ORIGINAL RESEARCH

Generating noisy monotone ordinal datasets

Irena Milstein *[1], Arie Ben-David[1], Rob Potharst[2]

[1] *Holon Institute of Technology, Israel*
[2] *Erasmus School of Economics, Netherlands*

Received: September 16, 2013 **Accepted:** October 21, 2013 **Online Published:** December 25, 2013
DOI: 10.5430/air.v3n1p30 **URL:** http://dx.doi.org/10.5430/air.v3n1p30

Abstract

Ordinal decision problems are very common in real-life. As a result, ordinal classification models have drawn much attention in recent years. Many ordinal problem domains assume that the output is monotonously related to the input, and some ordinal data mining models ensure this property while classifying. However, no one has ever reported how accurate these models are in presence of varying levels of non-monotone noise. In order to do that researchers need an easy-to-use tool for generating artificial ordinal datasets which contain both an arbitrary monotone pattern as well as user-specified levels of non-monotone noise. An algorithm that generates such datasets is presented here in detail for the first time. Two versions of the algorithm are discussed. The first is more time consuming. It generates purely monotone datasets as the base of the computation. Later, non-monotone noise is incrementally inserted to the dataset. The second version is basically similar, but it is significantly faster. It begins with the generation of almost monotone datasets before introducing the noise. Theoretical and empirical studies of the two versions are provided, showing that the second, faster, algorithm is sufficient for almost all practical applications. Some useful information about the two algorithms and suggestions for further research are also discussed.

Key Words: Ordinal classification, Monotone classification, Non-monotonicity index

1 Introduction

Decision making problems in which the dependent variable values are ordinal (i.e., ordered) are very common. This is due to the fact that we, human beings, often tend to think in symbolic ordinal terms. In bond rating, for instance, even the most reputable agencies grade the risk of nonpayment in ordinal symbols such as AAA, AA, A, A- and so on. Generally speaking, we call any problem in which the dependent variable (which is later referred to as *"the class"* values are expressed in ordinal terms - an *ordinal problem*. We assume that there is only one dependent variable. Numerical attributes, if present, are usually converted to ordinal scales. Such a conversion is desirable, for instance, when one considers granting a credit card limit to an applicant. Usually it does not make any difference whether the applicant's net assets are worth 99,000 USD or 102,000 USD. Instead, we tend to think about such a feature in ordinal terms such as 'low', 'high', etc. Otherwise, the decision maker will be overloaded with useless information. What we do know is that all other things being equal, an applicant with net worth of 102,000 USD should be granted at least the same credit limit as the one granted to a customer with net worth of 99,000 USD. Otherwise, his/her decisions will contradict each other. This very common type of problem domains is called *monotone ordinal problems*. Monotone ordinal problems should satisfy a *monotonicity constraint*: the class is expected to be a monotone function of the attribute values.

To further clarify what a monotone ordinal problem means in the context of this work consider an assessment of a course given by students. The students are typically requested by the end of each semester to fill some kind of form in which they are presented with several attributes, such as "significance of the material to your future career", "interesting presentation of the material", and so on. The dependent variable may be "a general evaluation of the course". Usually a student selects a single ordinal value for each attribute out of a predefined possible ordinal set of values. Later he/she selects one ordinal value which best describes the dependent variable. In the terminology of this paper, the student has selected a specific class value. Note that the problem has a single class which may take one out of several possible ordinal values, and that all the attribute possi-

*Correspondence: Dr. Irena Milstein; Email: irenam@hit.ac.il; Address: Holon Institute of Technology, Israel.

ble values are ordinal too. It is expected that the "higher" the selected attribute values are, the "higher" the general score will be. In other words, we expect the dependent variable (or class) values to be some non-decreasing function of the attribute (ordinal) values. Examples of monotone ordinal problems include credit loan approval, bankruptcy prediction, insurance underwriting, and various other selection and preference problems. So far, an impressive number of data mining models have been developed for these monotone problems.

It is expected that any classification model for an ordinal monotone problem will be accurate and also will provide purely monotone classifications. However, real-life datasets, upon which learning is done, usually include noise. For example, a life insurance dataset may include cases where younger and healthier people were asked to pay higher life insurance premium relative to older and sicker applicants. If such a pair exists in the dataset we call it a *clash-pair* or a *non-monotone pair*. Another common example of a clash-pair arises when a pair of examples that have the same attribute values belong to different classes. The existence of such clash-pairs in datasets that should ideally be purely monotone is called *non-monotone noise*. Non-monotone noise is a fact of life in virtually all real-life datasets which result of human decision-making. This is due to the fact that a group of decision-makers may have different preferences. Even if a dataset is the result of decisions that were made by a single decision-maker over some period of time one should expect it to include non-monotone noise due to inconsistencies in subsequent decision, fatigue, overloading, etc.[1,2] These and many other studies of human decision-making show that decision-makers tend to simplify complex decisions. For example, we do not usually take more than 7 plus or minus two attributes into account simultaneously (also known as "Miller's Magical Numbers"). In this research we follow this guideline and concentrate on decision making problems which are manageable by humans in terms of problem space size. To be more specific, we are not interested here in problems that have 100 or 1000 attributes, since most of them are ignored by decision-makers anyway. Rather we are focused on problems which we nickname *"real world" ordinal decision problems*. Such problems typically have one class and less than ten attributes. Each attribute may have 2 to 10 possible ordinal values, and so is the class. We refer to "real world" problems later on while analyzing the proposed algorithm in Section 4.

Some data mining algorithms for monotone decision-making problems expect the training set to be purely monotone. Other ordinal data mining models can cope with non-monotone noise and even provide monotone classifications. Others can learn from non-monotone datasets but do not guarantee monotone classifications. We refer to these issues in detail in the Related Work section.

Some very innovative and diversified approaches to ordinal data mining have evolved in recent years. However, some key issues were not addressed yet. A solution to one of these open issues is proposed in this research, namely how to generate artificial ordinal datasets that have both monotone patterns as well as specified levels of non-monotone noise. The proposed algorithm can be used by any researcher who wishes to test how sensitive his/her ordinal classification model is to varying levels of non-monotone noise. Clearly, one does expect the models' performance to deteriorate when this type of noise increases, but surprisingly, no one has ever reported how fast this deterioration is for any monotone ordinal model.

In the next section we give an overview of related work. Section 3 deals with indices of non-monotonicity and introduces the notations used throughout the paper. Section 4 presents and discusses the proposed algorithm in detail. Section 5 includes conclusions and offers some future research topics.

2 Related work

Ordinal datasets are quite common in human decision making. For this reason they drew the attention of researchers from various fields for decades. Perhaps the most famous statistical ordinal model is McCullagh's Ordinal Regression.[3] It does not require datasets to be purely monotone, but it does not guarantee monotone classifications afterwards. The Ordinal Learning Model is an early, simple, ordinal classification model.[4] It also can work with non-monotone datasets, but unlike Ordinal Regression, it does ensure monotone classifications.

Monotone decision tree models have been introduced by Makino et al.[5] and later by Potharst and Bioch,[6] Cao-Van and De Baets,[7] Feelders and Pardoel[8] and others. They all require purely monotone datasets to begin with, and they assure the monotonicity of subsequent classifications. Lievens et al.'s probabilistic OSDL[9] can work with noisy ordinal datasets and also provides monotone classifications. Monotone neural network models were described by Daniels and Velikova.[10] Their algorithms can cope with noisy data but do not guarantee monotone classifications. Other approaches include, but are not limited to, Ben David's hybrid approach,[11] Frank and Hall's Ordinal Class Classifier meta-model,[12] and Popova and Bioch classification by function decomposition.[13] The last three models can cope with non-monotone noisy data but do not guarantee subsequent monotone classifications. This is unlike the rule ensembles proposed by Dembczynski et al.[14] that can work with noisy data and do provide monotone classifications.

Two papers have been published so far on the generation of monotone artificial datasets. The paper by De Loof et al. is about generating completely random monotone datasets.[15] It uses the computationally intensive Markov Chain Monte Carlo method. The datasets which are generated by this method have no underlying pattern, while in

our approach the user can embed any monotone function of his/her choice. The paper by Potharst et al. makes use of much simpler computation, and proposes a method to incorporate an underlying structure into the artificial monotone dataset.[16] Our algorithm begins with the generation of either purely monotone or almost purely monotone dataset (depending on the version to be shortly explained). In other words, only the first phase of our algorithm has a similar goal as defined in De Loof et al.[15] and Potharst et al.[16] Similar to Potharst et al.,[16] our algorithm does generate a monotone pattern as the base of the computation. However, our algorithm significantly differs from these two approaches by: (a) The resulting dataset in our approach contains user-defined level of noise, and (b) Our algorithms do not use class relabeling. Daniels and Velikova[10] briefly mention an algorithm which generates noisy monotone datasets. Similar to their approach, we convert a purely monotone dataset into a non-monotone one. However, since they have not presented a detailed description or a pseudo code of their algorithm, nor an analysis of its complexity, it is not possible to compare it with our algorithm.

3 Indices of non-monotonicity

Before presenting the algorithm in the coming section, let us review here some notation and clarify some important definitions. Let D be a dataset with k ordinal attributes A_1, ..., A_k and class variable Y which has C possible ordinal values. The dataset consists of n examples x. A partial ordering $\preceq$ on D is defined as

$$x \preceq x' \Leftrightarrow A_j(x) \leq A_j(x'), for j = 1, ..., k \tag{1}$$

Thus, two examples x and x' in space D are $comparable$, if either $x \preceq x'$ or $x' \preceq x$, otherwise x and x' are $incomparable$. $Identical$ examples denoted as $x = x'$, and $non-identical$ as $x \neq x'$.

Having this notation in mind, we call a pair of comparable examples (x, x') $monotone$ if

$$x \preceq x' \wedge x \neq x' \ \wedge Y(x) \leq Y(x') \tag{2}$$

or

$$x = x' \wedge Y(x) = Y(x') \tag{3}$$

A dataset consisting of n examples is $monotone$ if all possible pairs of examples are either monotone or incomparable. Example x from $Dclashes$ with example x' from D if

$$x \preceq x' \wedge x \neq x' \ \wedge Y(x) > Y(x') \tag{4}$$

or

$$x = x' \wedge Y(x) \neq Y(x') \tag{5}$$

Furthermore, we use the following notation: if x is an example from dataset D, then $NClash(x)$ is the number of examples from D that clash with x. $Clash(x) = 1$ if x clashes with some examples in D, and 0 otherwise. If $Clash(x) = 1$, x is called a non-monotone example.

Following Daniels and Velikova,[10] we call the first, and most obvious, index of non-monotonicity to be introduced here $NMI1$, the number of clash-pairs divided by the total number of pairs of examples in the dataset. So,

$$NMI1 = \frac{1}{n(n-1)} \sum_{x \epsilon D} NClash(x) \tag{6}$$

Horvath et al.[17] suggested to divide by the number of all comparable pairs in the dataset.

The second index is called here $NMI2$, the number of non-monotone examples divided by the total number of examples. So,

$$NMI2 = \frac{1}{n} \sum_{x \epsilon D} Clash(x) \tag{7}$$

We call the third index here *NMI3*, the minimum number of class label changes needed to make a dataset monotone, divided by the total number of examples. The lowest value of all three indices is 0, when the dataset is purely monotone. The highest value of the first two indices is 1, when every possible pair in the dataset clashes, which means also that all examples are non-monotone with respect to each other. The third index is always less than 1 since there is at least one label that does not needed to be relabeled.

NMI1 has been chosen by us due to its intuitiveness rather than its simplicity. Unlike *NMI2* and *NMI3*, *NMI1* reflects the fact that non-monotonicity occurs in pairs of examples. It counts all the clashing pairs in the dataset.

Before trying to actually generate artificial ordinal datasets it was of interest to us to check what values of *NMI1* are expected in real world ordinal datasets in which the monotonicity constraint does make sense. Daniels and Velikova who have checked two datasets found that they were almost monotone.[10] On the other hand, Horvath et al. found only 5 purely monotone datasets out of 40.[17] We also expected that the values of *NMI1* will be rather low since high values of noise may imply that a pattern if exists in the data may be corrupted by noise. We have chosen four real world ordinal datasets: ESL, ERA, LEV, and SWD. They all can be found and freely downloaded from the Weka web site (http://www.cs.waikato.ac.nz/ml/). Table 1 presents a short description of the datasets and the values of *NMI1*.

Table 1: The values of *NMI1* in real-world datasets.

Name	Description	Number of examples, *n*	Number of attributes, *k*	Value of *NMI1*
ERA	Degree of acceptance of applicants to a job.	1000	4	0.039
ESL	Profiles of applicants for a certain type of industrial jobs.	488	8	0.009
LEV	Evaluation of MBA courses.	1000	4	0.013
SWD	Assessments of qualified social workers.	1000	10	0.009

The results presented in Table 1 are confirmatory to our hypothesis and the values of *NMI1* in real-world datasets are rather low (vary between 1 percent and 4 percent in Table 1). For the sake of simplicity we refer to *NMI1* (which we use throughout this paper) in the coming sections as the *non-monotonicity index*. The following section describes the algorithm that generates a monotone ordinal dataset of size n (examples) with a user-specified value of the *non-monotonicity index* (i.e., *NMI1*).

4 The algorithm

Our goal while generating a non-monotone ordinal dataset is to artificially create a set of *n* examples which has both some pre-defined monotone pattern as well as a specific (also user-defined) *non-monotonicity index*. There are several ways to approach this problem. For example, one can begin the process with the generation of a monotone dataset with *n* examples and later exchange some of them with noisy examples incrementally. Another approach is to generate the examples with the monotone pattern and the non-monotone noise simultaneously. We have chosen the first approach since generating monotone datasets is a problem that has already been studied, so the results can be used here.

4.1 Version A

The proposed algorithm works in two phases. The first phase generates a purely monotone dataset with *n* examples. Several ways of how purely monotone datasets can be generated were discussed earlier in the Related Work section. For this algorithm, we have chosen another, simpler, method which is described in steps A to E of Figure 1. The second phase, step F of Figure 1, changes the existing examples in the dataset till the desired level of non-monotonicity index is reached. This phase may be incrementally repeated for the different (increasing) levels of the non-monotonicity index, such that several datasets (each with the same number of examples and similar pattern, but with different non-monotone indices) will be generated in a single run.

Input: Number of examples (n), Desired Non-monotonicity Index (NMI), Number of Attributes (k), Number of ordinal values of each attribute (O_j), Number of ordinal class values (C).

Output: A dataset with the above specifications.

A. **For** each example {

A.1 Assign random values $(0,1,\dots,O_j-1)$ to the attributes

A.2 Compute the output as a non-decreasing monotone function of the attribute values}.

B. Sort all the examples in increasing order of output values.

C. Assign ordinal values $(0,1,\dots,C-1)$ to the class such that the class values will be balanced.

D. **Do while** the dataset includes non-monotone pairs {

D.1 If two examples have similar attribute values but different class values then

Randomly assign the class of one example to the other}.

E. Arrange the examples in random order.

F. **Do while** the current value of non-monotonicity index is less than the lower bound of NMI {

F.1 Randomly select an example, x

F.2 Generate a new example, x', which clashes with the selected example, x

F.3 Randomly select an example, x'', different from x, which will be replaced by the new created example, x'

F.4 Recalculate the value of non-monotonicity index

F.4.1 Compute the number of examples that clash with the example to be replaced, $NClash(x'')$

F.4.2 Compute the number examples that clash with the new created example, $NClash(x')$

F.4.3 Update the current value of the non-monotonicity index

F.5 If the current value the non-monotonicity index is less than the upper bound of NMI then

Replace a randomly chosen example, x'', with the newly generated example, x';

Otherwise, cancel the changes done in step F.4}.

Figure 1: Version A algorithm

In step A.1 of Figure 1 the attribute values, A_j, of n examples are generated, each having O_j possible integer values ranging from zero to O_j-1 (i.e. $A_j \in 0, 1, ..., O_j - 1), j = 1, ..., k$. These values can be randomly chosen from any statistical distribution, but in the current version a Uniform distribution has been used for simplicity. Once the integer values of the attributes are assigned to all k attributes of an example, the class numerical value is calculated using a monotonically increasing function. We have used several of those in our experiments such as $Y = \sum_j A_j, Y = \sum_j jA_j, Y = \sum_j jA_j^j, Y = \sum_j (A_j^3 + jA_j^j)$, and more.

The monotone examples are later sorted in increasing order of (still numeric) class values. This is shown in step B of Figure 1. Step C maps these class values to (approximately) equally balanced integer class values ranging from zero to *C*-1. This is done in the following manner: Suppose that there are 700 examples and 7 possible ordinal class values, the first 100 examples (after sorting in increasing order of class values) are assigned the value 0, the next 100 examples – the value 1, ... , and the last 100 examples are assigned the value 6. This is shown in step C of Figure 1.

However, the dataset which is generated in steps A-C is not guaranteed to be monotone. If two examples, say *x* and *x'*, have similar attribute values – they surly will have similar values of *Y* by the end of step A. On the other hand, *x* and *x'* may be assigned different class values in step C. In the above example, if the 100th and the 101th example have similar attribute values, the first will be assigned with class 0 and the second will be assigned the value 1. According to our definition of non-monotonicity (see Section 3), these two examples clash with one another. Step D resolves this kind of non-monotone conflicts, and once this step is done – the generated dataset is monotone.

Step E arranges the examples in random order (unlike step B). It is easy to show that the proposed algorithm for generating a monotone dataset is admissible, complete and unbiased.[16]

At the second phase the generated purely monotone dataset is modified according to the desired non-monotonicity index. This phase is described in step F of Figure 1. In step F.1 some example, say x, is randomly selected from the dataset, and then in step F.2 a new example, x', which is provided to clash with example x, is generated. This is done by randomly selecting attribute values for x', $A_j(x')$, that are '$\leq$' (or '$\geq$') relative to those of x, $A_j(x)$, and a class value, $Y(x')$, that is '>' (or '<') to $Y(x)$. In step F.3 another example, say x'', is randomly selected from the dataset. This example has to be replaced later by a new example, x'. In step F.4 the current value of the non-monotonicity index is updated by the difference between the number of examples that clash with x' and the number of examples that clash with x''. If the updated value of the non-monotonicity index is less than the desired value, then x' replaces x'' in step F.5. Note that the current value of the non-monotonicity index may decrease. In the general case it may happen that the randomly selected example which is deleted from the dataset (to give way to x') has been in many non-monotone conflicts with the rest of the examples. However, this occurs very rarely since the dataset is initially monotone or almost monotone and we usually are interested in generating datasets with low non-monotonicity indices.

4.2 Discussion and empirical results

Let us evaluate the complexity of version A. The first phase of generating a monotone dataset is implemented in O(n^2), due to step D that involves the checking of every pair of examples in the dataset. The time complexity of the second phase is O($n^2 \cdot NMI$), in the worst case, when all new examples clash with only one example in the dataset. Since step D is more time consuming, the time complexity of version A is O(n^2). One way to reduce the complexity of step D is to compare the class values for pairs of only adjacent examples. However, it seems that the non-monotonicity which is resolved in step D is rather rare when we generate the so-called "real world" ordinal datasets (i.e., with 2 to 10 attributes, each attribute and the class having 2 to 10 possible ordinal values) with a couple of hundreds of examples. If this hypothesis is correct, step D may be omitted altogether in most cases. The hypothesis was checked in the following experiment.

We have first generated datasets as described in steps A-C in Figure 1 and then calculated the non-monotonicity index. The experiment was repeated 5 times for different numbers of examples, n, attributes, k, ordinal values of each attribute, O_j, and ordinal class values, C. We assume the following initial values of the parameters: k = 5, O_j = 5, C = 4, and n = 1000 and then we change the value of only one parameter while the others remain constant. Tables 2 - 5 present the results of the experiment. The number of attributes, k, and the number of attribute values, O_j, are changed in Table 2 and Table 3, respectively, and equal to 3, 5, 7, and 10. The number of class values, C, is changed in Table 4 and equals to 2, 4, 6, and 8. Table 5 presents the results for *n*= 500, 1000, 1500 and 2000.

Tables 2 and 3 show that the value of the non-monotonicity index is decreasing in an increasing rate with the number of attributes, k, and the number of ordinal values of each attribute, O_j. The intuition behind this result is based on the fact that the initially generated dataset may be non-monotone only if identical examples have the different class values. Obviously, the probability of generating two identical examples in the dataset is decreasing when the number of attributes and/or the number of ordinal values of each attribute are increasing. On the other hand, the probability of a clash with identical attribute values is increasing with the number of ordinal class values, C, and/or with the size of the dataset, n. This intuition is confirmed by the results presented in Tables 4 and 5.

Table 2: The values of *NMI1* in initially generated datasets for different values of k ($O_j = 5$, $C = 4$, and $n = 1000$).

Number of attributes, k	Average number of clash-pairs	Average value of *NMI1*
3	167.4	3.35E-04
5	2	4.00E-06
7	0	0.00E+00
10	0	0.00E+00

Table 3: The values of *NMI1* in initially generated datasets for different values of O_j ($k = 5$, $C = 4$, and $n = 1000$).

Number of attribute values, O_j	Average number of clash-pairs	Average value of *NMI1*
3	156.6	3.14E-04
5	2	4.00E-06
7	0.2	4.00E-07
10	0	0.00E+00

Table 4: The values of *NMI1* in initially generated datasets for different values of C ($k = 5$, $O_j = 5$, and $n = 1000$).

Number of class values, C	Average number of clash-pairs	Average value of *NMI1*
2	0.8	1.60E-06
4	2	4.00E-06
6	3.4	6.81E-06
8	9.4	1.88E-05

The results of the above experiment show very low values of non-monotonicity indices in the initially generated datasets. Thus, step D is not essential, provided that a purely monotone dataset is not required. For example, if the purpose of an experiment is to generate dataset with some noise index, then getting a purely monotone one on the way is not essential at all. In this case, step D is not required. On the other hand, if the purpose is to compare the performance of models on both a purely monotone and noisy datasets, we suggest that a decision whether step D is needed or not will be based upon Tables 2-5. Usually, step D is not required when the number of attributes and/or attribute values is large, and/or the number of class values is small. Since most real-world monotone ordinal problems we have encountered so far do not require step D, a faster algorithm is provided in the next section, where we also explain why it skips step E of Figure 1.

Table 5: The values of *NMI1* in initially generated datasets for different values of n ($k = 5$, $O_j = 5$, and $c = 4$).

Number of examples, n	Average number of clash-pairs	Average value of *NMI1*
500	0.2	3.21E-06
1000	2	4.00E-06
1500	7.6	6.76E-06
2000	7.6	3.8E-06

4.3 Version B

It has been argued above that step D of Version A is usually not needed in practice. Furthermore step E may be also skipped since the examples are chosen randomly in step F anyway. The faster algorithm, called Version B, is presented in Figure 2.

Input: Number of examples (n), Desired Non-monotonicity Index (*NMI*), Number of Attributes (k), Number of ordinal values of each attribute (O_j), Number of ordinal class values (C).

Output: A dataset with the above specifications.

A. **For** each example {

A.1 Assign random values $(0,1,\dots,O_j-1)$ to the attributes

A.2 Compute the output as a non-decreasing monotone function of the attribute values}.

B. Sort all the examples in increasing order of output values.

C. Assign ordinal values $(0,1,\dots,C-1)$ to the class such that the class values will be balanced.

D. **Do while** the current value of non-monotonicity index is less than the lower bound of *NMI* {

D.1 Randomly select an example, x

D.2 Generate a new example, x', which clashes with the selected example, x

D.3 Randomly select an example x'', different from x, which will be replaced by the new created example x'

D.4 Recalculate the value of non-monotonicity index

D.4.1 Compute the number of examples that clash with the example to be replaced, $NClash(x'')$

D.4.2 Compute the number examples that clash with the new created example, $NClash(x')$

D.4.3 Update the current value of the non-monotonicity index

D.5 If the current value the non-monotonicity index is less than the upper bound of *NMI* then

Replace a randomly chosen example, x'', with the newly generated example, x';

Otherwise, cancel the changes done in step D.4}.

Figure 2: Version B algorithm

The total time complexity of Version B of the algorithm is defined by step D in Figure 2, that is O($n^2 \cdot NMI$), in the worst case, when all new examples clash with only one example in the dataset. However, this happens very rarely. The results of our experiments show that most often, the value is close to the minimum, O($n^2 \cdot NMI$). In the latter case, the most time consuming step may be step B, which is O($n^2 \cdot lgn$).

To test the algorithms in practice, a working prototype has been written using VBA interpreter. The prototype creates Excel spreadsheets as output, which can be read by most data mining software. The prototype is rather slow. For example, it takes about 32 seconds on the average to generate a noisy file of 1000 examples with 5 attributes (5 possible ordinal values for each attribute as well as for the output) on a 3.2 GHz i5 processor with 4GB memory using Version A of the algorithm. Version B runs about 44 percent faster. Compiled versions, which are written now, are expected to significantly improve the performance. However, even with the current interpreted Versions A and B one can generate a sequence of datasets with a user-defined monotone structure and varying (also user-defined) levels of noise in a matter of minutes on a common desktop computer.

5 Conclusions and further research

Ordinal datasets, in which the class is expressed in ordinal terms are very common in human decision-making. This explains the rapidly growing number of data mining models that use the ordinal information within the datasets. Some of these models assure monotone classifications, a property which is desired in many problem domains. This work is geared towards these algorithms as it proposes means to test their capabilities under controlled levels of non-monotone noise.

Most of the scientific work which has been done so far was focused on developing algorithms which get rid of non-monotone clashes. Surprisingly, we have not encountered articles that do that by means other than relabeling. However, it is of interest to compare the complexities of these algorithms to those that were presented here. Daniels and Velikova[10] proposed a method which is polynomial in the number of examples and class values, O(n^3C). Rademaker et al.[18] presented a method which is optimal in the sense that it relabels as few examples as possible. Time complexity of their algorithm is O(n^3). In an as yet unpublished paper, Pijls and Potharst[19] show that Rademaker et al.'s method may be streamlined. They also show that there are non-optimal methods that perform relabeling with time complexity of O(n^2). Again, all these algorithms aim at cleaning noisy datasets from non-monotone clashes, and they all are based on relabeling. Our algorithms work in the opposite direction. They generate noisy datasets from purely (or almost purely) monotone datasets, and they do not use relabeling.

Two versions of an algorithm which generates artificial monotone datasets with user-defined level of non-monotone noise were proposed and analyzed. This is the first time such algorithms are presented in detail. Based on empirical observations as well as on decision-making theory it has been argued that the faster version (Version B) is sufficient for most practical implementations. However, it has been shown that even the slower version (Version A) can produce useful datasets on a common desktop computer for most real-world human decision-making problems in a matter of minutes. This observation is based on the performance of a rather slow VBA based interpreted prototype we have written for both versions. This prototype is available to anyone upon request from the authors. Currently we are working on implementing a new, faster and more user friendly, version which is written in Java.

This work has been focused on describing the algorithm itself. It has opened, however, many interesting questions to future work. For example, exploring which of the currently known or newly proposed ordinal data mining models is sensitive to non-monotone noise. Which model requires almost purely monotone data sets to be relatively accurate? At which non-monotone noise level(s) the accuracy deteriorates? At what pace this deterioration occurs with the rise of noise levels? Another interesting question which has not been addressed here is the study of the effect of the monotone function chosen on step A.2 (see Figure 1 and Figure 2) on the generated data set. Will there be a significant difference (if any) on the generated dataset if one chooses linear versus non-linear (or highly non-linear) monotone function? We have also confined the experiments to what we have called "real world" problems, and have not tried generating data sets with dozens or more attributes. We intend to re-check this issue in the future using the compiled Java version.

References

[1] Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological Review, 63(2), 81-97. PMid:13310704 http://dx.doi.org/10.1037/h0043158

[2] Ganzach, Y. (1993). Goals as determinants of nonlinear noncompensatory judgment strategies. Organizational Behavior and Human Decision Processes, 56, 422-440. http://dx.doi.org/10.1006/obhd.1993.1062

[3] McCullagh, P. (1980). Regression models for ordinal data, Journal of the Royal Statistical Society, 42(2), 109-142.

[4] Ben-David, A., Sterling, L., and Pao, Y. H. (1989). Learning and classification of monotonic ordinal concepts. Computational Intelligence, 5, 45-49. http://dx.doi.org/10.1111/j.1467-8640.1989.tb00314.x

[5] Makino, K., Suda, T., Ono, H., and Ibaraki, T. (1999). Data analysis by positive decision trees. IEICE Transactions on Information Systems, E82-D (1), 76-88.

[6] Potharst, R. and Bioch, J. C. (2000). Decision trees for ordinal clas-

sification. Intelligent Data Analysis, 4, 97-111.

[7] Cao-Van, K. and De Baets, B. (2003). Growing decision trees in an ordinal setting. International Journal of Intelligent Systems, 18, 733-750. http://dx.doi.org/10.1002/int.10113

[8] Feelders, A. and Pardoel, M. (2003). Pruning for monotone classification trees. Lecture Notes in Computer Science, 2810, 1-12. http://dx.doi.org/10.1007/978-3-540-45231-7_1

[9] Lievens, S., De Baets, B., and Cao-Van, K. (2006). A probabilistic framework for the design of instance-based supervised ranking algorithms in an ordinal setting. Annals of Operations Research, 163, 115-142. http://dx.doi.org/10.1007/s10479-008-0326-1

[10] Daniels, H. A. M. and Velikova, M. V. (2006). Derivation of monotone decision models from noisy data. IEEE Transactions on Systems, Man and Cybernetics - Part C, 36, 705-710. http://dx.doi.org/10.1109/TSMCC.2005.855493

[11] Ben-David, A. (1995). Monotonicity maintenance in information-theoretic machine learning algorithms. Machine Learning, 19, 29-43. http://dx.doi.org/10.1007/BF00994659

[12] Frank, E. and Hall, M. (2001). A simple approach to ordinal classification. The 12th European Conference on Machine Learning, 145-156.

[13] Popova, V. and Bioch, J. C. (2005). Monotone classification by function decomposition. Lecture Notes in Computer Science, 3735, 203-214. http://dx.doi.org/10.1007/11563983_18

[14] Dembczynski, K., Kotlowski, W., and Slowinski, R. (2009). Learning rule ensembles for ordinal classification with monotonicity constraints. Fundamenta Informaticae, 94, 163-179.

[15] De Loof, K., De Baets, B., and De Meyer, H. (2008). On the random generation of monotone data sets. Information Processing Letters, 107, 216-220. http://dx.doi.org/10.1016/j.ipl.2008.03.007

[16] Potharst, R., Ben-David, A., and van Wezel, M. (2009). Two algorithms for generating structured and unstructured monotone ordinal data sets. Engineering Applications of Artificial Intelligence, 22, 491-497. http://dx.doi.org/10.1016/j.engappai.2009.02.004

[17] Horvath, T., Eckhardt, A., Buza, K., Vojtas, P., and Schmidt-Thieme, L. (2011). Value-transformation for monotone prediction by approximating fuzzy membership functions. The 12th IEEE International Symposium on Computational Intelligence and Informatics, 367-372.

[18] Rademaker, M., De Baets, B., and De Meyer, H. (2012). Optimal monotone relabelling of partially non-monotone ordinal data. Optimization Methods and Software, 27(1), 17-31. http://dx.doi.org/10.1080/10556788.2010.507272

[19] Pijls, W. and Potharst, R. (2013). Repairing non-monotone data sets. Technical Report Erasmus University.