

# Democratic Approximation of Lexicographic Preference Models

Fusun Yaman<sup>†</sup> and Thomas J. Walsh<sup>‡</sup> Michael L. Littman<sup>‡</sup> and Marie desJardins<sup>†</sup>

<sup>†</sup>Department of Computer Science and Electrical Engineering  
University of Maryland, Baltimore County  
1000 Hilltop Circle, Baltimore, MD 21250  
{fusun,mariedj}@cs.umbc.edu

<sup>‡</sup>Department of Computer Science  
Rutgers University  
110 Frelinghuysen Rd., Piscataway, NJ 08854  
{thomaswa,mlittman}@cs.rutgers.edu

## Abstract

Previous algorithms for learning lexicographic preference models (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. We also introduce an intuitive yet powerful learning bias to prune some of the possible LPMs, incorporate this bias into our algorithms, and demonstrate its effectiveness when data is scarce.

## Introduction

Lexicographic preference models (LPMs) are one of the simplest 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. Previous work on learning LPMs from a set of preference observations has been limited to autocratic approaches: one of many possible LPMs is picked arbitrarily and used for future decisions. However, autocratic methods will likely 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 on the variable level and samples the consistent LPMs implicitly. Model voting explicitly samples the consistent LPMs and employs weighted voting where the weights are computed using Bayesian priors. The additional complexity of voting-based algorithms is tolerable: both algorithms have low-order polynomial time complexity. Our experiments demonstrate these democratic algorithms outperform the “state of the art” greedy approach. We also

show how to incorporate domain knowledge, in the form of a powerful learning bias, into these algorithms.

## Lexicographic Preference Models

In this section, we briefly introduce the lexicographic preference model (LPM) and summarize previous results on learning LPMs. Throughout this work, we consider binary variables whose domain is  $\{0, 1\}$ <sup>1</sup>, where the preferred value of each variable is known. 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 to 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 respect to  $\mathcal{L}$ ; similarly, any variable in  $X - R$  is *irrelevant* with respect to  $\mathcal{L}$ . If  $A$  and  $B$  are two objects, then the preferred object given  $\mathcal{L}$  is determined as follows:

- Find the smallest 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  $B$ . However, preference observations are often gathered from expert demonstrations, with ties broken arbitrarily. Thus, for some observations,  $A$  and  $B$  may actually be tied. An LPM  $\mathcal{L}$  is *consistent* with an observation  $(A, B)$  iff  $\mathcal{L}$  implies that  $A$  is preferred to  $B$  or that  $A$  and  $B$  are equally preferred.

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

<sup>1</sup>The representation can easily be generalized to monotonic preferences with ordinal variables.

---

**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  $i = 1, \dots, n$  do
2:   Arbitrarily pick one of  $X_j \in X$  such that
      $MISS(X_j, O) = \min_{X_k \in X} MISS(X_k, O)$ 
3:    $\pi(X_j) := i$ , 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
      $\pi(X_i) < \pi(X_j)$ 
```

---

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.

Schmitt and Martignon (2006) proposed a greedy polynomial-time 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 data case, finding an LPM that does not violate more than a constant number of the observations is NP-complete. Algorithm 1 is Schmitt and Martignon (2006)’s greedy variable-permutation algorithm, which we use as a performance baseline. The algorithm refers to a function  $MISS(X_i, O)$ , which is defined as  $|\{(A, B) \in O : A(X_i) < B(X_i)\}|$ ; 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.

Dombi, Imreh, and Vincze (2007) 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, the algorithm does not return an answer.

In this paper, 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.

## Voting Algorithms

Instead of finding just one of the consistent LPMs, we propose a democratic approach that 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. However, enumerating the exponentially many models is impractical. Instead, 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.

## Variable Voting

Variable voting uses a generalization of the LPM representation. Instead of a total order on the variables, variable voting reasons with a *partial* order ( $\preceq$ ) to find the preferred object in a given pair. Among the variables that are different in the objects, the ones that have the smallest rank (the most salient) in the partial order vote choose the preferred object.

**Definition 1 (Variable Voting)** Suppose  $X$  is a set of variables and  $\preceq$  is a partial 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  $D$ , the set of variables that differ in  $A$  and  $B$ .
- Define  $D^*$ , the set of variables in  $D$  that have the smallest rank among  $D$  with respect to  $\preceq$ .
- Define  $N_A$  as the number of variables in  $D^*$  that favor  $A$  (i.e., that have value 1 in  $A$  and 0 in  $B$ ) and  $N_B$ , as 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.

**Example 2** Suppose  $\preceq$  is the partial 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 partial 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 partial orders that define equivalence classes on the set of variables. 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. If it is correct, then the ranks stay the same. Otherwise, the ranks of the variables that voted for the wrong object are incremented, thus reducing their importance<sup>2</sup>. Finally, the algorithm builds a partial order  $\preceq$ , where  $x \preceq y$  iff  $x$  has a lower rank than  $y$ . We next provide an example and some theoretical properties of variable voting.

**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 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, thus, the loop terminates. The partial order  $\preceq$  based on ranks of the variables is the same as the order given in Example 2.

<sup>2</sup>In our empirical results, we also update the ranks when the prediction was correct but not unanimous.

---

**Algorithm 2** *learnVariableRank*

---

**Require:** A set of  $X$  of variables, and a set  $O$  of observations  
**Ensure:** A partial order on  $X$ .  
1:  $\Pi(x) = 1, \forall x \in X$   
2: **while**  $\Pi$  can change **do**  
3:   **for** Every observation  $(A, B) \in O$  **do**  
4:     Let  $D$  be the variables that differ in  $A$  and  $B$   
5:      $D^* = \{x \in D \mid \forall y \in D, \Pi(x) \leq \Pi(y)\}$   
6:      $V_A$  is the set of variables in  $D^*$  that are 1 in  $A$ .  
7:      $V_B$  is the set of variables in  $D^*$  that are 1 in  $B$ .  
8:     **if**  $|V_B| \geq |V_A|$  **then**  
9:       **for**  $x \in V_B$  such that  $\Pi(x) < |X|$  **do**  
10:          $\Pi(x) = \Pi(x) + 1$ ;  
11:     Return partial order  $\preceq$  on  $X$  such that  $x \preceq y$  iff  $\Pi(x) < \Pi(y)$ .

---

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

Observations	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$
<i>Initially</i>	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

**Correctness:** Suppose  $\preceq$  is a partial order returned by *learnVariableRank*( $X, O$ ). It can be shown that any LPM  $\mathcal{L}$  such that  $\sqsubset_{\mathcal{L}}$  is a topological sort of  $\preceq$  is consistent with  $O$ . Furthermore, *learnVariableRank* never increments the ranks of the relevant variables beyond their actual rank in the target LPM. The ranks of the irrelevant variables can be incremented as far as the number of variables.

**Convergence:** *learnVariableRank* has a mistake-bound of  $O(n^2)$ , where  $n$  is the number of variables, 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  $O(n^2)$ . Thus, given enough observations, *learnVariableRank* will converge to a partial order  $\preceq$  such 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.

**Complexity:** A cursory inspection of the time complexity of the algorithm yields a loose upper bound of  $O(n^3m)$ , where  $n$  is the number of variables and  $m$  is the number of observations.

## 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

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

---

**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  $\{Y : Y \notin \text{rulePrefix} \mid \forall (A, B) \in O, A(Y) = 1 \text{ or } A(Y) = B(Y)\}$ .  
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*

---

where  $V_{(A>B)}^{\mathcal{L}}$  is 1 if  $A$  is preferred with respect to  $\mathcal{L}$  and 0 otherwise.  $V_{(B>A)}^{\mathcal{L}}$  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  $S$  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. It is generally not feasible to have all consistent LPMs—in practice, the sample has to be small enough to be feasible and large enough to be representative.

To satisfy these conflicting criteria, we introduce *aggregated LPMs*, which exploit the fact that many consistent LPMs share prefixes in the total order that they define. An aggregated LPM,  $(X_1, X_2, \dots, X_k, *)$ , represents a set of LPMs that define a total order with the prefix  $X_1 < X_2 < \dots < X_k$ . Intuitively, an aggregated LPM states that any possible completion of the prefix is consistent with the observations.

The algorithm *sampleModels* (Algorithm 3) uses 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) 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 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. If no observations remain, then we return the aggregated form of the prefix. 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. The 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 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 *sampleModels* is run, it randomly generates one of

the aggregated LPMs:  $(X_2, X_3, X_1, *)$ ,  $(X_3, X_1, *)$ , or  $(X_3, X_2, X_1, *)$ .

An aggregate LPM in a sample saves us from enumerating 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  $B$ . 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:

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

Intuitively,  $f_m$  counts every possible permutation with at most  $m$  items. Note that  $f_m$  can be computed efficiently and that  $f_n$  counts all possible LPM's for  $n$  variables.

Consider a pair of objects,  $A$  and  $B$ . We wish to determine how many extensions of an aggregate LPM  $\mathcal{L} = (X_1, X_2, \dots, X_k, *)$  would vote for one of the objects. We will call the variables  $X_1 \dots X_k$  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.

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  $|a|/(|a| + |b|)$ . The number of extensions of  $\mathcal{L}$  that will vote for  $A$  over  $B$  is denoted by  $N_{A>B}^{\mathcal{L}}$ :

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

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

**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.

Algorithm 4 (*modelVote*) takes a sample of consistent LPMs and a pair of objects as input, and predicts the preferred object using the weighted votes of the sampled LPMs. Returning to Example 5, the reader can verify that model voting will prefer  $A$  over  $B$ . Next, we will present our theoretical results for *sampleModels* and *modelVote*.

**Complexity:** The time complexity of *sampleModels* is bounded by  $O(n^2m)$ , where  $n$  is the number of variables

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

LPMs	$P(L S_1)$	$P(L S_2)$	$N_{A>B}^{\mathcal{L}}$	$N_{B>A}^{\mathcal{L}}$
$()$	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

#### 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  $\mathcal{L}$  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 an extension of  $\mathcal{L}$  referring only the prefix.
12:      winner is the object  $\mathcal{L}^*$  prefers among  $A$  and  $B$ 
13:       $Vote(winner) += F_{\mathcal{L}}/sampleSize$ .
14:    else
15:       $Vote(A) += N_{A>B}^{\mathcal{L}}/sampleSize$ .
16:       $Vote(B) += N_{B>A}^{\mathcal{L}}/sampleSize$ .
17: if  $Vote(A) = Vote(B)$  then
18:   Return a tie
19: else
20:   Return the object obj with the highest  $Vote(obj)$ .
```

and  $m$  is the number of observations. If we call *sampleModels*  $s$  times, then the total complexity of sampling is  $O(sn^2m)$ . For constant  $s$ , 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. 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 partial order *learnVariableRank* returns.

## Introducing Bias

In general, when there are not many training examples for a learning algorithm, the space of consistent LPMs is large. To overcome this problem, we can introduce bias (domain knowledge), indicating that certain solutions should be favored over the others. In this section, we propose a bias in the form 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.

**Definition 3 (Learning Bias)** A learning bias  $\mathcal{B}$  for learning a lexicographic preference model on a set of variables  $X$  is a total order on a partition of  $X$ .  $\mathcal{B}$  has the form  $E_1 < E_2 < \dots < E_k$ , where  $\cup_i E_i = X$ . Intuitively,  $\mathcal{B}$  defines a partial order ( $\preceq_{\mathcal{B}}$ ) on  $X$  such that for any two variables  $x \in E_i$  and  $y \in E_j$ ,  $x < y$  iff  $E_i < E_j$ .

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

Intuitively, an LPM that is consistent with a learning bias respects the variable orderings induced by the learning bias. The learning bias prunes the space of possible LPMs. The size of the partition determines the strength of the bias; for example, if there is a single variable per set, then the bias defines a specific LPM. In general, the number of LPMs consistent with learning bias ( $E_1 < E_2 < \dots < E_k$ ) can be computed with the following:

$$G([e_1, \dots, e_k,]) = f_{e_1} + e_1! \times (G([e_2, \dots, e_k]) - 1), \quad (4)$$

where  $e_i = |E_i|$  and the base case for the recursion is  $G([]) = 1$ . The first term 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  $e_1!$ .

We can generalize *learnVariableRank* to utilize the learning bias defined above by changing only the first line of *learnVariableRank*, which initializes the ranks of the variables. Given a bias of the form  $S_1 < \dots < S_k$ , the new 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.

The algorithm *modelVote* can also be generalized to use a learning bias  $\mathcal{B}$ . In the sample generation phase, we use *sampleModels* as presented earlier, and then eliminate all rules whose prefixes are not consistent with the bias. Note that even if the prefix of an aggregated LPM  $\mathcal{L}$  is consistent with a bias, 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  $N_{A < B}^{\mathcal{L}}$  (or  $N_{B < A}^{\mathcal{L}}$ ) with  $F_{\mathcal{L}}^{\mathcal{B}}$  and  $N_{A < B}^{\mathcal{L}, \mathcal{B}}$  (or  $N_{B < A}^{\mathcal{L}, \mathcal{B}}$ ), respectively, where:

- $F_{\mathcal{L}}^{\mathcal{B}}$  is the number of extensions of  $\mathcal{L}$  that are consistent with  $\mathcal{B}$ , and
- $N_{A < B}^{\mathcal{L}, \mathcal{B}}$  is the number of extensions of  $\mathcal{L}$  that are consistent with  $\mathcal{B}$  and prefer  $A$ . ( $N_{B < A}^{\mathcal{L}, \mathcal{B}}$  is similar.)

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

In counting the number of extensions of  $\mathcal{L}$  that are consistent with  $\mathcal{B}$  and prefer  $A$ , as in *modelVote*, we 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  $N_{A > B}^{\mathcal{L}, \mathcal{B}}$ , the number of extensions of  $\mathcal{L}$  that are consistent with  $\mathcal{B}$  and prefer  $A$ , can be computed as follows:

$$N_{A > B}^{\mathcal{L}, \mathcal{B}} = \frac{|a|}{|a| + |b|} \times (F_{\mathcal{L}}^{\mathcal{B}} - G([d_j \dots d_{k-1}, |w|])). \quad (5)$$

## Experiments

We empirically evaluated the described algorithms, using a metric of *prediction performance*,  $P$ , with respect to a set of test observations,  $T$ :

$$\text{performance}(P, T) = \frac{\text{Correct}(P, T) + 0.5 \times \text{Tie}(P, T)}{|T|} \quad (6)$$

where  $\text{Correct}(P, T)$  is the number of observations in  $T$  that are predicted correctly by  $P$  and  $\text{Tie}(P, T)$  is the number of observations in  $T$  that  $P$  predicted as a tie. We will use  $MV$ ,  $VV$ , and  $G$  to denote the model voting, variable voting, and the greedy approximations (respectively) of an LPM.

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  $G$  or  $MV$  with sampling. To mitigate this, we ran  $G$  200 times for every  $(O, T)$ , and  $MV$  10 times for each  $(O, T)$  pair. For  $MV$ , we called *sampleModels*  $S$  times per run.

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, we used  $S = 50$  and  $S = 200$ . Larger sample sizes (e.g. 800) improved performance slightly, but are omitted for space. For fixed values of  $R$  and  $I$ , an LPM  $\mathcal{L}$  is randomly generated. We randomly generated  $N_O$  and  $N_T$  pairs of objects, each with  $I + R$  variables. Finally, we labeled the preferred objects in accordance with  $\mathcal{L}$ .

Figure 1a shows the average performance of  $G$ ,  $MV$  with two different sample sizes and  $VV$  for  $R = 15$ ,  $I = 0$ , and  $N_T = 20$ , as  $N_O$  ranges from 2 to 20. Figure 1b 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 performances of  $VV$  and  $MV$  are better than the average performance of  $G$ , and the difference is significant at every data point. Also, the worst case performance of  $G$  after seeing two observations is around 0.3, suggesting a very poor approximation of the target.  $VV$  and  $MV$ 's worst case performances are much better than the worst case performance of  $G$ , justifying the additional complexity of these two algorithms. We have observed the same behavior for other values of  $R$  and  $I$ , and

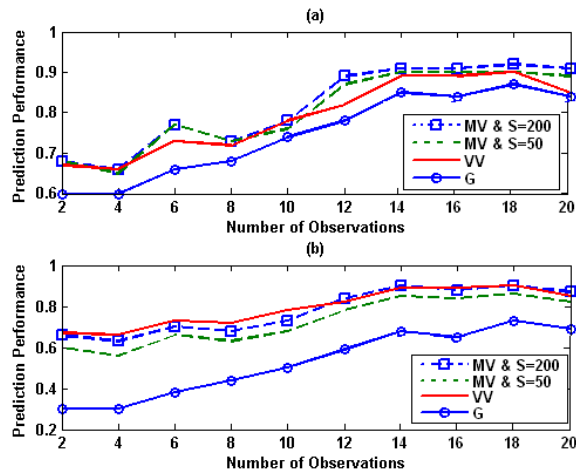


Figure 1: The average and worst prediction performance of the greedy algorithm, variable voting and model voting

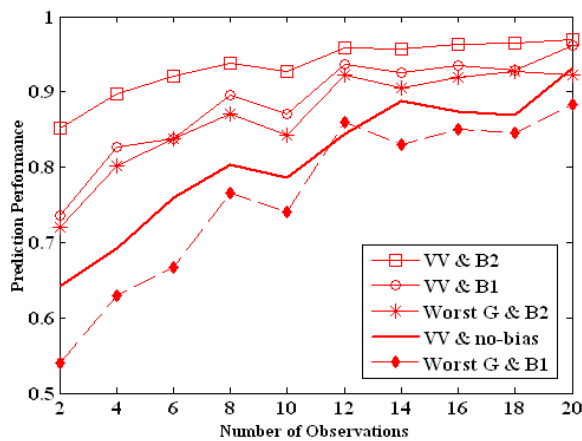


Figure 2: The effect of bias on VV and G.

have also witnessed a significant performance advantage for *MV* over *VV* in the presence of irrelevant variables when training data is scarce (omitted for space).

Figure 2 shows the positive effect of learning bias on the performance of voting algorithms for  $R = 10$ ,  $I = 0$ , and  $N_T = 20$ , as  $N_O$  ranges from 2 to 20. We have trivially generalized  $G$  to produce LPMs that are consistent with a given bias. The data points are averages over 20 different pairs of training and test sets  $(O, T)$ . We have arbitrarily picked two biases:  $B_1 : \{X_1, X_2, X_3, X_4, X_5\} < \{X_6, X_7, X_8, X_9, X_{10}\}$  and  $B_2 : \{X_1, X_2, X_3\} < \{X_4, X_5\} < \{X_6, X_7, X_8\} < \{X_9, X_{10}\}$ . The performance of *VV* improved greatly with the introduction of learning biases.  $B_2$  is a stronger bias than  $B_1$  and prunes the space of consistent LPMs more than  $B_1$ , resulting in a greater performance gain due to  $B_2$ . The differences between the bias curves and the non-bias curve are statistically significant, except at the last point of each. Note that the biases are particularly effective when the number of training observations is small. For both biases, the worst case performance of  $G$  is significantly lower than the performance of *VV* with the corresponding bias. We obtained very similar results with *MV* (omitted for space).

## Related Work

Lexicographic orders and other preference models have been utilized in several research areas, including multicriteria optimization (Bertsekas and Tsitsiklis 1997), linear programming, and game theory (Quesada 2003). The lexicographic model and its applications were surveyed by Fishburn (1974). The most relevant existing works for learning and/or approximating LPMs are by Schmitt and Martignon (2006) and Dombi, Imreh, and Vincze (2007), which were summarized earlier. In addition, the ranking problem as described by Cohen, Schapire, and Singer (1999) is similar to the problem of learning an LPM. However, that line of work poses learning as an optimization problem, finding the ranking that maximally agrees with the given preference function. Our work assumes noise-free data, for which an optimization approach is not needed. Another analogy (Schmitt and Martignon 2006), is between LPMs and decision lists (Rivest 1987). Specifically, it was shown that LPMs are a special case of 2-decision lists, and that the algorithms for learning these two classes of models are not directly applicable to each other.

## Conclusions and Future Work

We presented two democratic approximation methods, *variable voting* and *model voting*, for learning a lexicographic preference model (LPM). 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 learning bias for when the number of observations is small and incorporated this bias into the voting-based methods. In the future, we plan to generalize our algorithms to learn the preferred values of a variable as well as the total order on the variables. We also intend to develop democratic approximation techniques for other kinds of preference models.

## Acknowledgments

This work was supported by DARPA/USAF through BBN contract FA8650-06-C-7606.

## References

- Bertsekas, D., and Tsitsiklis, J. 1997. *Parallel and distributed computation: numerical methods*. Athena Scientific.
- Cohen, W.; Schapire, R.; and Singer, Y. 1999. Learning to order things. *Journal of Artificial Intelligence Research* 10:243–270.
- Dombi, J.; Imreh, C.; and Vincze, N. 2007. Learning lexicographic orders. *European Journal of Operational Research* 183(2):748–756.
- Fishburn, P. 1974. Lexicographic Orders, Utilities and Decision Rules: A Survey. *Management Science* 20(11):1442–1471.
- Quesada, A. 2003. Negative results in the theory of games with lexicographic utilities. *Economics Bulletin* 3(20):1–7.
- Rivest, R. 1987. Learning decision lists. *Machine Learning* 2(3):229–246.
- Schmitt, M., and Martignon, L. 2006. On the complexity of learning lexicographic strategies. *Journal of Machine Learning Research* 7:55–83.