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. In this paper, we present an effective approach to address the pattern ordering problem by combining the rank information gathered from disparate 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 [4] is a pervasive problem that encompasses many diverse applications; from mundane tasks such as predicting the top 3 winners of a talent competition to more complex applications such as product recommender systems. 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 [2][12]. Pattern ordering is an important task 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 key component of association-based predictive modeling techniques such as CBA [10][9] and RBA [11], where highly ranked association rules are used for building classification and regression models.

Despite its importance, pattern ordering is a challenging task due to the wide variety of metrics and expert's opinions available for ranking patterns. As shown in [12], many existing metrics may produce conflicting ordering results. In this paper, we present an effective approach to address the pattern ordering problem by combining the rank information obtained from disparate sources. The key assumption here is that given a collection of "reasonably consistent" rankings, we may learn a suitable *ordering function* for patterns by taking a linear combination of the rankings obtained from each individual source. Combining rankings from multiple sources is also useful for distributed data mining applications such as spam detection and cyber-threat analysis, where the objective is, to obtain a consistent global ordering of spam or intrusion detection rules based on the ordering provided by each detector.

While rank aggregation techniques have been developed for 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 [4]) 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 $(ABC) \rightarrow (XY)$, $(ABC) \rightarrow (X)$, $(AB) \rightarrow (X)$, etc. Violation of the i.i.d. assumption has an adverse effect on many 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 [4], which can effectively combine the ranking information from multiple sources. Our algorithm also

reduces the 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 demonstrate the effectiveness of our algorithm.

2. PROBLEM STATEMENT

This section illustrates how pattern ordering can be formulated as an ensemble learning problem.

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., $f(x): x \in \mathbf{X} \to \Re$. f may correspond to an evaluation metric or an expert's opinion about a pattern. Each pattern x_j is assigned a rank value $r_f(x \mid \mathbf{X})$ depending on the magnitude of $f(x_j)$. The vector of rankings produced by f is also known as its *rank list*.

A pattern ordering problem is defined as the task of learning a function that maps each pattern x_j into its actual (ideal) rank within a given set $\mathbf{X}$ using the information contained in the training set $\left\{r_f(x \mid \mathbf{X})\right\}$. 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 of each pattern x_j may change as well, and thus, requiring the ranking function to be re-trained.

2.2 Pair-wise Preference Function

Alternatively, we may transform the original data $r_f(x \mid \mathbf{X})$ into a collection of ordered pairs, (x_p, x_q). 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 mapping function depends only on the pair of input objects, rather than the entire collection of objects in $\mathbf{X}$. More precisely, a *pair-wise preference function* $R(x, x')$, is defined as a mapping R: $\mathbf{X} \times \mathbf{X} \to \{0, 1\}$ [6]. The output of this relation is equal to 1 if x is preferred over x' and zero otherwise. Adding a new object into the collection does not affect the pair-wise ordering of other objects.

2.3 Multi-criteria Preference Function

Let $\Phi = \{f_1, f_1, \ldots, f_m\}$ denote a collection of ordering functions. A pair-wise preference function $R_i\,(x, x')$ can be induced from each ordering function $f_i(x)$ in the following way:

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

In general, the pair-wise ordering relations induced by different f_i's do not always agree with each other. Our goal is to learn the *true preference function* $F(u,v)$ using a linear combination of the pair-wise preference functions:

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

where the w_i's are the weights associated with each pair-wise preference function $R_i(x, x')$. A naïve strategy is to consider the weighted average ranking approach, where the weights are assigned identically to each preference function. A better strategy is to learn the weights adaptively, assigning lower weights to preference functions that order many of the patterns incorrectly. A well-known online algorithm that uses such a strategy is the Hedge algorithm [4]. 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 [7] and applied in [4] 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 captured 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 will be exactly the same as $|F|$, thus reducing the loss function to zero.

In order to compute the loss function, the Hedge algorithm requires prior knowledge about the true ordering for some pairs of objects, $T^t \subseteq F$. (This is why the algorithm can only be used 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 pairs $(u,v) \in T^t$, whose true ordering are known, are used by the learning algorithm as a feedback [4] 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. [4] 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 patterns is often unknown. This section presents our proposed unsupervised Hedge algorithm for combining the rank information produced by 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 ordering function should be minimal. Hence, at each iterative step of our unsupervised Hedge algorithm, instead of comparing the estimated preference function against the true preference function for the known ordered pairs (Equation 2), we compare the rankings produced by the estimated preference function 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, our initial estimated preference function is equivalent to the weighted average ranking approach, i.e. R_{avg}. The estimated preference function is subsequently used to determine the loss function of each ordering function. The loss function shown in Table 1 looks slightly different than the definition given in Equation (2). We will explain their difference in Section 3.2. The loss function is then used to update the weights using the multiplicative rule given in Equation (3). This formula reduces the weight of any preference function whose rank list differs substantially from the rank list produced by the estimated preference function.

There are several assumptions made by our unsupervised Hedge algorithm. First, we assume that the rank list provided 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 combining the results from multiple sources.

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.
        $f_{comb}^t = -\sum_{i=1}^m r_{f_i}(x \mid \mathbf{X}) w_i^t$
  2. Generate the rank 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
        $Loss(f_i, f_{comb}^t) = \frac{\sum_{x \in X} |r_{f_i}(x \mid \mathbf{X}) - r_{f_{comb}^t}(x \mid \mathbf{X})|}{\max_j \sum_{x \in X} |r_{f_j}(x \mid \mathbf{X}) - r_{f_{comb}^t}(x \mid \mathbf{X})|}$
  4. Update the weights using the following equation:
        $w_i^{t+1} = \frac{w_i^t \beta^{Loss(f_i, f_{comb}^t)}}{\sum_{j=1}^m w_j^t \beta^{Loss(f_j, f_{comb}^t)}}$
```

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 approach 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? As shown in [4], 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 [3] 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'), we learn the ordering function $f(x)$ directly. More precisely, we will learn the weighted combination of ordering functions $f_{comb}(x) = \sum_{i=1}^m w_i f_i(x)$ that is most consistent with all the rank lists $r_{f_i}(x \mid \mathbf{X})$. A rank list can be induced from $f_{comb}(x)$ by sorting the patterns in ascending order of their $f_{comb}(x)$ values. This approach is much more efficient than inducing a rank list from pair-wise preference functions. It effectively reduces the computational complexity to O($n$log$n$). Second, although the ordering function can be 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 $r_{f_i}(x \mid \mathbf{X})$ denote the rank of pattern x within the data set **X**. Third, computing the loss function using Equation (2) is very expensive because it requires pair-wise comparison between various pairs of patterns. We simplify this calculation by computing the absolute difference between the rank lists 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 task is linear in the number of patterns. As a result, the overall complexity of unsupervised Hedge algorithm is only O($n$log$n$), which is considerably 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.,

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

   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 [8]. (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 break the dependency among the patterns. We will demonstrate later the effect of i.i.d. violation on our unsupervised Hedge algorithm in Section 4.2.

4. EXPERIMENTAL EVALUATION

We have conducted a series of experiments to evaluate the effectiveness of the unsupervised Hedge algorithm using both synthetic data and data obtained from the UCI KDD Archive. 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.

The first approach, which we called random method, uses a set of random numbers to represent the quality of various patterns. The true ordering for the oracle is obtained by sorting these random numbers. To simulate the ordering functions, the random numbers are corrupted by Gaussian noise. The variance of the noise is varied to ensure diversity in the ensemble of ordering functions.

The second approach, called sampling method, simulates a distributed data mining application. The global data is obtained from the UCI KDD archive. Local databases are simulated by sampling with replacement from the global data set, where the size of each local database is determined from an inverse Gamma distribution. Data mining patterns such as association rules and decision trees are subsequently extracted from these databases – the oracle ranks the patterns derived from the global data set while each distributed site ranks their local patterns. The details of this approach are omitted due to space limitation. Interested readers may refer to our technical report [13].

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 their absolute differences, the closer are their rank lists.

In some cases, we may only be interested in the similarity between the top-K highest rated 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.

Finally, we compared our 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.

4.1 Effectiveness of 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 2. Figure 2 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.

The results for sampling method using the original ‘spam’ data set are presented in Table 5 and Figure 3. 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 4. 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.

4.2 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 4.3 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.

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

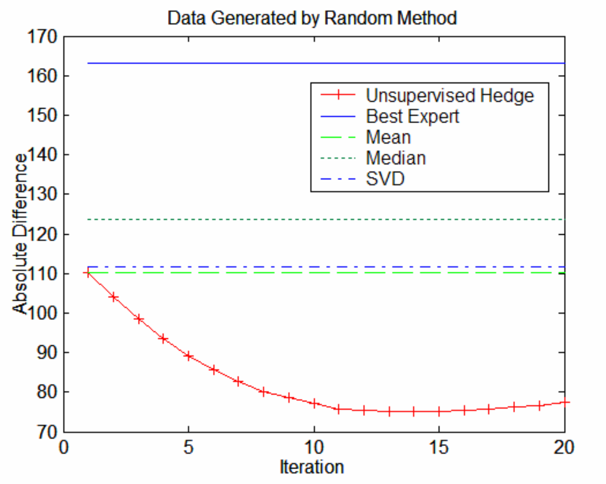

Figure 2: Absolute difference for the adapted Hedge algorithm and the four baseline algorithms.

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

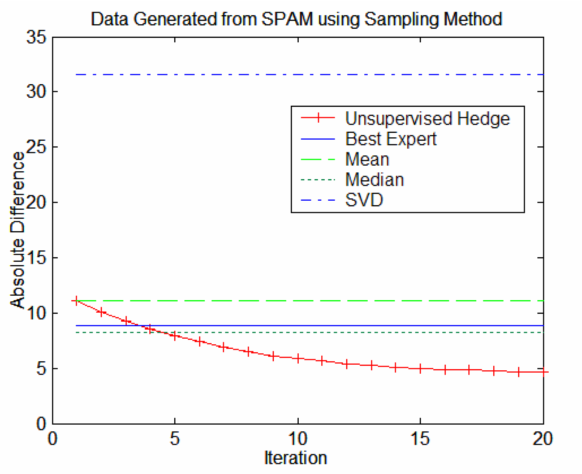

Figure 3: Absolute difference for the unsupervised Hedge algorithm and the four baseline algorithms.

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

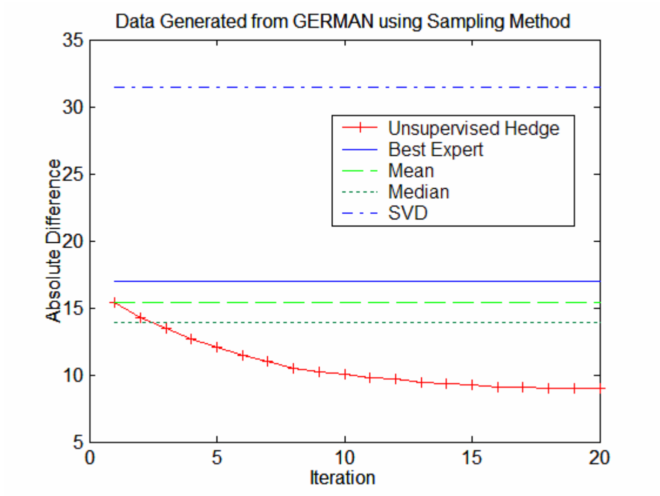

Figure 4: Absolute difference for the unsupervised Hedge algorithm and the four baseline algorithms.

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.

	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

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

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 average 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 ACM-SIGMOD*, 207-216, 1993.

[2] 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.

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

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

[5] Dietterich, T.G. Ensemble Methods in Machine Learning. In *Proc of Multiple Classifier Systems*, Cagliari, Italy, 2000.

[6] 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*, 1998.

[7] 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.

[8] Jensen, D., Neville, J. And Hay, M.: Avoiding Bias when Aggregating Relational Data with Degree Disparity. In *Proc. of the 20th Int'l Conf on Machine Learning*, 2003.

[9] 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.

[10] 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.

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

[12] 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.

[13] Tan, P.N., and Jin, R., Ordering Patterns by Combining Opinions from Multiple Sources, *Technical Report*, Michigan State University, 2004.