A Learning Approach to SQL Query Results Ranking Using Skyline and Users' Current Navigational Behavior

Zhiyuan Chen, Tao Li, and Yanan Sun

Abstract—Users often find that their queries against a database return too many answers, many of them irrelevant. A common solution is to rank the query results. The effectiveness of a ranking function depends on how well it captures users' preferences. However, database systems often do not have the complete information about users' preferences and users' preferences are often heterogeneous (i.e., some preferences are static and common to all users while some are dynamic and diverse). Existing solutions do not address these two issues. In this paper, we propose a novel approach to address these shortcomings: 1) it addresses the heterogeneous issue by using skyline to capture users' static and common preferences and using users' current navigational behavior to capture users' dynamic and diverse preferences; 2) it addresses the incompleteness issue by using machine learning technique to learn a ranking function based on training examples constructed from the above two types of information. Experimental results demonstrate the benefits of our approach.

Index Terms—Data and knowledge visualization, interactive data exploration and discovery.

1 Introduction

Users often ask SQL queries through a form-based interface. However, they often cannot form a query that returns exactly the answers matching their preferences. As a result, their queries often return too many answers, many of them irrelevant. A common solution is to rank query results [3], [11], [2]. The effectiveness of a ranking function depends on how well it captures users' preferences.

- *Z. Chen and Y. Sun are with University of Maryland Baltimore County, Baltimore, Maryland, USA. E-mail: {zhchen,sun8}@umbc.edu*
- *T. Li is with Florida International University, Miami, FL, USA. Email: taoli@cs.fiu.edu*

Heterogeneity and incompleteness problem: However, there are two common challenges to capture users' preferences: 1) users' preferences are often heterogeneous in nature, 2) the database system often only has incomplete information of users preferences. Some preferences are static and common to all users. For example, if everything else is equal, consumers always prefer products with lower prices. On the other hand, some preferences are dynamic and diverse. E.g., customers who prefer luxury cars are often less sensitive to prices than those who prefer economic ones. Even the same customer's preferences may change over time. E.g., customers who bought a small car when he was single might buy a bigger car once he got married. Due the complexity of user preferences, it is unrealistic that the system will learn a user's preference completely. For example, the system can ask a user to specify whether he prefers car A over car B, but this information does not say whether car A is better than car C, especially if car C and car B are substantially different.

Existing solutions have used query history [3], [11] or partial orders between records [2] to capture users' preferences. However, these solutions do not address the incompleteness issue and the heterogeneous issue of users' preferences. For example, a new user may not have any query history. Query history only captures users' past behavior and may not reflect users' *current* preferences.

Overview of our approach: In this paper, we propose a novel learning approach to rank SQL query results. This approach has two benefits: 1) it addresses the heterogeneous issue by using skyline to capture users' static and common preferences and using users' current navigational behavior to capture users' dynamic and diverse preferences; 2)

Digital Object Indentifier 10.1109/TKDE.2012.128

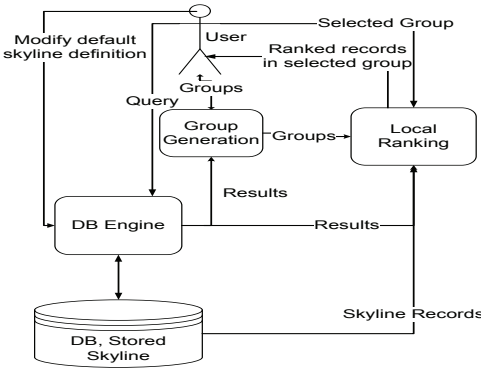


Fig. 1. Architecture of our system.

it addresses the incompleteness issue by using machine learning technique to learn a ranking function based on training examples constructed from the above two types of information. Commonly used ranking functions have many parameters and machine learning techniques allow us to automatically tune these parameters according to diverse users' preferences [22]. In this paper, we use machine learning techniques to tune the importance (or weight) of each attribute in ranking. This machine learning technique also will not overfit the training examples so it works well for incomplete preference information.

Fig. 1 shows an overview of our approach. The system stores a default skyline operator [6]. For a numerical attribute, the skyline operator indicates whether it should be maximized (e.g., the number of features of a product) or minimized (e.g., price). For a categorical attribute, the operator specifies some desired value (e.g., the color should be red). The default skyline captures users' common and static preferences (e.g., price should be low). Users can also modify the default skyline definition (e.g., define his favorite color). The system will then pre-compute the skyline records [6], which are not dominated by other records. These records will be later used in the ranking process.

Our system also uses users' current navigational behavior, which captures users' more dynamic and diverse preferences. The navigational behavior is collected as follows. Users can submit form-based SQL queries in our system. Once the database engine executes the query and returns the results, the system provides users with several options to navigate the results. These navigation techniques divide results into smaller groups. E.g., users can use the clustering technique proposed in [21], which will return a number of clusters along with the representative records for each cluster. They can use automatic categorization techniques [8], [12] which will organize the results into a decision tree like structure. They can also use group-by operator by specifying the group by columns.

Each group will be associated with an informative label making it easy for users to decide which group is of interest. For example, if groups are generated by group-by, the label contains the values of the group-by columns. If groups are generated using clustering, the label is the representative record in each cluster. If groups are generated using automatic categorization, the label is the condition associated with the category. Each label also includes the number of rows included in the group.

Users can then take one of following three actions: 1) click and rank, i.e., to select some groups based on their labels and look at records in those groups (which will be automatically ranked); 2) click and expand, i.e., to select a group and further divide it into smaller groups using one of the navigational techniques; 3) select some rows of interest in a clicked group. Existing information retrieval techniques have used the third type of action as user feedback but not the first two types. Further, there are usually very few groups so it is much easier for users to select a group rather than records (which may require sifting through hundreds of records rather than examining a few labels associated with groups). Thus we focus on the feedback generated by the first two types of actions. In both types of actions, users will select some groups of interest. We will use this information as implicit user feedback when ranking results in a group.

E.g., suppose users are searching a used car database. They can post query conditions on various attributes of used cars. The results will then be grouped by models. They can then select a car model, and the system will rank cars of that model. Users can then select some records in the selected model or divide that model into smaller subgroups (e.g., by trims levels).

The main challenge is how to automatically rank results in selected groups. Ranking is necessary because a group may still contain hundreds or more records, making it difficult for users to sift through. Our approach uses users' navigational behavior (i.e., group selection) along with skyline records to learn a ranking function for each group that the user has selected. The skyline records in a selected group g become positive training examples and all other records (including those in the selected group

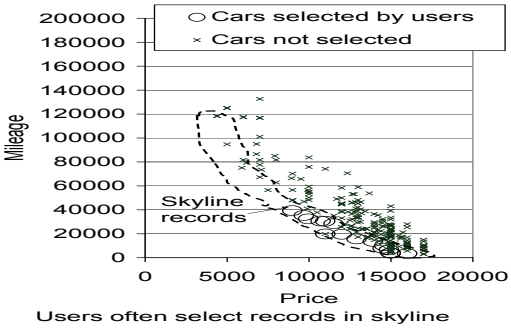


Fig. 2. The group of Japanese cars

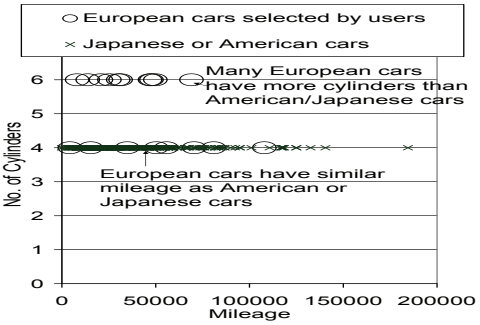


Fig. 3. European cars vs. American/Japanese cars

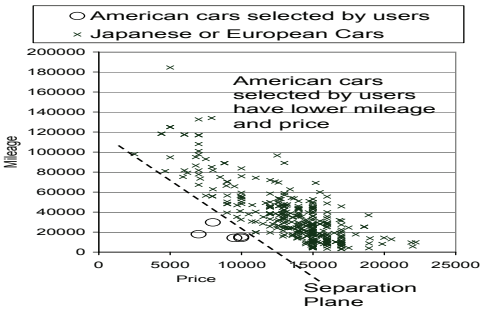


Fig. 4. American cars vs. European/Japanese cars

or in other groups) become negative examples. The rationale is that the selection of g implies that users are more interested in records in g than records in other groups so records in g should be ranked higher. Similarly, skyline represents users' static preferences so skyline records should be ranked higher.

In learning the ranking function, we could collect the record pairs from the positive examples and negative examples, assign a label to each pair representing the relative rank of the two records, and then train a classification model based on the labeled data. However, this approach is computationally expensive as the number of record pairs is very large. In addition, this approach implicitly assumes the equal importance of each record pair and often leads to biased classification model [22].

Instead, we simplify the learning problem to a binary classification problem. We use support vector machine (SVM) [7] to learn a hyperplane that best separates the positive examples from the negative examples. We choose SVM for two reasons. First, the separation plane learned from SVM indicates the importance (or weights) of different attributes in the ranking function (see Section 3.2). Thus SVM automatically tunes the weights of attributes in the ranking function. Since we construct a different set of training examples for each selected group, SVM allows us to dynamically tune the ranking functions for different groups. Different users often select different groups, thus our approach addresses the diversity issue of user preferences. Second, SVM is less likely to overfit the training examples and thus works well for incomplete preference information [7].

Intuition behind the learning method: Next we use an example to show the intuition behind our learning method.

Example 1. We used an Auto data set consisting of used cars. The detail of the data set is given in Section 5.1. Some users asked a query that returns 3 entry level car models (one Japanese, one European, and one American model). Results in Example 1 are divided into 3 groups, one for each car model. We also asked some users to select some cars in each group.

Intuitively, SVM will try to find out what properties distinguish positive examples from negative examples. Next we will show that such distinctions often provide useful insights into the importance of attributes.

Fig. 2 shows the price and mileage of Japanese cars. The circles are cars finally selected by these users. The dotted shape indicates records in the skyline. Clearly, most selected records belong to the skyline, that is, they have more desired values (e.g., lower price or mileage) than other cars.

Fig. 3 depicts mileage and no. of cylinders for European cars selected by users and cars in other two groups (i.e., American or Japanese). It shows that European cars selected by users often have more cylinders than cars in the other groups. On the other hand, the selected European cars have similar mileage as cars in other groups. Thus the ranking function for European cars should put a higher weight on no. of cylinders than on mileage. A possible explanation is that European cars are more luxury so those who are interested in European cars

want a powerful engine (i.e., with more cylinders).

Fig. 4 plots price and mileage for selected American cars and cars in other groups. It shows that the ranking function for American cars should put significant weights on both mileage and price because selected American cars have either lower mileage or lower price than Japanese and European cars. A possible explanation is that buyers of American cars know that American cars with high mileage are likely to break down so they prefer cars with lower mileage. They also know that American cars have lower resale values so they want to buy them at a cheap price upfront.

The above examples show that the distinctions between positive training examples and negative examples indicate which attributes are more important in the ranking function. Since SVM tries to find such distinctions, it is suitable for the learning task. However, the basic SVM method may still be expensive for large data sets. It also has some quality problems because as shown in Fig. 2, some of the skyline records (e.g., those in the upper left part of the skyline in Fig. 2) are not selected by users and thus should not be ranked high. We propose an Iterative method that addresses these shortcomings. Our contributions are summarized as follows:

- We propose a basic learn-to-rank method using skyline records and users' current navigational behavior.
- We propose an Iterative method that improves both quality and efficiency of the basic method.
- We conduct an empirical study which demonstrates the superiority of our approach over existing solutions.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 and Section 4 describe the basic method and the Iterative method, respectively. Section 5 presents experimental results. Section 6 concludes the paper.

2 Related Work

Database ranking and top-k queries: There has been recent work on ranking results of database queries as well as linked objects using query history or partial orders [3], [11], [2], [9]. However, as mentioned in Section 1, existing solutions do not address the heterogeneous issue and the incompleteness issue. More specifically, existing solutions do not capture users' current preferences, and do not work well when there is only limited information about users' preferences (e.g., when query history is not available). This paper proposes a solution that addresses these two issues.

Query result navigation: There has been a lot of work on query result navigation using automatic categorization [8], [12], clustering [21], or multiple facets [26]. Navigation can be also done using group-by and OLAP queries [16]. There also has been work on providing keyword search over data cubes [30]. Li et al. [20] proposed a method to combine ranking and clustering approaches. However, the focus of their work is on efficiency and their work uses fixed ranking functions.

The effectiveness of navigational techniques certainly has some impact on the effectiveness of our methods (e.g., an ineffective clustering technique probably will not lead to meaningful navigational behavior). However in this paper we focus on learning ranking functions based on users' navigational behavior and will study how to choose the most effective navigational technique as future work.

Skyline queries: There has been a rich body of work on computing skyline queries [6], [19]. However, most such work focuses on the efficiency issue. Further, as shown in Fig. 2, there are many skyline records that users will not select. Thus simply returning skyline points as top ranked records is insufficient. [28] used skyline to visualize the results. Recent efforts have also been reported on refining skyline queries using positive/negative examples [17], [24]. However, they do not consider how to learn the ranking function based on skyline.

Information retrieval: There also has been work on information retrieval [27], [5] using query history or click-through history. However such information may not represent users' *current* preferences.

A more recent paper [23] considered how to cluster and rank text documents at the same time. The basic idea is that a document that is more similar to the centroid of a cluster will be ranked higher. However, this is often inappropriate for ranking relational data because users often want to find records with optimized attribute values (e.g., cars with lower price and mileage) rather than records that are close to the centroid of a cluster (e.g., cars with average price and mileage).

RankSVM is a popular learning to rank method in information retrieval (IR) community [10], [18]. The basic idea is to formalize learning to rank as a problem of binary classification on instance pairs,

and then to solve the problem using Support Vector Machines [10], [18]. In [25], a Bayesian approach was developed to present users a selected lists of results so that users can click on some results to form more informative training data. However, such work has the following limitations: 1) it does not consider relational data; 2) it assumes that users have explicitly provided some ranked pairs or ranked lists as training examples; 3) the same ranking function is used to rank all records in the result, which ignores the diversity of user preferences; 4) the learning algorithm is often quite expensive due to the need to consider all pairs. Our approach aims at relational data. It also does not require ranked pairs or lists. Instead, training examples are inferred from skyline and users' navigational behavior. Our approach also dynamically adjusts the ranking function for each cluster, thus it addresses the diversity of user interest. Finally, our approach is more efficient than pair-wise learning approaches because it simplifies the learning problem to a binary classification problem.

In [32], an SVM-ranking mechanism was used to learn a global ranking function which is amenable to existing relational DBMS by allowing users to qualitatively and intuitively express their preferences via ordering some samples. Our work differs from [32] from the following two aspects: 1) Instead of providing a global ranking function, our approach dynamically adjusts the ranking function for each cluster, thus it addresses the diversity of user interest. 2) our approach does not require ranked pairs/lists or ordered samples. Instead, training examples are inferred from skyline and users' navigational behavior.

3 Basic Learn-to-Rank Method

We first introduce some definitions. We then discuss the relationship between ranking functions and the separation plane (a concept used in SVM). Finally we describe the basic learn-to-rank method.

3.1 Definitions

Let D be a relation with n records and m attributes $A_1, ..., A_m$. Let Q be a query and the results of Q are partitioned into t non overlapping groups $c_1, ..., c_t$ using any existing navigational technique. Let $r \in D$ be a record and r_j be the value of A_j in r. We assume that a skyline operator has been defined by a user or some domain experts.

Definition 1: Skyline operator SO: SO consists of a set of rules for optimizing one attribute or a subset of attributes. Let l be the number of rules. Each rule $SO_j (1 \leq j \leq l)$ is a triplet $(SA_j, s_j(r), sign_j)$ where SA_j is a subset of attributes. $s_j(r)$ is a function defined over values of attributes in SA_j in a record r. s_j is called a ranking term. $sign_j$ equals MAX (meaning s_j needs to be maximized), MIN (s_j needs to be minimized), or $DIFF$ (users are interested in all distinct values of s_j).

For example, when people are buying antiques, they are more interested in rare features. Thus we can define a rule for a feature attribute A_j s.t. ($SA_j = \{A_j\}$, $s_j(r) = IDF(r_j)$, $sign_j = MAX$), where IDF is inverse document frequency [4]. For a numerical attribute such as price, we know we need to minimize it. Thus the rule is ($SA_j = \{A_j\}$, $s_j(r) = r_j$, $sign_j = MIN$). For a categorical attribute, users may specify some desired values such as color is red. This can be represented as ($SA_j = \{A_j\}$, $s_j(r)$ equals 1 if r_j is red and 0 otherwise, $sign_j = MAX$). We can even define a rule that optimizes a function over multiple attributes. For example, we want to minimize the distance from a hotel to the beach. Let A_x and A_y be the coordinate attributes of a hotel. The rule is ($SA_j = \{A_x, A_y\}$, $s_j(r) = dist(r_x, r_y)$, $sign_j = MIN$) where $dist(r_x, r_y)$ is the distance from (r_x, r_y) to the beach. It should be pointed out that domain experts can define these rules and users can modify them. Our work focuses on how to weight/optimize different terms, not on the definition of each rule.

Next we put these ranking terms (s_j) together into a weighted sum function. We choose weighted sum for two reasons: 1) many existing ranking functions (e.g., TF/IDF, Okapi BM25 [4]) can be represented in weighted sum format; 2) it simplifies the learning process (see Section 3.2).

Definition 2: Ranking function: A ranking function F_i for group c_i can be represented as for all $r \in c_i$

$$F_i(r, Q) = \sum_{j=1}^{l} w_{ij} s_j(r)$$

where s_j is a ranking term obtained from the skyline operator, w_{ij} is the weight of that term, and l is number of rules in skyline operator.

E.g., uniform ranking function $F = \sum_{j=1}^{m} \frac{sign(A_j)}{m} r_j$, where $s_j = r_j$ and $sign(A_j)$ equals +1 if the attribute should be maximized and equals -1 if it should be minimized.

A record r dominates a record r' if for every term s_j that needs to be maximized, $s_j(r) \geq s_j(r')$

and for every term s_k that needs to be minimized, $s_k(r) \leq s_k(r')$. The skyline consists of all records r such that there does not exist any other record r' that dominates r. We use $skyline(c_i)$ to represent the set of records in the skyline of group c_i.

Definition 3: Learn-to-rank problem: Suppose users have selected a group c_i. Let $P = skyline(c_i)$ be the set of positive examples, and $N = (c_i - skyline(c_i)) \bigcup (\cup_{k \neq i} c_k)$ be the set of negative examples. We want to learn a ranking function F_i^* s.t. for any $r \in P, r' \in N$, $F_i^*(r, Q) > F_i^*(r', Q)$.

Since s_j is defined by users, the problem of finding local ranking function for group c_i is reduced to the problem of finding the most appropriate weights w_{ij}.

3.2 Relationship between Ranking Functions and The Separation Plane

Given a record r, we can view $s(r) = (s_1(r), \ldots, s_l(r))$ as a point in a l dimensional space, where the score of each term s_j forms a dimension. The weights $(w_{i1}, w_{i2}, ..., w_{il})$ for ranking function F_i can be also viewed as a l-dimensional *weight vector* $\mathbf{w_i}$. The ranking function $F_i = \sum_{j=1}^{l} w_{ij} s_j(r)$. Thus $F_i(r) = \mathbf{w_i} \cdot \mathbf{s(r)}$ where $\cdot$ is the inner product.

Next we explain the relationship between the separation plane and the learn-to-rank problem described in Definition 3. Suppose there is a separation hyperplane $\mathbf{w_i} \cdot \mathbf{x} + \mathbf{b} = \mathbf{0}$ that separates P from N s.t. for any record $r \in P$, $\mathbf{w_i} \cdot s(r) + b > 0$ and for any record $r' \in N$, $\mathbf{w_i} \cdot s(r') + b < 0$. For example, Fig. 4 shows a linear separation plane. We have proved the following theorem.

Theorem 1: Given a separation plane $\mathbf{w_i} \cdot x + b = 0$ that correctly separates P and N, a ranking function $F_i(x, Q) = \mathbf{w_i} \cdot \mathbf{s(x)}$ will rank all records in P ahead of records in N.

The proof is straightforward. For any $r \in P$ and $r' \in N$, $F_i(r, Q) = \mathbf{w_i} \cdot s(r) > -b > F_i(r', Q) = \mathbf{w_i} \cdot s(r')$.

Theorem 1 also shows that the weights of the ranking function equal the weights of the separation plane. Thus the problem of learning the ranking function is reduced to the problem of finding the separation plane.

3.3 Basic SVM method

Linear SVM: When there are multiple possible separation planes, we need to find the separation plane with less chance of overfitting. It is important not to overfit the training examples because as mentioned before, the system often has incomplete information about users' preferences. SVM is a well-known method that has less chance of overfitting. It tries to maximizes the margin of error (more specifically, the distance from the separation plane to the nearest positive or negative examples). It has been proved [7] that SVM provides upper bounds for both empirical error (i.e., error over training examples) and generalization error (error over other possible data sets) and these error bounds are minimized by SVM. Please refer to Appendix for details.

We will use SVM to find the separation plane. To improve efficiency, we can precompute the skyline, and store in the database a flag column to indicate whether a record is in the skyline. When users ask a query, the flag column will be returned along with query results. Thus we can easily identify skyline in each group. We will then generate P and N as described in Definition 3 and then use SVM to learn the ranking function.

Non Linear SVM: When P and N are not linearly separable (e.g., Fig. 2 and 3), we can use the kernel method [7] to map data to a high dimensional space using a nonlinear transformation ϕ. More specifically, there is a transformation $\phi(x)$ and kernel function $K(x, y)$ (x, y are two data vectors) s.t. $\phi(x) \cdot \phi(y) = K(x, y)$. Now the ranking function $F_i(r, Q) = \mathbf{w_i} \cdot \phi(s(r))$. Usually it is difficult to compute ϕ directly, but we can directly compute the ranking score as

$$F_i(r, Q) = \sum_{y \in P \cup N} \alpha_y c_y K(s(r), s(y)) \quad (1)$$

Here $\alpha_y (y \in P \cup N)$ are returned by SVM. $c_y = 1$ if $y \in P$ and $c_y = -1$ if $y \in N$. $K(s(r), s(y))$ can be computed using the kernel function K. Usually there are very few $\alpha_y \neq 0$. These y are called support vectors, and they lie on the boundary of P and N. Thus we just need to use support vectors when computing the ranking score.

4 Iterative Method

The basic SVM method has two problems. First, SVM is quite expensive. Second, some skyline records are not selected by users and thus should not be ranked high. We propose an Iterative method that addresses these shortcomings. Section 4.1 and Section 4.2 describe these improvements. Section 4.3 presents the pseudo code of the

iterative method. The theoretical analysis of the iterative method is given in Appendix.

4.1 Quality Improvement

We use an iterative refinement method to improve the quality of the basic SVM method. The iterative method works as follows. Initially P and N are generated as in Definition 3 and SVM is used to learn the current ranking function F_i. The problem is that some of the records in P may not be selected by users eventually because they will receive a very low ranking score from the final ranking function. Thus we want to remove such records. Of course the final ranking function F_i^* is not known yet. But we can use the current ranking function F_i as a good approximation. So at each round we use F_i to rank results in $P \cup N$ and move the last n_r records in the rank order from P to N. For example, in Fig. 2, the initial P includes cars with very high mileage (the top most car in the skyline has mileage over 100,000). Such cars are likely to receive a very low ranking score by F_i and will be moved to N.

The process is then repeated over the new P and N. Let $F_i(r, Q) = \mathbf{w_i} \cdot s(r)$ be the ranking function learned in the current round and $F_i^0(r, Q) = \mathbf{w_i^0} \cdot s(r)$ be the function learned in the previous round. The algorithm stops when $|\mathbf{w_i^0} - \mathbf{w_i}| < t_\epsilon$ where t_ϵ is a threshold and $|\mathbf{w_i^0} - \mathbf{w_i}|$ is the Euclidean distance between $\mathbf{w_i}$ and $\mathbf{w_i^0}$. We also normalize these weight vectors such that their L_2 norm is one. The pseudo code is shown in Fig. 5.

Convergence: We call records in final P *winners* and records in the final N *losers*. Suppose a "good" ranking function has a higher probability to rank a winner ahead of a loser. We have proved that when $n_r << |P \cup N|$ and F_i is a good ranking function, the probability of a winner is ranked at bottom n_r positions by F_i is extremely low. The proof is in Appendix. Thus when the bottom n_r records are removed from P at each iteration, most likely only losers are removed. After a few rounds, all losers are removed from P and the algorithm will converge.

A too small n_r or t_ϵ may also cause slow convergence. We set $n_r = 10$ because it leads to quick convergence in our experiments and it is much smaller than $|P \cup N|$ in most cases. We set t_ϵ to 0.01 because it also leads to quick convergence. More details are given in Section 5.4.

4.2 Efficiency Improvement

The most expensive step of our approach is SVM. A selective sampling approach has been proposed to only asking users to label samples close to the class boundary [31]. However, our approach does not require explicit user labeling. So we use several other methods to improve efficiency.

Reducing the size of N: N consists of non skyline records in group c_i and records in other groups. This is unnecessary because only support vectors (i.e., those with $\alpha_y \neq 0$ in Equation 1) will affect the result of SVM [7]. We have proved in Appendix that any non skyline record r' in c_k ($k \neq i$) cannot be a support vector. The intuition is that support vectors in N must have the highest ranking scores in N [7]. Since r' is not a skyline record, there exists a record $r \in skyline(c_k)$ and r has a higher ranking score than r'. Both r and r' belong to N. Thus r' does not have the highest score in N and cannot be a support vector. Hence we only need to include in N non skyline records in c_i and skyline records in other groups, i.e.,

$$N = (c_i - skyline(c_i)) \bigcup (\cup_{k \neq i} skyline(c_k)). \quad (2)$$

Incremental SVM: Since SVM is used repeatedly in the Iterative method, we use incremental SVM [14] to reduce the cost of SVM in subsequent rounds. At each round, the bottom n_r records are moved from P to N. Using incremental SVM, we only need to check whether moving these records will affect the support vectors in the previous separation plane. Since $n_r << |P \cup N|$, the cost of incremental SVM is much lower than rerunning SVM.

Pre-Ranking: Incremental SVM does not apply to the SVM step in the first round, so we want to further reduce the cost of the first round. By [7] the separation plane only depends on support vectors. Thus we just need to ensure that support vectors in the final SVM are included in the training data. Let SV be the set of support vectors. We have proved in the Appendix that a support vector will be ranked at top $|SV| + |P|$ positions by the final ranking function. In practice we expect $|P|$ (the size of final P) be quite small because users will not select many records. The number of support vectors is also quite small in general. Thus we can use an initial ranking function (e.g., the uniform ranking function in Section 3.1) to prerank the records in $P \cup N$. When $n_0 >> |SV| + |P|$, any record that is ranked below the top n_0 positions is unlikely to belong to SV. Thus we only use the first n_0 records

Input: Groups $c_1, \ldots, c_t$, current selected group c_i, and two parameters: n_r as the number of records to move from P to N in each round, and t_ϵ as the stopping threshold.
Output: local ranking functions $F_i(r, Q)$
(1) $P = skyline(c_i)$
(2) $N = (c_i - P) \cup (\cup_{j \neq i} skyline(c_j))$
(3) $\mathbf{w_i}$ = the weight vector for a uniform ranking function where $w_{ij} = \frac{sign(s_j)}{l}$. Here l is the number of rules in the skyline operator, $sign(s_j)$ equals +1 if s_j should be maximized and equals -1 if it should be minimized
(4) repeat
(5) $\mathbf{w_i^0} = \mathbf{w^i}$
(6) $(\mathbf{w_i}, \{\alpha_y | y \in P \cup N\}) = SVM(P, N)$
/* SVM returns weight vector $\mathbf{w_i}$ and support vectors α_y */
(7) normalize $\mathbf{w_i}$ s.t. $\sum_{j=1}^{l} w_{ij}^2 = 1$
(8) $F_i(r, Q) = \mathbf{w_i} \cdot s(r)$ when P and N are linearly separable
$F_i(r, Q) = \sum_{y \in P \cup N} \alpha_y c_y K(s(r), s(y))$ when P and N are not linearly separable
(9) rank records in $P \cup N$ using F_i
(10) D_i = the bottom n_r records
(11) $P = P - D_i$, $N = N \cup D_i$
(12) until $|\mathbf{w_i}^0 - \mathbf{w_i}| < t_\epsilon$
(13) Return $F_i(r, Q)$

Fig. 5. The Iterative Algorithm.

in the learning process. We set $n_0 = 500$ in this paper because it is much larger than $|SV| + |P|$ and we did not observe any noticeable impact on the effectiveness of SVM in our experiments.

We can precompute the initial ranking scores and store them at an "order" column in the database. At run time, the order column is returned along with the query results. At run time, we can use a binary heap to select the n_0 top ranked records using the order information.

Complexity analysis: The skyline is precomputed thus the cost of retrieving skyline operator at run time is $O(n)$, where n is the result size. Pre-ranking costs $O(n \log n_0)$. Let $O(S(nm))$ be the complexity of SVM over an input with n rows and m attributes (it is difficult to give exact definition of S [7]). Suppose $l \sim m$ (i.e., the number of terms is close to the number of attributes). The first SVM step takes $O(S(n_0 m))$. It takes $O(mn_0 \log n_r)$ time to find the bottom n_r records. A subsequent round of SVM takes $O(S(n_r m))$ using incremental SVM. Suppose the algorithm stops in t_n rounds. Thus the total cost for one group is

$$O(t_n(S(n_r m) + mn_0 \log n_r) + S(n_0 m) + n \log n_0) \quad (3)$$

4.3 Algorithm Description of the Iterative Method

The pseudo code is shown in Fig. 5. Line 1 to 3 initialize P, N, and an initial uniform ranking function. Line 4 to 12 repeatedly use SVM to learn the current ranking function, use the ranking function to rank records in $P \cup N$, and move the last n_r records from P to N. The process stops when the weight vector (also the ranking function) converges.

5 Experimental Evaluation

5.1 Setup

We conducted our experiments on a machine with Intel PIII 3.2 GHZ CPU, 2 GB RAM, and running Windows XP.

Dataset: We used two real data sets. The first is a Fund data set that contains 14604 mutual funds downloaded from *www.scottrade.com*. There are 33 attributes, 28 numerical and 5 categorical. The total data size is 20 MB. We used 6 attributes in the ranking algorithms: "1 year return", "3 year return", "standard deviation", "minimal investment", "expenses", and "manager year". The second data set is part of an online used-car dealer's nationwide database. It contains 15192 used cars. We used 6 attributes in ranking: "year", "price", "mileage", "number of cylinders", "has cd charger", and "has alloy wheel". All attribute values are also normalized to the range of 0 to 1.

Queries and ground truth: We used 10 test queries (5 for each data set). Table 1 describes the test queries in our experiments, along with the result size, the number of relevant records in ground

truth, and the number of groups. We chose the queries such that they represented various user interests. For the Fund data, we used 5 test queries ($Q1$ to $Q5$ in Table 1), where the first 4 queries had conditions on various attributes and the last one returned all records in the data set. For the Auto data, we used 5 queries ($Q6$ to $Q10$ in Table 1). Since buyers of used cars often compare different models of cars or SUVs in similar classes (e.g., compare 3 entry level car models), we used queries to cover these cases. $Q6$ returned 3 entry level car models. $Q7$ returned 3 middle level car models. $Q8$ returned 3 luxury models. $Q9$ returned 3 compact SUV models and $Q10$ returned 3 midsize SUV models. For each query, one model was Japanese, one was European, and one was American.

We used a user-study to collect ground truth. We enrolled 55 participants (students). We have implemented two navigational techniques: k-means clustering (using ranking attributes) and group-by. The number of clusters (k) is automatically decided using a commonly used "Elbow" method [13], which selects the number of clusters where adding a new cluster will not significantly reduce the sum of square error. Each participant was asked to look at the query results for each test query and select a navigational technique. We found that all participants selected k-means clustering for Fund data because all ranking attributes in that data set were numerical so it was difficult to use group-by. K-means clustering also returned meaningful clusters. For most queries, 2 clusters were generated: one containing stock funds (with high risk) and the other containing bond funds (with low risk). For Auto data set, all participants selected group by car models or group by price ranges. Price was divided into 3 ranges: 0 to \$14,999, \$15,000 to \$29,999, and \$30,000 and above.

Next each participant was asked to look at the groups or clusters generated by their selected navigational technique, and select some groups to visit. For each selected group, the participant looked at all records in the group and selected some. To remove randomness, we considered any record that was selected by at least 2 participants as relevant. We also observed that each group was selected by at least two participants. The number of groups and the number of relevant records are also listed in Table 1.

Metrics: We used precision/recall at position k as the metric. For a group c_i, let R_i be the set of relevant records, and let S_i be the set of top-k records returned by a ranking method. The precision at position k equals $\frac{|S_i \cap R_i|}{k}$. The recall at position k equals $\frac{|S_i \cap R_i|}{|R_i|}$

Since the size of $|R_i|$ varies in each group, it is inappropriate to compute the precision and recall at a fixed position. Instead we compute precision and recall at position $|R_i|$ at each group. In this case $k = |R_i|$, thus precision equals recall. Since there are multiple groups, we also report the average precision (recall) across groups.

Algorithms: We implemented both the Iterative method and the Basic method in Matlab 7.0. We asked some domain experts to specify whether an attribute should be maximized or minimized. Thus $s_j(r) = r_j$ and our ranking function is $F_i = \sum_{i=1}^{m} w_{ij} r_j$. We set $t_\epsilon = 0.01$ and n_r=10 for the Iterative method. We also compared these two methods with three existing methods below. All ranking algorithms were used to rank only records in the selected groups.

1) Probabilistic ranking: this is the method proposed in [11]. Details can be found in Appendix.
2) Uniform ranking (described in Section 3.1).
3) Centroid ranking: this method computes the centroid for each group, and ranks the records in a group in the ascending order of their distance to the centroid. This is similar to the method proposed in [23].

5.2 Average Precision/Recall

Fig. 6 shows the average precision and recall for the 5 test queries for Fund data. Fig. 7 shows the results for 5 testing queries for the Auto data using group-by model. Fig. 8 shows the results for Auto using group-by price.

To measure the impact of using users' current navigational behavior, we also report the results for a variant of Iterative method which does not use the navigational behavior. This variant only includes non skyline records in a selected group c_i in negative examples, i.e., $N = c_i - skyline(c_i)$.

The results show that the precision and recall of the Iterative method are significantly higher than existing ranking methods because it uses both users' navigational behavior and skyline records. The Iterative method is also better than the Basic method because it uses the iterative process to remove lower ranked records in the skyline. The Iterative method also outperforms the variant with-

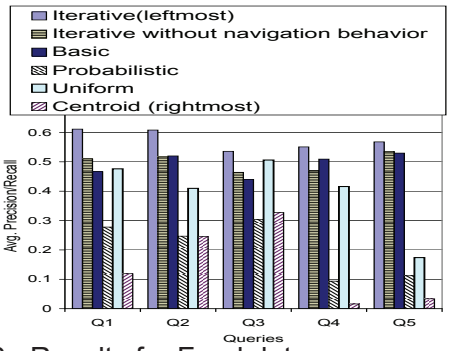


Fig. 6. Results for Fund data

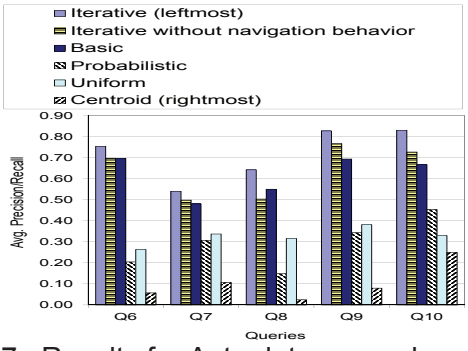


Fig. 7. Results for Auto data, group by model

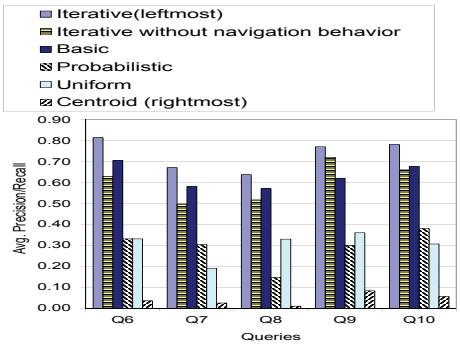


Fig. 8. Results for Auto data, group by price

Attributes	Japanese	European	American
year	0.1244	0.2247	0.1278
price	-0.2277	-0.225	-0.4164
mileage	-0.4162	0	-0.4166
no. of cylinders	0.1987	0.5503	0
has cd changer	0.0252	0	0.0152
has alloy wheel	0.0078	0	0.024

Fig. 9. Weights assigned by the Iterative method for different groups for $Q6$ over Auto data

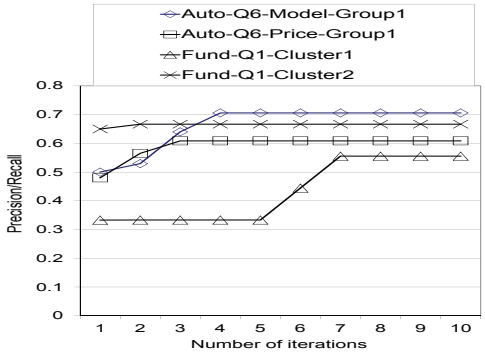


Fig. 10. Varying number of iterations.

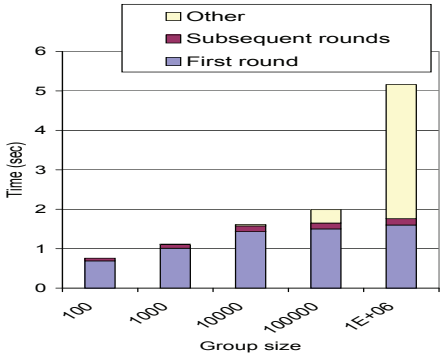


Fig. 11. Execution time of the iterative method when varying group size

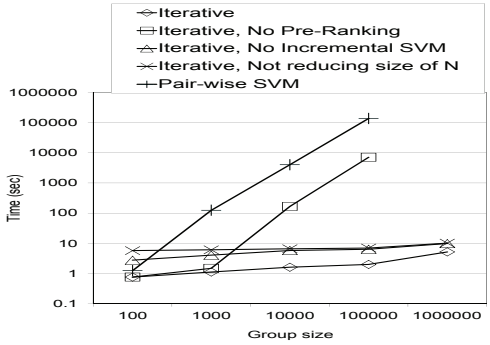


Fig. 12. Execution time when varying group sizes, using all 6 attributes

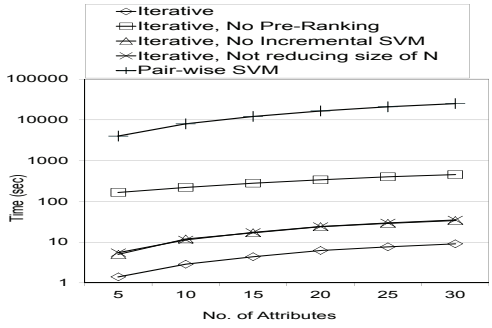


Fig. 13. Execution time when varying number of attributes, group size = 10,000

TABLE 1
Test queries

Data set	Query Condition	Result size	No. of relevant records in ground truth	Number of groups
Fund	Q1: Fund Name like '%Vanguard%'	139	30	2
Fund	Q2: Expense Ratio ≤ 0.2	290	71	2
Fund	Q3: Manager Year ≥ 20	140	40	2
Fund	Q4: Minimal investment ≤ 100	1264	128	4
Fund	Q5: All funds	14604	694	2
Auto	Q6: Model = Corolla or Jetta or Focus	534	41	3
Auto	Q7: Model=Camry or Passat or Malibu	511	56	3
Auto	Q8: Model= Acura or BMW or Cardilac	1277	67	3
Auto	Q9: Model= Honda CR-V or BMW X3 or Ford Espace	203	32	3
Auto	Q10: Model= Honda Pilot or BMW X5 or Ford Explorer	514	81	3

out using the navigational behavior. This verifies the benefits of using navigational behavior.

Centroid ranking has the worst performance among all methods. This indicates that most users are interested in optimized values rather than average values. Uniform ranking is always better than Centroid ranking and is often better than Probabilistic ranking, possibly because it uses static and common preferences to some degree (by assigning $sign(A_j)$ based on whether A_j should be maximized or minimized).

We also ran a t-test between Iterative and other methods to verify the sensitivity of results. The p-values for the null hypothesis are below 0.001 in all cases. Thus the results are statistically significant.

5.3 Analysis of An Example

This section revisits the 3 groups shown in Fig. 2 to Fig. 4 (they are generated from the results of $Q6$ over Auto data using group-by model). Fig. 9 shows the weights assigned by the Iterative method. Interestingly, the assigned weights agree with the insights we gained from Fig. 2 to Fig. 4. For example, for the European model, the highest weight is assigned to number of cylinders (i.e., a more powerful engine is more important for those who want to buy a European car). For American and the Japanese models, mileage receives the highest weight (it is negative because mileage needs to be minimized) because people who buy Japanese and American models care more about how long the car will last. Price is also assigned a higher weight for American model than the Japanese model, due to the lower resale values of American cars.

5.4 Impact of Number of Iterations

This section examines the impact of the number of iterations for the Iterative method. Fig. 10 shows the results for Q1 (over Fund) and Q6 (over Auto). For the Auto data set, we report the results for group 1 using both group-by methods. For the Fund data set, we report results for cluster 1 and 2. The results for other queries and groups are similar. The results show that the precision and recall converge very quickly in general, though at a different pace for different queries or groups. On average, the Iterative method took 5.58 iterations for the Auto data set with a standard deviation of 1.86. It took on average 4.45 iterations for the Fund data set with a standard deviation of 1.75. This shows that the Iterative method converges quickly.

5.5 Execution Time

The precomputation time of our approach was about 1 second for both data sets. Thus we focus on the execution time at run time.

To study the scalability of our methods, we generated a synthetic data set from the Fund data set based on the condensation method proposed in [1], which will preserve the statistical properties (including correlations between attributes) of the original data. We scaled up the data set by 100 times. We used query Q5 (returning all funds). The groups are generated as follows. We first used the same clustering techniques for our experiments on the original Fund data to generate two clusters. The selected group contains records in the larger cluster that are closest to the cluster center and the unselected group contains all the remaining rows. E.g., if the selected group's size is 100, then the first 100 closest records are included in the selected

group. We varied the size of the selected group from 100 to 1 million.

We have also considered a pair-wise variant of our method, which used the pair-wise learning approach in [32]. The pair-wise approach considers each pair of positive and negative examples. Let r_p be a positive example and r_n be a negative example. This method generates two new data points: $r_p - r_n$ and $r_n - r_p$, where the first one is labeled in class 1 and the second is labeled in class 2. SVM is then used over all generated points. The rest of the method is the same as ours. Since the pair-wise variant uses the same training examples as ours, we do not find significant difference in precision and recall. So we only report execution time here.

Fig. 11 reports the execution time of the Iterative method when we varied the number of records in a group from 100 to 1,000,000. The time is also broken down into the time of first round of SVM, the time for subsequent rounds, and the time for other operations, including the time to return pre-ranked records (if the group size is over 500) and to return final ranked results.

The results show that the execution time grows at a much slower pace than the group size due to pre-ranking (which limits the number of records used in SVM to 500). The time for subsequent SVM is also quite small compared to the time of the first round SVM due to incremental SVM. The total execution time was around 5.2 seconds for 1,000,000 records. Thus our method can be used in an interactive environment. The time for pre-ranking and final ranking (denoted as other) increases almost linearly with group sizes.

Fig. 12 reports the execution time of the Iterative method with three variants where pre-ranking, incremental SVM, or reducing the size of N was turned off. We also reported the time for pair-wise variant. We varied the group sizes as in Fig. 11. We were not able to report the results for the Basic method because it did not finish within 3 days for the scaled up data set. The variant without pre-ranking and the pair-wise variant also did not finish within 3 days when there were 1 million records in the group. We also varied the number of attributes from 5 to 30 for the data set with group size 10000 (the original Fund data has 33 attributes) by including different number of attributes in the ranking function. The results are shown in Fig. 13.

The results show that the Iterative method scales almost linearly to the number of attributes and the size of group. The pair-wise variant has the worst performance because the number of pairs is quadratic to the number of training examples. The results also show the benefits of incremental SVM, pre-ranking, and reducing the size of N. The execution time is significantly longer if any of these techniques is turned off (the time is in logarithmic scale). The improvement brought by pre-ranking is the most significant because without it SVM cannot handle large data sets. We also find that there is no noticeable difference in the precision and recall between these variants and the Iterative method. Thus using these techniques will not affect the effectiveness of our method.

5.6 Usability of Our Approach

We also conducted a usability study. We want to compare our group wise ranking approach (results are first divided into groups and then ranked) with a global ranking approach (simply ranking all results). Among the three ranking methods described in Section 5.1, both Uniform and Probabilistic ranking are global ranking methods. Centroid is a group wise method but as shown in results in previous sections it has the worst performance among all methods. So we used Uniform and Probabilistic ranking in the global ranking approach. We used Iterative ranking (our approach) in the group wise approach.

We enrolled 51 participants (all students and with no overlap with those participated in the study of collecting ground truth). We designed a user interface which allows users to search for used cars using a SQL query form. The form allows users to enter conditions on the 6 attributes used in our experiment. Once the results are returned, users will see the list of results as well as 4 buttons: "group by model", "group by price range", "rank (method 1)", and "rank (method 2)". The first button will group results by model name and the second button will group results by the 3 price ranges. If users click on the third or fourth button, then either Uniform or Probabilistic ranking will be used to rank all results (we also randomly assigned Uniform or Probabilistic to Method 1 or 2 to avoid bias). If users click on one of the first two buttons, they will see summary information for each group, including the number of rows in each group and the model name or price range associated with the group. They can then click on one of the groups, where our Iterative method will be used to rank results in the selected group. They are asked to try all ranking

TABLE 2
Results of Usability Study

Approach	# participants called it the best
Our approach	39
Uniform ranking	4
Probabilistic ranking	3
Did not respond	5

options (i.e., all four buttons) and at the end they are asked to indicate which approach they consider the best. For our method, the skyline operator is also predefined to suit the common preferences: all users should prefer lower values on price and mileage and higher values on the other 4 attributes.

Table 2 summarizes the results. 39 participants prefer our group wise approach (roughly equal number of them prefer each group by option) while only 4 prefer the uniform approach, 3 prefer probabilistic approach, and 5 did not respond.

6 Conclusion

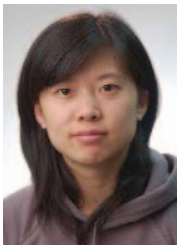
This paper proposes a local learn-to-rank approach that uses skyline records and users' current navigational behavior. This approach works well for limited and heterogeneous users' preferences. One interesting future research direction is to study how we can improve user's navigational experience using ranking method. For example, can we generate better clusters based on the ranking function?

References

[1] C. C. Aggarwal and P. S. Yu. A condensation approach to privacy preserving data mining. In *EDBT*, 2004.
[2] R. Agrawal, R. Rantzau, and E. Terzi. Context-sensitive ranking. In *SIGMOD Conference*, pages 383–394, 2006.
[3] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In *CIDR*, 2003.
[4] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison& Wesley, 1999.
[5] R. A. Baeza-Yates and A. Tiberi. Extracting semantic relations from query logs. In *KDD*, pages 76–85, 2007.
[6] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.
[7] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining Knowledge Discovery*, 2(2):121–167, 1998.
[8] K. Chakrabarti, S. Chaudhuri, and S. won Hwang. Automatic categorization of query results. In *SIGMOD*, pages 755–766, 2004.
[9] K. Chakrabarti, V. Ganti, J. Han, and D. Xin. Ranking objects based on relationships. In *SIGMOD Conference*, pages 371–382, 2006.
[10] O. Chapelle and S. S. Keerthi. Efficient algorithms for ranking with svms. *Inf. Retr.*, 13:201–215, June 2010.
[11] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic ranking of database query results. In *VLDB*, pages 888–899, 2004.
[12] Z. Chen and T. Li. Addressing diverse user preferences in sql-query-result navigation. In *SIGMOD*, pages 641–652, New York, NY, USA, 2007. ACM.
[13] J. David J. Ketchen and C. L. Shook. The application of cluster analysis in strategic management research: An analysis and critique. *Strategic Management Journal*, 17(6):441?58, 1996.
[14] C. P. Diehl and G. Cauwenberghs. SVM incremental learning, adaptation and optimization. In *In Proceedings of the 2003 International Joint Conference on Neural Networks*, pages 2685–2690, 2003.
[15] T. Evgeniou, L. Perez-Breva, M. Pontil, and T. Poggio. Bounds on the generalization performance of kernel machine ensembles. In *ICML*, pages 271–278, 2000.
[16] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. *Data Min. Knowl. Discov.*, 1(1):29–53, 1997.
[17] B. Jiang, J. Pei, X. Lin, D. W. Cheung, and J. Han. Mining preferences from superior and inferior examples. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 390–398, 2008.
[18] T. Joachims. Optimizing search engines using clickthrough data. In *KDD'02*, pages 133–142, New York, NY, USA, 2002. ACM.
[19] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, pages 275–286, 2002.
[20] C. Li, M. Wang, L. Lim, H. Wang, and K. C.-C. Chang. Supporting ranking and clustering as generalized order-by and group-by. In *SIGMOD Conference*, pages 127–138, 2007.
[21] B. Liu and H. V. Jagadish. Using trees to depict a forest. *PVLDB*, 2(1):133–144, 2009.
[22] T.-Y. Liu. Learning to rank for information retrieval (tutorial). In *WWW'08*, 2008.
[23] Y. Liu, W. Li, Y. Lin, and L. Jing. Spectral geometry for simultaneously clustering and ranking query search results. In *SIGIR*, pages 539–546, 2008.
[24] D. Mindolin and J. Chomicki. Discovering relative importance of skyline attributes. *Proc. VLDB Endow.*, 2:610–621, August 2009.
[25] F. Radlinski and T. Joachims. Active exploration for learning rankings from clickthrough data. In *KDD'07*, pages 570–579, New York, NY, USA, 2007. ACM.
[26] S. B. Roy, H. Wang, U. Nambiar, G. Das, and M. Mohania. Dynacet: Building dynamic faceted search systems over databases. In *ICDE*, pages 1463–1466, 2009.
[27] X. Shen, B. Tan, and C. Zhai. Context-sensitive information retrieval using implicit feedback. In *SIGIR*, pages 43–50, 2005.
[28] J. Stoyanovich, W. Mee, and K. Ross. Semantic ranking and result visualization for life sciences publications. In *ICDE 2010*, 2010.
[29] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, NY, 1998.
[30] P. Wu, Y. Sismanis, and B. Reinwald. Towards keyword-driven analytical processing. In *SIGMOD Conference*, pages 617–628, 2007.
[31] H. Yu. SVM selective sampling for ranking with application to data retrieval. In *KDD*, KDD '05, pages 354–363, New York, NY, USA, 2005. ACM.
[32] H. Yu, S.-w. Hwang, and K. C.-C. Chang. Enabling soft queries for data retrieval. *Inf. Syst.*, 32:560–574, June 2007.

Zhiyuan Chen Zhiyuan Chen is an associate professor at the Department of Information Systems, University of Maryland Baltimore County. He received a PhD in Computer Science from Cornell University, Ithaca, NY, in 2002. His research interests include privacy preserving data mining, data navigation and visualization, XML, automatic database tuning, and database compression.

Tao Li Tao Li received the Ph.D. degree in computer science from the Department of Computer Science, University of Rochester, Rochester, NY, in 2004. He is currently an Associate Professor with the School of Computing and Information Sciences, Florida International University, Miami. His research interests are data mining, machine learning, and information retrieval. He is a recipient of NSF CAREER Award and multiple IBM Faculty Research Awards.



Yanan Sun Yanan Sun received her M.S in Information system from University of Maryland Baltimore County in 2009 and B.A from University of Maine at Augusta in 2006. She is currently a PhD student at the Department of Information Systems, University of Maryland Baltimore County. Her general area of research is data mining.