Democratic Approximation of Lexicographic Preference Models ☆

Fusun Yaman[a,*], Thomas J. Walsh[b], Michael L. Littman[c], Marie desJardins[d]

[a]*BBN Technologies, 10 Moulton St. Cambridge, MA 02138,USA*
[b]*University of Arizona, Department of Computer Science, Tucson, AZ 85721*
[c]*Rutgers University, Department of Computer Science, Piscataway, NJ 08854 USA*
[d]*University of Maryland Baltimore County, Computer Science and Electrical Eng. Department, Baltimore, MD 21250 USA*

Abstract

Lexicographic preference models (LPMs) are an intuitive representation that corresponds to many real-world preferences exhibited by human decision makers. Previous algorithms for learning LPMs produce a "best guess" LPM that is consistent with the observations. Our approach is more democratic: we do not commit to a single LPM. Instead, we approximate the target using the votes of a *collection* of consistent LPMs. We present two variations of this method—*variable voting* and *model voting*—and empirically show that these democratic algorithms outperform the existing methods. Versions of these democratic algorithms are presented in both the case where the preferred values of attributes are known and the case where they are unknown. We also introduce an intuitive yet powerful form of background knowledge to prune some of the possible LPMs. We demonstrate how this background knowledge can be incorporated into variable and model voting and show that doing so improves performance significantly, especially when the number of observations is small.

Key words: lexicographic models, preference learning, Bayesian methods.

*Corresponding author
Email addresses: `fusun@bbn.com` (Fusun Yaman), `twalsh@cs.arizona.edu` (Thomas J. Walsh), `mlittman@cs.rutgers.edu` (Michael L. Littman), `mariedj@cs.umbc.edu` (Marie desJardins)

1. Introduction

Lexicographic preference models (LPMs) are one of the simplest yet most intuitive preference representations. An LPM defines an order of importance on the variables that describe the objects in a domain and uses this order to make preference decisions. For example, the meal preference of a vegetarian with a weak stomach could be represented by an LPM such that a vegetarian dish is always preferred over a non-vegetarian dish, and among vegetarian or non-vegetarian items, mild dishes are preferred to spicy ones.

Despite the simplicity of lexicographic LPMs, several studies on human decision making [4, 20, 9] experimentally demonstrate that humans often make decisions using lexicographic reasoning instead of mathematically more sophisticated methods such as linear additive value maximization [6].

Previous work on learning LPMs from a set of preference observations has been limited to *autocratic* approaches: one of many possible consistent LPMs is picked heuristically and used for future decisions. However, it is highly likely that autocratic methods will produce poor approximations of the target when there are few observations.

In this paper, we present a *democratic* approach to LPM learning, which does not commit to a single LPM. Instead, we approximate a target preference using the votes of a collection of consistent LPMs. We present two variations of this method: *variable voting* and *model voting*. Variable voting operates at the variable level and samples the consistent LPMs implicitly. The learning algorithm based on variable voting learns a weak order on the variables, such that each linearization corresponds to an LPM that is consistent with the observations. Model voting explicitly samples the consistent LPMs and employs a weighted vote, where the weights are computed using Bayesian priors. The additional complexity of voting-based algorithms (compared to autocratic methods) is tolerable: both algorithms have low-order polynomial time complexity. Our experiments show that these democratic algorithms outperform both the average and worst-case performance of the state-of-the-art autocratic algorithm.

We also investigate the effect of imperfect data on the learning algorithms. We consider two kinds of imperfections: *faulty* observations (noise) and *hidden ties* (ties that are broken arbitrarily). Our empirical evaluation demonstrates that all of the algorithms we consider are robust in the presence of hidden ties. However, even a small number of faulty observations significantly reduce the performance of the voting algorithms. On the other hand, the greedy algorithm is resilient: that is, the performance decline is proportional to the amount of noise in the data. We take a lesson from this, and adapting the voting methods to consider the amount of noise in an environment, we empirically show the resulting heuristic is on par with the greedy approach in the case of noisy observations.

To further improve the performance of the learning algorithms when the number of observations is small, we introduce an intuitive yet powerful form of background knowledge. The background knowledge defines equivalence classes on the variables, indicating the most important set of variables, the second most important set, and so on. This representation permits a user or designer to provide partial information about an LPM (or a class of LPMs) that can be used by the learner to reduce the search space. We demonstrate how this background knowledge can be used with variable and model voting and show that doing so improves performance significantly, especially when the number of observations is small.

In the rest of the paper, we give some background on LPMs (Section 2), then describe our voting-based methods (Section 3). After introducing these methods in the case where the preferred values of all attributes are known, we present extensions of these algorithms to the case where preferred values are not known *a priori* (Section 4). We then introduce our background knowledge representation, show how we can generalize the voting methods to exploit this background knowledge (Section 5), present an approach for handling noisy data (Section 6), and present experimental results of this work (Section 7). Finally, we present related work (Section 8) and discuss our future work and conclusions (Section 9).

2. Lexicographic Preference Models

In this section, we briefly introduce the lexicographic preference model (LPM) and summarize previous results on learning LPMs. In this work, we only consider binary variables whose domain is $\{0, 1\}$.[1] For clarity in the introduction of our algorithms, we assume for now that the preferred value of each variable is known. This assumption will be removed in Section 4. Without loss of generality, we will assume that 1 is always preferred to 0.

Given a set of variables, $X = \{X_1 \dots X_n\}$, an object A over X is a vector of the form $[x_1, \dots, x_n]$. We use the notation $A(X_i)$ to refer the value of X_i in the object A. A *lexicographic preference model* $\mathcal{L}$ on X is a total order on a subset R of X. We denote this total order with $\sqsubset_{\mathcal{L}}$. Any variable in R is *relevant* with

[1]The representation can easily be generalized to monotonic preferences with ordinal variables, such that 1 corresponds to a preference on the values in increasing order, and 0 to a decreasing order, as shown by Yaman and desJardins [21] for conditional preference networks (CP-nets).

respect to $\mathcal{L}$; similarly, any variable in $I = X - R$ is *irrelevant* with respect to $\mathcal{L}$. If a variable A appears earlier in this total order than B ($A < B$), then A is said to be *more important* or to have a *smaller rank* than B.

If A and B are two objects, then the preferred object given $\mathcal{L}$ is determined as follows:

- Find the smallest (most important) variable X^* in $\sqsubset_{\mathcal{L}}$ such that X^* has different values in A and B. The object that has the value 1 for X^* is the most preferred.

- If all relevant variables in $\mathcal{L}$ have the same value in A and B, then the objects are equally preferred (a tie).

Example 1. *Suppose $X_1 < X_2 < X_3$ is the total order defined by an LPM $\mathcal{L}$, and consider objects $A = [1, 0, 1, 1]$, $B = [0, 1, 0, 0]$, $C = [0, 0, 1, 1]$, and $D = [0, 0, 1, 0]$. A is preferred over B because $A(X_1) = 1$, and X_1 is the most important variable in $\mathcal{L}$. B is preferred over C because $B(X_2) = 1$ and both objects have the same value for X_1. Finally, C and D are equally preferred because they have the same values for the relevant variables.*

An *observation* $o = (A, B)$ is an ordered pair of objects, connoting that A is preferred to B. In many practical applications, however, preference observations are gathered from demonstration of an expert who breaks ties arbitrarily. That is, when presented with a situation in which a decision or choice must be made, if the expert judges the two alternatives to be *equally good*, the expert will in fact be indifferent, and will therefore be equally likely to choose either alternative. Thus, for some observations, A and B may actually be tied in the preference order, although we cannot determine this directly from the observations. Therefore, an LPM $\mathcal{L}$ is said to be *consistent* with an observation (A, B) iff $\mathcal{L}$ implies that A is preferred to B or that A and B are equally preferred.

The problem of learning an LPM is defined as follows. Given a set of observations, find an LPM $\mathcal{L}$ that is consistent with the observations. Previous work on learning LPMs was limited to the case where all variables are relevant. This assumption entails that, in every observation (A, B), A is strictly preferred to B, since ties can only happen when there are irrelevant attributes.

The best published algorithm for learning LPMs from observations was presented by Schmitt and Martignon [16], who proposed a greedy variable-permutation algorithm that is guaranteed to find one of the LPMs that is consistent with the observations, if one exists. They have also shown that for the noisy

Algorithm 1 *greedyPermutation*

```
Require: A set of variables $X$ and a set of observations $O$.
Ensure: An LPM that is consistent with $O$, if one exists.
1: for $i = 1, \ldots, n$ do
2:     Arbitrarily pick one of $X_j \in X$ such that
       $\mathit{MISS}(X_j, O) = \min_{X_k \in X} \mathit{MISS}(X_k, O)$
3:     $Rank(X_j) := i$, assign the rank $i$ to $X_j$
4:     Remove $X_j$ from $X$
5:     Remove all observations $(A, B)$ from $O$ such that $A(X_j) \neq B(X_j)$
6: Return the total order $\sqsubset$ on $X$ such that $X_i < X_j$ iff $Rank(X_i) < Rank(X_j)$
```

data case, finding an LPM that does not violate more than a constant number of the observations is NP-complete. We use this greedy algorithm, which is shown in Algorithm 1, as a performance baseline. The algorithm refers to a function *MISS*(X_i, O), which is defined as $|\{(A, B) \in O : B(X_i)\text{is preferred to}A(X_i)\}|$; that is, the number of observations violated in O if the most important variable is selected as X_i. Basically, the algorithm greedily constructs a total order by choosing the variable at each step that causes the minimum number of inconsistencies with the observations. If multiple variables have the same minimum, then one of them is chosen arbitrarily. The algorithm runs in polynomial time, specifically $O(n^2m)$, where n is the number of variables and m is the number of observations.

Dombi et al. [7] have shown that if there are n variables, all of which are relevant, then $O(n \log n)$ queries to an oracle suffice to learn an LPM. Furthermore, it is possible to learn any LPM with $O(n^2)$ observations if all pairs differ in only two variables. They proposed an algorithm that can find the unique LPM induced by the observations. In case of noise due to irrelevant attributes (with ties reported arbitrarily), the algorithm does not return an answer.

In the net section, we investigate the following problem: Given a set of observations with no noise, but possibly with arbitrarily broken ties, find a rule for predicting preferences that agrees with the target LPM that produced the observations. Later in the paper, we will relax this assumption to permit noisy data (Section 6).

3. Voting Algorithms

We propose a democratic approach for approximating the target LPM that produced a set of observations. Instead of finding just one of the consistent LPMs, it reasons with a *collection* of LPMs that are consistent with the observations. Given two objects, such an approach prefers the one that a majority of its models prefer. A naive implementation of a voting algorithm would enumerate all LPMs that are consistent with a set of observations. However, since the number of models that are consistent with a set of observations can be exponential, the naive implementation is infeasible.

In this section, we describe two methods—*variable voting* and *model voting*—that sample the set of consistent LPMs and use voting to predict the preferred object. Unlike existing algorithms that learn LPMs, these methods do not require all variables to be relevant or observations to be tie-free. The following subsections explain the variable-voting and model-voting methods and summarize our theoretical results.

3.1. Variable Voting

Variable voting uses a generalization of the LPM representation. Instead of a total order on the variables, variable voting reasons with a *weak* order[2] ($\preceq$) to find the preferred object in a given pair. Among the variables that differ in the two objects, the ones that have the smallest rank (and are hence the most salient) in the weak order vote to choose the preferred object. The object that has the most "1" values for the voting variables is declared to be the preferred one. If the votes are equal, then the objects are equally preferred.

Definition 1 (Variable Voting). *Suppose X is a set of variables and $\preceq$ is a weak order on X. Given two objects, A and B, the variable-voting process with respect to $\preceq$ for determining which of the two objects is preferred is:*

- *Define D to be the set of variables that differ in A and B.*
- *Define D^* to be the set of variables in D that have the smallest rank among D with respect to $\preceq$.*
- *Define N_A to be the number of variables in D^* that favor A (i.e., that have value 1 in A and 0 in B) and N_B to be the number of variables in D^* that favor B.*
- *If $N_A > N_B$, then A is preferred. If $N_A < N_B$, then B is preferred. Otherwise, they are equally preferred.*

[2]A weak order is an asymmetric, reflexive, and transitive order. In other words, a weak order defines an ordering over sets of objects; within each set, the objects are unordered with respect to each other.

Algorithm 2 *learnVariableRank*

```
Require: A set of variables $X$, and a set of observations $O$.
Ensure: A weak order on $X$.
 1: $\Pi(x) = 1, \forall\, x \in X$
 2: while $\Pi$ has changed on the last iteration do
 3:     for Every observation $(A, B) \in O$ do
 4:         $D = \{x | A(x) \neq B(x)\}$
 5:         $D^* = \{x \in D | \forall y \in D, \Pi(x) \leq \Pi(y)\}$
 6:         $V_A = \{x \in D^* | A(x) = 1\}$
 7:         $V_B = \{x \in D^* | B(x) = 1\}$
 8:         VariableVote predicts a preferred object based on $V_A > V_B$.
 9:         for $x \in V_B$ such that $\Pi(x) < |X|$ do
10:             $\Pi(x) = \Pi(x) + 1$;
11: Return weak order $\preceq$ on $X$ such that $x \preceq y$ iff $\Pi(x) < \Pi(y)$.
```

Example 2. *Suppose $\preceq$ is the weak order $\{X_2, X_3\} < \{X_1\} < \{X_4, X_5\}$. Consider objects $A = [0, 1, 1, 0, 0]$ and $B = [0, 0, 1, 0, 1]$. D is $\{X_2, X_5\}$. D^* is $\{X_2\}$ because X_2 is the smallest ranking variable in D with respect to $\preceq$. X_2 favors A because $A(X_2) = 1$. Thus, variable voting with $\preceq$ prefers A over B.*

Algorithm 2 presents the algorithm *learnVariableRank*, which learns a weak order $\preceq$ on the variables from a set of observations such that variable voting with respect to $\preceq$ will correctly predict the preferred objects in the observations. Specifically, it finds weak orders that define equivalence classes on the set of variables. The algorithm maintains the minimum possible rank for every variable that does not violate an observation with respect to variable voting. Initially, all variables are considered equally important (rank of 1). The algorithm loops over the set of observations until the ranks converge. At every iteration and for every pair, variable voting predicts a winner, which allows us to use this algorithm in the online-learning setting where examples (O) need to be classified during the learning process. Regardless of this prediction, the ranks of the variables that voted for the wrong object are incremented, thus reducing their importance. Finally, the algorithm builds a weak order $\preceq$ based on the ranks such that $x \preceq y$ if and only if x has a lower rank than y. In the offline-learning setting (where O is a set of training examples), this weak order can then be given directly to variable voting to classify examples in the test set.

Example 3. *Suppose $X = \{X_1, X_2, X_3, X_4, X_5\}$ and O consists of* $([0, 1, 1, 0, 0]$, $[1, 1, 0, 1, 1])$, $([0, 1, 1, 0, 1]$, $[1, 0, 0, 1, 0])$ *and* $([1, 0, 1, 0, 0]$, $[0, 0, 1, 1, 1])$. *Table*

Table 1: The rank of the variables after each iteration of the for-loop in line 3 of the algorithm *learnVariableRank*.

Observations	X_1	X_2	X_3	X_4	X_5
$Initially$	1	1	1	1	1
$[0, 1, 1, 0, 0], [1, 1, 0, 1, 1]$	2	1	1	2	2
$[0, 1, 1, 0, 1], [1, 0, 0, 1, 0]$	2	1	1	2	2
$[1, 0, 1, 0, 0], [0, 0, 1, 1, 1]$	2	1	1	3	3

1 illustrates the ranks of every variable in X after each iteration of the for-loop in line 3 of the algorithm learnVariableRank. The ranks of the variables stay the same during the second iteration of the while-loop; therefore, the loop terminates. The weak order $\preceq$ based on ranks of the variables is the same as the order given in Example 2.

We now summarize our theoretical results about the algorithm *learnVariableRank*.

Correctness. Suppose $\preceq$ is a weak order returned by $learnVariableRank(X, O)$. Any LPM $\mathcal{L}$ based on a corresponding topological sort $\sqsubset_{\mathcal{L}}$ of $\preceq$ will be consistent with the observation set O. This can be proven simply by contradiction. Suppose an observation o_i existed such that a majority of the variables within an existing class led to an incorrect classification of o_i based on the returned weak order $\preceq$. Since the learning algorithm loops over all of the observations, this process would have resulted in an increment of a value and the algorithm would not have completed with the current $\preceq$, thus all LPM consistent with $\preceq$ are consistent with O. Furthermore, $learnVariableRank$ never increments the ranks of the relevant variables beyond their actual rank in the target LPM. This can be seen by considering the cases of both relevant and irrelevant variables. First, for relevant variables, the number of times a variable is in the set of variables that actually vote (and are therefore potentially incremented) *and* votes incorrectly, is simply its true rank t. Once it reaches this true rank, by definition it cannot vote incorrectly because this variable must vote correctly when all other values are tied (otherwise this would not be its true rank). Second, the ranks of the irrelevant variables can be incremented only as far as the number of variables, thus the algorithm is guaranteed to terminate, even in the presence of irrelevant variables.

Convergence. In either the online-learning setting (O being incrementally fed to the learner) or the offline setting (O provided as a batch), *learnVariableRank* has a mistake-bound of $O(n^2)$, where n is the number of variables. To see this, we consider two cases for any given observation o_t at time t, assuming for ease of exposition that A is preferred to B. First, if $V_A > V_B$, then a mistake will not be made, though some variables may have their rank increased anyway (because they would have voted the wrong way). The second case is where $V_A < V_B$, in which case a mistake is made, and therefore we need to limit the number of such cases. Ignoring any increments in the first case, we see that because each mistake increases the sum of the potential ranks by at least 1 and the sum of the ranks the target LPM induces is $O(n^2)$, the second case can occur no more than $O(n^2)$ times . This bound guarantees that given enough observations (as described in the background section), *learnVariableRank* will converge to a weak order $\preceq$ that, when used in conjunction with variable voting, consistently classifies all preferred objects with respect to the target LPM. Furthermore, the incrementing of ranks in case 1 gives us the stronger result (mentioned above) that every topological sort of $\preceq$ has the same prefix as the total order induced by the target LPM. If all variables are relevant, then $\preceq$ will converge to the total order induced by the target LPM.

Computational Complexity. We consider the computational complexity of *learnVariableRank* in the offline-learning setting where O is provided as a training set. A loose upper bound on the time complexity of *learnVariableRank* is $O(n^3m)$, where n is the number of variables and m is the number of observations. This bound holds because the while-loop on line 2 runs at most $O(n^2)$ times (because this is the max sum of the maximum possible ranks) and the for-loop in line 3 runs for m observations (by definition). The time complexity of one iteration of the for-loop is $O(n)$ (since all variables need to be considered in the worst case); therefore, the overall complexity is $O(n^3m)$. We leave the investigation of tighter bounds, improved data structures, and the average case analysis for future work.

3.2. Model Voting

The second method we present employs a Bayesian approach. This method randomly generates a sample set, S, of distinct LPMs that are consistent with the observations. When a pair of objects is presented, the preferred one is predicted using weighted voting. That is, each $\mathcal{L} \in S$ casts a vote for the object it prefers, and this vote is weighted according to its posterior probability $P(\mathcal{L}|S)$.

Definition 2 (Model Voting). *Let U be the set of all LPMs, O be a set of observations, and $S \subset U$ be a set of LPMs that are consistent with O. Given two objects*

A and B, model voting prefers A over B with respect to S if

$$\sum_{\mathcal{L} \in U} P(\mathcal{L}|S) V^{\mathcal{L}}_{(A>B)} > \sum_{\mathcal{L} \in U} P(\mathcal{L}|S) V^{\mathcal{L}}_{(B>A)}, \tag{1}$$

where $V^{\mathcal{L}}_{(A>B)}$ is 1 if A is preferred with respect to $\mathcal{L}$, and 0 otherwise. $V^{\mathcal{L}}_{(B>A)}$ is defined analogously. $P(\mathcal{L}|S)$ is the posterior probability of $\mathcal{L}$ being the target LPM given S, calculated as discussed below.

We first assume that all LPMs are equally likely *a priori*. In this case, given a sample of LPMs S of size k, the posterior probability of an LPM $\mathcal{L}$ will be $1/k$ if and only if $\mathcal{L} \in S$, and 0 otherwise. Note that when S is maximal, this case degenerates into the naive voting algorithm. However, it is generally not feasible to enumerate all consistent LPMs—in practice, the sample has to be small enough to be feasible and large enough to be representative.

In constructing S, we exploit the fact that many consistent LPMs share prefixes in the total order that they define on the variables. We wish to discover and compactly represent such LPMs. To this end, we introduce the idea of *aggregated LPMs*. An aggregated LPM, $(X_1, X_2 \ldots, X_k, *)$, represents a set of LPMs that define a total order with the prefix $X_1 < X_2 < \ldots < X_k$. Intuitively, an aggregated LPM states that any possible completion of the prefix is consistent with the observations. The algorithm *sampleModels* in Algorithm 3 implements a "smart sampling" approach by constructing an LPM that is consistent with the given observations, returning an aggregated LPM when possible. We start with an arbitrary consistent LPM (such as the empty set, which is always consistent) and add more variable orderings extending the input LPM. We first identify the variables that can be used in extending the prefix—that is, all variables X_i such that in every observation, either X_i is 1 in the preferred object or X_i is the same in both objects. We then select one of those variables randomly and extend the prefix. Finally, we remove the observations that are explained with this selection and continue with the rest of the observations. If at any point, no observations remain, then we return the aggregated form of the prefix, since every completion of the prefix will be consistent with the null observation. Running *sampleModels* several times and eliminating duplicates will produce a set of (possibly aggregated) LPMs.

Example 4. *Consider the same set of observations O as in Example 3. Then, the aggregated LPMs that are consistent with O are as follows: $()$, (X_2), (X_2, X_3), $(X_2, X_3, X_1, *)$, (X_3), $(X_3, X_1, *)$, (X_3, X_2) and $(X_3, X_2, X_1, *)$. To illustrate the set of LPMs that an aggregate LPM represents, consider $(X_2, X_3, X_1, *)$, which*

Algorithm 3 *sampleModels*

```
Require: A set of variables $X$, a set of observations $O$, and rulePrefix, an LPM to be extended.
Ensure: An LPM (possibly aggregated) consistent with $O$.
1: candidates is the set of variables $\{Y : Y \notin rulePrefix \mid \forall(A,B) \in O, A(Y) = 1\ or A(Y) = B(Y)\}$.
2: while candidates $\neq \emptyset$ do
3:     if $O = \emptyset$ then
4:         return $(rulePrefix, *)$.
5:     Randomly remove a variable $Z$ from candidates.
6:     Remove any observation $(C, D)$ from $O$ such that $C(Z) \neq D(Z)$.
7:     Extend rulePrefix: $rulePrefix = (rulePrefix, Z)$.
8:     Recompute candidates.
9: return rulePrefix
```

has a total of 5 extensions: (X_2, X_3, X_1)*,* (X_2, X_3, X_1, X_4)*,* (X_2, X_3, X_1, X_5)*,* $(X_2, X_3, X_1, X_4, X_5)$*,* $(X_2, X_3, X_1, X_5, X_4)$*. Every time the algorithm sampleModels runs on the set of observations O from Example 3, it will randomly generate one of the aggregated LPMs:* $(X_2, X_3, X_1, *)$*,* $(X_3, X_1, *)$*, or* $(X_3, X_2, X_1, *)$*. Note that the shorter models that are not produced by sampleModels are all subprefixes of the aggregated LPMs and it is easy to modify sampleModels to return those models as well.*

An aggregate LPM in a sample saves us from having to enumerate all possible extensions of a prefix, but it also introduces complications in computing the weights (posteriors) of the LPMs, as well as their votes. For example, when comparing two objects A and B, some extensions of an aggregate LPM might vote for A and some for B. Thus, we need to find the total number of LPMs that an aggregate LPM represents and determine what proportion of them favor A over B (or vice versa), without enumerating all extensions. Suppose there are n variables and $\mathcal{L}$ is an aggregated LPM with a prefix of length k. Then the number of extensions of $\mathcal{L}$ is denoted by $F_{\mathcal{L}}$ and is equal to f_{n-k}, where f_m is defined to be:

$$f_m = \sum_{i=0}^{m} \binom{m}{i} \times i! = \sum_{i=0}^{m} \frac{(m)!}{(m-i)!}. \tag{2}$$

Intuitively, f_m counts every possible permutation with at most m items. Note that f_m can be computed efficiently and that the number of all possible LPMs when there are n variables is given by f_n.

While the above formula calculates the total number of extensions, we still need to determine how many votes an aggregate LPM $\mathcal{L} = (X_1, X_2, \ldots, X_k, *)$ will allocate to each of two compared objects A and B. We will call the variables $X_1 \ldots X_k$ the *prefix variables*. If A and B have different values for at least one prefix variable, then all extensions will vote in accordance with the smallest such variable. Suppose all prefix variables are tied and m is the set of all non-prefix variables. Then m is composed of three disjoint sets a, b, and w, such that a is the set of variables that favor A, b is the set of variables that favor B, and w is the set of variables that are neutral (that is, that have the same value in A and B).

An extension $\mathcal{L}'$ of $\mathcal{L}$ will produce a tie iff all variables in a and b are irrelevant in $\mathcal{L}'$. The number of such extensions is $f_{|w|}$. The number of extensions that favor A over B is directly proportional to $|a|/(|a| + |b|)$. Therefore, the number of extensions of $\mathcal{L}$ that will vote for A over B (denoted by $N^{\mathcal{L}}_{A>B}$) is:

$$N^{\mathcal{L}}_{A>B} = \frac{|a|}{|b| + |a|} \times (f_m - f_{|w|}). \tag{3}$$

The number of extensions of $\mathcal{L}$ that will vote for B over A is computed similarly. Note that the computation of $N^{\mathcal{L}}_{A>B}$, $N^{\mathcal{L}}_{B>A}$, and $F_{\mathcal{L}}$ can be done in linear time by caching the recurring values.

Example 5. *Suppose X and O are as defined in Example 3. The first column of Table 2 lists all LPMs that are consistent with O. The second column gives the posterior probabilities of these models given the sample S_1, which is the set of all consistent LPMs. The third column is the posterior probability of the models given the sample $S_2 = \{(X_2, X_3, X_1, *), (X_3, X_1, *), (X_3, X_2, X_1, *)\}$. Given two objects $A = [0, 1, 1, 0, 0]$ and $B = [0, 0, 1, 0, 1]$, the number of votes for each object based on each LPM is given in the last two columns. Note that the total number of votes for A and B does not add up to the total number of extensions of $(X_3, X_1, *)$ because two of its extensions—(X_3, X_1) and (X_3, X_1, X_4)—prefer A and B equally.*

Algorithm 4 describes *modelVote*, which takes a sample of consistent LPMs (produced, for instance, by *sampleModels*) and a pair of objects as input, and predicts the preferred object using the weighted votes of the LPMs in the sample.

Returning to Example 5, the reader can verify that model voting will prefer A over B. Next, we present our theoretical results on the *sampleModels* and *modelVote* algorithms.

Table 2: The posterior probabilities and number of votes of all LPMs in Example 5.

LPMs	$P(L\|S_1)$	$P(L\|S_2)$	$N^{\mathcal{L}}_{A>B}$	$N^{\mathcal{L}}_{B>A}$
()	1/31	0	0	0
(X_2)	1/31	0	1	0
(X_2, X_3)	1/31	0	1	0
$(X_2, X_3, X_1, *)$	5/31	5/26	5	0
(X_3)	1/31	0	0	0
$(X_3, X_1, *)$	16/31	16/26	7	7
(X_3, X_2)	1/31	0	1	0
$(X_3, X_2, X_1, *)$	5/31	5/26	5	0

Complexity. The time complexity of *sampleModels* is bounded by $O(n^2m)$, where n is the number of variables and m is the number of observations: the while-loop in line 2 runs at most n times (the worst case is that each variable needs to be removed one at a time from *candidates*). At each iteration, we have to process every observation, each time performing computations in $O(n)$ time. If we call *sampleModels* s times (to generate a sample of size s) then the total complexity of sampling is $O(sn^2m)$. For constant s, or with s bounded by a polynomial function of the other relevant quantities (n and m), this bound is still polynomial. Similarly, the complexity of *modelVote* is $O(sn)$ because it considers each of the s rules in the sample, counting the votes of each rule, which can be done in $O(n)$ time.

Comparison to variable voting. The set of LPMs that is sampled via *learnVariableRank* is a subset of the LPMs that *sampleModels* can produce and there are cases where this relationship is strict (models(*learnVariableRank*) $\subset$ models(*sampleModels*). For inclusion, we see that *sampleModels* without aggregates considers all possible models (since every variable is considered in every location in the LPM recursively). Thus, the LPMs consistent with the weak order returned by *learnVariableRank* must be a subset of these models. The strictness can be shown with the running example in the paper demonstrates that *sampleModels* can generate the LPM $(X_3, X_1, *)$; however, none of its extensions is consistent with the weak order returned by *learnVariableRank*.

Algorithm 4 *modelVote*

```
Require: A set of LPMs, $S$, and two objects, $A$ and $B$.
Ensure: Returns either one of $A$ or $B$ or $tie$.
 1: Initialize $sampleSize$ to the number of non-aggregated LPMs in $S$.
 2: for every aggregated LPM $\mathcal{L} \in S$ do
 3:     $sampleSize$+=$F_{\mathcal{L}}$.
 4: $Vote(A) = 0$; $Vote(B) = 0$;
 5: for every LPM $\mathcal{L} \in S$ do
 6:     if $\mathcal{L}$ is not an aggregate rule then
 7:         $winner$ is the object that $\mathcal{L}$ prefers among $A$ and $B$.
 8:         Increment $Vote(winner)$ by $1/sampleSize$.
 9:     else
10:         if $A$ and $B$ differ in at least one prefix variable of $\mathcal{L}$ then
11:             $\mathcal{L}^*$ is any extension of $\mathcal{L}$
12:             $winner$ is the object that $\mathcal{L}^*$ prefers among $A$ and $B$
13:             $Vote(winner)$ += $F_{\mathcal{L}}/sampleSize$.
14:         else
15:             $Vote(A)$ += $N^L_{A>B}/sampleSize$.
16:             $Vote(B)$ += $N^L_{B>A}/sampleSize$.
17: if $Vote(A) = Vote(B)$ then
18:     Return a $tie$
19: else
20:     Return the object $obj$ with the highest $Vote(obj)$.
```

4. Learning Preferred Attribute Values

In the previous sections, we assumed that the preferred value for each binary variable was known, so only the order of importance on the variables needed to be learned. In this section, we generalize the definition of an LPM to explicitly state the preferred value for each relevant variable. The motivation for this generalization is that the preferred value of a variable is not always known *a priori*. For example, in a meal preference learning situation, different groups of people might prefer different values of the “spicy” variable.

To represent this larger model space, we will use a pair of literals to represent each variable, similar to the trick used in learning Boolean formulae over binary variables. There are two *literals* l based on a variable X: the variable X (*positive literal*) and its negation $\neg X$ (*negative literal*). Given a set of variables V, let $L(V)$ be the set of all literals based on variables in V. A *generalized LPM* $\mathcal{L}$ on $L(V)$ is a total order on a subset R of $L(V)$ such that R does not contain both a positive and a negative literal based on the same variable. If A and B are two objects, then the preferred object given $\mathcal{L}$ is determined as follows:

- Find the smallest literal l in $\mathcal{L}$ such that if X is the variable l is based on, then $A(X)$ and $B(X)$ have different values. If l is a positive literal, then the object that has the value 1 for X is preferred; otherwise, the object that has the value 0 for X is preferred.

- If all relevant variables in $\mathcal{L}$ have the same value in A and B, then the objects are equally preferred (a tie).

Example 6. *Suppose $\neg X_1 < X_2 < X_3$ is the total order defined by a generalized LPM $\mathcal{L}$, and consider objects $A = [1, 0, 1, 1]$, $B = [0, 1, 0, 0]$ and $C = [0, 0, 1, 1]$. B is preferred over A because $B(X_1) = 0$, and $\neg X_1$ is the most important literal in $\mathcal{L}$. B is preferred over C because $B(X_2) = 1$ and both objects have the same value for X_1.*

Next we will adapt the voting algorithms described in previous sections to learn generalized LPMs.

4.1. Generalized Variable Voting

We can adapt the definition of variable voting (Definition 1) in a similar way to the LPM generalization above. Essentially, we need to define the weak order $\preceq$ over a set of literals instead of a set of variables. We also need to modify the way we count votes (N_A and N_B), such that among the voting literals, positive (negative) literals vote for the object that has 1 (0) for the variable the literal is based on. To avoid repetition, we will not formally define generalized variable voting, but the following example demonstrates the new vote-counting procedure.

Example 7. *Suppose that $\preceq$ is the weak order $\{X_2, X_3\} \preceq \{X_1, \neg X_2, \neg X_3, X_4, \neg X_5\} \preceq \{\neg X_1, \neg X_4, X_5\}$. Consider objects $A = [1, 0, 1, 1, 0]$ and $B = [0, 0, 1, 0, 1]$. The literals based on variables that are different in A and B are $D = \{X_1, \neg X_1, X_4, \neg X_4, X_5, \neg X_5\}$. The literals that get to vote are X_1, X_4, and $\neg X_5$ since they are the smallest ranking variables in D with respect to $\preceq$. X_1 votes for A because X_1 is a positive literal and $A(X_1) = 1$. Similarly, X_4 votes for A. $\neg X_5$ votes for B because it is a negative literal and $B(X_5) = 0$. Therefore, variable voting with $\preceq$ prefers A over B.*

Algorithm 5 *genLearnVariableRank*

```
Require: A set of variables $X$, and a set of observations $O$.
Ensure: A weak order on literals based on variables in $X$.
 1: $\Pi(x) = 1$ and $\Pi(\neg x) = 1, \forall\, x \in X$
 2: while $\Pi$ can change do
 3:     for every observation $(A, B) \in O$ do
 4:         $D$ is the set of literals based on variables that differ in $A$ and $B$
 5:         $D^* = \{x \in D | \forall y \in D, \Pi(x) \leq \Pi(y)\}$
 6:         $V_A$ is the set of positive (negative) literals in $D^*$ that are 1 (0) in $A$.
 7:         $V_B$ is the set of positive (negative) literals in $D^*$ that are 1 (0) in $B$.
 8:         for $x \in V_B$ such that $\Pi(x) < |X| + 1$ do
 9:             $\Pi(x) = \Pi(x) + 1$;
10: Return weak order $\preceq$ on $X$ such that $x \preceq y$ iff $\Pi(x) < \Pi(y)$.
```

As the previous example demonstrates, a literal votes only if its complement does not have a smaller rank. Furthermore, if both a literal and its complement have the same ranking, then their votes will cancel each other out and will not affect the preference decision. We note that in either case, since both the positive and negative literals appear in the variable ranking, special care must be taken when constructing an LPM from this weak ordering, a topic we return to at the end of this section.

Algorithm 5 presents the algorithm *genLearnVariableRank*. Given a set of observations, this algorithm learns the ranking of each literal and returns a weak order $\preceq$ on a subset of the literals. Generalized variable voting (as outlined above) with respect to $\preceq$ will correctly predict the preferred objects in the observations. The *genLearnVariableRank* algorithm is very similar to *learnVariableRank*; the major difference is that in *genLearnVariableRank*, the ranking function Π is over all possible literals and is updated when the prediction was wrong or correct but not unanimous. The rank of a literal is not incremented beyond one more than the number of variables.

Example 8. *Suppose* $X = \{X_1, X_2, X_3, X_4, X_5\}$ *and* O *consists of* $([0, 1, 1, 0, 0], [1, 1, 0, 1, 1])$, $([0, 1, 1, 0, 1], [1, 0, 0, 1, 0])$ *and* $([1, 0, 1, 1, 0], [0, 0, 1, 0, 1])$. *Table 3 illustrates the ranks of every literal based on variables in* X *after each iteration of the for-loop in line 3 of the algorithm genLearnVariableRank. The while-loop in line 2 of the algorithm terminates after two iterations. The algorithm genLearnVariableRank returns the weak order* $\{X_2, X_3\} \preceq \{X_1, \neg X_2, \neg X_3, X_4, \neg X_5\} \preceq \{\neg X_1, \neg X_4, X_5\}$.

Table 3: The rank of the literals after each iteration of the for-loop in line 3 of the algorithm *genLearnVariableRank*.

Observations	X_1	X_2	X_3	X_4	X_5	$\neg X_1$	$\neg X_2$	$\neg X_3$	$\neg X_4$	$\neg X_5$
$Initially$	1	1	1	1	1	1	1	1	1	1
$[0,1,1,0,0], [1,1,0,1,1]$	2	1	1	2	2	1	1	2	1	1
$[0,1,1,0,1], [1,0,0,1,0]$	2	1	1	2	2	1	2	2	1	2
$[1,0,1,1,0], [0,0,1,0,1]$	2	1	1	2	2	2	2	2	2	2
$[0,1,1,0,0], [1,1,0,1,1]$	2	1	1	2	2	2	2	2	2	2
$[0,1,1,0,1], [1,0,0,1,0]$	2	1	1	2	2	2	2	2	2	2
$[1,0,1,1,0], [0,0,1,0,1]$	2	1	1	2	3	3	2	2	3	2

The asymptotic bounds for the complexity and convergence of *learnVariableRank* also hold for *genLearnVariableRank*. However, for correctness, the relationship between the weak order $\preceq$ returned by *genLearnVariableRank* and the generalized LPMs consistent with the observations needs to be revised. Specifically, a topological sort of $\preceq$ will not be a valid generalized LPM because it will contain literals that are negations of each other. To correct this problem, we can simply discard any literal that has a higher rank than its opposite; if a pair of such literals appears in the same bin, we can discard them both. A topological sort can then be performed on the resulting weak order to produce a generalized LPM $\mathcal{L}$. Any LPM produced in such a manner is consistent with O because the discarded variables could never have been used by the learning algorithm to make, or even influence, a prediction.

4.2. Generalized Model Voting

The modifications to model voting can be considered in two parts. First, we need to extend the algorithm *sampleModels* to produce generalized LPMs. Algorithm 6 presents the algorithm *genSampleModels*, which operates on the level of literals and returns (possibly aggregated) generalized LPMs. Similar to *sampleModels*, a positive literal X is considered as a candidate for rule extension only if in every observation, either X is 1 in the preferred object or is the same in both objects. A negative literal $\neg X$ is a candidate when in every observation, either X is 0 in the preferred object or is the same in both objects. Note that to produce a valid LPM, we also need to ensure that the prefix contains at most one literal based on the same variable.

Second, we need to generalize the counting of model extensions for aggregate LPMs and the distribution of votes when comparing two objects. The number of

Algorithm 6 *genSampleModels*

```
Require: A set of variables $X$, a set of observations $O$, and rulePrefix, an LPM to be extended.
Ensure: An LPM (possibly aggregated) consistent with $O$.
 1: $candidates^+$ is the set of positive literals $\{Y : Y, \neg Y \notin rulePrefix \mid \forall(A,B) \in O, A(Y) = 1 \; or A(Y) = B(Y)\}$.
 2: $candidates^-$ is the set of negative literals $\{\neg Y : Y, \neg Y \notin rulePrefix \mid \forall(A,B) \in O, A(Y) = 0 \; or A(Y) = B(Y)\}$.
 3: $candidates = candidates^+ \cup candidates^-$
 4: while $candidates \neq \emptyset$ do
 5:     if $O = \emptyset$ then
 6:         return $(rulePrefix, *)$.
 7:     Randomly remove a variable $Z$ from $candidates$.
 8:     Remove any observation $(C, D)$ from $O$ such that $C(Z) \neq D(Z)$.
 9:     Extend $rulePrefix$: $rulePrefix = (rulePrefix, Z)$.
10:     Recompute $candidates$.
11: return $rulePrefix$
```

extensions of $\mathcal{L}$ is denoted by $F_{\mathcal{L}}$ and is equal to f_{n-k}, where n is the set of of variables the literals are based on, k is the length of the prefix in $\mathcal{L}$, and f_m is redefined as:

$$f_m = \sum_{i=0}^{m} \binom{m}{i} \times i! \times 2^i = \sum_{i=0}^{m} \frac{m! \times 2^i}{(m-i)!}. \tag{4}$$

The new definition of f_m has an extra 2^i term inside the summation because the extensions of i fixed variables include every combination of literals (positive or negative) for each variable.

Now consider a pair of objects, A and B, and an aggregated generalized LPM $\mathcal{L}$. As before, if A and B have different values for at least one prefix literal in $\mathcal{L}$, then all extensions will vote in accordance with the smallest such literal. However, if all prefix variables are tied, then in generalized model voting, the votes will be divided equally because there is an equal number of extensions with positive and negative literals based on the rest of the variables. Thus, the algorithm for generalized model voting will be the same as *modelVote*, except that the first line will call the algorithm *genSampleModels* and lines 14 to 16 (which compute the distribution of votes for aggregate LPMs when the prefix variables are tied) will be deleted.

5. Introducing Background Knowledge

In general, when there are not many training examples for a learning algorithm, the space of consistent LPMs is large. In this case, it is not possible to find a good approximation of the target model. To overcome this problem, we can introduce background knowledge, indicating that certain solutions should be favored over the others. In this section, we propose a form of background knowledge consisting of equivalence classes over the set of attributes. These equivalence classes indicate the set of most important attributes, second most important attributes, and so on. For example, when buying a used car, most people consider the most important attributes of a car to be the mileage, the year, and the make of the car. The second most important set of attributes is the color, number of doors, and body type. Finally, perhaps the least important properties are the interior color and the wheel covers. Throughout this section, we assume that the preferred value of a variable is known or is given in the background knowledge. We now formally define our representation for background knowledge and what it means for an LPM to be consistent with the background knowledge.

Definition 3 (Background knowledge). *The background knowledge $\mathcal{B}$ for learning a lexicographic preference model on a set of variables X is a weak order: that is, a total order on a partition of X. $\mathcal{B}$ has the form $E_1 < E_2 < \ldots < E_k$, where $\cup_i E_i = X$. $\mathcal{B}$ defines a weak order on X such that for any two variables $x \in E_i$ and $y \in E_j$, $x < y$ iff $E_i < E_j$. We denote this weak order by $\preceq_B$.*

Definition 4. *Suppose that $X = \{X_1, \ldots X_n\}$ is a set of variables, $\mathcal{B}$ the background knowledge, and $\mathcal{L}$ an LPM. $\mathcal{L}$ is consistent with $\mathcal{B}$ iff the total order $\sqsubset_{\mathcal{L}}$ is consistent with the weak order $\preceq_{\mathcal{B}}$.*

Intuitively, an LPM that is *consistent* with background knowledge $\mathcal{B}$ respects the variable orderings induced by $\mathcal{B}$. The background knowledge prunes the space of possible LPMs. The size of the partition determines the strength of $\mathcal{B}$; for example, if there is a single variable per set, then $\mathcal{B}$ defines a specific LPM. In general, the number of LPMs that is consistent with background knowledge of the form $E_1 < E_2 < \ldots < E_k$ can be computed with the following recursive formula:

$$G([e_1, \ldots e_k,]) = f_{e_1} + e_1! \times (G([e_2, \ldots e_k]) - 1), \tag{5}$$

where $e_i = |E_i|$ and the base case for the recursion is $G([]) = 1$. The first term in the formula counts the number of possible LPMs using only the variables in E_1, which are the most important variables. The definition of consistency entails that a variable can appear in $\sqsubset_{\mathcal{L}}$ iff all of the more important variables are already in $\sqsubset_{\mathcal{L}}$, hence the term $e_1!$. Note that the recursion on G is limited to the number of sets in the partition, which is bounded by the number of variables; therefore, it can also be computed in linear time by caching precomputed values of f.

To illustrate the potential power of background knowledge, consider a learning problem with nine variables. Without background knowledge, the total number of LPMs is 905,970. If the background knowledge $\mathcal{B}$ partitions the variables into three sets, each with three elements, then the number of LPMs consistent with $\mathcal{B}$ is only 646. If $\mathcal{B}$ has four sets, where the first set has three variables and the rest have two, limits the number to 190.

We can easily generalize the *learnVariableRank* algorithm to utilize background knowledge, by changing only the first line of *learnVariableRank*, which initializes the ranks of the variables. Given background knowledge of the form $S_1 < \ldots < S_k$, the generalized algorithm assigns the rank 1 (most important rank) to the variables in S_1, rank $|S_1| + 1$ to those in S_2, and so forth. This initialization ensures that an observation (A, B) is used for learning the order of variables in a class S_i only when A and B have the same values for all variables in classes $S_1 \ldots S_{i-1}$ and have different values for at least one variable in S_i.

The algorithm *modelVote* can also be generalized to use background knowledge $\mathcal{B}$. In the sample generation phase, we use *sampleModels* as presented earlier, and then eliminate all rules whose prefixes are not consistent with $\mathcal{B}$. Note that even if the prefix of an aggregated LPM $\mathcal{L}$ is consistent with $\mathcal{B}$, this may not be the case for every extension of $\mathcal{L}$. Thus, in the algorithm *modelVote*, we need to change any references to $F_{\mathcal{L}}$ and $N^{\mathcal{L}}_{A<B}$ (or $N^{\mathcal{L}}_{B<A}$) with $F^{\mathcal{B}}_{\mathcal{L}}$ and $N^{\mathcal{L},\mathcal{B}}_{A<B}$ (or $N^{\mathcal{L},\mathcal{B}}_{B<A}$), respectively, where:

- $F^{\mathcal{B}}_{\mathcal{L}}$ is the number of extensions of $\mathcal{L}$ that are consistent with $\mathcal{B}$, and
- $N^{\mathcal{L},\mathcal{B}}_{A<B}$ is the number of extensions of $\mathcal{L}$ that are consistent with $\mathcal{B}$ and prefer A. ($N^{\mathcal{L},\mathcal{B}}_{B<A}$ is analogous.)

Suppose that $\mathcal{B}$ is given as $E_1 < \ldots < E_m$. Let Y denote the prefix variables of an aggregate LPM $\mathcal{L}$ and let E_k be the first set such that at least one variable in E_k is not in Y. Then, $F^{\mathcal{B}}_{\mathcal{L}} = G([|E_k - Y|, |E_{k+1} - Y|, \ldots |E_m - Y|])$.

When counting the number of extensions of $\mathcal{L}$ that are consistent with $\mathcal{B}$ and prefer A, we again need to examine the case where the prefix variables equally prefer the objects. Suppose Y is as defined as above and D_i denotes the set difference between E_i and Y. Let D_j be the first non-empty set and D_k be the first set such that at least one variable in D_k has different values in the two objects. Obviously, only the variables in D_k will influence the prediction of the preferred object. If

- $d_i = |D_i|$, the cardinality of D_i, and
- a is the set of variables in D_k that favor A, b is the set of variables in D_k that favor B, and w is the set of variables in D_k that are neutral,

then $N^{\mathcal{L},\mathcal{B}}_{A>B}$, the number of extensions of $\mathcal{L}$ that are consistent with $\mathcal{B}$ and prefer A, can be computed as follows:

$$N^{\mathcal{L},\mathcal{B}}_{A>B} = \frac{|a|}{|a|+|b|} \times (F^{\mathcal{B}}_{\mathcal{L}} - G([d_j \ldots d_{k-1}, |w|])). \tag{6}$$

6. Handling Noisy Data

Although inferring an LPM from noisy data is NP-complete [16], the moderate empirical success of the Greedy algorithm (as seen in Section 7.7) give us an intuition as to how a heuristic solution with voting can be developed. Specifically, the greedy approach iteratively constructed an LPM where each added attribute violated the fewest number of observations. We can borrow this intuition to build heuristic extensions of *learnVariableRank* and *modelVote*. If the expected number of noisy observations ϵ in a data set is provided, then the new algorithm, Noise-Aware Model Vote ($NAMV$), changes *sampleModels* to only consider a variable as a *candidate* if adding it will not violate more (in total with the other variables) than ϵ observations. Notice that this remains a stochastic LPM construction and can still consider many more LPMs than the greedy approach. Following a similar path, we can develop a noise-aware version of *learnVariableRank* in which the ranks of the variables are not updated unless at least ϵ observations are mispredicted.

If we do not know the expected amount of noise in our observations, then we can employ a hybrid approach between *greedyPermutation* and *modelVote*, which we call *greedyVote*, by simply replacing the sampling algorithm *sampleModels* with *greedyPermutation*. In doing so, we will be losing the advantage of aggregate LPMs (since *greedyPermutation* produces a single LPM) and we will be confining our samples to the ones that can only be generated by the “make minimum mistakes” heuristic. We will investigate the resilience of both extensions empirically in Section 7.7.

7. Experiments

In this section, we explain our experimental methodology and discuss the results of our empirical evaluations. We define the *prediction performance* of an algorithm P with respect to a set of test observations T as:

$$\textit{performance}(P,T) = \frac{\textit{Correct}(P,T) + 0.5 \times \textit{Tie}(P,T)}{|T|}, \tag{7}$$

where *Correct*(P,T) is the number of observations in T that are predicted correctly by P (including any prediction for $t \in T$ where t is actually a tie) and *Tie*(P,T) is the number of observations in T that P predicted as a tie when one object should actually have been preferred over the other. Note that an LPM returned by *greedyPermutation* never returns a tie. In contrast, variable voting with respect to a weak order in which every variable is equally important will only return ties, so the overall performance will be 0.5, which is no better than randomly selecting the preferred objects. We will use MV, VV, and G to denote the model voting, variable voting, and the greedy approximations of an LPM. Similarly we will use $NAMV$ and GV to denote Noise-Aware Model Vote and *greedyVote*.

Given sets of training and test observations, (O,T), we measure the *average* and *worst* performances of VV, MV and G. When combined with *learnVariableRank*, VV is a deterministic algorithm, so the average and worst performances of VV are the same. However, this is not the case for MV with sampling, because *sampleModels* is randomized. Even for the same training and test data (O,T), the performance of MV can vary. To mitigate this effect, we ran MV 10 times for each (O,T) pair, and called *sampleModels* S times on each run (thus the sample size is at most S), recording the average and worst of its performance. The greedy algorithm G is also randomized (in line 2, one variable is picked arbitrarily), so we ran G 200 times for every (O,T), recording its average and worst performance. In all of the figures below, the data points are averages over 20 runs with different (O,T). We employed two-tailed T-test with 95% confidence interval to test significance. In our discussion of the results, when applicable, we note the statistically significant differences.

For our experiments, the control variables are R, the number of relevant variables in the target LPM; I, the number of irrelevant variables; N_O, the number of training observations; and N_T, the number of test observations. For MV experiments, the sample size (S) is also a control parameter. For fixed values of R and I, an LPM $\mathcal{L}$ is randomly generated. (If background knowledge $\mathcal{B}$ is given, then $\mathcal{L}$ is also consistent with $\mathcal{B}$.) Unless otherwise noted (as in Section 7.5),

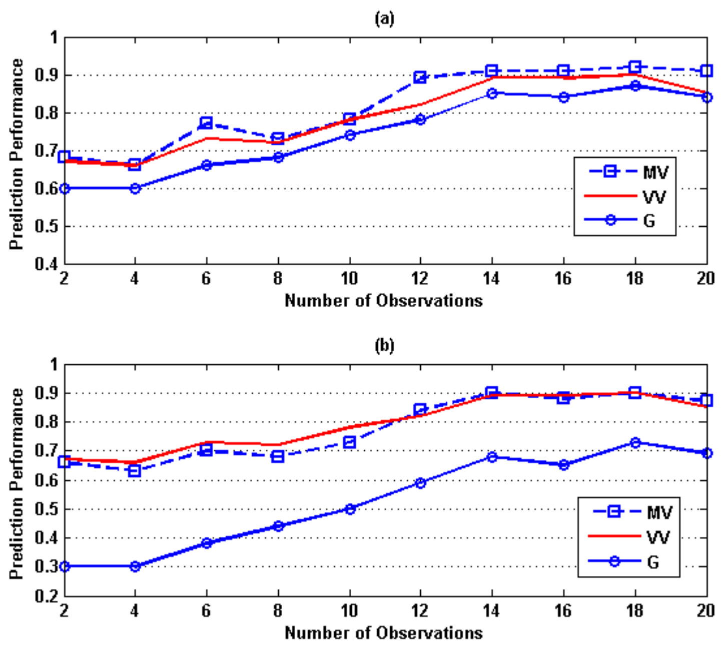

Figure 1: (a) Average prediction performance and (b) worst prediction performance of the greedy algorithm, variable voting, and model voting.

the preferred value of each attribute is 1; otherwise, the preferred value of each attribute is chosen randomly. We randomly generated N_O and N_T pairs of objects, each with $I + R$ variables. Finally, we labeled the preferred objects according to $\mathcal{L}$. In order to allow other researchers replicate our results, we posted the data files and the scripts that we used for data generation on the web at `http://maple.cs.umbc.edu/LPM`.

7.1. Comparison of MV, VV and G

Figure 1(a) shows the average performance of G; MV with sample size $S = 200$; and VV for $R = 15$, $I = 0$, and $N_T = 20$, as N_O ranges from 2 to 20. Figure 1(b) shows the worst performance for each algorithm. In these figures, the data points are averages over 20 different pairs of training and test sets (O, T). The average performance of VV and MV is better than the average performance of G, and the difference is significant at every data point. Also, note that the worst-case performance of G after seeing two observations is around 0.3, which suggests a very poor approximation of the target. VV and MV's worst-case performance are much better than the worst-case performance of G, justifying the additional complexity of the algorithms MV and VV.

7.2. Greedy Voting

Even though we proposed *greedyVote* as a noise-aware adaptation of model vote we also investigated its performance for the noise-free case. For this experiment, we used the same data set and control variables explained in 7.1. In addition, the sample size for GV is set to 200 as it is for MV. Figure 2 contains the average performance of MV, VV and G and as well as the worst performance of MV which were already reported in Figure 1 (a) and (b). Figure 2 demonstrates the average and worst case performance of GV. Just like MV, GV is a randomized algorithm, thus for each data set we ran GV ten times and used the average of ten runs as the prediction performance. An interesting result is GV's average performance is very close to VV's performance and its worst case performance is almost same as the average performance of G. Therefore, by virtue of being a voting-based algorithm, GV demonstrates a much better worst case performance than the greedy algorithm; however, the worst case performance is still significantly worse than MV and VV. We believe that this behavior occurs because GV uses the greedy approach for sampling the space of consistent LPMs, as reflected in the tight overlap between worst GV and average G performances.

7.3. Effect of Sample Size on MV Performance

Figure 3 shows the worst and average prediction performance of MV with sample sizes $S = 10$, $S = 50$, $S = 200$ and $S = 1200$ for problems with 10 relevant variables ($R = 10$) and and no irrelevant variables ($I = 0$). The number of observations (N_o) increases from 2 to 20 along the x-axis. In general, as the sample size increases, the prediction performance increases. The effect of sample size on worst-case performance is more evident than the average performance.

7.4. Irrelevant Variables

Irrelevant variables hamper the pruning of the space of possible LPMs. Of the two voting-based algorithms, MV has the ability to ignore some of the variables early on if *sampleModels* produces LPMs that do not use all of the variables. VV, on the other hand, operates with the entire set of variables, although given enough observations, it eventually discovers the irrelevant ones. In our experiments, we

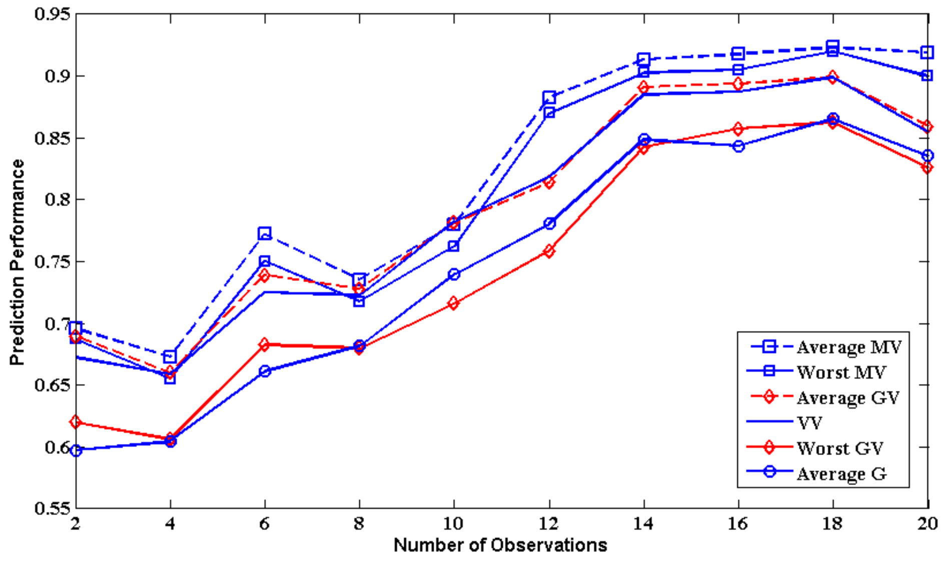

Figure 2: The average and worst case prediction performance of *greedyVote* compared to model voting, variable voting, and greedy approaches for noise-free data.

compared the average performance of MV and VV for cases where the total number of attributes were constant and the number of irrelevant attributes varied. Our results showed that both algorithms are robust in the presence of irrelevant variables. Furthermore, for several test cases where the number of irrelevant variables dominated the number of relevant variables, or when the total number of attributes was small, the performance of both algorithms improved over the case where only relevant variables appeared. That is, all things being equal, the algorithms found it easier to learn to ignore an irrelevant variable than to use a relevant one.

Figure 4 depicts two such cases. The graph on the left shows the average prediction performance of MV and VV for $R = 2, I = 3$ and $R = 5, I = 0$. The data sets with fewer relevant variables are learned more easily. The right shows the results of the same comparison and the same pattern of results with more attributes: $R = 3$, $I = 12$ and $R = 15$, $I = 0$. We have seen this pattern in several other similar situations. Note that the graphs show model voting outperforming variable voting, but we have not observed this difference to be statistically significant. Our experiments indicate that both algorithms are robust in the presence of irrelevant variables, still achieving high accuracy values with relatively few samples.

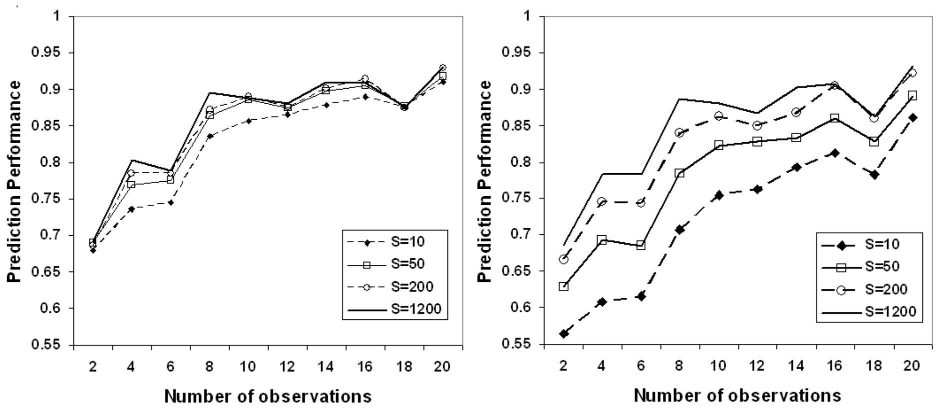

Figure 3: (a) The average prediction performance and (b) the worst prediction performance of model voting for different sample sizes.

7.5. *Learning Preferred Variable Values*

We implemented the generalized versions of MV and VV, which we will refer as gMV and gVV, respectively. We tested the performance of gMV and gVV using two different data sets. The first data set is same as the one used for comparing MV and VV where $R = 15$ and $I = 0$ and for all variables 1 is always preferred over 0. Figure 5(a) shows the average and worst performance of MV and VV. A quick comparison to Figure 1 reveals the decreased prediction performance (which is statistically significant) in all three. This was an expected result, given the increase in the search space. The second data set has $R = 5$ and $I = 10$ and the preferred value of variables are chosen randomly. Figure 5(b) demonstrates the performance of MV and VV in a very similar pattern to Figure 5(a).

7.6. *Effect of Hidden Ties*

Figure 6 shows the prediction performance of VV and G (average and worst case) for problems with 10 relevant variables ($R = 10$) and five irrelevant variables ($I = 5$). The number of observations (N_o) is always 50, but the number of these observations that are hidden ties varies from 0 to 45 (x-axis).

In general, as the number of hidden ties increase in the observations, the prediction performance degrades. This result was expected because the number of useful observations that can help the algorithms learn the ranking on the relevant attributes decreases as the number of hidden ties increases. The performance of

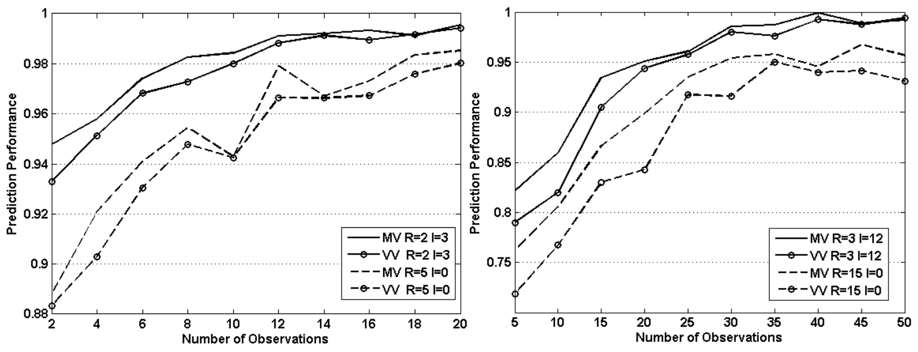

Figure 4: Average MV performance and VV performance for different numbers of relevant and irrelevant variables.

MV on the same data sets was very similar to VV and has thus been omitted for readability.

A more interesting result, however, is that the existence of hidden-ties in the data actually *improves* the performance of both algorithms, compared to a smaller dataset with the hidden tie observations omitted (the "filtered" versions in Figure 6). Our explanation for this phenomenon is that although hidden ties do not provide useful information for learning the order on the relevant attributes, their existence helps the algorithms identify the irrelevant attributes (because hidden ties can increase the number of mistakes that would be caused only by the irrelevant attributes) and push them further up in the ranking (decreasing their importance), thus allowing the other observations to clarify the ordering of the identified relevant attributes.

7.7. Effect of Noise

Figure 7 shows the average prediction performance of MV and G for problems with 10 relevant variables ($R = 10$) and five irrelevant variables ($I = 5$). The total number of observations (N_o) is always 50 but the number of these observations that are faulty varies from 0 to 45 (x-axis). Figure 8 shows the worst performance for the same setting.

The results show that both the average and the worst performance of MV (which operates under the assumption of noise-free data) are significantly compromised by even small amounts of noise, which it interprets as a refutation of

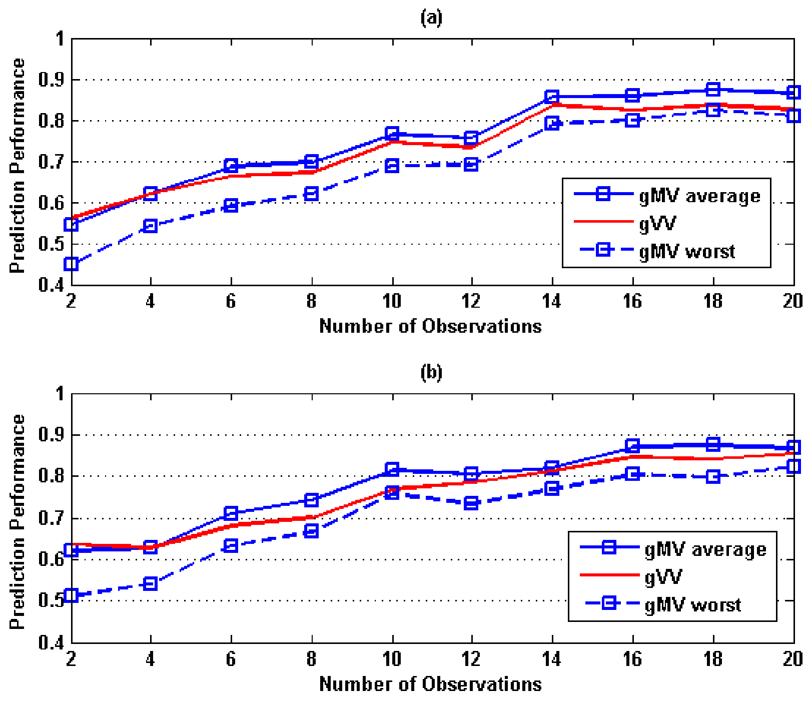

Figure 5: (a) Comparison of gMV and gVV for a data set with 15 relevant variables and no irrelevant variables, where 1 is always the preferred value. (b) Comparison of gMV and gVV for a data set with 5 relevant and 10 irrelevant variables, where the preferred value of each attribute is chosen randomly.

the correct model (as well as many of the "almost correct" models). The asymptotic performance at 0.5 reflects the fact that this noise causes MV to eliminate all of the models from its version space, causing it to predict a tie for every testing observation (essentially making it a random selection algorithm). We omitted the results for VV from the figure since VV's behavior closely resembled that of MV. The performance of G decays far more gracefully than MV or VV because G allows for some of the observations to be discarded. In Figures 7 and 8, we compare the average and worst performance of $NAMV$ and GV to the other algorithms presented in this paper, including "filtered" versions where the noisy data was omitted (which provided as baselines). Notice that unlike the original $modelVote$, which was confounded by even a small amount of noise, the gentle

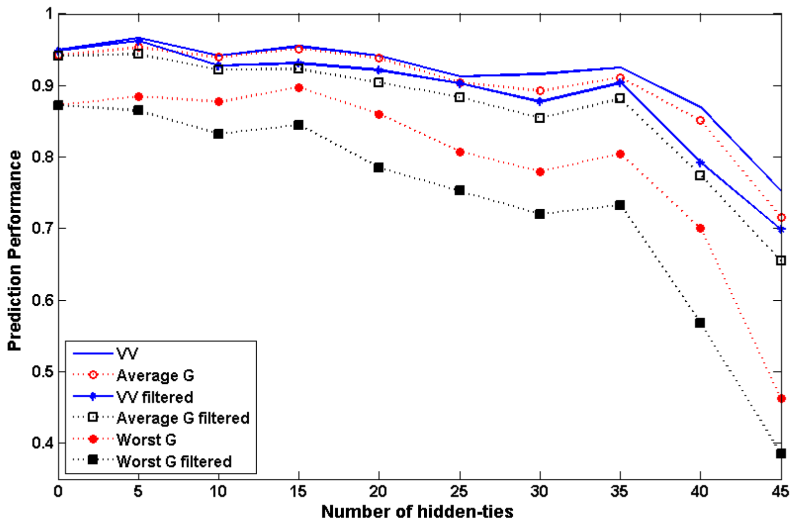

Figure 6: The prediction performance of VV and G for varying number of hidden ties in 50 observations. The results for filtered VV and G are obtained by eliminating the hidden ties from the observations.

decays of $NAMV$ and GV mirror the greedy approach's robustness to noise and perform comparably on this data set. Asymptotically, if all of the observations are noisy, $NAMV$ still performs better than $modelVote$ in the average case (a statistically significant result), because it does not eliminate all of the possible models, so instead of defaulting to random selection, it favors the test observation with the most 1s. Among the two noise adaptations of MV, GV demonstrates the best worst-case performance (for noise levels 10 to 25 the difference between GV and $NAMV$ is statistically significant).

7.8. Effect of Background Knowledge on Performance

Figure 9 shows the positive effect of incorporating background knowledge on the performance of voting algorithms for $R = 10$, $I = 0$, and $N_T = 20$, as N_O ranges from 2 to 20. In addition, this experiment aims to show that background knowledge does not undermine the advantage that voting algorithms held over the greedy algorithm in the knowledge-free case. To this end, we have trivially generalized G to produce LPMs that are consistent with given background knowledge $\mathcal{B}$. The data points are averages over 20 different pairs of training and

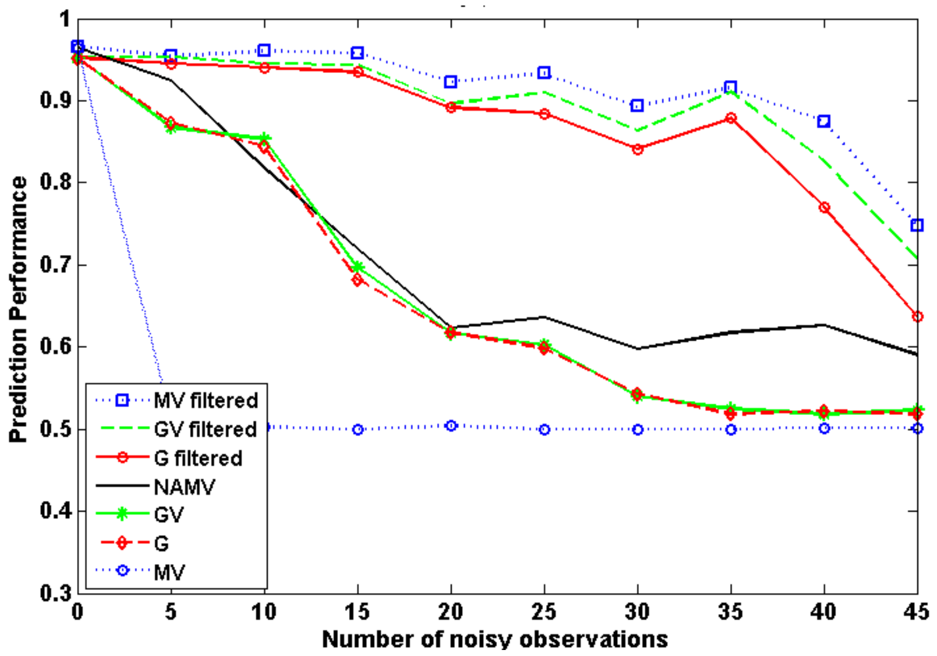

Figure 7: The average prediction performance of model voting, greedy, $NAMV$, and GV as the number of noisy observations increases.

test sets (O, T). We have arbitrarily picked two weak orderings to use as background knowledge: $\mathcal{B}_1 : \{X_1, X_2, X_3, X_4, X_5\} < \{X_6, X_7, X_8, X_9, X_{10}\}$ and $\mathcal{B}_2 : \{X_1, X_2, X_3\} < \{X_4, X_5\} < \{X_6, X_7, X_8\} < \{X_9, X_{10}\}$. The performance of VV improved greatly with the introduction of each ordering as background knowledge. $\mathcal{B}_2$ is stronger than $\mathcal{B}_1$, and therefore prunes the space of consistent LPMs more than $\mathcal{B}_1$. As a result, the performance gain due to $\mathcal{B}_2$ is greater than that due to $\mathcal{B}_1$. The difference between the performance with background knowledge and without background knowledge is statistically significant except at the last point. Note that using background knowledge is are particularly effective when the number of training observations is small. The worst-case performance of G with background knowledge $\mathcal{B}_1$ and $\mathcal{B}_2$ are also shown in Figure 9. In both cases, the worst-case performance of G is significantly lower than the performance of VV with the corresponding background knowledge.

Using the same experimental scenario, we obtained very similar results with MV, as seen in Figure 10. In summary, the worst case performance of greedy al-

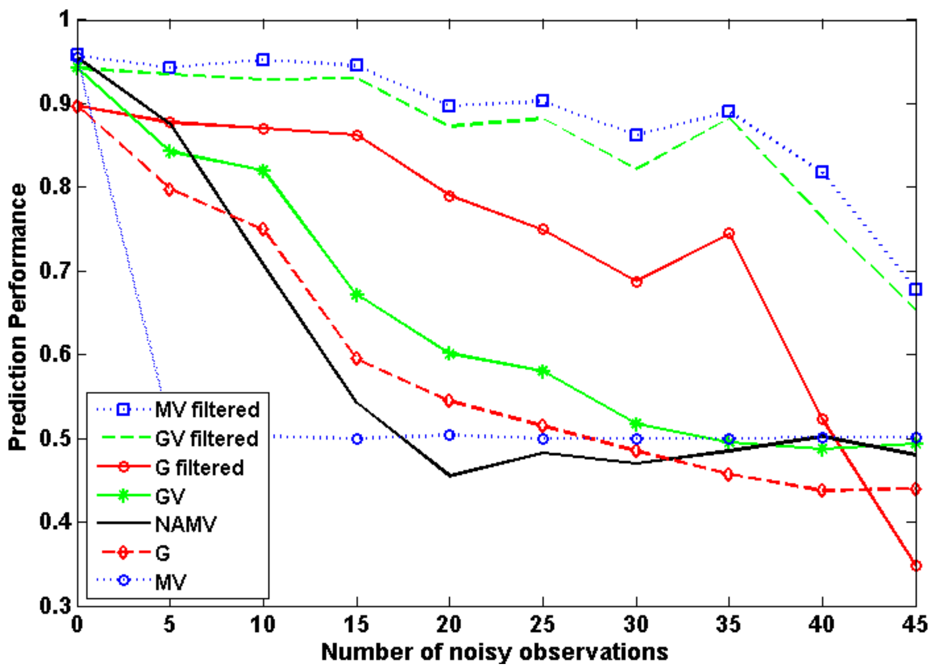

Figure 8: The worst prediction performance of model voting, greedy, $NAMV$, and GV as the number of noisy observations increases.

gorithm with background knowledge $\mathcal{B}_2$ outperforms the average performance of MV without any background knowledge. However, even with weaker knowledge, such as $\mathcal{B}_1$, the average performance of MV is better than G with $\mathcal{B}_2$.

8. Related Work

The concept of lexicographic comparisons is commonplace in everyday life. Expressions such as "safety first," "above all else do no harm," or "quality is job one" all evoke lexicographic preferences. Lexicographic rankings are often used in sporting events. For example, countries competing in the Olympics are typically ranked by total medals, then gold medals, then silver medals [17]. The winner of the Netflix prize was chosen by ranking submissions first by minimum test error, then by earliest submission [13].

Lexicographic utilities have been applied to understanding human preferences [19, 11]. They have an extensive mathematical foundation that has been

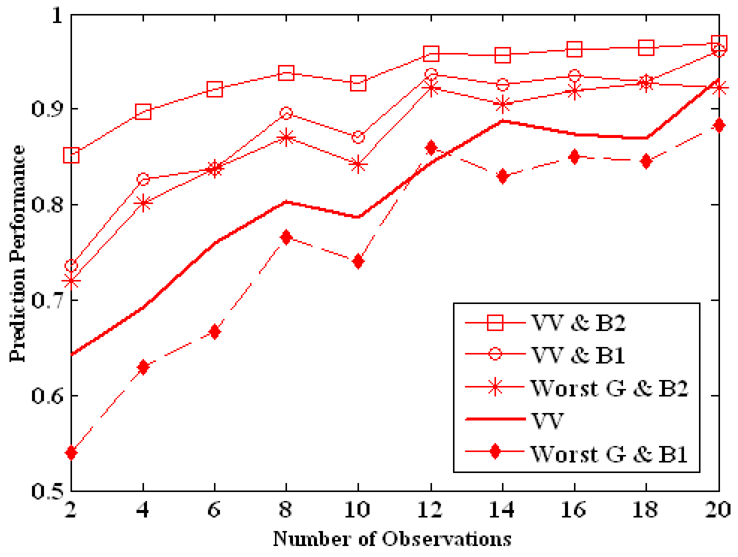

Figure 9: The effect of background knowledge on VV and G, using two arbitrarily selected weak orderings as background knowledge, where $\mathcal{B}_2$ is stronger (more constraining) than $\mathcal{B}_1$.

studied in the economics, psychological, and management science literatures [8]. Lexicographic orders and other preference models have been utilized in several research areas, including multicriteria optimization [1], linear programming [5], and game theory [14].

The most relevant existing work for learning and/or approximating LPMs is by Schmitt and Martignon [16] and Dombi et al. [7], which were summarized in Section 2.

In general, preferences and ranking are similar. The ranking problem, as described by Cohen et al. [3], is similar to the problem of learning an LPM. However, that line of work poses learning as an optimization problem, with the goal of finding the single ranking that maximally agrees with the given preference function. In particular, their approach constructs a collection of domain-specific “ranking experts” whose predictions are combined using a model voting scheme. The voting concept is similar in spirit to our approach, but the underlying representations are quite different.

Torrey et al. [18] employ an inductive logic programming approach to learn multi-attribute ranking rules. In principle, these rules can represent lexicographic preference models.

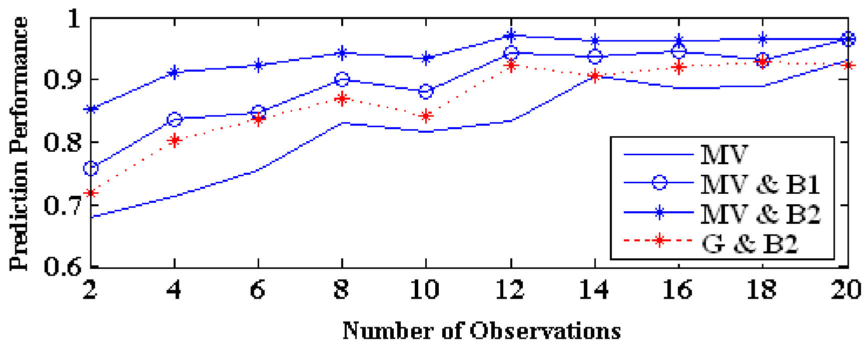

Figure 10: The effect of background knowledge on average MV performance, using two arbitrarily selected weak orderings as background knowledge, where $\mathcal{B}2$ is stronger than $\mathcal{B}1$.

Fürnkranz and Hüllermeier [10] investigate the pairwise preference learning and ranking problem. The observations are a set of partially ranked objects and the goal is to learn how to rank a new set of objects. Their approach is to reduce the original problem to a number of binary classification problems, one for each pair of labels. Hence, they make no assumptions about the underlying preference model.

Boutilier et al. [2] consider a preference learning algorithm and representation (CP-Nets) for modeling preferences under a *ceteris paribus* (all else being equal) assumption. However, this representation will not necessarily capture lexicographic preference models, and is therefore not directly applicable to the problem we have considered.

Another analogy, described by Schmitt and Martignon [16], is between LPMs and decision lists [15]. Specifically, it was shown that LPMs are a special case of 2-decision lists, but that the algorithms for learning these two classes of models are not directly applicable to each other.

9. Conclusions and Future Work

In this paper, we presented democratic approximation methods for learning lexicographic preference models (LPMs) given a set of preference observations. Instead of committing to just one of the consistent LPMs, we maintain a set of models and predict based on the majority of votes. We described two such methods: *variable voting* and *model voting*. We showed that both methods can be implemented in polynomial time and exhibit much better worst- and average-case performance than the existing methods. Finally, we have defined a form of background knowledge that can be used to improve performance when the number of observations is small; we incorporated this background knowledge into the voting-based methods, significantly improving their empirical performance.

Future directions of this work allow for a number of extensions and further theoretical investigations. Many of the theoretical bounds presented in this paper could be tightened (such as the complexity bound of *learnVariableRank*) or potentially defined in terms of simpler parameters (such as the sample size parameter for model vote), while still maintaining performance guarantees. We have recently extended the basic LPM representation and learning techniques to support context-dependent preferences in the form of *branching LPMs*, including methods for learning branching LPMs from noisy data [12]. We are also continuing to investigate heuristics like $NAMV$ and *greedyVote* that make them more robust against noise. While the problem of learning LPMs from noisy data is NP-complete, the superior performance of the voting algorithms over the greedy method in the noise-free case indicates that it may be possible to identify and characterize other restricted problem settings in which heuristic extensions such as $NAMV$ and *greedyVote* would significantly outperform the state-of-the-art greedy approach.

Acknowledgments

This work was supported by the Defense Advanced Research Projects Agency and the U. S. Air Force through BBN Technologies Corp. under contract number FA8650-06-C-7606.

References

[1] Bertsekas, D., Tsitsiklis, J., 1997. Parallel and distributed computation: numerical methods. Athena Scientific.

[2] Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., Poole, D., 2004. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. Journal of Artificial Intelligence Research 21, 135–191.

[3] Cohen, W., Schapire, R., Singer, Y., 1999. Learning to order things. Journal of Artificial Intelligence Research 10, 243–270.

[4] Colman, A. M., Stirk, J. A., December 1999. Singleton bias and lexicographic preferences among equally valued alternatives. Journal of Economic Behavior & Organization 40 (4), 337–351.

[5] Dantzig, G., Orden, A., Wolfe, P., 1955. The generalized simplex method for minimizing a linear form under linear inequality restraints. Pacific Journal of Mathematics 5, 183–195.

[6] Dawes, R. M., 1979. The robust beauty of improper linear models in decision making. American Psychologist 34, 571–582.

[7] Dombi, J., Imreh, C., Vincze, N., 2007. Learning lexicographic orders. European Journal of Operational Research 183 (2), 748–756.

[8] Fishburn, P., 1974. Lexicographic orders, utilities and decision rules: A survey. Management Science 20 (11), 1442–1471.

[9] Ford, J. K., Schmitt, N., Schechtman, S. L., Hults, B. M., Doherty, M., 1989. Process tracing methods: contributions, problems and neglected research issues. Organizational Behavior and Human Decision Processes 43, 75–117.

[10] Fürnkranz, J., Hüllermeier, E., 2003. Pairwise preference learning and ranking. In: In Proc. ECML-03, Cavtat-Dubrovnik. Springer-Verlag, pp. 145–156.

[11] Gigerenzer, G., Goldstein, D. G., 1996. Reasoning the fast and frugal way: Models of bounded rationality. Psychological Review 103 (4), 650–669.

[12] Jones, J., desJardins, M., June 2010. Learning of branching lexicographic preference models. Tech. Rep. TR-CS-10-05, UMBC Dept. of CS&EE.

[13] Netflix, 2009. The Netflix prize rules. http://www.netflixprize.com/rules.

[14] Quesada, A., 2003. Negative results in the theory of games with lexicographic utilities. Economics Bulletin 3 (20), 1–7.

[15] Rivest, R., 1987. Learning decision lists. Machine Learning 2 (3), 229–246.

[16] Schmitt, M., Martignon, L., 2006. On the complexity of learning lexicographic strategies. Journal of Machine Learning Research 7, 55–83.

[17] Sports Illustrated, 2008. Medal tracker. http://sportsillustrated.cnn.com/olympics/2008/medals/tracker/.

[18] Torrey, L., Shavlik, J., Walker, T., Maclin, R., 2007. Relational macros for transfer in reinforcement learning. In: Proceedings of the Seventeenth Conference on Inductive Logic Programming. Corvallis, Oregon.

[19] Tversky, A., 1969. Intransitivity of preferences. Psychological Review 76, 31–48.

[20] Westenberg, M. R., Koele, P., 1994. Multi-attribute evaluation processes: methodological and conceptual issues. Acta Psychologica 87, 65–84.

[21] Yaman, F., desJardins, M., 2007. More-or-less CP-Networks. In: Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence.

Fusun Yaman is a computer scientist at BBN Technologies. She received her Ph.D. (2006) in Computer Science from University of Maryland at College Park. Prior to joining BBN Technologies she was a postdoctoral research assistant at the University Maryland, Baltimore County under the supervision of Marie desJardins and Tim Oates. Her research interests are AI planning, specifically plan management frameworks for supporting distributed planning, and learning processes and preferences from demonstrations.

Thomas J. Walsh is a postdoctoral research assistant at the University of Arizona. He received his Ph.D. in 2010 from the Department of Computer Science at Rutgers University under the direction of Michael L. Littman in the Rutgers Laboratory for Real Life Reinforcement Learning (RL^3). His research focuses on efficient learning in sequential decision making problems with rich structure.

Michael L. Littman is professor and chair of the Department of Computer Science at Rutgers University and directs the Rutgers Laboratory for Real-Life Reinforcement Learning (RL^3). His research in machine learning examines algorithms for decision making under uncertainty. He has served on the editorial boards for several machine-learning journals and was Program Co-chair of ICML 2009.

Dr. Marie desJardins is an associate professor in the Department of Computer Science and Electrical Engineering at the University of Maryland, Baltimore County. Prior to joining the faculty in 2001, Dr. desJardins was a senior computer scientist at SRI International. Her research is in AI, focusing on the areas of machine learning, multi-agent systems, planning, interactive AI techniques, information management, reasoning with uncertainty, and decision theory.