Repairing non-monotone ordinal data sets by changing class labels

Wim Pijls and Rob Potharst

Econometric Institute Report* EI 2014-29

Abstract

Ordinal data sets often contain a certain amount of non-monotone noise. This paper proposes three algorithms for removing these non-monotonicities by relabeling the noisy instances. The first one is a naive algorithm. The second one is a refinement of this naive algorithm which minimizes the difference between the old and the new label. The third one is optimal in the sense that the number of unchanged instances is maximized. The last algorithm is a refinement of the second. In addition, the runtime complexities are discussed.

1 Introduction

For ordinal classification problems the class values have an ordinal nature, while attribute values should be at least ordinal (they may be either ordinal or numeric). For such problems, the *monotonicity* constraint is often used as a leading principle for finding predictive models. The reason behind this is as follows: if object A is better than object B according to all criteria used, A should get a better rating (class value) than B, or at least, A should not get a lower rating than B. Examples include credit loan approval, investment decisions, selection problems and evaluation problems. Thus, there exists a fairly extensive literature about the construction of monotone predictive models (classification trees, neural networks, etc.) for ordinal classification problems, see for instance [1, 2, 4, 8]. Some of these methods only work properly if their input data set is strictly monotone. Other methods allow a certain degree of non-monotone noise in the data. Data sets in real life usually contain a relatively low percentage of non-monotone noise, which should be removed before some classification methods can be used. Moreover, the performance of a classification algorithm often improves when a strictly monotone data is used[6]. However, how to (optimally) remove non-monotone noise from a data set may also be considered as a legitimate research problem by itself [10].

So far, three approaches for this problem have been proposed in the literature. The first one is a greedy algorithm to relabel the non-monotone examples one at a time, presented by Daniels and Velikova [4]. At each step, it searches for the instance and the new label to maximize the increase in monotonicity of the data set. Although at each step, it maximizes the jump towards complete monotonicity, the algorithm relabels more examples than is needed. Also, the maximizing process at each step is costly, resulting in a relatively slow

*Econometric Institute, Erasmus University Rotterdam, P.O.Box 1738, 3000 DR Rotterdam, The Netherlands, e-mail: pijls@ese.eur.nl

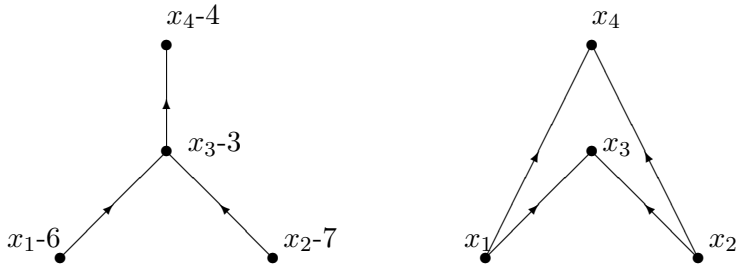

Figure 1: A data set as a poset (left) and the violation graph (right).

algorithm.
The second approach is presented by Rademaker et al.[10]; this algorithm solves a minimum flow problem in a transport network, that is associated with the non-monotone part of the data set, in order to find a maximum independent set in the violation graph of the non-monotone data set. This results in an algorithm that is optimal in the sense that it minimizes the number of relabelings performed to make the data set monotone.
The similar problem of relabeling a data set such that the total distance of the new labels to the old ones is minimized, is treated by Feelders [6] and Rademaker et al.[9], and can also be approached by isotonic regression methods, see for instance Dykstra, et al. [5] or Stout[11]. Our problem of minimizing the number of new labels, is equivalent to the use of 0/1-loss. However, this is not a convex function except for binary classification problems (i.e. when there are only two different label values). So the methods proposed by Feelders are not relevant to our problem. Neither is 0/1-loss equivalent to an L_p-distance, so also Dykstra's and Stout's methods do not apply. However, since these methods may be used to repair non-monotone ordinal data sets, we will compare their run-times with those of our algorithms.
In this paper we present three algorithms for removing all non-monotonic noise from a data set: 1) a naive algorithm, which appears to be a building block to the algorithms that follow, 2) the Borders algorithm, a fast alternative to the greedy algorithm mentioned above and 3) the Antichain algorithm, which minimizes the total number of relabelings.
The paper is structured as follows. In Section 2 the setting is described and a number of definitions is given. In Section 3 the naive algorithm and in Section 4 the Borders algorithm are proposed and proven to be correct. Section 5 contains similar content for the Antichain algorithm. In Section 6 we state the runtime complexity of the proposed algorithms and their competitors from the literature, and the conclusion follows in the last section.

2 Monotone data sets

Suppose we have a data set S with instances x and (class-) label $f(x)$. Each instance has a number of attribute values. Two instances x_1 and x_2 can be comparable or incomparable. In case of comparability we have $x_1 < x_2$ or $x_1 > x_2$ or $x_1 = x_2$.

Definition 1 *A data set S is called monotone if for each pair of instances x_1, x_2 from S:*
a) $x_1 < x_2 \Rightarrow f(x_1) \leq f(x_2)$,
b) $x_1 = x_2 \Rightarrow f(x_1) = f(x_2)$.

Definition 2 *An ordered pair x_1, x_2 is called a violation pair if $x_1 \leq x_2$ and $f(x_1) > f(x_2)$.*

Algorithm 1 Naive algorithm

1: **for all** $x \in M$ **do**
2: $f'(x) = f(x)$;
3: **end for**
4: $R = \emptyset$;
5: **while** $V \not\subset R$ **do**
6: select $x \in V \backslash R$;
7: choose $f'(x) \in [\ell(x), u(x)]$;
8: add x to R;
9: **end while**

Note that a data set is monotone iff it does not contain any violation pair.
The data set S splits into two disjoint subsets M and V. M is the set of instances that are not involved in any violation pair. V is the complementary set, thus the set of instances included in at least one violation pair.
The above set V is the underlying set of vertices in a directed graph $G = (V, E)$, where E is the set of arcs (x_1, x_2) such that x_1, x_2 is a violation pair. The graph $G = (V, E)$ is called the *violation graph* of data set S. This directed graph is anti-symmetric, so if $(x_1, x_2) \in E$, then $(x_2, x_1) \notin E$. Moreover, it is transitive, i.e. if $(x_1, x_2) \in E$ and $(x_2, x_3) \in E$, then also $(x_1, x_3) \in E$. Therefore, the violation graph can be regarded as a *poset* (partially ordered set).
A data set S with order relation $<$ is a poset only if S has no equal pairs, so $x_1 \neq x_2$ for any pair x_1, x_2. (This is not a requirement in the current paper.) A poset is usually pictured as a directed acyclic graph (transitive relations in the poset are not pictured). In the left part of Figure 1, a data set S is represented by a directed acyclic graph. The right part of Figure 1 shows its associated violation graph. The numbers in the graphs represent the f-values.

3 A Naive algorithm

For repairing a non-monotone data set Algorithm 1 is our first naive algorithm. The repaired label function is denoted by f'. The instances from V are put, after taking a repaired label, into set R. On termination, $R = V$ and hence $S = R \cup M$. This set with the new label f' is monotone, as will be proved. The new label is determined using two so-called border functions defined for $v \in V \backslash R$:

$$\begin{aligned} \ell(v) &= \max\{f'(y) \mid y \leq v \wedge y \in R \cup M\}, \\ u(v) &= \min\{f'(z) \mid v \leq z \wedge z \in R \cup M\}. \end{aligned}$$

As usual, $\ell(v) = -\infty$ (respectively, $u(v) = +\infty$), if an empty set is maximized (minimized). The correctness of the algorithm follows from Theorem 1.

Theorem 1 *An invariamt of the while loop is: the set $R \cup M$ with label function f' is monotone.*

Proof The invariant holds when the while loop starts. Then $R = \emptyset$ and M is monotone. Suppose a new instance x is selected to be added to R.
a) Let y be an instance in $R \cup M$ with $y < x$. (The case for an instance z with $x < z$ is

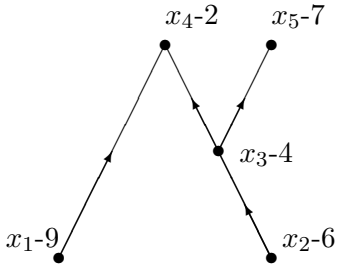

Figure 2: A data set S; set M should be taken into account.

analogous). Then $f'(y) \leq \ell(x) \leq f'(x)$ and the monotonicity is maintained.
b) Let y be an instance in $R \cup M$ with $y = x$. Then $\ell(x) = u(x) = f'(y)$ and hence $f'(x) = f'(y)$, which conforms to the definition of monotonicity. □

The set $R \cup M$ is identical to $S = V \cup M$ after execution and this set is monotone. So the algorithm is correct.
Algorithm 1 is not correct if the set $R \cup M$ is replaced with R in lines 7 and 8. In Figure 2 a data set S is displayed where $M = \{x_5\}$ and $V = \{x_1, x_2, x_3, x_4\}$. Suppose x_1 and x_2 are the first selected instances. Their class labels 9 and 6, respectively, are not changed. Subsequently, x_4 is selected and gets the new label 9. If, finally, x_3 is selected without considering M, its new label should be chosen from the interval $[6, 9]$. When $8 \in [6, 9]$ is chosen as the new label, a collision with x_5 is introduced.

4 The Borders algorithm

The naive algorithm can be made more specific by minimizing the deviations between the new labels and the original ones. This means that, if $f(x) \notin [\ell(x), u(x)]$ we choose $f'(x)$ to be equal to the closest border of the interval $[\ell(x), u(x)]$. Hence, if $f(x) < \ell(x)$ or $f(x) > u(x)$, we state $f'(x) = \ell(x)$ or $f'(x) = u(x)$ respectively. If $f(x) \in [\ell(x), u(x)]$, the obvious option remains $f'(x) = f(x)$. We call the naive algorithm with these additional specifications the *naive borders algorithm*.
When these new choices for relabeling are applied, a simpler computation of the borders is allowed, as we will show. The functions $\ell(v)$ and $u(v)$ can be replaced with new functions $\ell'(v)$ and $u'(v)$, $v \in V \backslash R$, defined as:

$$\begin{aligned} \ell'(v) &= \max\{f'(y) \mid y \in R \wedge (y, v) \in E\}, \\ u'(v) &= \min\{f'(z) \mid z \in R \wedge (x, z) \in E\}. \end{aligned}$$

Algorithm 2 displays the algorithm applying these new restricted border functions. These functions can be computed more speedily due to the inspection of a smaller set. Looking for a new label value $f'(x)$ for an instance x, only label values within R are inspected, instead of those within $R \cup M$. In addition, only arcs in the violation graph are taken into into consideration. (It is possible that $x < y$ with $y \in R$, but $(x, y) \notin E$; such an instance y is neglected.) So, the set of labels that need to be considered, has been substantially restricted.
Theorem 2 is the key to the correctness of the the Borders algorithm. Since the naive algorithm is correct, so is the Borders algorithm.

Algorithm 2 The Borders algorithm

```
1: for all x ∈ M do
2:    f'(x) = f(x);
3: end for
4: R = ∅;
5: while V ⊄ R do
6:    select x ∈ V\R;
7:    if f(x) < ℓ'(x) then
8:       f'(x) = ℓ'(x);        // x is an ascending instance
9:    else
10:      if f(x) > u'(x) then
11:         f'(x) = u'(x);   // x is a descending instance
12:      else
13:         f'(x) = f(x);   // x is a neutral instance
14:      end if
15:   end if
16:   add x to R;
17: end while
```

Lemma 1 *During execution of the while loop, any instance $v \in V\backslash R$ satisfies*
a) $\ell'(v) < \ell(v) \leq f(v)$ *or* $\ell'(v) = \ell(v)$*;*
b) $u'(v) > u(v) \geq f(v)$ *or* $u'(v) = u(v)$*.*

Proof of a)
The relation $\ell'(v) < \ell(v) \leq f(v)$ holds before the while loop starts. As soon as an instance $x < v$ with $f'(x) \geq \ell(v)$ is added to S, this element x generates values for $\ell'(v)$ and $\ell(v)$ such that $f'(x) = \ell'(v) = \ell(v)$. From that time R contains an element with maximal f'-value in the set $M \cup R$ and hence the equality $\ell'(v) = \ell(v)$ is maintained as long as $v \in V\backslash R$.
Part b) is symmetric □

Theorem 2 *If the Borders algorithm and the naive borders algorithm select the instances x in the same order, the function $f'(x)$ in the Borders algorithm is identical to the corresponding function in the naive borders algorithm.*

Proof
The three categories, ascending, descending and neutral, are considered. We show that for these three categories $f'(x)$ in the Borders algorithm receives the values $\ell(x)$, $u(x)$ and $f(x)$ respectively.
An ascending instance x satisfies the inequality $f(x) < \ell'(x) \leq \ell(x)$. As a consequence of Lemma 1, we must have $\ell'(x) = \ell(x)$ and hence $f'(x) = \ell'(x) = \ell(x)$.
The case that x is descending can be treated analogously.
For a neutral instance x we have $\ell'(x) \leq f(x) = f'(x) \leq u'(x)$. If $\ell'(x) < \ell(x)$, Lemma 1 implies that $f'(x) = f(x) \geq \ell(x)$. If, on the other hand, $\ell'(x) = \ell(x)$, the equality $f'(x) = f(x) \geq \ell'(x) = \ell(x)$ applies. Similarly, we can prove $f'(x) = f(x) \leq u(x)$. □

5 The Antichain algorithm

An *antichain* in a poset is a subset of vertices that do not share any arc. Suppose the instances in the violation graph have been furnished with a new label f', such that the total

Algorithm 3 The Antichain algorithm

1: **for all** $x \in M$ **do**
2: $f'(x) = f(x)$;
3: **end for**
4: $R = \emptyset$;
5: **while** $A \not\subset R$ **do**
6: select $x \in A \backslash R$
7: $f'(x) = f(x)$;
8: add x to R;
9: **end while**
10: **while** $U \not\subset R$ **do**
11: select $x \in U \backslash R$;
12: $f'(x) = \ell''(x)$;
13: add x to R;
14: **end while**
15: **while** $D \not\subset R$ **do**
16: select $x \in D \backslash R$;
17: $f'(x) = u''(x)$;
18: add x to R;
19: **end while**

data set S is monotone. The neutral instances, i.e. the instances that did not receive a new label, constitute an antichain in the violation graph. (For, if two instances x_1 and x_2 that did not receive a new label, share an arc in the violation graph, their non-monotonicity would not have been removed.) Thus, the number of neutral instances is at most equal to the size of the maximum antichain of the violation graph. So, to find a relabeling with a maximum number of neutral points, we should construct a maximum antichain A of the violation graph.
To find a maximum antichain in a poset the reader is referred to [7]. This problem can be translated to a maximum flow problem which may be solved efficiently by the methods of [3]. Note that a maximum antichain is not unique, in general.
For a given maximum antichain A we define the *up-set* U and the *down-set* D of A as follows:

$$\begin{aligned} U &= \{x \mid \exists a \;\; a \in A \wedge (a, x) \in E\}, \\ D &= \{x \mid \exists a \;\; a \in A \wedge (x, z) \in E\}. \end{aligned}$$

The sets U and D are disjoint. For, if $U \cap D$ included an instance x, the violation graph would contain arcs (a, x) and (x, a') and thus also (a, a') with $a, a' \in A$, which is not allowed in antichain A. Thus, set V can be partitioned as follows : $V = A \cup U \cup D$.
In relation to these sets we define the following functions for any $v \in V \backslash R$:

$$\begin{aligned} \ell''(v) &= \max\{f'(a) \mid a \in A \wedge (a, v) \in E\}, \\ u''(v) &= \min\{f'(b) \mid b \in A \cup U \wedge (v, b) \in E\}. \end{aligned}$$

Algorithm 3 is a special case of the Borders algorithm, where a maximum antichain A plays a crucial role. In the first while loop the labels of the antichain instances remain unchanged, in the second while loop the instances of the up-set U get new labels, considering only labels

from A; in the third while loop, the same is done for the down-set D, considering A and the new labels of U.

Lemma 2 *During execution of the second while loop, $f(v) < \ell''(v) = \ell'(v)$ for any $v \in U \backslash R$. During execution of the third while loop, $f(v) > u''(v) = u'(v)$ for any $v \in D \backslash R$.*

Proof
The relation holds when the second while loop starts, since $R = A$. Notice that $\ell''(v)$ does not change during execution of the loop.
Assume an instance x is added to R with $(x, v) \in E$ and $f'(x) > \ell'(v)$. (Otherwise, $\ell'(v)$ does not change.) Then, there must be an instance $a \in A$ with $f'(a) = f'(x) > \ell'(v) = \ell''(v)$. Since also $(a, v) \in E$, the inequality $f'(a) > \ell''(v)$ is not compatible with the definition of $\ell''(v)$.
The proof for the third while loop is similar. □.

Theorem 3 *If the Antichain algorithm and the Borders algorithm select the instances x in the same order, the function $f'(x)$ in the Antichain algorithm is identical to the corresponding function in the Borders algorithm.*

Proof In the first while loop of the Antichain algorithm x is a neutral instance. Lemma 2 shows that the second while loop deals with only ascending instances and the third loop with only descending instances. The new labels are equal to the functions of the Borders algorithm according to Lemma 2. □

As mentioned before, there may be several maximum antichains. The above algorithm is correct for any choice of a maximum antichain. When one chooses a maximum antichain with high label values, a so-called optimistic relabeling is obtained. Analogously, a maximum antichain with low label values results in a pessimistic relabeling.

6 The runtime complexity of the proposed algorithms

In this section we consider the cost of each of the proposed algorithms and their competitors from the literature in terms of runtime complexity. Each of the proposed algorithms starts with determining the violation graph $G = (V, E)$. This means that each instance should be compared with all other instances, and this costs $O(n^2)$, where n is the number of instances in data set S. The naive algorithm has runtime complexity $O(n|V|)$ and the Borders algorithm $O(|V|^2)$. The Antichain algorithm consists of two parts: first, a maximum antichain A should be determined, which can be done in $O(|V|^3)$, according to [3]; the second part, the relabeling of all instances of $V \backslash A$ has runtime complexity $O(|V \backslash A|^2)$.
As to the competing algorithms from the literature, the greedy Daniels and Velikova algorithm from [4] has runtime complexity $O(n^3C)$, where C is the number of class values. Rademaker et al.'s algorithm [10] resembles our Antichain algorithm, but considers the full set S, so its runtime complexity is $O(n^3)$. The algorithm proposed by Feelders [6] has runtime complexity $O(n^3C)$, and the fastest method to perform isotonic regression costs $O(n^3 log n)$, according to [11].

7 Conclusion and future work

In this paper we propose three related algorithms to repair a non-monotone data set by changing the class labels of some instances contained in the data set: the naive algorithm,

the Borders algorithm and the Antichain algorithm. The naive algorithm is the easiest to describe, and it acts as a building block for the other two algorithms. However, on all data sets it is slower than the Borders algorithm.
The gain of the Borders algorithm over the naive algorithm is that during relabeling the new label chosen depends only on labels in the violation set. So, if the violation set is small compared to the complete data set (which is often the case), this is a significant gain.
For most data sets the Borders algorithm is the fastest of the three; however, it is not optimal: it may relabel more instances than is actually needed. The Antichain algorithm is optimal; it relabels the least possible number of instances. However, finding a maximum antichain of the violation graph (as should be performed before running the antichain algorithm) is costly, except in cases where the violation graph is small.
In comparison with existing algorithms for repairing non-monotone data sets, the Borders algorithm is again the fastest. If we restrict our attention to optimal algorithms (in any sense) the Antichain algorithm. Rademaker's algorithms is similar but considers the full set S for relabeling where our algorithm considers only V.
To our knowledge, an algorithm similar to the Borders algorithm is not mentioned in the literature. As far as we know, there are no studies on how a non-monotone data set can be repaired by changing attribute values, instead of class labels. Also, both class labels and attribute values could be allowed to change. This topic might be interesting for future work.

References

[1] N. Barile and A. Feelders. Nonparametric monotone classification with MOCA. In *Proceedings of the Eighth IEEE International Conference on Data Mining*, pages 731–736, 2008.

[2] K. Cao-Van and B. De Baets. Growing decision trees in an ordinal setting. *Int. J. of Intelligent Systems*, 18:733–750, 2003.

[3] B.V. Cherkassky and A.V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19:390–410, 1997.

[4] H. Daniels and M. Velikova. Derivation of monotone decision models from noisy data. *IEEE Transactions on Systems, Man and Cybernetics–Part C*, 36(5):705–710, 2006.

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

[6] A. Feelders. Monotone relabeling in ordinal classification. In *Proceedings of the Tenth IEEE International Conference on Data Mining, ICDM'10*, 2010.

[7] W. Pijls and R. Potharst. Another note on Dilworth's decomposition theorem. *Journal of Discrete Mathematics*, 2013. http://dx.doi.org/10.1155/2013/692645.

[8] R. Potharst and J.C. Bioch. Decision trees for ordinal classification. *Intelligent Data Analysis*, 4:97–111, 2000.

[9] M. Rademaker, B. De Baets, and H. De Meyer. Loss optimal monotone relabeling of noisy multi-criteria data sets. *Information Sciences*, 179:4089–4096, 2009.

[10] M. Rademaker, B. De Baets, and H. De Meyer. Optimal monotone relabelling of partially non-monotone ordinal data. *Optimization Methods and Software*, 27(1):17–31, 2012.

[11] Q. F. Stout. Isotonic regression via partitioning. *Algorithmica*, 66:93–112, 2013.