Ordering Patterns by Combining Opinions from Multiple Sources

Pang-Ning Tan
Department of Computer Science and Engineering
Michigan State University
ptan@cse.msu.edu

Rong Jin
Department of Computer Science and Engineering
Michigan State University
rongjin@cse.msu.edu

ABSTRACT
Pattern ordering is an important task in data mining because the number of patterns extracted by standard data mining algorithms often exceeds our capacity to manually analyze them. A standard approach for handling this problem is to rank the patterns according to an evaluation metric and then presents only the highest ranked patterns to the users. This approach may not be trivial due to the wide variety of metrics available, some of which may lead to conflicting ranking results. In this paper, we present an effective approach to address the pattern ordering problem by combining the rank information gathered from multiple sources. Although rank aggregation techniques have been developed for applications such as meta-search engines, they are not directly applicable to pattern ordering for two reasons. First, the techniques are mostly supervised, i.e., they require a sufficient amount of labeled data. Second, the objects to be ranked are assumed to be independent and identically distributed (i.i.d), an assumption that seldom holds in pattern ordering. The method proposed in this paper is an adaptation of the original Hedge algorithm, modified to work in an unsupervised learning setting. Techniques for addressing the i.i.d. violation in pattern ordering are also presented. Experimental results demonstrate that our unsupervised Hedge algorithm outperforms many alternative techniques such as those based on weighted average ranking and singular value decomposition.

Categories and Subject Descriptors
H.2.8 [**Database Management**]: Database Applications – *Data Mining.*

Keywords
Pattern Ordering, Multi-criteria optimization, Ensemble Learning

1. INTRODUCTION
Learning to order data objects [5] is a pervasive problem that encompasses many diverse applications. For instance, a key problem in information retrieval is to rank documents based on their relevance to the user-specified queries. Another example is in recommender systems, where products such as movies and books are ordered in a way that reflects the interest of individual users. A common feature among these applications is that a large number of objects are available, but only a subset of them interesting to the users.

Recently researchers have begun to examine the problem of ordering data mining patterns [22]. Pattern ordering is an important problem because the number of patterns extracted by standard data mining algorithms often exceeds our capacity to manually analyze them. By ordering the patterns, we may reduce the amount of information to be digested by considering only the highest ranked patterns. Pattern ordering is also a critical component of association-based predictive modeling techniques such as CBA [18][17] and RBA [20]. Such techniques utilize the highest ranked patterns generated from association rule discovery algorithms to build a classification or regression model.

Despite its importance, pattern ordering is a challenging task due to the wide range of metrics and expert's opinions available for ranking patterns. As shown in [22], many existing metrics such as support, confidence, lift, correlation, χ^2, entropy, conviction, collective strength, etc., may produce conflicting ordering results when applied to the same set of patterns. In this paper, we present an effective approach to address the pattern ordering problem by combining the rank information obtained from disparate sources (e.g., different evaluation metrics or expert's opinions). The key assumption here is that given a collection of "reasonably consistent" rankings, we may learn an appropriate *ordering function* for patterns by taking a linear combination of the rankings obtained from each individual source. This idea is somewhat similar to the ensemble approach used for predictive modeling problems [7].

Combining rankings from multiple sources is also useful for distributed data mining applications such as spam detection and cyber-threat analysis, where the patterns of interest (e.g., rules describing signatures of a spam or cyber-attack) must be communicated and shared among a group of cooperating agents. Since the priority of each rule may be different, the agents must consolidate not only the rules obtained from other agents, but their rank information as well. The goal is to obtain a consistent global ordering of patterns based upon the individual rankings provided by each agent.

While advanced rank aggregation techniques have been developed in applications such as meta-search engines, they are not directly applicable to pattern ordering for two reasons:

1. *Current rank aggregation algorithms (such as the Hedge algorithm [5]) are mostly supervised*, i.e., they require a sufficient amount of labeled data. This may not be practical because obtaining the true (ideal) rankings of patterns is a very expensive task.

2. *Many patterns are correlated with each other*. For example, in association rule mining, it is common to extract patterns with similar substructures such as (ABC) → (XY), (ABC) → X, (AB) → X, etc. Therefore, one may no longer assume that the objects to be ranked are independent and identically distributed. Violation of the i.i.d. assumption affects the applicability of many existing rank aggregation algorithms.

The main contributions of our work are summarized below:

1. We introduce a methodology for solving the pattern ordering task by combining rank information from multiple sources.

2. We develop a novel, unsupervised version of the Hedge algorithm, which can effectively combine the ranking information from multiple sources. Our algorithm also reduces the learning complexity of the original Hedge algorithm from $O(n^2)$ to $O(n \log n)$, where n is the number of patterns.

3. We propose several strategies to address the issue of i.i.d. violation in pattern ordering.

4. We perform extensive experiments on a variety of data sets to illustrate the effectiveness and efficiency of our algorithm.

The remainder of this section is organized as follows. A formal definition of the pattern ordering problem is given in Section 2. We then describe our proposed methodology in Section 3. Section 4 presents our experimental evaluation and Section 5 concludes with a summary of our results along with directions for future work.

2. PROBLEM STATEMENT

This section presents a high-level discussion of the pattern ordering problem and illustrates how it can be formulated as an ensemble learning problem. A supervised algorithm for combining rank information from multiple sources is also presented.

2.1 Pattern Ordering Problem

Let $\mathbf{X} = \{x_1, x_2, .., x_n\}$ denote a collection of n patterns extracted from a given data set D. An *ordering function* (or *evaluation criterion*) f maps an input pattern x into a real number, i.e., $f(x): x \in \mathbf{X} \rightarrow \Re$. $f(x)$ may correspond to an evaluation metric (such as support, confidence, lift, and χ^2) or an expert's opinion about the quality of a pattern. Each pattern x is then assigned a rank value $r_f(x \mid \mathbf{X})$ depending on the magnitude of $f(x)$. The ranking vector produced by f is also known as its *rank list*.

Let $\Phi = \{f_1, f_1, \ldots, f_m\}$ denote a collection of ordering functions used to rank the patterns in $\mathbf{X}$ and $\{r_{f_1}(x \mid \mathbf{X}), \ldots, r_{f_m}(x \mid \mathbf{X})\}$ be their corresponding rank lists. For convenience, we may sometimes drop the reference to f when referring to the same ordering function – e.g., the notation $r_{f_i}(x \mid \mathbf{X})$ may be simplified as $r_i(x \mid \mathbf{X})$.

A pattern ordering problem is defined as the task of learning a ranking function that maps each pattern x into its true (ideal) rank within a given set $\mathbf{X}$ using the information gathered from the training set $\{r_f(x \mid \mathbf{X}) \mid \forall f \in \Phi\}$. Learning the explicit form of such function may not be a good idea because the rank of a pattern is defined relative to other patterns in $\mathbf{X}$. If the collection of patterns in $\mathbf{X}$ changes, then the rank value may change as well, and thus, requires the ranking function to be re-trained.

2.2 Pair-wise Preference Function

Alternatively, we may transform the original data into a collection of ordered pairs, $\{(x_p, x_q) \mid x_p \in \mathbf{X}, x_q \in \mathbf{X}\}$. Learning the relative ordering among three objects, x_1, x_2, x_3 is equivalent to finding the pair-wise ordering relation between (x_1, x_2), (x_2, x_3), and (x_1, x_3). The latter problem is more tractable because the function to be learnt depends only on the pair of input objects, rather than the entire collection of objects in $\mathbf{X}$.

A *pair-wise preference function,* $R(x, x')$, is defined as a mapping R: $\mathbf{X} \times \mathbf{X} \rightarrow \{0, 1\}$ [9]. The output of this relation is equal to 1 if x is preferred over x' and zero otherwise. The pair-wise preference function $R_i(x, x')$ can be induced from an ordering function $f_i(x)$ in the following way:

$$R_i(u,v) = \begin{cases} 1 & \text{if } f_i(u) > f_i(v) \\ 0 & \text{otherwise} \end{cases}$$

Example 1: *Let D={a,b,c,d} be a set of patterns extracted by a data mining algorithm. Suppose f(a)=0.71, f(b)=0.53, f(c)=0.22, and f(d) = 0.07, according to an evaluation metric f. Then, R(a,b) = R(a,c) = R(a,d) = R(b,c) = R(b,d) = R(c,d) = 1 and R(b,a) = R(c,a) = R(d,a) = R(c,b) = R(d,b) = R(d,c) = 0.*

Adding a new object into the collection does not affect the pair-wise ordering of other objects.

2.3 Learning a Multi-criteria Preference Function

In general, the ordered pairs induced by different ordering functions may not always agree with each other. The task of pattern ordering is to learn the *true preference function* $F(u,v)$ using a linear combination of the pair-wise preference functions defined for each ordering function:

$$\text{Pref}(u,v) = \sum_{i=1}^{m} w_i R_i(u,v) \qquad (1)$$

where the w_i's are the weights associated with each pair-wise preference function $R_i(x,x')$, subjected to the constraint $\sum_i w_i = 1$.

A naïve strategy is to consider the weighted average ranking approach, where the weights are assigned equally to each preference function, i.e., $w_i = 1/m$. Such weighting strategy assumes that the preference functions are equally reliable, which may not seem realistic in most practical applications. A better alternative is to learn the weights adaptively, assigning higher weights to preference functions that order many of the patterns correctly and lower weights to unreliable preference functions. A well-known online algorithm that uses such a strategy is the Hedge algorithm [5]. We briefly describe how the algorithm works in the next section.

2.4 Supervised Hedge Algorithm

The (supervised) Hedge algorithm was originally developed by Freund and Schapire [10] and applied in [5] to combine the rank information obtained from multiple sources in a supervised learning setting. The goal of this algorithm is to learn the appropriate weights $\{w_i\}_{i=1}^{m}$ such that the estimated preference function shown in Equation (1) best approximates the true preference function $F(u,v)$.

The difference between a preference function R from F is given by the following loss function:

$$Loss(R,F) = 1 - \frac{\sum_{(u \succ v)\in F} R(u,v)}{|F|} \qquad (2)$$

where $(u \succ v) \in F$ indicates that object u is preferred over object v according to the true preference function F, and $|F|$ denote the number of pairs ordered by F. If $R(x, x')$ is consistent with F, then the summation term is exactly the same as $|F|$, thus reducing the loss function to zero. If the ordered pairs produced by the preference function $R(x, x')$ is completely opposite of the ordering produced by F, then the loss function is equal to 1.

In order to compute the loss function, the Hedge algorithm requires that the true ordering for some pairs of patterns, $T^t \subseteq F$, must be known. (This is why the algorithm is only applicable in a supervised learning setting.) The algorithm uses this information to iteratively adjust the weights of the estimated preference function until the loss function is minimized.

Initially, the weights in Equation (1) are set to be equal, i.e., $w_i^1 = 1/m$ where the superscript "1" refers to the first iteration. During each subsequent iteration, the learning algorithm uses the labeled pairs $(u,v) \in T^t$ as a feedback [5] to modify the estimated preference function given in Equation (1). The loss function $Loss(R_i, T^t)$ is then computed for each preference function and the weights are updated using the following multiplicative rule:

$$w_i^{t+1} = \frac{w_i^t \beta^{Loss(R_i,F^t)}}{\sum_{i=1}^{m} w_i^t \beta^{Loss(R_i,F^t)}} \qquad (3)$$

Cohen et al. [5] have shown that the errors committed by the estimated preference function Pref(u,v) after T rounds of iteration are bounded by the following inequality:

$$\sum_{t=1}^{T} Loss(\text{Pref}^t, F^t) \le a_\beta \min_i \sum_{t=1}^{T} Loss(R_i, F^t) + c_\beta \ln m \qquad (4)$$

where $a_\beta = \ln(1/\beta)(1-\beta)$ and $c_\beta = 1/(1-\beta)$. The parameter β controls the tradeoff between generalization error and convergence rate. A smaller β leads to a slower convergence rate but with lower generalization error, whereas a larger β leads to faster convergence but higher generalization error.

3. Methodology

The supervised Hedge algorithm may not be suitable for pattern ordering since the true ordering for most patterns are often unknown. This section presents our proposed unsupervised Hedge algorithm for combining rank information from multiple sources.

3.1 Unsupervised Hedge Algorithm

To adaptively learn the weights of Equation (1), a feedback mechanism is needed to inform us whether the weights of our estimated preference function have been updated correctly. In a supervised learning setting, the feedback is provided by the pairs T^t whose true orderings are known in the training data. For unsupervised learning, we need to develop a methodology for simulating the feedback mechanism.

In this paper, we adopt the following heuristic to approximate the true preference function:

The ordering produced by the true preference function F is consistent with the ordering produced by most of the available ordering functions f_i's.

As a corollary, the above heuristic suggests that the difference between the rankings provided by the true preference function and the rankings provided by each of the individual ordering functions should be minimal. At each iterative step of our unsupervised Hedge algorithm, instead of comparing the estimated preference function *Pref* against the true preference function F for all labeled pairs (Equation 2), our strategy is to compare the rankings produced by *Pref* against the rankings produced by each f_i for all patterns. If the above heuristic holds, then our estimated preference function should approach the true preference function when the observed difference in rankings is minimal.

Table 1 presents a high-level description of our unsupervised Hedge algorithm. Initially, since there is no prior knowledge about the quality of each ordering function, we assign them equal weights. Hence, the initial estimated preference function *Pref* is equivalent to the weighted average preference function. We then compute the loss function by comparing the estimated preference function against the preference function R_i of each ordering function. The difference between the loss function shown in Table 1 and the formula given in Equation (2) is explained in Section 3.2. The loss function is used to update the weights using the multiplicative rule given in Equation (3). This formula reduces the weight of any preference function R_i whose rank list differs substantially from the rank list produced by the estimated preference function. After adjusting the weights, we re-compute the estimated preference function and repeat the procedure.

Our procedure for computing the estimated preference function and updating the weight combination is somewhat similar to the well-known Expectation-Maximization (EM) algorithm. The ideal preference function is treated as a hidden variable while the weights correspond to the model parameters. The procedure for computing the estimated ideal preference function corresponds to the E-step, in which the ideal preference function is estimated by combining multiple preference functions. The procedure for

Table 1: Pseudo-code for unsupervised Hedge algorithm.

```
Parameters:
    X: set of objects $\{x_i\}_{i=1}^{n}$;
    $\beta \in (0,1)$; Initial weights;
    $\{w_i^1\}_{i=1}^{m}$ with $w_i^1 = 1/m$, where $m$ is the number of ranking experts;
    T: number of iterations.
Do for t = 1, 2, …, T
  1. Compute the combined ordering function $f_{comb}^t = -\sum_{i=1}^{m} r_{f_i}(x \mid X) w_i^t$.
  2. Generate the ranking list $r_{f_{comb}}(x \mid X)$ for any object $x \in X$ in the descending order of their $f_{comb}^t$ value.
  3. Compute the loss function $Loss(f_i, f_{comb}^t)$ for every ordering function $f_i$ as
     $$Loss(f_i, f_{comb}^t) = \frac{\sum_{x \in X} |r_{f_i}(x \mid X) - r_{f_{comb}^t}(x \mid X)|}{\max_j \sum_{x \in X} |r_{f_j}(x \mid X) - r_{f_{comb}^t}(x \mid X)|}$$
  4. Set the new weights as
     $$w_i^{t+1} = \frac{w_i^t \beta^{Loss(f_i, f_{comb}^t)}}{\sum_{j=1}^{m} w_j^t \beta^{Loss(f_j, f_{comb}^t)}}$$
```

updating the weight combination corresponds to the M-step, in which the appropriate model parameters are updated using the loss function.

There are several assumptions made by our unsupervised Hedge algorithm. First, we assume that the rank list produced by each ordering function is "reasonably consistent" with the true preference function. Second, we assume that the errors committed by each ordering function are somewhat independent of each other. Under these assumptions, the ranking errors can be reduced by aggregating the results from multiple sources.

3.2 Implementation Issues

In order to efficiently implement the unsupervised Hedge algorithm, two important issues must be considered:

1. How to efficiently compute the loss function between a pair of preference functions? In the original Hedge algorithm (based on Equation (2)), we need to compare the ordering results for all pairs of patterns, a computation that requires O(n^2) complexity (where n is the number of patterns to be ordered).
2. How to efficiently induce a rank list from a given preference function? For example, given the pair-wise relations $R(a,e) = 1$, $R(c,b) = 1$, $R(c,a) = 1$, $R(a,d) = 0$, and $R(b,d) = 0$, is it possible to efficiently determine the total order among objects a, b, c, d, and e? As shown in [5], finding a rank list consistent with a collection of pair-wise preference functions is an NP-hard problem. Even with the simplified algorithms provided in [4], the computational complexity is still super-linear.

To resolve both issues, our algorithm departs from the approach used in the supervised Hedge algorithm. First, instead of learning the pair-wise preference function *Pref*(x, x') explicitly, we learn the combined ordering function $f_{comb}(x) = \sum_{i=1}^{m} w_i f_i(x)$ instead. The learning algorithm must find a combined ordering function f_{comb} whose rank list is most consistent with the rank lists $r_{f_i}(x \mid \mathbf{X})$ of all ordering functions f_i. We may induce a rank list from $f_{comb}(x)$ simply by sorting the patterns in ascending order of their $f_{comb}(x)$ values. This approach reduces the computational complexity from O(n^2) to O($n\log n$). Second, although the ordering function f_i can, in principle, take any real values, we represent the function in terms of the rank assigned to each pattern. More precisely, $f_i(x) = -r_{f_i}(x \mid \mathbf{X})$, where $r_{f_i}(x \mid X)$ denote the rank of pattern x within the data set $\mathbf{X}$. Third, computing the loss function using Equation (2) is very expensive because it requires pair-wise comparison between all pairs of patterns. We simplify this calculation by computing only the absolute difference between the rankings provided by two ordering functions on the same set of patterns:

$$Loss(f, f') \propto \sum_{x \in \mathbf{X}} \left| r_f(x \mid \mathbf{X}) - r_{f'}(x \mid \mathbf{X}) \right| \qquad (5)$$

where $r_f(x \mid \mathbf{X})$ denote the rank of pattern x among the set $\mathbf{X}$ by ordering function $f(x)$. The complexity of this simplified approach reduces from quadratic to linear in the number of patterns. Hence, the overall complexity of unsupervised Hedge algorithm is only O($n\log n$), which is lower than O(n^2).

3.3 Handling I.I.D. Violation

Another issue in pattern ordering is how to deal with correlated patterns. For example, some of the extracted patterns may be subsumed by other patterns that share similar features or have almost identical statistics. For instance, a rule (AB) $\rightarrow$ X may be subsumed by another rule (ABC) $\rightarrow$ X that have very similar support and confidence.

We present two techniques to recover the i.i.d. assumption so that existing methods such as the Hedge algorithm are still applicable:

1. Techniques for removing redundancy in patterns will be employed. For example, given an association rule, r: X $\rightarrow$ Y, where X and Y are itemsets [1], a related rule r': X' $\rightarrow$ Y' is considered to be redundant if X' $\subset$ X, Y' $\subset$ Y, and both r and r' have very similar support and confidence, i.e.,

$$|\text{support(r)} - \text{support(r')}| \le \varepsilon_1 \text{ and}$$
$$|\text{confidence(r)} - \text{confidence(r')}| \le \varepsilon_2$$

   where ε_1 and ε_2 are user-specified parameters. By removing the redundant rules, the remaining patterns have less dependency among each other.

2. Another approach is to cluster the patterns based on their characteristic features (e.g., attributes in the rules and their

corresponding contingency tables). For each cluster, we randomly select data points as representatives for the cluster [13]. (Ideally, we would like to have only one representative point for each cluster, but if the number of clusters is too small, it may be possible to select *m* independent points from each cluster). Standard clustering algorithms such as hierarchical and k-means clustering can be used in this approach.

Both of the methods described above may potentially reduce the degree of inter-dependencies among patterns. We will demonstrate later the effect of i.i.d. violation on our unsupervised Hedge algorithm in Section 4.4.

4. EXPERIMENTAL EVALUATION

We have conducted a series of experiments to evaluate the efficiency and effectiveness of the unsupervised Hedge algorithm using both synthetic data and data obtained from the UCI KDD Archive [23]. The details of the experiments and their corresponding results are presented next.

4.1 Experiment Setup

This section begins with a description of the datasets used in our experiments, followed by discussion on the metrics and baseline methods used for algorithm evaluation purposes.

4.1.1 Datasets

A key challenge encountered in our experiments is the lack of real-world data sets for which the true ordering of patterns are given. To address this issue, we need an *oracle* that produces the ideal rank list and a set of ordering functions that generates approximated versions of the list. In our experiments, we simulate the oracle in two ways: (i) using a synthetic data generator and (ii) using patterns and models extracted from real data sets. In the latter case, we assume a distributed data mining environment where each host has access only to its own local database. The oracle, on the other hand, has access to all databases. As a result, the local rank list produced by each individual host is somewhat different than the ideal rank list produced by the oracle.

The data sets used for our experiments must satisfy the following three criteria:

1) *Consistency criterion*. The rank list produced by each ordering function must be reasonably consistent with the ideal rank list of the oracle. Without this criterion, it is unlikely to find a combination of ordering functions that produces a better rank list than the individual criterion.

2) *Diverse ordering functions*. The ordering functions must be diverse enough in their opinions about the quality of different patterns. Similar to the arguments made by Breiman [4], by combining the diverse ranking results from multiple sources, we will be able to reduce the variance in ordering patterns and therefore improve the overall quality of the ranking results.

3) *Diverse quality criterion*. Diversity in the ordering functions does not necessarily translate to diversity in the ordering results (in terms of how close they reflect the rankings of the oracle). Ideally, some of the ordering functions should produce rankings that are quite close to the true ordering of the oracle while others produce rankings that are somewhat further from the ground truth. The diverse quality criterion ensures that the quality of rankings is diverse enough such that some ordering function commits more errors than others.

As previously mentioned, there are two methods used to generate the data sets for our experiment. The first approach, which we called the *random method*, uses a set of random numbers to represent the quality of patterns. These random numbers are generated from a Gaussian distribution with zero mean and variance equals to 1. There are altogether 1000 random numbers generated, each of which represents the quality of a pattern. The ideal ranking (for the oracle) is obtained by sorting the random numbers in descending order. To simulate an ordering function, the original random numbers are corrupted with Gaussian noise with mean equals to zero. The variance of the noise is varied to ensure diversity in the ensemble of ordering functions. A standard inverse Gamma distribution is used to sample the variance of the Gaussian noise. A rank list is then created by sorting the patterns in descending order of their corrupted values. Fifty such rank lists are created to represent fifty diverse ordering functions. Table 2 summarizes the details of this method.

Table 2: The random method used for generating the oracle and other ordering functions.

Random Method

1. Generate 1000 random numbers $\{\theta_1, \theta_2, ..., \theta_{1000}\}$ from a Gaussian distribution $N(0,1)$. The ideal rank list is created by sorting the random numbers in descending order.
2. Generate the rank lists for 50 ordering functions. Each rank list is generated as follows:
   a. Sample a variance σ from an inverse Gammar distribution with parameter '*a*' and '*b*' set to be 1.
   b. Generate 1000 noise values $\{\rho_1, \rho_2, ..., \rho_{1000}\}$ from Gaussian distribution $N(0,\sigma)$.
   c. Add the Gaussian noises to the original random numbers to create 'corrupted' random values, i.e., $\{\rho_1 + \theta_1, \rho_2 + \theta_2, ..., \rho_{1000} + \theta_{1000}\}$
   d. The rank list for an ordering function is obtained by sorting the 'corrupted' random numbers in descending order.

The second approach, called the *sampling method*, simulates a distributed data mining application. The oracle is assumed to have access to a global data set, which corresponds to one of the data sets obtained from the UCI KDD archive. Local databases are created by sampling with replacement from the global data set, where the size of each local database is determined from an inverse Gamma distribution. These local databases are used to simulate the rank lists of different ordering functions.

There are two steps in this approach: (i) pattern generation, and (ii) pattern evaluation. In the first step, data mining patterns such as association rules and decision trees are extracted either from the global database or from the local databases. If the patterns are extracted from the local databases, they must be consolidated before being evaluated by the oracle or the distributed agents. In the second step, the distributed agents will apply the extracted patterns or models to their local databases to evaluate their quality. We illustrate the sampling method for decision tree learning in Table 3.

Table 3: A sampling method for generating the oracle and other ordering functions.

Sampling Method

Let D be a data set from the UCI KDD archive.

- Create 200 different decision trees $\{M_1, M_2, ..., M_{200}\}$:
    1. Sample a number m from an inverse Gammar distribution with parameters $a = 1, b = 0.2 \times |D|$.
    2. Sample $\lfloor m \rfloor$ examples from data set D to form training set D_{train}.
    3. Construct a decision tree M over the sampled set D_{train}.
- Create the rank lists for 50 ordering functions:
    1. Sample a number m from an inverse Gammar distribution with $a = 1/4, b = 0.8 \times |D|$.
    2. Sample $\lfloor m \rfloor$ examples from dataset D to form the validation set D_{val}.
    3. Compute the accuracies $\{a_1, a_2, ..., a_{200}\}$ for the 200 decision trees over the validation set D_{val}.
    4. Generate the rank list of an ordering function by sorting the trees in descending order of their classification accuracies.
- The ideal rank list of the oracle is created by evaluating the 200 decision trees over the entire dataset D.

We begin with a set of 200 decision trees induced from training sets that are generated via sampling with replacement from the global database. To ensure diversity in the induced trees, the size of each training set is varied using an inverse Gamma distribution. We then select a metric such as classification accuracy to represent the quality of each tree. A validation set is needed to evaluate the decision trees. For the oracle, the validation set corresponds to the entire global data set. For ordering functions, the validation sets correspond to the local databases. Furthermore, to ensure diversity in the quality of ordering results, the size of the validation set is sampled from an inverse Gamma distribution. The larger the validation set, the more reliable is the ordering results. There are two UCI data sets used for this experiment – the 'spam' data set and 'German' data set.

4.1.2 Evaluation Metrics

To compare the similarity/difference between the rank list of the oracle and the rank list of the empirical preference function, two evaluation metrics are used. The first metric, absolute difference,

$$Diff(f_1, f_2) = \sum_{i=1}^{N} \left| r_{f_1}(x_i) - r_{f_2}(x_i) \right| \qquad (5)$$

computes the difference between the values of two rank lists. The smaller is the absolute difference, the closer are their rank lists. One potential drawback of this metric is that it treats each ranking position to be equally important. In many applications, users may be more interested in higher ranked patterns compared to lower ranked patterns.

A measure that compares the top-K results returned by two ordering functions is precision:

$$\text{Prec}(f_{comb}, f_{perf}, K) = \frac{\left| \left\{ x \mid r_{f_{comb}}(x \mid X) \le K \text{ and } r_{f_{perf}}(x \mid X) \le K \right\} \right|}{K} \qquad (6)$$

In our experiments, we reported the precision results at top-20 and top-50 rankings.

4.1.3 Baseline Methods

We compared the proposed algorithm against four baseline methods. Each baseline method produces an estimated rank list that will be compared against the rank list of the oracle:

1. *Mean method.* This method is identical to the weighted average ranking approach, where the rank list is estimated by averaging the rank lists of different sources.
2. *Median method.* In this method, the rank list is estimated by taking the median value of the rank lists provided by different sources.
3. *Best Expert method.* In this method, the ordering function that produces a rank list most similar (in terms of absolute difference) to the ideal rank list of the oracle is used.
4. *Singular Value Decomposition (SVD) method.* In this method, we decompose the original matrix that contains the rank lists for all sources. The first singular component is then chosen as the estimated preference function.

4.2 Effectiveness of the Unsupervised Hedge Algorithm

This section examines the effectiveness of the proposed algorithm using both synthetic and UCI simulated data sets. The results for the data set generated by the random method are shown in Table 4 and Figure 1. Figure 1 illustrates how the absolute difference for the unsupervised Hedge algorithm varies as the number of iterations increases, in comparison to other baseline methods. The Best Expert method, which selects the rank list with minimum absolute difference relative to the oracle, performs substantially worse than the other four methods. This observation was further confirmed by the results using the precision metric, as depicted in Table 4. The results suggest that combining rank information from multiple sources indeed help to improve the overall ranking results. Furthermore, the unsupervised Hedge algorithm performs substantially better than the four baseline models in terms of their absolute difference and precision at top-20 and top-50 rankings. The absolute difference also drops substantially after the first ten iterations. Nevertheless, after the 14th iteration, the absolute difference increases slightly, indicating the potential danger of overfitting.

Table 4: Precision of the dataset generated by random method.

	Unsupervised Hedge	Best Expert	Mean	Median	SVD
Prec@20	0.3	0.1	0.15	0.15	0.2
Prec@50	0.5	0.28	0.42	0.44	0.32

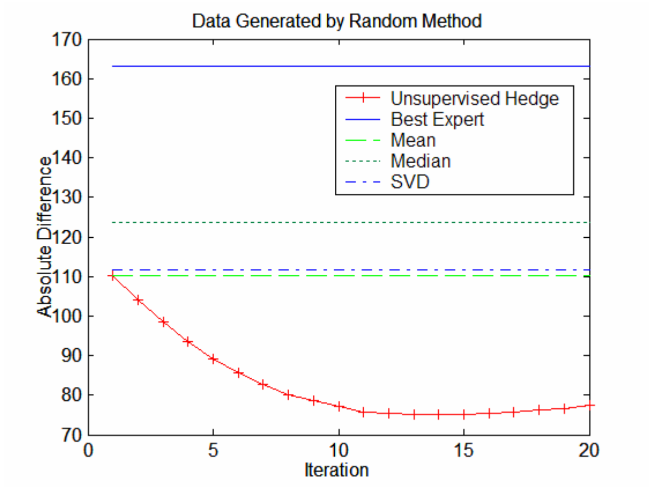

Figure 1: Absolute difference for the adapted Hedge algorithm and the four baseline algorithms.

Table 5: Precision results for the sampling method using the Spam data set.

	Unsupervised Hedge	Best Expert	Mean	Median	SVD
Prec@20	1	0.85	0.85	0.85	0.5
Prec@50	0.88	0.86	0.78	0.78	0.64

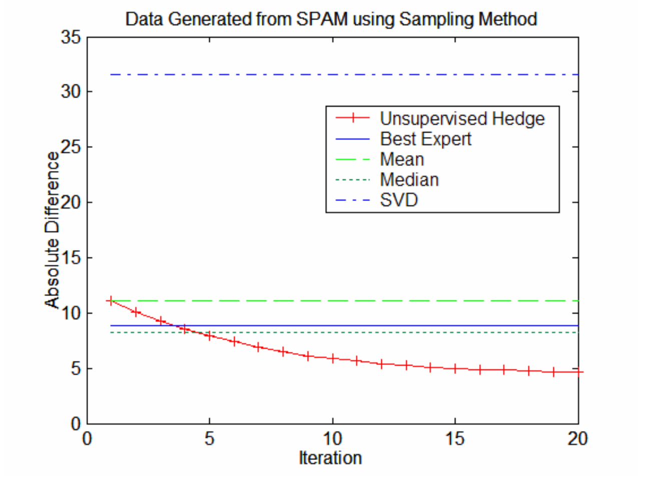

Figure 2: Absolute difference for the unsupervised Hedge algorithm and the four baseline algorithms.

Table 6: Precision results for the sampling method using the German data set.

	Unsupervised Hedge	Best Expert	Mean	Median	SVD
Prec@20	0.85	0.65	0.8	0.8	0.5
Prec@50	0.84	0.74	0.78	0.74	0.64

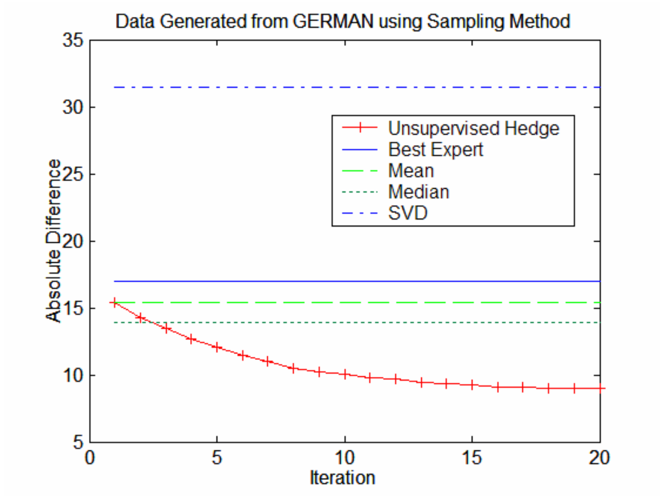

Figure 3: Absolute difference for the unsupervised Hedge algorithm and the four baseline algorithms.

The results for sampling method using the original 'spam' data set are presented in Table 5 and Figure 2. Unlike the previous data set where Best expert has the worst performance, in this case, SVD performs substantially worse than all other methods. The median method which has the second worst performance in the previous data set achieves the second best performance in the spam data set. Once again, the unsupervised Hedge algorithm outperforms the rest of the baseline methods.

Finally, the results for the sampling method using the 'German' UCI data set are presented in Table 6 and Figure 3. Much like the 'spam' data set, the SVD method performs substantially worse than the other four methods. Unsupervised Hedge algorithm again achieves the best results in terms of both precision and the absolute difference.

Overall, our proposed unsupervised Hedge algorithm outperforms all four baseline methods because it effectively combines the ordering results produced by multiple sources. Not only does this method reduce the difference in rankings, it also improves the precision of identifying the highest ranked patterns.

4.3 Efficiency of Unsupervised Hedge Algorithm

The synthetic data set generated by the random method is also used to evaluate the efficiency of the proposed algorithm, where efficiency is measured in terms of the total CPU time (in seconds) needed to learn the combined ordering function.

Figure 4 shows the scalability of the algorithm as the number of patterns is varied from 1000 to 128,000. The number of ordering functions used in this experiment is fixed at 50. Observe that the time complexity is almost linear with respect to the number of

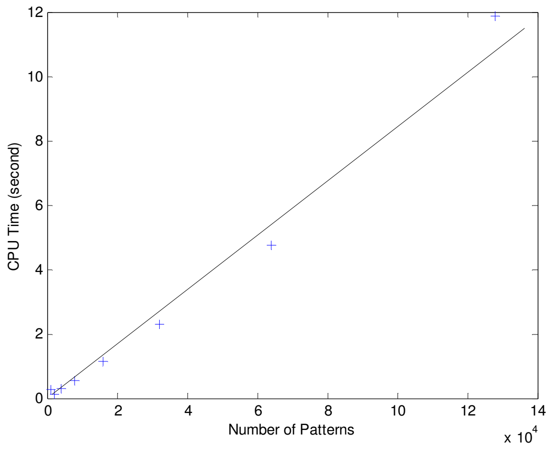

Figure 4: The CPU time for computing the proposed algorithm for different number of patterns. The number of experts is fixed to be 50.

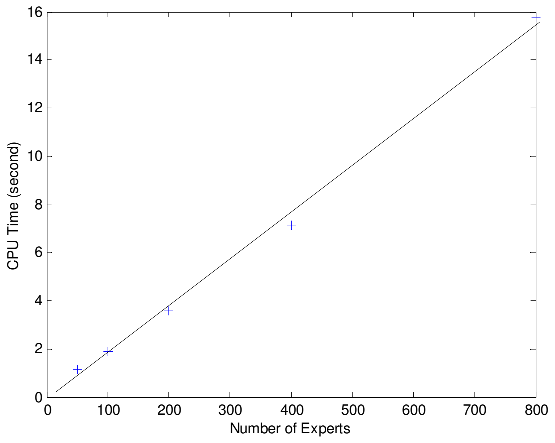

Figure 5: The CPU time for computing the proposed algorithm for different number of patterns. The number of patterns is fixed to be 16,000.

patterns, a result that is slightly better than our analysis in Section 3.2. Figure 5 shows the scalability of the algorithm as the number of ordering functions is increased from 50 to 800. The number of pattern used in this experiment is fixed at 16,000. Again, observe the linear relationship between CPU time and the number of ordering functions. The results from both of these experiments suggest that the proposed algorithm scales almost linearly with the number of patterns and the number of ordering functions.

4.4 Testing the Effect of I.I.D. Violation

In order to evaluate the impact of i.i.d assumption, we generate two sets of association patterns. The first set of patterns is pruned using the technique described in subsection 3.3 while the second set is generated without any pruning. Since the purpose of pruning is to reduce the amount of redundant patterns, we expect the pruned set to contain more independent patterns and to achieve better ranking results. Table 7 summarizes the absolute difference for the five algorithms over both pruned and unpruned patterns. First, similar to the previous experiments, unsupervised Hedge algorithm has smaller absolute difference compared to other baseline methods. Second, the absolute difference for the Best Expert method is unaffected by pruning because it does not perform any weighted aggregation on the rank lists. Finally, almost all algorithms that combine ranking results from multiple criteria (with the exception of the SVD method) improve their absolute difference values after pruning. For Unsupervised Hedge, pruning helps to reduce the absolute difference by almost 40%. Based on the above observation, we conclude that our proposed pruning method can effectively reduce the dependency among patterns and improves the performance of ensemble algorithms that combine ranking information from multiple sources.

Table 7: Absolute difference results for five algorithms in the case of pruning and not pruning patterns.

	Unsup. Hedge	Best Expert	Mean	Median	SVD
Without Pruning	123.00	246.11	154.54	136.42	575.90
With Pruning	78.05	246.11	91.87	106.23	618.39

5. CONCLUSIONS

This paper introduces the pattern ordering problem and presents an effective algorithm for addressing this problem by combining the rank information provided by disparate sources. Our algorithm is an adaptation of the original Hedge algorithm, modified to work in an unsupervised learning setting. Techniques for addressing the issue of i.i.d. violation are also presented. Our experimental results demonstrate that unsupervised Hedge algorithm outperforms many alternative techniques including those based on weighted averaged rankings and singular value decomposition.

Our experimental results also suggest the possibility of overfitting when the number of iterations is too large. This situation is analogous to the problem encountered by the AdaBoost algorithm when applied to noisy data [5]. For future work, we will consider using regularization methods to handle the overfitting problem.

6. ACKNOWLEDGMENTS

The work was partially supported by IRGP grant #71-4823 from the Michigan State University.

7. REFERENCES

[1] Agrawal, R., Imielinski, T., and Swami, A. Mining Associations between Sets of Items in Massive Databases. In *Proc. of the ACM-SIGMOD Int'l Conf on Management of Data*, 207-216, 1993.

[2] Agrawal, R. and Srikant, R. Mining Sequential Patterns. In *Proc. of the 11th Int'l Conf on Data Engineering*, 3-14, 1995.

[3] Bayardo, R.J., and Agrawal, R. Mining the Most Interesting Rules. In Proc of the 5th Int'l Conf on Knowledge Discovery and Data Mining, 145-154, 1999.

[4] Breiman, L. Bagging predictors. *Machine Learning*, 24(2):123-140, 1996.

[5] Cohen, W.W., Schapire, R.E., and Singer, Y. Learning to Order Things. *Adv in Neural Processing Systems* 10, 1997.

[6] Crammer, K. and Singer, Y. Pranking with Ranking. In *Proc of the 14th Annual Conference on Neural Information Processing Systems*, 2001.

[7] Dietterich, T.G. Ensemble Methods in Machine Learning. Multiple Classifier Systems, Cagliari, Italy, 2000.

[8] Ertoz, L., Lazarevic, A., Eilertson, E., Tan, P.N., Dokas, P., Kumar, V., and Srivastava, J. Protecting Against Cyber Threats in Networked Information Systems. In *Proc. of SPIE Annual Symposium on AeroSense, Battlespace Digitization and Network Centric Systems III*, 2003.

[9] Freund, Y., Iyer, R., Schapire, R. E., and Singer, Y. An Efficient Boosting Algorithm for Combining Preferences. In *Proc of 15th Int'l Conf on Machine Learning*, 170-178, 1998.

[10] Feund, Y. and Schapire, R. A Decision-Theoretic Generalization of on-line algorithm for combining preferences. Journal of Computer and System Sciences, 55(1), 119-139.

[11] Hand, D., Mannila, H., and Smyth, P. *Principles of Data Mining*. MIT Press, 2001.

[12] Herbrich, R., Graepel, T., and Obermayer, K. Large Margin Rank Boundaries for Ordinal Regression. In *Advances in Large Margin Classifiers*, pp. 115-132, 2000

[13] Jensen, D., Neville, J. And Hay, M.: Avoiding Bias when Aggregating Relational Data with Degree Disparity. In *Proc. of the Twenth Int'l Conference on Machine Learning*, 2003.

[14] Joachims, T. Optimizing Search Engine using Clickthrough Data. In *Proc. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2000.

[15] Lebanon, G. and Lafferty, J. Conditional Models on the Ranking Poset. *Advances in Neural Information Processing Systems (NIPS) 15*, 2003.

[16] Lee, W. and Stolfo, S. J. Data Mining Approaches for Intrusion Detection. In *Proc. of the 1998 USENIX Security Symposium*, 1998.

[17] Li, W., Han, J., and Pei, J. CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules. In *Proc of IEEE Int'l Conf on Data Mining*, 369-376, 2001.

[18] Liu, B., W. Hsu, and Y. Ma. Integrating Classification and Association Rule Mining. In *Proc of the 4th Int'l Conf on Knowledge Discovery and Data Mining*, 80-86, 1998.

[19] Nakhaeizadeh, G. and Schnabl, A. Development of Multi-Criteria Metrics for Evaluation of Data Mining Algorithms. In *Proc of the 3rd Int'l Conf on Knowledge Discovery and Data Mining*, 37-42, 1997.

[20] Ozgur, A. Tan, P.N., and Kumar, V. RBA: An Integrated Framework for Regression based on Association Rules. To appear in *Proc of the 4th SIAM Int'l Conf. on Data Mining*, April 2004.

[21] Shusha, A. and Levin, A. Ranking with Large Margin Principle: Two Approaches. In *Proc of Neural Information System Processing*, 2002.

[22] Tan, P.N., Kumar, V., and Srivastava, J. Selecting the Right Interestingness Measure for Association Patterns. In *Proc of the Eighth ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, 2002.

[23] UCI KDD Archive. http://kdd.ics.uci.edu/