Efficient Feature Weighting Methods for Ranking

Hwanjo Yu and Jinoh Oh
Department of Computer Science and Engineering
POSTECH
Pohang, South Korea
{hwanjoyu,kurin}@postech.ac.kr

Wook-Shin Han†
Department of Computer Engineering
Kyungpook National University
Daegu, South Korea
wshan@knu.ac.kr

ABSTRACT

Feature weighting or selection is a crucial process to identify an important subset of features from a data set. Removing irrelevant or redundant features can improve the generalization performance of ranking functions in information retrieval. Due to fundamental differences between classification and ranking, feature weighting methods developed for classification cannot be readily applied to feature weighting for ranking. A state of the art feature selection method for ranking, called GAS, has been recently proposed, which exploits importance of each feature and similarity between every pair of features. However, GAS must compute the similarity scores of all pairs of features, thus it is not scalable for high-dimensional data and its performance degrades on nonlinear ranking functions. This paper proposes novel algorithms, RankWrapper and RankFilter, which is scalable for high-dimensional data and also performs reasonably well on nonlinear ranking functions. RankWrapper and RankFilter are designed based on the key idea of Relief algorithm. Relief is a feature selection algorithm for classification, which exploits the notions of hits (data points within the same class) and misses (data points from different classes) for classification. However, there is no such notion of hits or misses in ranking. The proposed algorithms instead utilize the ranking distances of nearest data points in order to identify the key features for ranking. Our extensive experiments show that RankWrapper and RankFilter generate higher accuracy overall than the GAS and traditional Relief algorithms adapted for ranking, and run substantially faster than the GAS on high dimensional data.

Categories and Subject Descriptors
H.m [Information Systems]: Miscellaneous

General Terms
Algorithms, Performance

†This work was supported by the Brain Korea 21 Project in 2009 and Mid-career Researcher Program through NRF grant funded by the MEST (No. KRF-2009-0080667).
.corresponding author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CIKM ’09, November 2–6, 2009, Hong Kong, China.
Copyright 2009 ACM 978-1-60558-512-3/09/11 ...$10.00.

Keywords
Feature Weighting, Rank Learning

1. INTRODUCTION

1.1 Background and Motivation

Feature weighting, or selection, is a crucial process for identifying an important subset of features from a data set [5, 10, 12]. The feature weighting methods for classification identify a subset of discriminant features by removing irrelevant features. Removing such features usually improves generalization performance of classification. Identifying key features also helps to interpret the classification model and results.

Feature selection methods for classification typically fall into three categories [10]. (1) Filter methods: Feature selection is done as a preprocessing step before learning. A filter method computes a score for each feature before learning, and thus runs independently of the learning machine [18, 24, 7]. (2) Wrapper methods: Inputs and outputs of a learning machine are used to select features. Thus, the feature selection quality is influenced by the accuracy of the learning machine. However, wrapper methods often generate better learning performance than filter methods, as they interact with the learning machine. Sensitivity analysis belongs to this category. (3) Embedded methods: Feature selection is performed within the process of learning. RFE (Recursive Feature Elimination), which is embedded in SVM [11], belongs to this category.

Learning ranking functions has recently gained much attention in data mining and machine learning communities and produced various applications for information retrieval [2, 4, 8, 13, 15, 19, 23, 25, 26]. Unlike classification functions, a ranking function \( F(\vec{x}) \) returns a ranking score of a data point \( \vec{x} \), such that \( F(\vec{x}_i) > F(\vec{x}_j) \) when \( \vec{x}_i \) is ranked higher than \( \vec{x}_j \). Such a global ranking function is learned from either relative orderings or multi-level relevance judgments. An example of relative ordering is \( \{(\vec{x}_2 \succ \vec{x}_3 \succ \vec{x}_1), (\vec{x}_5 \succ \vec{x}_3 \succ \vec{x}_4)\} \), where “A \succ B” means “A is preferred to B.” An example of multi-level relevance judgments is \( \{\text{(highly relevant : } \vec{x}_1, \vec{x}_5), (\text{partially relevant : } \vec{x}_2), (\text{not relevant : } \vec{x}_3, \vec{x}_5)\} \). Many rank learning methods have been developed to learn an accurate ranking function from such training data, and RankSVM is one of the earliest rank learning methods.

Feature weighting for ranking is an important task. However, it has not been researched extensively yet. Removing irrelevant features improves the generalization performance of ranking functions and helps to interpret the ranking function. For example, when learning preference functions from the user’s implicit feedback on Web pages [15, 21], feature selection can play crucial roles in identifying key features that explain the user’s preference.
1.2 Related Work

GAS: A filter-based feature selection method for ranking, called GAS, has been recently proposed [9]. The method, however, is not very scalable to the number of dimensions (or features). The complexity of the method is at least quadratic to the number of features, i.e., $O(n^2)$ where $n$ is the number of features, because the method needs to compute the Kendall’s $\tau$ for every pair of features. GAS also does not fully consider the nonlinear relationships among features.

RFE: Recursive Feature Elimination (RFE) can also be used for ranking if the ranking function is linear or represented by the sum of weighted features [11]. RFE simply eliminates those features whose weights are insignificant at each iteration. As long as the function is linear, the RFE strategy can be used for both classification and ranking functions. However, the RFE cannot be directly applied to nonlinear functions, as the weight of each individual feature is often hidden in nonlinear functions.

Relief: Relief algorithms have been extensively researched and are considered to be the most successful feature weighting methods for classification [3, 16, 20, 22]. Relief algorithms compute the importance of features based on the feature distance of nearest hits (data points within the same class) and nearest misses (data points from different classes). These algorithms are efficient for high-dimensional data (of linear complexity w.r.t. the number of features) and can correctly estimate the quality of features when there exist nonlinear relationships among the features. However, traditional Relief algorithms cannot be directly applied to ranking data sets, because in ranking, there are no notions of hit or miss.

1.3 Our Contributions

Based on the key idea of Relief, this paper proposes two efficient and effective feature weighting methods for ranking – RankWrapper and RankFilter.

- RankWrapper is a wrapper method running on relative orderings as the training data. When the training data is given as a set of relative orderings, the information about the absolute relevance levels is missing, and thus there are no notions of hit or miss of a data point. RankWrapper instead utilizes the information about ranking distances among data points instead of using absolute relevance levels of data points.

- RankFilter is a filter method running on multi-level relevance judgments as the training data. RankFilter extends the ReliefF algorithm (Relief for multiclass classification) by regarding each relevance level as a class in the multiclass classification while considering the distances among the relevance levels.

By designing our algorithms based on the key idea of the Relief algorithm, we are able to exploit the inherent advantages of it. Thus, RankWrapper and RankFilter are scalable to high dimensional data and can correctly estimate the quality of features when there exist nonlinear relationships among the features. Our extensive experiments show that RankWrapper and RankFilter generate higher accuracy overall than the GAS and Relief algorithms adapted for ranking, and run substantially faster than the GAS on high dimensional data.

1.4 Paper Organization

This paper is organized as follows. We first overview Relief algorithm for classification (Section 2) and present the RankWrapper (Section 3) and RankFilter methods (Section 4). We then present our empirical analysis of the methods on synthetic and real data sets (Section 5). We conclude our study in Section 6.

2. RELIEF OVERVIEW

The Relief family of algorithms identify discriminating features for classification [20]. A training set for classification is denoted as $D = \{(\vec{x}_i, y_i), \ldots, (\vec{x}_m, y_m)\}$, where $\vec{x}_i$ is a data point, and $y_i$ is the class label of $\vec{x}_i$.

The key idea of Relief is to iteratively estimate the feature weights according to their ability to discriminate between neighboring data points. In each iteration, for a random data point $\vec{x}$, two nearest neighbors of $\vec{x}$ are found, one from the same class (termed the nearest hit or NH) and the other from the different class (termed the nearest miss or NM). When two neighboring data points from the same class have different values of some attributes, the attributes separate the two data points, which is not desirable, thus the attributes are not discriminant. On the contrary, when two neighboring data points from different classes have different values of some attributes, those attributes separate the two data points, which is desirable, thus the attributes are discriminant.

Based on this intuition, Relief updates the weight of the $i^{th}$ feature by the following formula.

$$w_i = w_i + \text{dist}(\vec{x}_i, NM^{(i)}(\vec{x})) - \text{dist}(\vec{x}_i, NH^{(i)}(\vec{x}))$$ (1)

where $\vec{x}^{(i)}$ is the $i^{th}$ feature of a data point $\vec{x}$, and $NM^{(i)}(\vec{x})$ and $NH^{(i)}(\vec{x})$ are the $i^{th}$ features of the nearest miss and nearest hit of $\vec{x}$ respectively. All the features are updated based on the feature distance between $\vec{x}$ and its nearest miss and hit, and this process is repeated on some training points $\vec{x}$. After some iterations, the weight vector $\vec{w}$ will have high values on discriminant attributes only.

Algorithm 1 Relief

Input: $m$ training vectors of $n$ attributes and the class label for each vector

Output: the weight vector $\vec{w}$ (= $w_1 \ldots w_n$)

set all weights $\vec{w} := 0$;

for each data point $\vec{x}$ do

find nearest hit NH and nearest miss NM;

for $j = 1 \ldots n$ do

$w_j = w_j + \text{dist}(\vec{x}^{(j)}, NM^{(j)}(\vec{x})) - \text{dist}(\vec{x}^{(j)}, NH^{(j)}(\vec{x}))$;

end for

end for

Algorithm 1 shows the Relief algorithm [20, 17]. Relief finds the nearest hit and nearest miss of a point $\vec{x}$. It decreases the weights that are discriminant between the point $\vec{x}$ and the nearest hit, and increases the weights that are discriminant between $\vec{x}$ and the nearest miss.

Note that, even when there are nonlinear relationships among the features, Relief correctly identifies high quality features. For example, suppose features $\vec{x}^{(3)}$, $\vec{x}^{(4)}$, and $\vec{x}^{(5)}$ play crucial roles only when they appear together. Then, although their individual importance is low, Relief will eventually increase the weights for them (i.e., $w_3$, $w_4$, and $w_5$), because, if the three features are discriminant only together, there will be some NM having different values in the three features together, which will increase the weights of the features according to the weight update rule of Relief.
3. RankWrapper: Feature Weighting from Relative Ordering

This section presents RankWrapper algorithm which computes weights of features when the training data is given as a set of relative orderings. The Relief algorithm cannot be used for such data because the information about classes or absolute relevance levels is missing, and thus there are no notions of hit or miss of a data point in the training set. To compute feature weights from a set of relative orderings, RankWrapper utilizes the information about ranking distances among data points instead of the absolute relevance levels of data points.

RankWrapper, as a wrapper method, first uses a rank learning method (e.g., RankSVM) to generate a set of data points and their ranking scores

\[ R = \{(x_1, y_1), ..., (x_m, y_m)\} \]

where \( y_i \) is the ranking score of \( x_i \) such that \( y_i > y_j \) when \( x_i \succ x_j \).

**Property 1.** Let \( NP \) be the nearest points or the points close to \( x \) in the feature space. Then, among the \( NP \), the points relatively far from \( x \) in the ranking scores contain more disparate and crucial features for ranking than the points relatively close to \( x \) in the ranking scores.

**Rationale 1.** Since the \( NP \) are close to \( x \) in the feature space, their feature values are likely to be similar to each other. If some of the \( NP \) are relatively far from \( x \) in the ranking scores, they must have some features containing disparate values from those of \( x \), and those features must play crucial roles in making their ranking distant from \( x \).

**Property 2.** Among the nearest points \( NP \) in the feature space, if the points relatively close to \( x \) in the ranking scores contain disparate feature values from \( x \), those features are unimportant for ranking.

The rationale of Property 2 is straightforward.

Based on Properties 1 and 2, we introduce new notions of median ranking distance and rank significance. The median ranking distance of \( x \) is the ranking score distance between \( x \) and the median point in a sequence of the \( NP \) ordered by the ranking distance from \( x \). For example, consider the data points in a two-dimensional space in Figure 1 and a linear ranking function \( f \) where the ranking score of each data point is determined based on the projection onto \( f \). Among the five nearest points \( x_1, ..., x_5 \) that are close to \( f \) in the feature space, the points closest and farthest to \( f \) in ranking are \( x_3 \) and \( x_4 \) respectively. The median point is \( x_2 \) and the median ranking distance in this example is \( d_1 \) that is the ranking distance of \( x_2 \) from \( f \).

The rank significance, denoted as \( \text{rsignificance} \), is a function returning a positive or negative magnitude so as to increase or decrease the weights of corresponding features, depending on the ranking distance from \( x \) compared to the median ranking distance. Specifically, the feature weight is updated according to the feature difference between \( x \) and a nearest point \( np \in NP \) multiplied by the rank significance as follows.

\[
\text{rsignificance}(\text{medianRankDist}, \text{rankDist}(x, np)) \cdot \text{dist}(x, np) \cdot \text{sim}(x, np)
\]

Note that, the \( NH \) of the Relief corresponds to the data points relatively close to \( x \) in ranking, thus the rank significance returns a negative magnitude for the data points whose ranking distance from \( x \) is smaller than the median ranking distance in order to decrease the corresponding feature weights. Likewise, the rank significance returns a positive magnitude for the data points whose ranking distance from \( x \) is larger than the median ranking distance. Thus, the rank significance is defined as follows.

\[
\text{rsignificance} = \frac{\text{rankDist} - \text{medianRankDist}}{\text{medianRankDist}}
\]

To illustrate the effect of the rank significance function, consider the example of Figure 1. Points \( x_1 \) and \( x_4 \) are located farther from \( f \) than from the median point according to \( f \), and their attribute values are disparate from \( f \) mainly in dimension \( d_1 \). Thus, the rank significance returns a positive magnitude to increase the weight of \( d_1 \). On the other hand, points \( x_2 \) and \( x_5 \) are closer to \( f \) than to the median point according to \( f \), and their attribute values are different from \( f \) mainly in dimension \( d_2 \). Thus, the weight of \( d_2 \) will be decreased by the rank significance. In this example, dimension \( d_2 \) will be determined to be more important than \( d_2 \) for ranking function \( f \).

Next, the distance between \( x \) and \( np \) in the feature space must also be considered in the weight update because closer \( np \) have greater influence on the weight update than farther \( np \). It is because closer data points have smaller feature differences overall, and thus, the same feature difference in the closer points becomes more meaningful than in the farther points. Accordingly we design the following function “\( \text{sim} \)”, which returns a magnitude of the similarity (opposite to distance) between \( x \) and \( np \) in the feature space.

Note that similar functions were also proposed in [20].

\[
\text{sim}(x, np) = \frac{\text{sim}_\text{un}(x, np)}{\sum_{np' \in NP} \text{sim}_\text{un}(x, np')} \quad \text{and} \quad \text{sim}_\text{un}(x, np') = e^{-\text{dist}(x, np')/\delta^2}
\]

“\( \text{sim}_\text{un}(x, np') \)” returns an unnormalized similarity in which the influence of \( np' \) is exponentially decreased with the distance from \( x \). “\( \text{dist}(x, np') \)” is the distance between \( x \) and \( np' \) measured by the rank of \( np' \) in a sequence of data points ordered by the distance from \( x \). \( \delta \) is a user defined parameter controlling the influence of the distance. The reason for using the difference instead of actual distance is to assure that the nearest data point always has the same impact on the weights.

Therefore, the final weight update formula becomes

\[
w_i = w_i + \text{rsignificance}(\text{medianRankDist}, \text{rankDist}(x, np)) \cdot \text{dist}(x, np) \cdot \text{sim}(x, np)
\]

RankWrapper algorithm is described in Algorithm 2. RankWrapper first finds \( k \) nearest points of a data point \( x \) (line 4), sorts the nearest points according to their ranking distances from \( x \), and finds the ranking distance between \( x \) and the median point (“\( \text{medianRankDist} \)” in line 5). \( k \) is not a sensitive parameter but desirable to set to an odd value since the rank significance is computed based on the median point. \( (k \) is safely set to 5 in our experiments.) The median ranking distances are used inside the function \( \text{rsignificance} \) in line 8. For each nearest point \( np \), weights are updated according to the equation of line 8.

The complexity of the RankWrapper is the same as that of the Relief algorithm, which is at most \( O(nm^2) \) where \( n \) and \( m \) are the numbers of features and data points respectively. It is possible to reduce it further using the same techniques used for the Relief algorithm such as k-d trees [20]. Moreover, the outer loop does not have to run for every data point in the training set. Instead, it is still
Algorithm 2 RankWrapper

**Input:** a set of relative ordering

**Output:** the weight vector $\vec{w} (= < w_1 \ldots w_n >)$

1: Compute a set $R = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_m, y_m)\}$ from the training data using a rank learning method (e.g., RankSVM)
2: Set all weights $\vec{w} = 0$
3: for each data point $\vec{x} \ do$
4: Select $k$ nearest points $NP$ in the feature space;
5: Set medianRankDist = median ranking distance between $\vec{x}$ and $NP$ according to $R$
6: for each $\vec{n}$ do
7: $w_j = w_j + \text{significance}(\text{medianRankDist, rankDist}(\vec{x}, \vec{n})) \cdot \text{sim}(\vec{x}, \vec{n}) \cdot \text{dist}(x^{(i)}, n_i^{(j)})$;
8: end for
9: end for
10: end for
11: end for

effective with a constant number of iterations especially when the data size is large. The complexity then becomes $O(nm)$.

Note that our feature weighting task becomes similar to the feature weighting for regression since regression task also learns from a set of $(\vec{x}_i, y_i)$ pairs where $y_i$ is a real value. We will thus compare our RankWrapper with RRelief (Relief for regression [20]) as well as the GAS in our experiments in Section 5.

4. RANKFILTER: FEATURE WEIGHTING FROM MULTI-LEVEL RELEVANCE JUDGMENTS

RankFilter computes weights of features when the training data is given as multi-level relevance judgments. When ranking involves multi-level relevance judgments, the training data is similar to that of multiclass classification in the sense that each data point belongs to one of the given categories or relevance levels.

In ranking, however, there is a notion of distance among the categories while the multiclass classification does not have such notion. For example, the distance between “highly relevant” and “partially relevant” categories is closer than that between “highly relevant” and “not relevant” categories. Thus, when weights are updated based on the nearest misses in the Relief algorithm, the contribution of each miss must be weighted according to its ranking distance from the data point $\vec{x}$. Specifically, the misses from the relevance levels that are farther from the relevance level of $\vec{x}$ must have a greater influence than the misses from closer levels.

Thus, the RankFilter’s weight update rule becomes:

$$w_i = w_i + \frac{P(C_{NM})}{1 - P(C_x)} \cdot \text{diff}(C, \vec{x}, NM) \cdot \text{dist}(x^{(i)}, NM^{(i)}) \ldots (3)$$

The contribution of each miss is also weighted according to the prior probability $P(C_{NM})$ estimated from the training set. Since the $P(C_{NM})$ excludes the relevance level of hits, we divide it by the sum of probabilities for the misses’ relevance levels, i.e., $1 - P(C_x)$.

Algorithm 3 RankFilter

**Input:** $m$ training vectors of $n$ attributes and the relevance judgment for each vector

**Output:** the weight vector $\vec{w} (= < w_1 \ldots w_n >)$

1: set all weights $\vec{w} := 0$
2: for each data point $\vec{x} \ do$
3: find $k$ nearest hit $NH$
4: // let $C_x$ be the relevance level of $\vec{x}$
5: for each relevance level $C \neq C_x$ do
6: from relevance level $C$, find $k$ nearest misses $NM(\vec{x})$
7: end for
8: for $j = 1 \ldots n$ do
9: $w_j = w_j + \sum_{C \neq C_x} \frac{P(C)}{P(C_x)} \cdot \text{diff}(C, \vec{x}, NM) \cdot \sum_{l=1}^{k} \text{dist}(x^{(l)}, NM_{l}^{(i)}(\vec{x}))/k$
10: end for
11: end for
12: end for
The time complexity of the RankFilter is the same as that of the RankWrapper as well as the Relief algorithms, i.e., $O(nm)$ (discussed at the end of Section 3).

5. EXPERIMENT

We experimentally evaluate our methods on synthetic datasets (Section 5.2) and real-world datasets (Section 5.3), comparing with the GAS [9], ReliefF (a Relief-based method for multiclass classification), and RRelief (a Relief-based method for regression) [20].

5.1 Evaluation Measures

To evaluate the qualities of feature weighting methods, we select highly weighted features after performing each feature weighting method and compute the ranking accuracy using only the selected features.

We use NDCG and Kendall’s $\tau$ as evaluation measures. NDCG is used when the ranking labels are given as multi-level relevance judgments, and Kendall’s $\tau$ is used when there is a desirable global ordering of data $R^*$. NDCG is popularly used for IR applications where ranking on top results is more important than that on bottom results [4, 2, 19, 23], and Kendall’s $\tau$ is favorably used to measure the overall accuracy based on the number of correctly ranked pairs [15, 21, 25]. There are other measures for evaluating the ranking accuracy such as AUC (Area Under the Curve) and MAP (Mean Average Precision). They are used when there are two levels of ranking. Descriptions of NDCG and Kendall’s $\tau$ follow.

Kendall’s $\tau$: Let $R^*$ be the optimal ranking of data in which the data is ordered perfectly according to the user’s preference. A ranking function $F$ is evaluated by how closely its ordering $R^F$ approximates $R^*$. Kendall’s $\tau$ has been widely used for measuring similarity between two orderings $R^*$ and $R^F$ [15, 9]. For two strict orderings $R^*$ and $R^F$, Kendall’s $\tau$ is defined based on the number $Q$ of discordant pairs. If $R^*$ and $R^F$ agree in how they order a pair, $\vec{x}_i$ and $\vec{x}_j$, the pair is concordant, otherwise, it is discordant. For ordering $R^*$ and $R^F$ on a dataset $D$, we define the similarity function $\tau$ as the following, which is equivalent to that defined in [9]:

$$\tau(R^*, R^F) = \frac{1 - Q}{\binom{|D|}{2}}$$  \hspace{1cm} (5)

To illustrate, suppose $R^*$ and $R^F$ order five vectors $\vec{x}_1, \ldots, \vec{x}_5$ as follow:

$$\vec{x}_1 \succ_{R^*} \vec{x}_2 \succ_{R^*} \vec{x}_3 \succ_{R^*} \vec{x}_4 \succ_{R^*} \vec{x}_5$$  \hspace{1cm} (6)

$$\vec{x}_3 \succ_{R^F} \vec{x}_2 \succ_{R^F} \vec{x}_4 \succ_{R^F} \vec{x}_1 \succ_{R^F} \vec{x}_5$$  \hspace{1cm} (7)

In this example, $\tau(R^*, R^F)$ is computed as 0.7, as the number of discordant pairs is 3, i.e., \{$(\vec{x}_1, \vec{x}_2)$, $(\vec{x}_1, \vec{x}_3)$, $(\vec{x}_2, \vec{x}_3)$\} while all remaining 7 pairs are concordant.

Normalized Discount Cumulative Gain (NDCG): NDCG focuses on the accuracy on the top results rather than on the bottom results. For example, when users rated movies from 1 (meaning ‘terrible’) to 5 (meaning ‘excellent’), the learning machine learned a ranking function from the training data and returned top $n$ results. Then, NDCG score is computed as follows.

$$NDCG(n) = Z_n \sum_{j=1}^{n} 2^{R(j)} \frac{1}{\log(1+j)}$$  \hspace{1cm} (8)

where $j$ is the position from the top, $R(j)$ is the rating of the $j$’s movie. $Z_n$ is a normalization factor to guarantee that the NDCG score of a perfect ranking equals to 1. As $j$ increases or as the returned movie becomes farther from the top, its impact on the NDCG score decreases logarithmically.

5.2 Experiment on Synthetic Data

Data Generation: The following is the procedure of our synthetic data generation.

1. We randomly generated a data set $D$, where $D$ contains 100 data points of 50 dimensions (i.e., 100-by-50 matrix), and each element in the matrix is a random number between zero and one.

2. We randomly generated a global ranking function $F^*$, by randomly generating the weight vectors $\vec{w}$ in $F^*(\vec{x}) = \sum_{i=1}^{10} exp(-||\vec{w}_i - \vec{x}||)^2$. (Thus, the RankSVM with RBF kernels will be used for evaluation.)

3. We ranked the data points in $D$ according to $F^*$. We then added 50 dimensions to the data set such that $D$ becomes a 100-by-100 matrix, and filled up the added elements with random numbers. In other words, the added 50 dimensions are noise.

Let $R^*$ be the $D$ ranked according to $F^*$. $R^*$ was used for evaluating RankWrapper, RRelief, and GAS. We then equally divided $R^*$ into five levels and used the five levels of data for evaluating RankFilter, ReliefF, and GAS.

5.2.1 Accuracy Comparison

We evaluate the accuracy of each feature selection method by investigating how well each feature selection method maximizes the ranking performance. We used RankSVM with RBF kernels and also Committee Perceptron as the learning method. Committee Perceptron is a recently developed Ensemble-based rank learning method [6]. At each iteration, we run the following procedure.

1. We run each feature selection method on the synthetic data set and remove five features from $D$ that are weighted the lowest according to each method.

2. We train a ranking function from the rest of the features for each method using both the RankSVM and Committee Perceptron and compare the accuracies of the functions. We ran 3-fold cross validations to tune the SVM parameters (i.e., soft margin parameter $C$), and tried $C = 10^{-6}, 10^{-5}, \ldots, 10^0$ and reported the highest accuracy. The kernel parameter, gamma, is fixed to 1 since $F^*$ is generated with the parameters.

Let the number of perceptrons (weak learners) to 20 in the Committee Perceptron.

Ranking from Relative Ordering: Figure 2(a) and (b) reports the Kendall’s $\tau$ of RankWrapper, RRelief, and GAS at each iteration for Committee Perceptron and RankSVM respectively. The methods run as wrapper methods learning from partial orderings of $R^*$ (thus Kendall’s $\tau$ is used here). That is, $R^*$ is randomly divided into three groups (for 3-fold cross validation), two groups of orderings are used to select the highly weighted features using each method, and the other group is used to evaluate the quality of selected features by computing the Kendall’s $\tau$ of the rank learning methods on the selected features. The results are averaged over 30 runs. The highest accuracies are achieved by RankWrapper. The
GAS’s performance is lower than the others overall. It is because the data is generated using the nonlinear function with the RBF kernel, but the GAS only considers the similarity of every pair of features to capture the nonlinearity of features.

**Figure 2: Accuracy. X-axis: # of features; Y-axis: NDCG@10**

**Ranking from Multi-Level Relevance Judgments:** Figure 3(a) and (b) reports the NDCG@10 of the RankFilter, ReliefF, and GAS at each iteration for Committee Perceptron and RankSVM respectively. The methods run as filter methods learning from five levels of absolute relevance judgments (thus NDCG is used here). 3-fold cross validations are used and the results are averaged over 30 runs. The highest accuracies are also achieved by RankFilter. The GAS’s performance is lower than the others for the same reason.

**Figure 3: Accuracy. X-axis: # of features; Y-axis: NDCG@10**

**5.2.2 Scalability Comparison**

We compare the execution times of the filter methods on varying size of features and data points. These experiments are done on a dual quadcore machine with 2660MHz and 48G memory.

We first fixed the number of data points into 100, varied the number of features from 10 to 100, and examined the time for each method to compute the weights of the features from the divided $R^n$. Figure 4 shows the execution times of the three filter methods. As the number of features increases, the execution time of the GAS increases quadratically while the other filter methods increase linearly. As discussed in Section 1.2, the complexity of GAS is at least quadratic to the number of features because it needs to compute the Kendall’s $\tau$ for every pair of features.

Next, we fixed the number of features to 100 and varied the number of data points from 10 to 100. Figures 5(a) and (b) show the execution times of the filter methods in different scales. The execution times of the filter methods in different scales. The execution times of the filter methods increase much more rapidly than the other methods.

Note that the scalabilities of wrapper methods, i.e., RankWrapper and RRelief, will be highly dependent on the scalability of the learning method, because wrapper methods need to execute a rank learning method first before computing the weights. If RankSVM is used as the learning method, the scalabilities of RankWrapper and RRelief will be better than that of GAS w.r.t. the number of features but worse w.r.t. the number of data points, because the complexity of RankSVM is linear w.r.t. the number of features but $O(m^2)$ where $m$ is the number of data points. If scalable rank learning methods such as AdaRank [23] or Committee Perceptron [6] are used, the scalabilities of the wrapper methods will improve w.r.t. the number of data points.

**5.3 Experiment on Real Data**

In this section, we evaluate the accuracy of each method on the OHSUMED data set. The OHSUMED data set consists of 348,566 documents and 106 queries [14]. In total, there are 16,140 query-document pairs on which relevance judgments are made. The relevance judgments are either ‘d’ (definitely relevant), ‘p’ (partially relevant), or ‘n’ (not relevant). The data has been used in many experiments in IR [23].

**5.3.1 OHSUMED - LETOR**

We first use the OHSUMED data set from the LETOR site [1] (Section 5.3.1). Each data point (query-document pair) in the data set is represented by 45 features. Detailed information about the data set can be found from the site. We used the first 45 queries of documents to run a 3-fold cross validation, using 30 queries for learning and the other 15 queries for testing and repeating this three times.

We used two rank learning methods for this experiment – (1) RankSVM with RBF kernel and (2) Committee Perceptron [6] that is a recently developed Ensemble-based rank learning method. We evaluate the accuracy of the feature selection methods (1) from relative ordering and also (2) from multi-level relevance judgments.

**Ranking from Relative Ordering:** To evaluate the Kendall’s $\tau$ on the data set, we learned a ranking function $F^*$ using the RankSVM from the training set and constructed a global ranking $R^*$ for each query of documents. Given $R^*$, we evaluated RankWrapper, RRelief, and GAS using the 3-fold cross validation: at each iteration,
(1) from the training set, we removed three features of the lowest weights using each feature selection method, (2) learned ranking functions $F'$ using RankSVM and Committee Perceptron from the training set, and (3) tested the accuracy of $F'$ on the testing set. This process was repeated until we consumed all the features. The results were averaged over three runs by the 3-fold cross validation.

Figure 6 shows the Kendall’s $\tau$ of the six methods at each number of features. The accuracies of the methods fluctuate a little, but the RankWrapper with Committee Perceptron shows the highest accuracy overall.

**Ranking from Multi-Level Relevance Judgments**: Here we evaluate RankFilter, ReliefF, and GAS using NDCG@10. We directly used the three levels of relevance given in the LETOR as the ranking labels. We used the same 3-fold cross validation method – 30 queries for training and the other 15 queries for testing – and also ran RankSVM and Committee Perceptron: Using each feature selection method, at each iteration we eliminated the lowest weighted three features and estimated NDCG@10 of the RankSVM and Committee Perceptron functions using the 3-fold cross validation.

While RankSVM and Committee Perceptron show comparable accuracies for the relative ordering, RankSVM showed much higher accuracy than Committee Perceptron in this experiment. Thus, we show the accuracies of the two learning methods in different graphs. Figures 7(a) and (b) show the results of the three methods with Committee Perceptron and RankSVM respectively.

The highest accuracies were achieved by RankFilter in both graphs. The accuracy differences between RankFilter and ReliefF were, however, not as significant as those in the experiments on synthetic dimensional data. RankWrapper and RRelief are only plausible solutions for nonlinear feature selection from relative ordering on high dimensional data. RankWrapper and ReliefF are on the testing set.

We used the first twelve queries (i.e., query id = 1 to 12) in this experiment excluding a few queries containing too few documents (i.e., query id = 8 and 10). Table 1 shows the number of documents and extracted features for each query.

<table>
<thead>
<tr>
<th>qid</th>
<th># documents</th>
<th># features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>134</td>
<td>2625</td>
</tr>
<tr>
<td>2</td>
<td>128</td>
<td>3096</td>
</tr>
<tr>
<td>3</td>
<td>182</td>
<td>3227</td>
</tr>
<tr>
<td>4</td>
<td>111</td>
<td>1910</td>
</tr>
<tr>
<td>5</td>
<td>56</td>
<td>1403</td>
</tr>
<tr>
<td>6</td>
<td>153</td>
<td>2354</td>
</tr>
<tr>
<td>7</td>
<td>54</td>
<td>1267</td>
</tr>
<tr>
<td>8</td>
<td>42</td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>140</td>
<td>2374</td>
</tr>
<tr>
<td>10</td>
<td>35</td>
<td>x</td>
</tr>
<tr>
<td>11</td>
<td>96</td>
<td>1830</td>
</tr>
<tr>
<td>12</td>
<td>84</td>
<td>1226</td>
</tr>
</tbody>
</table>

Table 1: The number of documents and extracted features for each query.

For each query of documents, we learned a ranking function $F^*$ using the RankSVM and constructed a global ranking $R^*$. The wrapper methods are run and evaluated based on the $R^*$. Assume the $R^*$ is a $m$-by-$n$ matrix, i.e., $m$ data points of $n$-dimensions, and the $m$ data points are ordered by the hidden function $F^*$. Using each feature selection method, we removed a half of lowly weighted dimensions (or features) at each iteration and repeated this until we have one dimension left. Let $R^*_{(i)}$ be a $m$-by-$n_{(i)}$ matrix after removing lowly weighted dimensions at $i^{th}$ iteration, such that $n_{(i)} = n_{(i-1)}/2$. The quality of $R^*_{(i)}$ or the quality of each feature selection method is evaluated by learning a RankSVM function on $R^*_{(i)}$ and evaluating the accuracy of the function.
For evaluation, we run 3-fold cross validations. At each iteration, we tuned the soft margin parameter $C$ and the kernel parameters, i.e., degree $p$ for polynomial and $\gamma$ for RBF, by trying $C = 10^{-6}, 10^{-5}, ..., 10^6, p = 1, 2, ..., 5$, and $\gamma = 10^{-6}, 10^{-5}, ..., 10^6$, and reported the highest accuracy. We experimented with polynomial and RBF kernels, but since the linear kernel is equivalent to the polynomial kernel with degree $p = 1$, we ended up trying linear, polynomial, and RBF kernels in this experiment.

Figure 8 shows the feature selection results. The highest performances are achieved by RankWrapper for most queries except for qid = 3 and 9 which show the highest performances by RRelief and RBF kernels. Note that the X-axis is logarithmically scaled, thus in most cases the highest performances are achieved when a substantial amount of features are removed. For example, the highest performances for qid = 1 and 2 are achieved when a half of the features are removed (by RankWrapper and RBF kernel for qid = 1, and by RankWrapper and polynomial kernel for qid = 2). For qid = 5, the highest performance is achieved by RankWrapper and polynomial kernel when 87.5% of features are removed.

6. CONCLUSIONS

This paper proposes efficient and effective feature weighting methods for ranking, i.e., RankWrapper and RankFilter. The methods compute the feature weights that maximize the ranking performance. RankWrapper, as a wrapper method, computes the weights from a set of relative orderings; it uses a rank learning method to precompute the ranking scores of data points and uses the scores for computing the feature weights. RankFilter, as a filter method, computes the weights from multi-level relevance judgments independently of rank learning methods. Based on the similar idea of Relief for classification, RankWrapper and RankFilter update the feature weights based on the ranking distance of points that are close to each other in the feature space. When two close points have a large ranking distance, the disparate features of the two points are important for discriminating the ranking of the points. On the other hand, when two close points have a small ranking distance, the disparate features of the points are unimportant for ranking. Based on this intuition, RankWrapper and RankFilter efficiently and effectively identify key features for ranking even when there exist nonlinear relationships among the features. We experimentally confirm that our proposing methods generate higher accuracy overall than the state-of-the-art feature weighting methods and are more scalable to high dimensional data.

7. REFERENCES

Figure 8: RankWrapper and RRelief on the OHSUMED documents. X-axis: # of features; Y-axis: Kendall’s τ