Exploiting Monotonicity Constraints in Active Learning for Ordinal Classification

Pieter Soons

Ad Feelders

Technical Report UU-CS-2014-001
January 2014

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Exploiting Monotonicity Constraints in Active Learning for Ordinal Classification

Pieter Soons
Universiteit Utrecht
Utrecht, The Netherlands
e-mail: p.s@live.nl

Ad Feelders
Universiteit Utrecht
Utrecht, The Netherlands
e-mail: A.J.Feelders@uu.nl

Abstract

We consider ordinal classification and instance ranking problems where each attribute is known to have an increasing or decreasing relation with the class label or rank. For example, it stands to reason that the number of query terms occurring in a document has a positive influence on its relevance to the query. We aim to exploit such monotonicity constraints by using labeled attribute vectors to draw conclusions about the class labels of order related unlabeled ones. Assuming we have a pool of unlabeled attribute vectors, and an oracle that can be queried for class labels, the central problem is to choose a query point whose label is expected to provide the most information. We evaluate different query strategies by comparing the number of inferred labels after some limited number of queries, as well as by comparing the prediction errors of models trained on the points whose labels have been determined so far. We present an efficient algorithm to determine the query point preferred by the well-known active learning strategy generalized binary search. This algorithm can be applied to binary classification on incomplete matrix orders. For non-binary classification, we propose to include attribute vectors in the training set whose class labels have not been uniquely determined yet. We perform experiments on artificial and real data.

Keywords: Active learning, ordinal classification, monotonicity constraints.

1 Introduction

In some learning problems, we know for each attribute whether it has a positive or negative influence on the response variable. For example, it stands to reason that the number of query terms occurring in a document has a positive influence on its relevance to the query. In the context of ordinal classification or instance ranking problems, prior knowledge of this kind provides us with the useful constraint that if a scores better (not worse) on all attributes than b, then the class label of a can't be lower than the class label of b. For example, if a and b apply for a loan, and a scores better than b on all criteria, then it would be rather surprising if the application of b is accepted, but a's is rejected. Quite some work has been done on the development of algorithms that enforce such monotonicity constraints on the models they produce, see for example [6, 1].

We propose to use this prior knowledge in the context of active learning to augment the training sample with data points whose labels can be (partially) inferred from those of order related points whose label was provided explicitly by an oracle. We consider the pool-based active learning setting where we initially have a pool of unlabeled data points, and we can ask an oracle for the label of (a limited number of) data points. The central problem is to decide which data point to submit to the oracle. Generally speaking, we would like to submit one for which we expect it will enable us to draw conclusions about the labels of many other points. The problem is how to formalize this general notion, and to find efficient algorithms to determine the corresponding query points.

This paper builds on the work of Barile and Feelders in [3]. The most important extensions are:

1. For binary classification, we have developed an efficient algorithm to evaluate the well-known active learning heuristic generalized binary search (see section 3.1). This algorithm can be used in case the order on the data points is an incomplete matrix order, or 2-dimensional order.
2. For non-binary classification, we have developed a way to include data points into the training sample whose label has not been uniquely determined yet (see section 3.2).

We compare the performance of these new algorithms with those of [3] in section 5. We have furthermore developed an instance of the Propp-Wilson algorithm with sandwiching that enables us to draw a random monotone classification over a given partial order. This algorithm is used to generate artificial data sets for the evaluation of our active learning algorithms, and is described in section 4. We first discuss some preliminaries however.

2 Preliminaries

Let $\mathcal{X}$ be an *attribute space* $\mathcal{X} = \mathcal{X}^1 \times \mathcal{X}^2 \times \ldots \times \mathcal{X}^p$ consisting of vectors $x = (x^1, x^2, \ldots, x^p)$ of values on p attributes. We assume that each attribute takes values in a linearly ordered set $\mathcal{X}^h$. The partial ordering $\preceq$ on $\mathcal{X}$ is the ordering induced by the order relations of its coordinates $\mathcal{X}^h$:

$$x_i \preceq x_j \Leftrightarrow \forall h = 1, \ldots, p : x_i^h \leq x_j^h.$$

Likewise, let $\mathcal{Y}$ be a finite linearly ordered set of *classes*. Without loss of generality, we assume that $\mathcal{Y} = \{1, 2, \ldots, k\}$ where k is the number of classes.

We have available an unlabeled training sample $X = \{x_1, \ldots, x_n\}$ with $X \subset \mathcal{X}$. The monotonicity assumption states that

$$x_i \preceq x_j \Rightarrow y_i \leq y_j. \tag{1}$$

This constraint expresses the prior knowledge that the class label is increasing in each of the attributes, and it is the basis of all inference. If the class label is decreasing rather than increasing in an attribute, we can simply invert the values of the attribute. A classification that satisfies (1) is called a monotone classification. The degree of comparability of X is the fraction of *pairs* of data points (x_i, x_j) with $x_i \preceq x_j$ or $x_j \preceq x_i$.

A lower set L is a subset of X that contains the downward closure of all its elements:

$$x_i \in L, x_j \preceq x_i \Rightarrow x_j \in L.$$

Likewise, an upper set U of X contains the upward closure of all its elements. The downset $\downarrow (x_i)$ of x_i contains all elements of X that are smaller than or equal to x_i:

$$\downarrow (x_i) = \{x_j \in X : x_j \preceq x_i\}.$$

The upset $\uparrow (x_i)$ of x_i is defined analogously. We use $d(x_i) = |\downarrow (x_i)|$ to denote the size of the down set of x_i, and $u(x_i)$ to denote the size of its upset.

3 Active Learning

We consider the setting of pool-based active learning. A fundamental assumption in this paper is that the oracle always produces monotone class labels, that is, class labels that satisfy constraint (1). We first consider the case of binary classification; section 3.2 discusses the non-binary case.

3.1 Binary Classification

Dasgupta [4] analyzes an active learning strategy that can be summarized as follows. Let H denote a hypothesis class (e.g. the class of monotone functions), and let $\widehat{H}$ denote the effective hypothesis class for a given sample of unlabeled points $X = \{x_1, \ldots, x_n\}$ (e.g. the class of monotone functions on X). Let π denote a probability distribution over $\widehat{H}$. The objective is to determine the unique $h \in \widehat{H}$ that is consistent with all the hidden labels, by querying as few of them as possible. Let $S \subseteq \widehat{H}$ be the set of hypotheses that is consistent with the labels queried so far. For each unlabeled x_i, let S_i^+ be the hypotheses which label x_i positive and S_i^- the ones which label it negative. The proposed strategy, which we will henceforth refer to as generalized binary search (or GBS for short) is to pick the x_i for which these sets are most nearly equal in probability, that is, the x_i for which $\pi(S_i^-)$ is closest to $\frac{1}{2}$. Another (equivalent) interpretation is that the query point whose class distribution has the highest entropy is selected. Dasgupta [4] shows that if the optimal query strategy (that is, the one that requires the fewest queries in expectation to determine h) requires Q^* queries in expectation, then the expected number of queries needed by GBS is at most $4Q^* \ln 1/(\min_h \pi(h))$. In particular, if π is the uniform distribution, it requires at most $4 \ln |\widehat{H}| Q^*$ queries in expectation. Henceforth we assume that π is uniform.

Let us consider how we can implement GBS for our problem. Let $\widehat{H}$ be the collection of all monotone binary classifications of the pool of unlabeled points X. Note that there is a one-to-one correspondence between monotone binary classifications on $(X, \preceq)$, and lower sets of $(X, \preceq)$: given a lower set $L \subseteq X$, the classification function

$$f(x) = \begin{cases} 1 & \text{if } x \in L \\ 2 & \text{otherwise} \end{cases}$$

is monotone, and vice versa, given a monotone classification f, the set

$$L = \{x \in X : f(x) = 1\}$$

is a lower set of X. Hence, to determine the size of S_i^- we need to count the number of lower sets that include x_i. Unfortunately, counting the lower sets of a partial order is #P complete [8]. For this reason, Barile and Feelders [3] proposed the easy-to-compute GREEDY heuristic

$$x^* = \arg\max_{x \in X} \min\{d(x), u(x)\},$$

where x^* denotes the selected query point. This heuristic maximizes the number of labels we can infer in the worst case, and only requires the computation of the size of the upset and downset of each potential query point.

We note however that for certain types of partial order the problem of counting lower sets is tractable. For example, the number of lower sets of a chain of length n is $n + 1$, and both GBS and GREEDY query the point in the middle of the chain. This is known to be optimal, and the number of queries required to determine all labels is $1 + \log_2 n$. A somewhat less restrictive order is the matrix order. We get an $m \times n$ matrix order for example if we have two attributes X_1 and X_2, with domain sizes $|\mathcal{X}^1| = m$ and $|\mathcal{X}^2| = n$. We have developed an efficient dynamic programming algorithm to count the lower sets for this type of order. We describe this algorithm in the remainder of this section (a proof of its correctness is given in Appendix A). Since it might be the case that not all possible vectors (i, j) $(i = 1, \ldots, m,\ j = 1, \ldots, n)$ are observed in the sample, we must take into account the possibility that some of the cells of the matrix are "empty". We count the lower sets of the partial order: $(i, j) \preceq (k, l) \Leftrightarrow i \leq k$ and $j \leq l$. Let $D_{i,j}$ denote the number of lower sets of the part of the matrix with row $\leq i$ and column $\geq j$. To count the lower sets of the whole matrix, we can use the following recursion:

$$D_{i,j} = \begin{cases} D_{i-1,j} + D_{i,j+1} - D_{i-1,j+1} & \text{if } (i,j) \text{ is empty} \\ D_{i-1,j} + D_{i,j+1} & \text{otherwise} \end{cases}$$

All we need to kick start the process is to determine the counts for row 1 and column n, but this is straightforward since the corresponding blocks are linear orders. If cell $(1, n)$ happens to be empty we enter a count of 1 since the empty set is a lower set.

4	22	18	10	3
3	12	8	7	3
2	9	5	4	2
1	4	3	2	1
	1	2	3	4

(a) Matrix D

4	1	2	3	3
3	1	2	5	8
2	2	3	8	16
1	3	6	14	22
	1	2	3	4

(b) Matrix E

4	18	10	3	1
3	18	17	9	3
2	18	17	13	8
1	21	19	16	16
	1	2	3	4

(c) Matrix C

Table 1: Counts for an example 4×4 matrix order. The grey cells correspond to attribute vectors that are not observed in the sample. The cells are numbered so as to correspond with the ordering in the plane, e.g., $(1, 1)$ is the bottom-left cell.

Consider the example of the 4×4 matrix given in Table 1(a). The grey cells are empty. To compute $D(3, 2)$ for example, we add $D(2, 2)$ and $D(3, 3)$, and because $(3, 2)$ is an empty cell, we subtract $D(2, 3)$ giving a count of 8 lower sets. We read in cell $(4, 1)$ that this matrix order has 22 lower sets (monotone binary classifications) in total.

This establishes the total number of lower sets of a matrix order, but we need to count the number of lower sets that include x. As it turns out this can be done with a few extra steps. The number of lower sets of X that include x is equal to the number of lower sets of $X \setminus \downarrow x$. This number cannot be read off immediately from D, except for the cells in the top row. For example, the down set of cell $(4, 1)$ is the entire first column of the matrix, so the required count can be found in cell $(4, 2)$. To compute the counts for cells not in the top row, we first define a matrix E which is constructed in a similar fashion as D, but working in the opposite direction, i.e. from the upper-left cell to the bottom-right cell. Cell (i, j) of E contains a count of the lower sets of the block extending from the upper-left cell to cell (i, j). The total count will now appear in the bottom-right cell of E (see Table 1(b)). Finally we define the matrix C where $C(i, j)$ contains the number of lower sets that include cell (i, j).

$$C_{i,j} = \begin{cases} D_{i,j+1} & \text{if } i = m \\ C_{i+1,j} + D_{i,j+1} \times (E_{i+1,j} - E_{i+2,j}) & \text{otherwise,} \end{cases}$$

where, all "out of bounds" cells should be taken to contain a count of 1. The matrix C for our example is given in Table 1(c). We can read from it for example that the number of lower sets that include $(2, 3)$ is 13. To determine the complexity of this algorithm, we note that we need to perform just a few elementary computations for each cell in the matrix, so the complexity is $O(mn)$, which is proportional to the number of observed data vectors.

Let us compute which query point would be selected by the GBS heuristic. In Table 2 we have listed the data required for its evaluation. Here we see that the class label distribution for the cell $(4, 2)$ has the highest entropy: in 45% (10 out of 22) of all monotone classifications it has label 1, and in 55% it has label 2. Since the entropy for all other query points is smaller, GBS will select $(4, 2)$. The necessary computations to evaluate the GREEDY heuristic are summarized in Table 3. If we query for the label of cell $(2, 3)$ we can determine the label of 5 cells, regardless of whether the true label is 1 or 2. All other cells have a lower worst case outcome, so the cell $(2, 3)$ is selected by GREEDY.

Finally, we note that after querying for the label of x, the labels of either the downset (if the label is 1) or the upset (if the label is 2) of x are determined due to the monotonicity constraint, and hence the corresponding vectors can be removed from the partial order. The remaining order is again an incomplete matrix order, so the next query point can be determined in the same way.

3.2 Non-binary Classification

In the binary case, the GBS heuristic selected the query point whose class distribution had the largest entropy. This principle can be extended to the non-binary case, but it requires even more complex computations, as was to be expected.

cell	$C(i,j)$	$P(y=1)$	cell	$C(i,j)$	$P(y=1)$
(1,1)	21	.95	(2,3)	13	.59
(2,1)	18	.82	(3,3)	9	.41
(1,2)	19	.86	(4,3)	3	.14
(4,2)	10	.45	(2,4)	8	.36
(1,3)	16	.73	(3,4)	3	.14

Table 2: Computations necessary for evaluating GBS: for every non-empty cell, the number of lower sets $C(i,j)$ that contain that cell, and the corresponding probability that this cell has class label 1 are given. GBS selects the query point $(4,2)$

cell	d	u	min	cell	d	u	min
(1,1)	1	10	1	(2,3)	5	5	5
(2,1)	2	7	2	(3,3)	6	3	3
(1,2)	2	8	2	(4,3)	8	1	1
(4,2)	4	2	2	(2,4)	6	2	2
(1,3)	3	6	3	(3,4)	8	1	1

Table 3: Computations necessary for evaluating GREEDY: for every non-empty cell, the size d of its downset, the size u of its upset, and the minimum of them is given. GREEDY selects the query point $(2,3)$.

Given a nested sequence $L_1 \subseteq L_2 \subseteq \ldots \subseteq L_{k-1}$ of lower sets of X, the classification function:

$$f(x) = \begin{cases} j & \text{if } x \in L_j \setminus \bigcup_{i<j} L_i, \quad j = 1, \ldots, k-1 \\ k & \text{otherwise} \end{cases},$$

is monotone. Conversely, given a monotone classification function $f(x), x \in X$, define $L_i = \{x \in X : f(x) \leq i\}$, $i = 1, \ldots, k-1$. The sequence $L_1, \ldots, L_{k-1}$ is a nested sequence of lower sets of X. Hence, to count the number of monotone classifications on X, we can count the number of nested sequences of lower sets of X instead. We have not been able to find an efficient way to do this even for a matrix order.

Therefore we extend the GREEDY heuristic to non-binary classification problems, as proposed in [3]. It doesn't make much sense now to count the number of class labels that can be inferred in the worst case, as that number will typically be zero (or one if you include the the label of query point itself) for all potential query points. Hence, we switch to counting the number of labels that can be *eliminated* in the worst case instead.

Let $[\ell_i, h_i]$ denote the interval of possible labels for x_i. Initially we have $\ell_i = 1$ and $h_i = k$ for all i, but as we learn the labels of points, using the monotonicity constraint these bounds are adjusted. Let $N(x_i, y)$ denote the number of values we can *eliminate* when $y_i = y$. We have

$$N(x_i, y) = \sum_{x_j \in \downarrow(x_i)} (h_j - y)_+ + \sum_{x_j \in \uparrow(x_i)} (y - \ell_j)_+,$$

where $z_+ = \max(0, z)$. Maximizing the worst case we select the query point

$$x^* = \arg\max_{x_i \in X} \min_{y \in [\ell_i, h_i]} \{N(x_i, y)\}.$$

As the number of class labels k increases, it becomes more and more difficult to infer unique class labels for points that are not queried. This means that it becomes very difficult to infer a sizeable training set from just a limited number of queries. On the other hand there may be quite a few points whose set of possible labels has been reduced considerably by inference from the query points. If we only include points whose label has been uniquely determined in the training set, then we completely ignore this potentially valuable information. Therefore we propose to exploit partial label information as follows. Let x_i be an attribute vector whose label set has been reduced to the interval $[\ell_i, h_i]$. The size of this interval is $s_i = h_i - \ell_i + 1$. We add s_i copies of x_i to the training set, with respective labels $\ell_i, \ldots, h_i$. Each copy gets the weight s_i^{-1}.

The question remains how many labels have to be eliminated before we decide to include a point in the training sample. This is determined by the value of the parameter $a \in [0, 1]$, where x_i is included if $s_i \leq a \cdot k$. So if $a = \frac{1}{2}$, we include a point if at least half of its possible labels have been eliminated. Algorithm 1 shows how we use the partial label information to predict the class of a new data point z with a K-nearest neighbor classifier. In the call, T denotes the training sample. The call to GetNearestNeighbors returns the K nearest neighbors of z in T, where only attribute vectors x_i in T with $s_i \leq a \cdot k$ are considered. In predicting the label of z, the weight of a training point is inversely proportional to its distance to z. In case a training point has multiple possible labels, the weight is distributed evenly over the different labels. To get the final prediction, the weighted average is rounded to the nearest integer. We will use Algorithm 1 to assess the quality of a query strategy by measuring the prediction error on an independent test sample of this classifier, using a training sample T that has been constructed with that query strategy. It should be noted that the classifier is merely an instrument to assess the quality of a query strategy, so we have not attempted to find the best classifier. The partial label information can also be used by other classification algorithms that allow the specification of case weights in training.

Algorithm 1 Classify(z,K,T,a)

```
1: $y \leftarrow 0$
2: $w \leftarrow 0$
3: $N \leftarrow$ GetNearestNeighbors($z$,$K$,$T$,$a$)
4: for all $x_i \in N$ do
5:     $d \leftarrow$ Distance$(z, x_i)$
6:     $s_i \leftarrow h_i - \ell_i + 1$
7:     for all $j = \ell_i \rightarrow h_i$ do
8:         $y \leftarrow y + j \times 1/(s_i \times d)$
9:         $w \leftarrow w + 1/(s_i \times d)$
10:    end for
11: end for
12: return Round$(y/w)$
```

4 Generating random monotone functions

In this section we present an algorithm to uniformly sample a random monotone classification function with (at most) k labels, defined on a given partial order $(X, \preceq)$. This algorithm will be used to generate artificial data sets for the evaluation of the proposed active learning heuristics. It is an instance of the Propp-Wilson algorithm [7] with sandwiching.

Recall that we established a one-to-one correspondence between nested sequences of lower sets and monotone classification functions in section 3.2. Let $\mathcal{S}$ denote the collection of all nested sequences of lower sets on $(X, \preceq)$. We define an order $(\mathcal{S}, \preceq_{\mathcal{S}})$ on the elements S of $\mathcal{S}$ as follows. Let $S_1 = L_1^{S_1}, \ldots, L_{k-1}^{S_1}$ and $S_2 = L_1^{S_2}, \ldots, L_{k-1}^{S_2}$ denote two nested sequences of lower sets. The order is defined as:

$$S_1 \preceq_{\mathcal{S}} S_2 \Leftrightarrow \forall i = 1, \ldots, k-1 : L_i^{S_2} \subseteq L_i^{S_1}.$$

Note that this order has a least element $S_\perp = \forall i = 1, \ldots, k-1 : L_i = X$ (all elements of X get the smallest label), and a greatest element $S^\top = \forall i = 1, \ldots, k-1 : L_i = \varnothing$ (all elements of X get the largest label).

We set up a number of Markov chains that have $\mathcal{S}$ as their state space. Let $\phi_t(S, u)$ denote the update function, where $S \in \mathcal{S}$ is the current state and u is a uniformly distributed random number in the interval $(0, 1)$. The chains are coupled, that is, the same random number u_t is used by all chains. Therefore, once two chains "meet" in the same state, they become "stuck forever". Such chains are said to have coalesced. Now suppose the update function is monotone, that is

$$S_1 \preceq_{\mathcal{S}} S_2 \Rightarrow \phi_t(S_1, u_t) \preceq_{\mathcal{S}} \phi_t(S_2, u_t).$$

If we have coupled chains that start in S_1, S_2 and S_3 with $S_1 \preceq_{\mathcal{S}} S_2 \preceq_{\mathcal{S}} S_3$, and at time t, the chains started at S_1 and S_3 have coalesced, then the monotone update function guarantees that the chain that started in S_2 has the same value. If in the partial order we have a least element and a greatest element, then we only need two chains, started in these two states. Once these two chains converge, all chains started in other states would have converged to the same value.

Hence, for the algorithm to work, it is essential to use a monotone update function. In Appendix B we present the update function, and prove that it is monotone.

To summarize the algorithm: we use two coupled chains, started at $S_{\perp}$ and $S^{\top}$ and run them "in the past" from time $-T$ to -1. If after this time, the chains have coalesced, we use their value as our sample. If they have not coalesced, we increase the amount of time and run the chains again. Once the chains have converged at $t = 0$ for some T, we would still get the same answer if we had started the chains even earlier. So in effect, we get a sample from a chain that has been run for an infinite amount of time. Propp and Wilson [7] show that by running this Markov chain we generate a uniform random element from $\mathcal{S}$.

5 Experiments

To evaluate the different query strategies, we compare their performance on artificial and real data sets. We would like to stress that we have not attempted to systematically optimize the parameters of the classification model (nearest neighbor), because it is merely instrumental in measuring the quality of the different query strategies. The value of K for the K-nearest neighbor algorithm was fixed at 5 because preliminary experiments demonstrated that this value worked quite well. Throughout the experiments we fixed the ambiguity parameter a at $\frac{1}{2}$, as it makes sense to only use the partial label information if more than half of the possible labels have been ruled out.

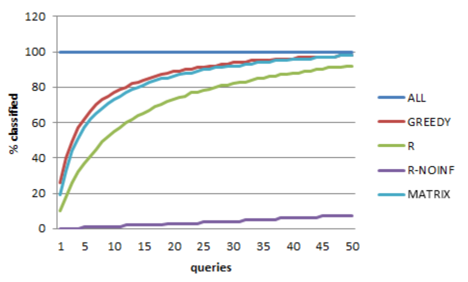
5.1 Experiments on Artificial Data

We generate artificial data in two different ways, one specifically intended for the incomplete matrix order, and one more general method. To generate data for the matrix order, we pick a value m for the number of rows and columns of the matrix (for simplicity we assume the matrix is square). Furthermore, we choose a value $d \in [0, 1]$ which denotes the fraction of observed cells. Given an incomplete matrix order generated in this way, we use the algorithm described in section 4 to generate a random monotone binary classification on this order. We only consider *binary* classifications on matrix orders, so that we can use the efficient lower sets counting algorithm of section 3.1 to compute the GBS query point.

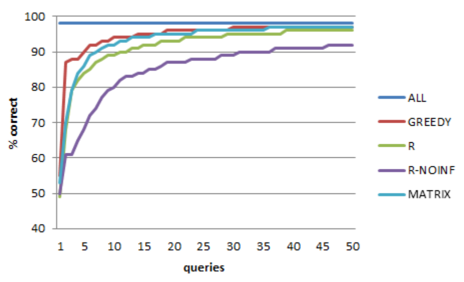
Table 4 and Figure 1 summarize some of our findings. In Table 4 we see for example that the GREEDY heuristic only needs 12 queries to infer 80% of the labels in the training set consisting of two thirds of $n = 1250$ is 833 attribute vectors. With respect to the number of inferred labels, GREEDY dominates GBS for a long time but is taken over by it at the very end: when it comes to classifying 100% of the training sample GBS requires fewer queries. Note that in the beginning we can infer many labels from just a few queries, but to obtain the last 20% of labels requires relatively many queries. For example, for $m = 50$ and $d = 0.2$, GBS requires 13.4 queries on average to infer 80% of the labels , but to obtain the last 20% an additional 28.6 queries are needed. Likely this is because towards the end, the partial order disintegrates into a number of smaller disconnected components, which limits the scope for inference. Figure 1 gives the detailed

	$m = 50, d = 0.2$				$m = 50, d = 0.5$			
% labeled	20%	50%	80%	100%	20%	50%	80%	100%
GREEDY	1	3	11	45	1	3.1	12	67
GBS	2.1	3.8	13.4	42	2.1	3.8	14	56
RANDOM	1.1	8	23	56	1.1	8.3	27	95

Table 4: Average number of queries needed to classify stated % of vectors averaged over 100 samples.



(a) % labeled vectors



(b) accuracy on test set

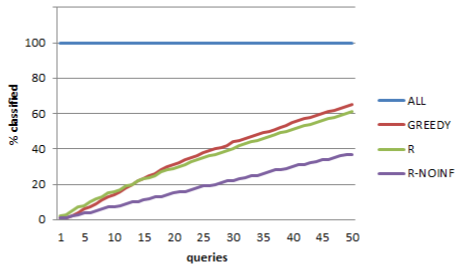
Figure 1: Average results on 100 artificial data sets
($k = 2, m = 50, d = 0.5$)

results for different query strategies for $k = 2$, $m = 50$, and $d = 0.5$. The RANDOM (R in Figure 1) strategy picks an unlabeled point at random but does use the monotonicity constraint to infer the labels of other points. The R-NOINF strategy doesn't even use the monotonicity constraint, so its training sample consists only of the points that were explicitly labeled by the oracle. It was merely included to show the benefit of the constraint. The line labeled ALL gives the results obtained when the complete labeled training sample is used, and provides a natural upper bound for performance. The results of GBS are given by the line labeled MATRIX in Figure 1.

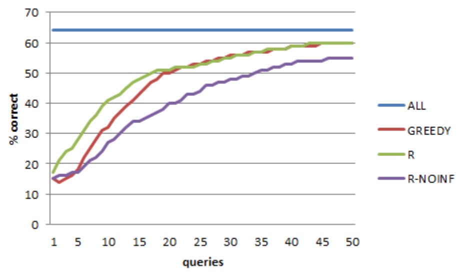
To study the behavior in case there are more than two class labels, we generate data from a bivariate normal distribution. The degree of comparability (see section 2) can be controlled by changing the correlation between the attributes: a strong positive correlation gives high comparability, and a strong negative correlation gives low comparability. We determine the partial order on the generated data points, and again use the algorithm described in section 4 to generate a random monotone classification on the order concerned.

An interesting phenomenon occurs as the number of class labels increases (see Table 5 and Figure 2). The RANDOM strategy actually starts to outperform the greedy heuristic. For example, for $k = 7$, after 5 queries the model produced by GREEDY classifies 18% of the test set correctly, against 28% for RANDOM. In the end GREEDY catches up, but it is never better. The problem with GREEDY if there are many class labels is that initially it is not able to uniquely determine the class label of sufficiently many data points, leading to relatively small training samples. In Figure 2(a) we can see that after about 15 queries GREEDY has catched up, but in any case the number of inferred class labels of GREEDY and RANDOM remain very close. However, if we also include points into the training sample whose label has not been uniquely determined yet, but whose label set has been reduced by at least 50% (indicated by PARTIAL), then GREEDY regains its edge (see Table 5 and Figure 2(c)).

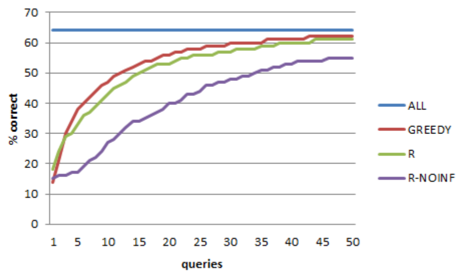
Finally, we consider the effect of the degree of comparability on the performance of the different query strategies. In Figure 3 the degree of comparability has been plotted on the x-axis. The number of queries has been fixed at seven in this case. First of all, we note that the performance increases with the degree of comparability of the data for all query strategies. This was to be expected, since the more comparable pairs of data points, the higher the benefit of the monotonicity



(a) % labeled vectors



(b) accuracy on test set

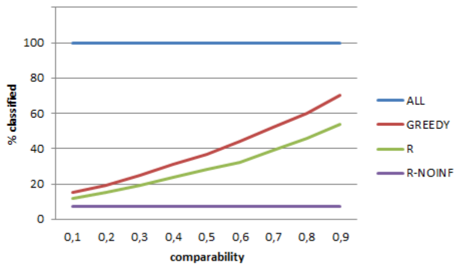


(c) accuracy with partial label information

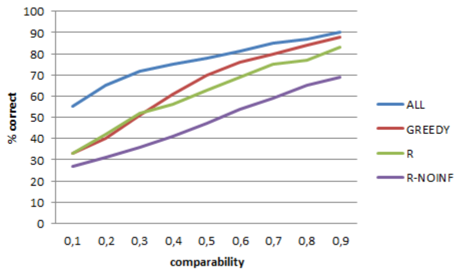
Figure 2: Average results on 100 artificial data sets
$(k = 7, n = 200, c = 0.5)$

	$k = 5$ (max = 73%)				$k = 7$ (max = 64%)			
q	5	10	20	50	5	10	20	50
GREEDY	31%	56%	67%	72%	18%	32%	50%	60%
RANDOM	42%	56%	64%	71%	28%	41%	51%	60%
GREEDY-PARTIAL	56%	64%	70%	73%	38%	47%	56%	62%
RANDOM-PARTIAL	50%	60%	66%	71%	33%	43%	53%	61%

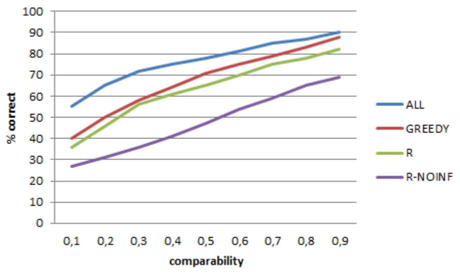
Table 5: % correctly labeled test vectors after stated number of queries averaged over 100 samples. max indicates the prediction error when the complete training set is used $(n = 200, c = 0.5)$



(a) % labeled vectors



(b) accuracy on test set



(c) accuracy with partial label information

Figure 3: For each value of c (comparability), the average result of tests on 100 artificial data sets ($k = 4, n = 200, q = 7$)

Name	n	k	#att	c
AutoMPG	392	7	4	0.81
CPU	209	4	6	0.48
Haberman	306	2	3	0.33
Windsor	546	4	11	0.26
Pima	768	2	8	0.07
Ohsumed	156	2	2	0.66

Table 6: Basic properties of real data sets after pre-processing; #att stands for number of attributes and c stands for comparability.

constraint. For example, if there are no comparable pairs at all, the order on the data points is one big anti chain, and the monotonicity constraint has no effect whatsoever. Conversely, if all pairs of points are comparable we have a linear order with maximum benefit of the monotonicity constraint. In figure 3(a) we see that the advantage of GREEDY over RANDOM increases with comparability. If comparability is low there is little scope for inference, and basically every query strategy will perform quite badly. As comparability increases, so does the scope for inference, and well-chosen queries start to pay off.

Comparing Figures 3(b) and 3(c) we see that including partial label information is more useful in data sets with low comparability. This makes perfect sense, since in such data sets there is little scope for inference. Therefore, after a limited number of queries few labels will have been uniquely determined, and the added benefit of partial label information can be substantial.

5.2 Experiments on Real Data

In this section we discuss the experiments that we performed on a collection of (almost) real data sets. Their basic properties are summarized in Table 6.

All data sets are available from the UCI machine learning repository [2], except for the Windsor housing data, which is available from the Journal of Applied Econometrics Data Archive[1] and the Ohsumed data set which is available from the LETOR web site [2]. Monotonicity judgments were based on common sense. For example, in the AutoMPG data, the weight of the car was taken to have a negative influence on miles per gallon, and in the Windsor housing data, lot size was taken to have a positive influence on selling price of the house. The numeric targets of AutoMPG, CPU, and Windsor were discretized into 7, 4, and 4 equal frequency bins respectively. For the Ohsumed data, we selected the data for query 3, and attributes 1 and 16. To obtain a monotone classification of the data sets we relabeled the data using the algorithm described in [5]. For example, to make the Pima data set monotone, the algorithm relabeled 53 of the 768 data points. Although the Ohsumed data originally had 3 labels ("irrelevant", "partially relevant" and "highly relevant"), after relabeling only 2 labels remained.

After preprocessing, the Ohsumed data set conformed to the restrictions of two class labels and two attributes, and hence we could apply our generalized binary search for matrix orders to it. The results are shown in table 7. Very few queries are required to infer all class labels. This is explained by the high comparability ($c = 0.66$), and in particular by the fact that the data set contains quite a few identical attribute vectors. Apart from that we observe the same phenomenon as in the artificial data: GREEDY requires fewer queries than GBS on average to infer 80% of the labels but GBS wins when 100% of the labels are to be inferred.

% labeled	20%	50%	80%	100%
GREEDY	1	1.2	3.2	15
GBS	1	1	4	13
RANDOM	1	2.2	6.3	24

Table 7: Oshumed: average number of queries needed to classify stated % of vectors averaged over 100 samples.

The results of our experiments on real data are summarized in Table 8. In general we observe that GREEDY outperforms RANDOM in terms of the number of inferred data points as well as in terms of the accuracy of induced models on the test set. We also see that using partial label information tends to improve the accuracy of induced models. The most extreme example is Windsor housing where after querying 20 data points the accuracy of GREEDY is 60% and for GREEDY-PARTIAL it's 65%.

6 Conclusion

We have shown that monotonicity constraints can be exploited to infer class labels of unlabeled points in ordinal classification problems. We have investigated different query strategies and evaluated their performance. For binary classification we developed an efficient algorithm to count monotone classifications on an incomplete matrix order. This enabled us to compare a well-known active learning heuristic called generalized binary search, to an easy-to-compute greedy heuristic. We found that the greedy heuristic initially performs better, but generalized binary search tends to win if the aim is to classify the whole training set. In the context of active learning this is rarely the case, so we may conclude that the greedy heuristic compares favorably to generalized binary search.

For non-binary classification we noticed that as the number of class labels increases, the greedy heuristic tends to be overtaken by the random strategy. At the same time we observed that it is rather wasteful to ignore the partial label information that was inferred from the query points. We have proposed a way to incorporate such partial information into the training sample. The experiments showed that this leads to improved performance. In the experiments on artificial

[1]`http://econ.queensu.ca/jae/`

[2]`http://research.microsoft.com/en-us/um/beijing/projects/letor/`

Name	Heuristic	20%	50%	80%	5	10	20	max
AutoMPG	GREEDY	3.4	34	74	39%	60%	69%	91%
	RANDOM	6.2	44	85	34%	52%	65%	
	GREEDY-PARTIAL				59%	66%	71%	
	RANDOM-PARTIAL				50%	60%	67%	
CPU	GREEDY	7.3	31	65	45%	67%	69%	74%
	RANDOM	12	39	79	40%	52%	65%	
	GREEDY-PARTIAL				59%	68%	70%	
	RANDOM-PARTIAL				49%	61%	66%	
Haberman	GREEDY	1.1	5.3	20	85%	89%	90%	91%
	RANDOM	2	8.5	25.3	84%	86%	89%	
Windsor	GREEDY	17	72	160	44%	51%	60%	72%
	RANDOM	18	93	176	37%	47%	56%	
	GREEDY-PARTIAL				53%	60%	65%	
	RANDOM-PARTIAL				44%	52%	59%	
Pima	GREEDY	12	116	256	74%	77%	78%	79%
	RANDOM	24	102	222	72%	75%	77%	
Ohsumed	GREEDY	1	1.2	3.2	97%	98%	99%	99%
	RANDOM	1	2.2	6.3	91%	97%	99%	

Table 8: Test results for the real data sets. All results are averages over 100 train-test partitions, where $\frac{2}{3}$ is used for training. The first three columns of numbers are the number of queries it takes to classify 20%, 50% and 80% of the training set respectively. The next three columns are the percentages of correctly labeled test vectors after 5, 10 and 20 queries respectively. The final column gives the accuracy on the test set when using all training data.

data we found that the greedy heuristic benefited more from the partial label information than the random strategy. The greedy heuristic using partial label information emerged as the overall best strategy on the real data as well.

Appendix A

This appendix contains the correctness proof for the algorithm to count monotone binary classifications on a 2-dimensional partial order, or (incomplete) matrix order. Figure 4 gives a schematic representation of the matrix and the way we count lower sets.

Let $B_{i,j}$ denote the block of cells with (i, j) as its upper-left cell, and extending all the way to the lower right cell $(1, n)$ of the matrix. So $B_{i-1,j} \cup B_{i,j+1}$ denotes the union of the blocks with $(i-1, j)$ and $(i, j+1)$ as their upper-left elements respectively. Let S be an arbitrary subset of the cells in the matrix, and let $\mathcal{L}(S)$ denote the number of lower sets of the order induced by only considering the cells in S. Finally, let $D(i, j)$ denote the number of lower sets of the order on block $B_{i,j}$, that is, $D(i, j) = \mathcal{L}(B_{i,j})$.

Proposition 1.

$$\mathcal{L}(B_{i-1,j} \cup B_{i,j+1}) = D(i-1, j) + D(i, j+1) - D(i-1, j+1).$$

Proof. We have $(i, j) \preceq (k, l) \Leftrightarrow i \leq k$ and $j \leq l$ (matrix order). Hence, every lower set of $B_{i-1,j}$ is also a lower set of $B_{i-1,j} \cup B_{i,j+1}$. This explains the inclusion of $D(i-1, j)$ on the right hand side of the proposition. This leaves the remaining term $D(i, j+1) - D(i-1, j+1)$ to be accounted for. None of the lower sets of $B_{i,j+1}$ are lower sets of $B_{i-1,j} \cup B_{i,j+1}$, but they could be extended to one by including elements from the first column of $B_{i-1,j}$ (the red column in figure 4). Partition the lower sets of block $B_{i,j+1}$ into those that contain elements from its top row (the green row

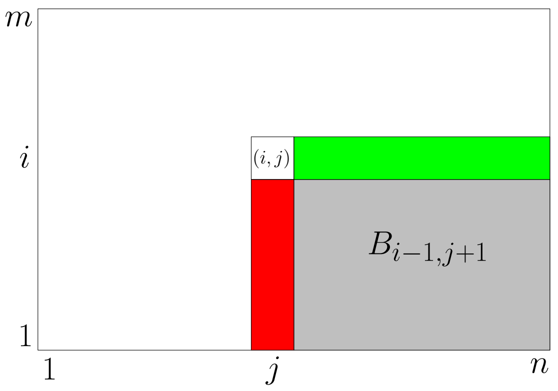


Figure 4: Schematic representation to support the proofs of Proposition 1 and Proposition 2. The cells of the matrix are arranged so that the order between cells coincides with the order in the cartesian plane. Hence the cell $(1,1)$ is in the lower-left corner. We have $(i,j) \preceq (k,l) \Leftrightarrow i \leq k$ and $j \leq l$.

in figure 4), and those that don't. Denote the number of lower sets that contain elements from the top row i by $\mathcal{L}_i(B_{i,j+1})$. We have $D(i,j+1) = \mathcal{L}_i(B_{i,j+1}) + D(i-1,j+1)$. Lower sets of $B_{i,j+1}$ that do not contain any elements from its top row are already accounted for because their possible extensions are lower sets of $B_{i-1,j}$. Hence we only count lower sets of $B_{i,j+1}$ that contain elements from its top row. Each such lower set has exactly one possible extension to a lower set of $B_{i-1,j} \cup B_{i,j+1}$, since it has to include all elements from the first column of $B_{i-1,j}$ (the red column in figure 4). This produces $\mathcal{L}_i(B_{i,j+1}) = D(i,j+1) - D(i-1,j+1)$ additional lower sets. □

Proposition 2.

$$D(i,j) = \mathcal{L}(B_{i-1,j} \cup B_{i,j+1}) + D(i-1,j+1).$$

Proof. Consider the question how many extra lower sets $B_{i,j}$ has compared to $B_{i-1,j} \cup B_{i,j+1}$. The extra lower sets must contain the cell (i,j) since this is the only new element. Consider any lower set from $B_{i-1,j} \cup B_{i,j+1}$ that contains elements from the top row of $B_{i,j+1}$. After addition of (i,j) to the order, we are forced to add (i,j) to this set in order for it to remain a lower set. Hence lower sets that include elements from the top row of $B_{i,j+1}$ do not produce any new lower sets after cell (i,j) is added to the order. On the other hand lower sets that include (i,j) must always include the down set of (i,j) as well, that is they must always include the first column of block $B_{i-1,j}$. Combining these two observations, we can conclude that the new lower sets are obtained by taking any lower set of block $B_{i-1,j+1}$, and adding the first column of $B_{i-1,j}$ to it. Hence, this produces $D(i-1,j+1)$ new lower sets. □

Combining Propositions 1 and 2, we find that $D(i,j) = D(i-1,j) + D(i,j+1)$, providing an efficient way to compute the number of lower sets of the complete matrix by starting at the lower-right cell and working our way to the upper-left cell. This cell contains the final solution after termination of the algorithm. However, so far we have not considered the possibility of empty cells. If (i,j) is an empty cell, then $D(i,j)$ will be equal to $\mathcal{L}(B_{i-1,j} \cup B_{i,j+1})$, so the proper count is given in proposition 1. All we need to kick start the process is to determine the counts for the bottom row and right-most column, but this is straightforward since the corresponding block are linear orders. If the bottom-right cell happens to be empty we enter a count of 1 since the empty set is a lower set. To summarize, we get the recursion given in table 9. By declaring out of bounds cells (row 0 and column $n+1$) to contain counts of 1, the counts for row 1 and column n are computed correctly.

So far, we have established a method to determine the total number of lower sets of a matrix order, but we need to count the number of lower sets that include x. As it turns out this can be done with a few extra steps. The number of lower sets of X that include x is equal to the number

$$D(i,j) = \begin{cases} D(i-1,j) + D(i,j+1) - D(i-1,j+1) & \text{if cell } (i,j) \text{ is empty} \\ D(i-1,j) + D(i,j+1) & \text{otherwise} \end{cases}$$

Table 9: Recursion to compute the number of lower sets in block $B_{i,j}$. Any required cell with index "out of bounds" should be taken to contain a count of 1.

$$C(i,j) = \begin{cases} D(i,j+1) & \text{if } i = m \\ C(i+1,j) + D(i,j+1) \times (E(i+1,j) - E(i+2,j)) & \text{otherwise} \end{cases}$$

Table 10: Recursion to compute the number of lower sets that include cell (i,j). Any required cell with index "out of bounds" should be taken to contain a count of 1.

of lower sets of $X \setminus \downarrow x$. Unfortunately, the number of lower sets of $X \setminus \downarrow x$ cannot be read off immediately from D, except for the cells in the top row. To compute the counts for cells not in the top row, we first define a matrix E which is constructed in a similar fashion as D, but working in the opposite direction, i.e. from the upper-left cell to the bottom-right cell. Cell (i,j) of E contains a count of the lower sets of the block extending from the upper-left cell to cell (i,j). The total count will now appear in the bottom-right cell of E. Finally we define the matrix C where $C(i,j)$ contains the number of lower sets that include cell (i,j).

Proposition 3.

$$C(i,j) = C(i+1,j) + D(i,j+1) \times (E(i+1,j) - E(i+2,j))$$

Proof. $C(i+1,j)$ contains a count of the lower sets of block $Q+U+V$ (see figure 5), because $X \setminus \downarrow (i+1,j) = Q+U+V$. Partition the lower sets of $(Q+U+V)$ into lower sets that contain elements from block U, and lower sets that don't contain elements from block U. We have

$$\mathcal{L}(Q+U+V) = \mathcal{L}_U(Q+U+V) + \mathcal{L}_{\bar{U}}(Q+U+V)$$

Now $\mathcal{L}_{\bar{U}}(Q+U+V) = \mathcal{L}(Q) \times \mathcal{L}(V)$ since Q and V are incomparable blocks, so we can produce a lower set by taking any lower set from Q and combining it with any lower set from V. By construction of the matrices D and E, we have $\mathcal{L}(Q) \times \mathcal{L}(V) = E(i+2,j) \times D(i,j+1)$. $C(i,j)$ should contain a count of the lower sets of block $Q+R+U+V$. Again partition them in lower sets that contain elements from U and those that don't.

$$\mathcal{L}(Q+R+U+V) = \mathcal{L}_U(Q+R+U+V) + \mathcal{L}_{\bar{U}}(Q+R+U+V)$$

Now we have

$$\begin{aligned} \mathcal{L}_{\bar{U}}(Q+R+U+V) &= \mathcal{L}(Q+R) \times \mathcal{L}(V) \\ &= E(i+1,j) \times D(i,j+1). \end{aligned}$$

Also we have $\mathcal{L}_U(Q+U+V) = \mathcal{L}_U(Q+R+U+V)$ because there is a one-to-one correspondence between lower sets from $Q+U+V$ that contain elements of U, and lower sets from $Q+R+U+V$ that contain elements of U: given one of the former, we get exactly one of the latter by adding the whole of block R. This is mandatory because the whole of block R is in the down set of any element of U. Given one of the latter, we get exactly one of the former by removing all elements from block R. This always produces a lower set of $Q+U+V$, since R is a lower set of $Q+R+U+V$. Summarizing we get that $C(i,j) - C(i+1,j) = (E(i+1,j) - E(i+2,j)) \times D(i,j+1)$. □

To summarize, we get the recursion given in table 10 to construct the matrix C.

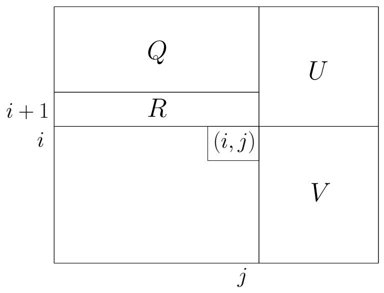


Figure 5: Schematic picture for the proof of proposition 3

$$\phi_t(S,u) = \begin{cases} S[L_j^S \to L_j^S \setminus \{x_t\}] & \text{if } u_t < \frac{1}{2} \text{ and } x_t \in (L_j^S \setminus L_{j-1}^S) \text{ and } L_j^S \setminus \{x_t\} \text{ is a lower set.} \\ S[L_j^S \to L_j^S \cup \{x_t\}] & \text{if } u_t \geq \frac{1}{2} \text{ and } x_t \in (L_{j+1}^S \setminus L_j^S) \text{ and } L_j^S \cup \{x_t\} \text{ is a lower set.} \\ S & \text{otherwise} \end{cases}$$

Table 11: The update function for the algorithm to draw a random monotone function. $S[Y \to Z]$ is identical to S except that Y is replaced by Z. In the first case, if $j = 1$, then L_{j-1}^S is taken to be the empty set. In the second case, if $j = k - 1$, then L_{j+1}^S is taken to be equal to X.

Appendix B

Section 6 contains a specification of the update function for the algorithm to draw a random monotone function, and the proof that the update function is monotone.

We first describe the proposed update function in words. We draw an element x_t at random from X and we draw a uniform random number u_t. If $u_t < \frac{1}{2}$, then we increase the label of x_t by one (unless x_t already has the highest label), provided that the resulting classification is still monotone. Increasing the label by one is accomplished by removing x_t from the first lower set in which it occurs in the nested sequence of lower sets. If $u_t \geq \frac{1}{2}$, then the label of x_t is decreased by one (unless x_t already has the smallest label), provided that the resulting classification is still monotone. Decreasing the label by one is accomplished by adding x_t to the last lower set in which it does not occur in the nested sequence of lower sets. In all other cases, S remains unchanged. The formal definition of the update function ϕ is given in Table 11.

Proposition 4.

$$S_1 \preceq_{\mathcal{S}} S_2 \Rightarrow \phi_t(S_1, u) \preceq_{\mathcal{S}} \phi_t(S_2, u).$$

Proof. We give the proof for the case $u_t < \frac{1}{2}$. The proof for the case $u_t \geq \frac{1}{2}$ is similar. Assume $S_1 \preceq_{\mathcal{S}} S_2$, that is, $L_i^{S_2} \subseteq L_i^{S_1}$ for $i = 1, \ldots, k-1$. If $\phi_t(S_1) = S_1$ then the consequence of the proposition is trivially true. Consider the case that $\phi_t(S_1) = S_1[L_{j_1}^{S_1} \to L_{j_1}^{S_1} \setminus \{x_t\}]$, that is, $x_t \in L_{j_1}^{S_1}$, $x_t \notin L_{j_1-1}^{S_1}$ and $L_{j_1}^{S_1} \setminus \{x_t\}$ is a lower set. The last condition holds if x_t is a maximal element of $L_{j_1}^{S_1}$. Let $L_{j_2}^{S_2}$ be the first lower set of S_2 that contains x_t. We have $j_2 \geq j_1$ because $S_1 \preceq_{\mathcal{S}} S_2$. We consider two cases

1. $j_2 > j_1$: Since $L_{j_1}^{S_2} \subseteq L_{j_1}^{S_1}$, and $x_t \notin L_{j_1}^{S_2}$, it follows that $L_{j_1}^{S_2} \subseteq L_{j_1}^{S_1} \setminus \{x_t\}$. This was the only condition that could have been violated by the update, so we conclude $\phi_t(S_1) \preceq_{\mathcal{S}} \phi_t(S_2)$.

2. $j_1 = j_2$: Since x_t is a maximal element of $L_{j_1}^{S_1}$, and by assumption $L_{j_1}^{S_2} \subseteq L_{j_1}^{S_1}$, we may conclude that x_t is also a maximal element of $L_{j_1}^{S_2}$. This implies that $\phi_t(S_2) = S_2[L_{j_1}^{S_2} \to L_{j_1}^{S_2} \setminus \{x_t\}]$. Since by assumption $L_{j_1}^{S_2} \subseteq L_{j_1}^{S_1}$, it follows that $L_{j_1}^{S_2} \setminus \{x_t\} \subseteq L_{j_1}^{S_1} \setminus \{x_t\}$. Again,

this was the only condition that could have been violated by the update, so we conclude $\phi_t(S_1) \preceq_{\mathcal{S}} \phi_t(S_2)$.

$\square$

References

[1] E.A. Altendorf, A.C. Restificar, and T.G. Dietterich. Learning from sparse data by exploiting monotonicity constraints. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 18–25, 2005.

[2] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[3] N. Barile and A. Feelders. Active learning with monotonicity constraints. In *SIAM International Conference on Data Mining (SDM 2012)*, pages 756–767, 2012.

[4] S. Dasgupta. Analysis of a greedy active learning strategy. In *Advances in Neural Information Processing Systems (NIPS)*, pages 337–344. MIT Press, 2004.

[5] A. Feelders. Monotone relabeling in ordinal classification. In *ICDM 2010, Proceedings of the 10th IEEE International Conference on Data Mining*, pages 803–808, 2010.

[6] W. Kotlowski and R. Slowinski. Rule learning with monotonicity constraints. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML 2009)*, pages 537–544, 2009.

[7] J. Propp and D. Wilson. Exact sampling with coupled markov chains and applications to statistical mechanics. *Random Struct. Algorithms*, 9(1-2):223–252, 1996.

[8] J. Provan and M. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4):777–788, 1983.