data generation is not always that easy. An example is the construction of data sets for concept learning problems [13]. The construction of data sets for monotone learning problems is another example. The monotonicity constraint on a model or data set has recently been studied by a number of authors [2, 17, 6, 11] and will be explained in Section 2. Application areas of monotone classification and regression include pricing and various selection problems [2, 17]. While working on algorithms for monotone classification and regression problems, we found that it is not a trivial task to come up with a monotone dataset, or a monotone classification or regression model. This has to do with the great number of constraints on the data values that is implied by the monotonicity constraint. In [18], an algorithm for constructing random monotone datasets is proposed, which has never been published in a regular medium. This algorithm is outlined in Section 3 of this paper. In Section 4 we present a new method to construct monotone classification or regression tree models, using the method of Section 3 and a special order relation on the leaves of a tree introduced in [7]. This method is illustrated with an example in Section 5. The last section contains the summary of this paper.

2 The concept of monotonicity

Even though data mining is often applied to domains where little theory is available, in many cases it is either known that the target function satisfies certain constraints, or it is simply required that the model constructed satisfies those constraints.

One type of constraint that is available in many applications states that the dependent variable (or its expected value) should be a monotone function of the independent variables. Economic theory would state for example that people tend to buy less of a product if its price increases (ceteris paribus), so price elasticity of demand should be negative. The strength of this relationship and the precise functional form are however usually not dictated by economic theory. Other well-known examples are labor wages as a function of age and education (see e.g. [15]) or so-called hedonic price models where the price of a consumer good depends on a bundle of characteristics for which a valuation exists [12].

Another class of problems where monotonicity constraints often apply are so-called selection problems, such as the selection of applicants for a job or a loan on the basis of their characteristics. As an example, consider a selection procedure for applicants to a job based on the outcomes of a

series of academic and/or psychological tests. If each of the test outcomes x_i is scored from low (bad performance) to high (good performance) and the classes are taken to be 0 = not selected and 1 = selected, then it would be very natural to demand the selection rule to be monotone. In fact, the requirement of monotonicity would be equivalent to excluding all situations in which applicant A scores better or at least as good on all tests as applicant B, whereas B gets selected and A does not.

Because the monotonicity constraint is quite common in practice, many data analysis techniques have been adapted to be able to handle such constraints.

Isotonic regression, for example, deals with regression problems with monotonicity constraints. The traditional method used in isotonic regression is the so-called pool-adjacent violators algorithm [20]. This method however only works in the one-dimensional case. A versatile non-parametric method is given in [15], see also [19].

Monotonicity constraints have also been investigated in the neural network literature. In [22] the monotonicity of the neural network is guaranteed by enforcing constraints on the weights during the training process. Daniels and Kamp [10] present a class of neural networks that are monotone by construction. This class is obtained by considering multilayer neural networks with non-negative weights.

Various methods have also been proposed for classification problems with monotonicity constraints[1], such as decision or classification trees [2, 14, 16, 17, 11, 9], decision lists [4], logical analysis of data [5], rough sets [8] and instance-based learning [3, 1].

We will now define the concept of monotonicity somewhat more formally. Let $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_p$ be an instance space with p attributes, $\mathcal{Y}$ a set of target values. (In a classification problem, $\mathcal{Y}$ is a finite set of classes, in a regression problem it is an interval of real numbers.) We suppose that each coordinate space $\mathcal{X}_i$ and the target space $\mathcal{Y}$ has an ordering $\leq$. This means that each attribute, including the target, has at least an ordinal scale. Then the instance space $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_p$ has a partial ordering $\leq$, defined by $\mathbf{x} = (x_1, x_2, \ldots, x_p) \leq \mathbf{x}' = (x'_1, x'_2, \ldots, x'_p)$ if and only if $x_i \leq x'_i$ for all i. Now, a model

$$f : \mathcal{X} \to \mathcal{Y}$$

will be called *monotone*, if

$$\mathbf{x} \leq \mathbf{x}' \Rightarrow f(\mathbf{x}) \leq f(\mathbf{x}') \tag{1}$$

[1] In [9] these are called supervised ranking problems.

for all instances $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$. A data set $\mathcal{D}$ is a set of examples $(\mathbf{x}, y)$ from $\mathcal{X} \times \mathcal{Y}$. A dataset is called *monotone* if for all pairs $(\mathbf{x}, y)$ and $(\mathbf{x}', y')$ from $\mathcal{D}$ we have

$$\mathbf{x} \leq \mathbf{x}' \Rightarrow y \leq y'. \tag{2}$$

A data set $\mathcal{D}$ is *consistent* if $\mathbf{x} = \mathbf{x}' \Rightarrow y = y'$ whenever both $(\mathbf{x}, y)$ and $(\mathbf{x}', y')$ are examples from $\mathcal{D}$. It is easy to see that a monotone data set is automatically consistent.

Table 1: The bank loan dataset

client	*income*	*education*	*crim.record*	*loan*
cl1	low	low	fair	no
cl2	low	low	excellent	low
cl3	average	intermediate	excellent	intermediate
cl4	high	low	excellent	intermediate
cl5	high	intermediate	excellent	high

We will now give an example of a monotone classification problem. Suppose a bank wants to base its loan policy on a number of features of its clients, for instance on income, education level and criminal record. If a client is granted a loan, it can be one in three classes: low, intermediate and high. So, together with the 'no loan' option, we have four classes. Suppose further that the bank wants to base its loan policy on a number of credit worthiness decisions in the past. These past decisions are given in Table 1. A client with features at least as high as those of another client may expect to get at least as high a loan as the other client. So, finding a loan policy compatible with past decisions amounts to solving a monotone classification problem with the dataset of the above table. In order to save space we often map the values of the attributes of a dataset to a set of numbers. For instance, the table just presented could be written like Table 2.

Table 2: Two types of shorthand for the dataset of Table 1

x_1	x_2	x_3	y	$\mathbf{x}$	y
0	0	1	0	001	0
0	0	2	1	002	1
1	1	2	2	112	2
2	0	2	2	202	2
2	1	2	3	212	3

3 Generating random monotone datasets

In this section we will propose a technique that produces monotone datasets with prescribed parameters. This technique could, for instance, generate a monotone data set of n elements from a given instance space $\mathcal{X}$ and a set of target values $\mathcal{Y}$, such that the frequency of the occurrence of these target values follows a prescribed pattern. This data set will be random, in the sense that it is a random sample from the space of all possible datasets satisfying the prescribed conditions. However, it is not guaranteed that all possible datasets have equal chances of being drawn. In order to describe the technique we will first introduce a mapping between datasets and certain graphs, that will facilitate the description.

Let $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_p$ be an instance space with p attributes, $\mathcal{Y}$ a set of target values and let $\mathcal{D}$ be a set of examples $(\mathbf{x}, y)$ from $\mathcal{X} \times \mathcal{Y}$. With $\mathcal{D}$ we will associate a labeled directed graph $\mathcal{G}(\mathcal{D})$ as follows: $\mathcal{D}_x$, the set of $\mathbf{x}$-values from dataset $\mathcal{D}$, will be the set of vertices of the directed graph, and let $[\mathbf{x}, \mathbf{x}']$ be an arc[2] of the graph iff $\mathbf{x} > \mathbf{x}'$. (Note, that this graph $\mathcal{G}(\mathcal{D})$ will always be acyclic.) Let further each vertex $\mathbf{x}$ of the graph be labeled with the target value y of data example $(\mathbf{x}, y)$. For example, the 3-attribute data set $\mathcal{D}$ of Table 2 will have as its associated graph $\mathcal{G}(\mathcal{D})$ the graph of Figure 1.

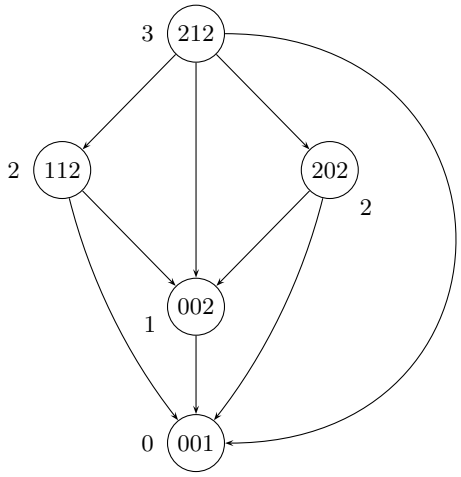


Figure 1: The Graph Associated with a Dataset

[2]An arc is an edge together with a direction.

We will call a path in a graph $\mathcal{G}(\mathcal{D})$ *non-increasing* if the labels of the successive vertices on the path form a non-increasing sequence. For instance, the path $[212, 112, 002, 001]$ is non-increasing since the labelings form the non-increasing sequence $[3, 2, 1, 0]$. Even if we would relabel 002 to 2, the path would stay non-increasing; however if we would relabel 001 to 2 the path would loose this property. The following lemma connects the monotonicity of a data set with the non-increasing nature of the paths in its associated graph.

Lemma 1 *Let $\mathcal{D}$ be a data set on $\mathcal{X} \times \mathcal{Y}$ and let $\mathcal{G}(\mathcal{D})$ be its associated labeled graph. Then we have*

$$\mathcal{D} \text{ is monotone} \iff \text{all paths in } \mathcal{G}(\mathcal{D}) \text{ are non-increasing.}$$

Proof: First we prove the $\Rightarrow$ part. In that case we suppose that (2) holds for all $(\mathbf{x}, y) \in \mathcal{D}$. Now, suppose we have a path $[\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n]$ in $\mathcal{G}(\mathcal{D})$, then $[\mathbf{x}_0, \mathbf{x}_1]$,$[\mathbf{x}_1, \mathbf{x}_2]$, etc. must be arcs of the graph, so from the definition of the associated graph it follows that $\mathbf{x}_0 > \mathbf{x}_1 > \ldots > \mathbf{x}_n$. Now, from (2) we can conclude that $y_0 \geq y_1 \geq \ldots \geq y_n$). Next we prove the $\Leftarrow$ part. So we suppose that all paths in $\mathcal{G}(\mathcal{D})$ are non-increasing. We must now prove (2) for all pairs $(\mathbf{x}, y)$ and $(\mathbf{x}', y')$ from $\mathcal{D}$. So let $(\mathbf{x}, y)$ an $(\mathbf{x}', y')$ be data examples from $\mathcal{D}$ with $\mathbf{x} \leq \mathbf{x}'$. If $\mathbf{x} = \mathbf{x}'$ then $y = y'$, so (2) is trivial. If $\mathbf{x} < \mathbf{x}'$ then $[\mathbf{x}', \mathbf{x}]$ is an arc of $G(\mathcal{D})$ so $[\mathbf{x}', \mathbf{x}]$ is also a path. Since all paths are non-increasing it follows that $y' \geq y$. $\square$

We proceed with an informal description of the algorithm for producing monotone datasets. Suppose, the required datasets must have N examples. We start with selecting N different vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ randomly from $\mathcal{X}$. Next, we select N not necessarily different target labels $y_1, \ldots, y_N$ from $\mathcal{Y}$. Now, each of the target labels $y_1, \ldots, y_N$ must be assigned to exactly one of the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ to obtain the required data set. This will be done as follows:

- construct the graph associated with $\mathbf{x}_1, \ldots, \mathbf{x}_N$
- sort the sequence $y_1, \ldots, y_N$ such that $y_1 \leq y_2 \leq \ldots \leq y_N$
- now, interpret the graph as a flow network: each of the target labels $y_1, \ldots, y_N$ is allowed to travel through the network, following a random path; each path starts at a randomly chosen source node (a node without incoming arcs) and ends at a sink node (a node without outgoing arcs); note that each path is finite since the graph is acyclic;

when the label comes to the end of its path, the target label is sticked to the last vertex on the path; this vertex is subsequently removed from the graph, and the next label is allowed to travel through the new graph. This is done until each vertex of the original graph has a target label sticked to it.

For instance, if the randomly drawn vectors are 001, 002, 112, 202, and 212 and the target labels are 0, 1, 2, 2, 3 we end up with the above graph $\mathcal{G}$, so the above data set $\mathcal{D}$ results. However, if with the same set of vectors we want to associate the target labels 0, 1, 1, 2, 3, one of the following datasets will result:

Dataset 1		Dataset 2	
001	0	001	0
002	1	002	1
112	1	112	2
202	2	202	1
212	3	212	3

each with probability $\frac{1}{2}$.

We will end with a formal description of the algorithm for producing monotone datasets:

1. Draw $\mathbf{x}_1, \ldots, \mathbf{x}_N$ randomly, without replacement, from $\mathcal{X}$.

2. Select $y_1, \ldots, y_N$ from $\mathcal{Y}$, see Remark 1 below.

3. Order $y_1, \ldots, y_N$ such that $y_1 \leq y_2 \leq \ldots \leq y_N$.

4. Define the $(N+1) \times N$ matrix $M = (m_{ij})$ as follows:
   for $1 \leq i, j \leq N$
   $$m_{ij} = \begin{cases} 1 & \text{if } \mathbf{x}_j < \mathbf{x}_i \\ 0 & \text{otherwise} \end{cases}$$
   and for $1 \leq j \leq N$:
   $$m_{0j} = \begin{cases} 1 & \text{if } \sum_{i=1}^{N} m_{ij} = 0 \\ 0 & \text{otherwise} \end{cases}$$

5. Perform the following algorithm on M:

```
for j := 1 to N do
    begin
        k := FindSink(M);
        label(k) := y_j;
        for i := 0 to N do m_ik := 0
    end
```

where the function FindSink, which returns a number between 1 and N, is defined as follows:

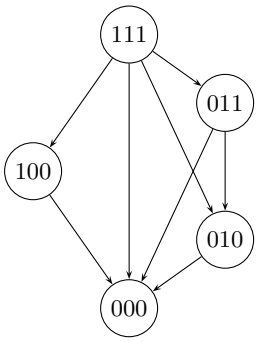
```
FindSink(var M):
    p := 0;
    while PathsFrom(p) is non-empty do
        p := random element from PathsFrom(p);
    return p
```

where the set PathsFrom(p) is defined as

$$\text{PathsFrom}(p) := \{i \in \{1, \ldots, N\} : m_{pi} > 0\}.$$

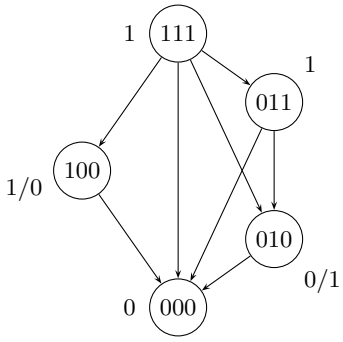
Remark 1 Note, that in step 2 of this algorithm, we can fill in our own selection mechanism. For instance, we could select the target labels according to a random mechanism or take some arbitrary set of labels at will.

Remark 2 It can be shown that for a given set of vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and an ordered list of target labels $[y_1, \ldots, y_N]$ the above algorithm can generate each possible monotone data set $\mathcal{D}$ with $D_x = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and $D_y = [y_1, \ldots, y_N]$. Sometimes, it also generates all the possible monotone datasets with equal probability, as is the case in the above example. However, this is not necessarily the case, as can be seen from the following example: If the set of vectors is 000, 100, 010, 011, 111 the associated graph has the following shape



Now, if the target label list is $[0, 0, 1, 1, 1]$ we can generate two different monotone datasets, like in Figure 2.

Figure 2: The graphs of two different monotone datasets



Here the notation 1/0 means class 1 for the first data set and class 0 for the second data set. Note that the first data set is generated with probability $\frac{2}{3}$ and the second one with probability $\frac{1}{3}$. Thus, the datasets generated by our algorithm, although guaranteed to be monotone, might contain a slight bias in the sense that some monotone datasets have a higher probability of being selected than others. This point needs further investigation.

4 Generating structured monotone datasets

In many cases, we want our artificial datasets to be structured or model-based. The reason for this is the following: usually, the algorithm we want to test on the artificial dataset is supposed to discover an underlying pattern or structure in the data; for instance, in classification or regression problems, a relation between the attributes and the target values is supposed to be discovered. Thus, our artificial dataset should be based on such an underlying relation or model: it is hard to discover something, when there is nothing to discover. In case we deal with a monotone problem, the underlying model should be monotone. If our model is to be tree-based (as will be the models in this paper), how are we going to label the leaves in such a fashion that the resulting tree will be monotone? In this section we will show that to label the leaves, we can essentially use the algorithm of the preceding section, applied to the leaves of the tree, provided these leaves are ordered according to the special ordering, introduced in [7].

Specifically, if $\mathcal{T}$ is a decision tree on instance space $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_p$ with instances $\mathbf{x} = (x_1, \ldots, x_p)$ and the test in each node of the tree has the form $X_i \leq c$ for some $c \in \mathcal{X}_i, 1 \leq i \leq p$, then the associated subset $T \subset \mathcal{X}$ of each node or leaf has the form $T = \{\mathbf{x} \in \mathcal{X} : \mathbf{a} < \mathbf{x} \leq \mathbf{b}\} = (\mathbf{a}, \mathbf{b}]$ for some $\mathbf{a}, \mathbf{b} \in \overline{\mathcal{X}}$ with $\mathbf{a} < \mathbf{b}$ and $\overline{\mathcal{X}} = \mathcal{X}$ extended with infinity elements. We will call $\mathbf{a} = \min(T)$ the minimal element or left corner of node T and $\mathbf{b} = \max(T)$ the maximal element or right corner. The set of leaves $\mathcal{L}$ of the tree form a partition of $\mathcal{X}$ and are labeled with values from $\mathcal{Y}$. Thus, the tree defines an associated labeling rule $f : \mathcal{X} \to \mathcal{Y}$. The tree will be called *monotone* if for its associated labeling rule $f(\mathbf{x})$ we have

$$\mathbf{x} \leq \mathbf{x}' \Rightarrow f(\mathbf{x}) \leq f(\mathbf{x}'). \tag{3}$$

The special ordering we need for nodes and leaves is the following: if T and T' are nodes or leaves, then we define

$$T \leq T' \iff \min(T) \leq \max(T'). \tag{4}$$

In [7] it is shown that this ordering is reflexive and anti-symmetric, but not transitive. Thus, it does not induce a partial ordering on the set of leaves. However, a partial ordering is not necessary for the following lemma to hold.

Lemma 2 *Let $\mathcal{T}$ be a classification or regression tree on $\mathcal{X}$ with leaves $\mathcal{L}$ and let f be its associated labeling rule. For all pairs $T, T' \in \mathcal{L}$ we have*

$$T \leq T' \Rightarrow f(T) \leq f(T'), \tag{5}$$

if and only if $\mathcal{T}$ is monotone.

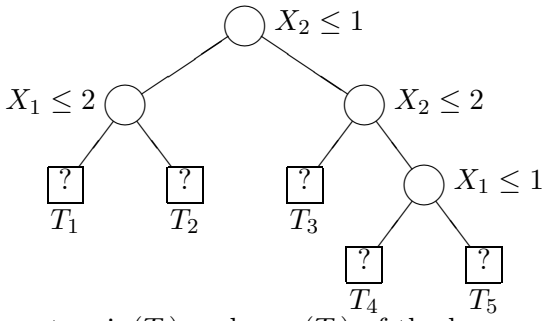
Proof: We first prove the 'only if' part. It is clear that with $f(T)$ we mean the labeling assigned to leaf T by tree $\mathcal{T}$. To prove that $\mathcal{T}$ is monotone we should prove (3) for all $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$. If $\mathbf{x}$ and $\mathbf{x}'$ belong to the same leaf, (3) is trivial. So, let us suppose that they belong to different leaves, with T the leaf that contains $\mathbf{x}$ and T' the leaf that contains $\mathbf{x}'$. Since by hypothesis $\mathbf{x} \leq \mathbf{x}'$, we have $\min(T) \leq \mathbf{x} \leq \mathbf{x}' \leq \max(T')$, so according to our definition (4) we have $T \leq T'$. Thus, by (5) we have $f(T) \leq f(T')$, which proves (3). The reverse part follows the same lines. □

We can use Lemma 2 to assign labels to an unlabeled decision tree $\mathcal{T}$ with leaves $\mathcal{L}$ such that the resulting decision tree will be monotone, as follows. First, generate as many labels $y \in \mathcal{Y}$ as there are leaves in the tree. This can

be done according to any distribution. Next, sort these labels in ascending order: $y_1 \leq y_2 \leq \ldots \leq y_k$. Next, we build the associated graph $\mathcal{G}(\mathcal{L})$ of the leaves just like we did we the instances in Section 2. Thus, the vertices of the graph are now leaves instead of instances. We now run the algorithm of Section 2 on this graph together with the labels $y_1 \leq y_2 \leq \ldots \leq y_k$. The proof that the resulting labeled tree will be monotone follows directly from Lemma 2.

5 Example

In this section we will show how we can use the method of Section 3 to label an unlabeled decision tree monotonically. Suppose we have the following unlabeled tree:



The corner elements $\min(T_i)$ and $\max(T_i)$ of the leaves of this tree are easily checked to be as follows:

leaf	$\min(T_i)$	$\max(T_i)$
T_1	$(-\infty, -\infty)$	$(2, 1)$
T_2	$(2, -\infty)$	$(\infty, 1)$
T_3	$(-\infty, 1)$	$(\infty, 2)$
T_4	$(-\infty, 2)$	$(1, \infty)$
T_5	$(1, 2)$	(∞, ∞)

By inspection of this table we can find all the arrows in the associated graph for these leaves, which is shown in Figure 3.

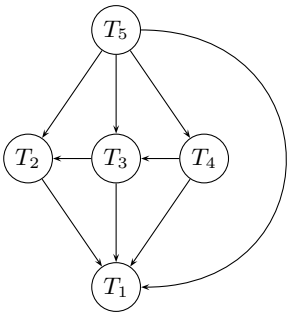


Figure 3: The associated graph for the leaves of the example tree.

Now, if we generate the following set of labels $\{0, 1, 1, 1, 2\}$, the algorithm of Section 2 returns the following label assignment: $f(T_1) = 0, f(T_2) = f(T_3) = f(T_4) = 1, f(T_5) = 2$ which gives indeed a monotone decision tree as can be easily checked.

As an aside, this example also shows the non-transitivity of the special ordering of the leaves: we have $T_2 \leq T_3$ and $T_3 \leq T_4$, but $T_2 \not\leq T_4$ since $(2, -\infty) \not\leq (1, \infty)$.

6 Conclusion

The algorithms of this paper are an important aid to help us with the difficult task of generating artificial data for monotone models. The first algorithm is guaranteed to give a monotone data set, the second algorithm yields a monotone decision tree based on any unlabeled tree. This monotone tree can subsequently be used to generate structured monotone data; if needed, random noise can be superimposed. As a side effect, in this paper interesting characterizations of monotone datasets and monotone decision trees are proved.

References

[1] A. Ben-David. Automatic generation of symbolic multi-attribute ordinal knowledge-based DSS's: methodology and applications. *Decision Sciences*, 23:1357–1372, 1992.

[2] A. Ben-David. Monotonicity maintenance in information-theoretic machine learning algorithms. *Machine Learning*, 19(1):29–43, 1995.

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

[4] J.C. Bioch. Dualization, decision lists and identification of monotone discrete functions. *Annals of Mathematics and Artificial Intelligence*, 24:69–91, 1998.

[5] J.C. Bioch and T. Ibaraki. Complexity of identification and dualization of positive Boolean functions. *Information and Computation*, 123:50–63, 1995.

[6] J.C. Bioch and V. Popova. Monotone decision trees and noisy data. In *ERIM Report Series Research in Management, ERS-2002-53-LIS*, 2002.

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

[8] J.C. Bioch and V.N. Popova. Rough sets and ordinal classification. In A. Sharma H. Arimura, S. Jain, editor, *Algorithmic Learning Theory*, Lecture Notes in Artificial Intelligence 1968, pages 291–305. Springer, 2000.

[9] Kim Cao-Van and Bernard de Baets. Growing decision trees in an ordinal setting. *International Journal of Intelligent Systems*, 18:733–750, 2003.

[10] H. Daniels and B. Kamp. Application of MLP networks to bond rating and house pricing. *Neural Computation and Applications*, 8:226–234, 1999.

[11] A.J. Feelders and M.A. Pardoel. Pruning for monotone classification trees. *Lecture Notes in Computer Science*, 2810:1–12, 2003.

[12] O. Harrison and D. Rubinfeld. Hedonic prices and the demand for clean air. *Journal of Environmental Economics and Management*, 53:81–102, 1978.

[13] T.J.C. Hunniford and R.J. Hickey. A simulated annealing technique for generating artificial data to assess concept learning algorithms. *Intelligent Data Analysis*, 3(3):177–189, 1999.

[14] Kazuhisa Makino, Takashi Susa, Kojin Yano, and Toshihide Ibaraki. Data analysis by positive decision trees. In Yahiko Kambayashi et al., editors, *Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications (CODAS)*, pages 257–264, Kyoto, Japan, December 1996. World Scientific.

[15] H. Mukarjee and S. Stern. Feasible nonparametric esimation of multiargument monotone functions. *Journal of the American Statistical Association*, 89(425):77–80, 1994.

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

[17] R. Potharst and A.J. Feelders. Classification trees for problems with monotonicity constraints. *SIGKDD Explorations*, 4(1):1–10, 2002.

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

[19] J.O. Ramsay. Monotone regression splines in action. *Statistical Science*, 3(4):425–441, 1988.

[20] T. Robertson, F. Wright, and R.L. Dykstra. *Order Restricted Statistical Inference*. Wiley, 1988.

[21] Sheldon M. Ross. *Simulation, 3rd edition*. Academic Press, New York, 2001.

[22] S. Wang. A neural network method of density estimation for univariate unimodal data. *Neural Computation & Applications*, 2:160–167, 1994.