# Query-Level Learning to Rank Using Isotonic Regression

Zhaohui Zheng
Yahoo! Inc.
701 First Avenue
Sunnyvale, CA 94089
{zhaohui, haoying, gzsun}@yahoo-inc.com

Hongyuan Zha
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
zha@cc.gatech.edu

Gordon Sun
Yahoo! Inc.
701 First Avenue
Sunnyvale, CA 94089
{zhaohui, haoying, gzsun}@yahoo-inc.com

## ABSTRACT

Ranking functions determine the relevance of search results of search engines, and learning ranking functions has become an active research area at the interface between Web search, information retrieval and machine learning. Generally, the training data for learning to rank come in two different forms: 1) absolute relevance judgments assessing the degree of relevance of a document with respect to a query. This type of judgments is also called labeled data and are usually obtained through human editorial efforts; and 2) relative relevance judgments indicating that a document is more relevant than another with respect to a query. This type of judgments is also called preference data and can usually be extracted from the abundantly available user clickthrough data recording users' interactions with the search results. Most existing learning to rank methods ignore the query boundaries, treating the labeled data or preference data equally across queries. In this paper, we propose a minimum effort optimization method that takes into account the entire training data within a query at each iteration. We tackle this optimization problem using functional iterative methods where the update at each iteration is computed by solving an *isotonic regression* problem. This more global approach results in faster convergency and signficantly improved performance of the learned ranking functions over existing state-of-the-art methods. We demonstrate the effectiveness of the proposed method using data sets obtained from a commercial search engine as well as publicly available data.

## Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; I.2.6 [**Artificial Intelligence**]: Learning; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

## General Terms

Algorithms, Experiments, Theory

## Keywords

Ranking functions, learning to rank, minimum effort optimization, labeled data, preference data, user clickthrough, functional iterative method, isotonic regression, quadratic programming

## 1. INTRODUCTION

Search engines are essential tools for finding and exploring information on the Web and other information systems. To a large extent the quality of a search engine is determined by the ranking function used to produce the search results in response to user queries. Research and experiments in information retrieval in the past have produced many fundamental methodologies and algorithms including vector space models, probabilistic models and the language modeling-based methodology [20, 19, 3]. More recently, machine learning approaches for learning ranking functions generated much renewed interest from the Web search and information retrieval community as well as the machine learning community. It has the promise of improved relevancy of search engines and reduced demand for manual parameter tuning [18].

Several machine learning methods for learning to rank have been proposed and we will present a brief review in the next section. Most of the methods are based on the supervised learning paradigm and requires training data which come mostly in two different forms: 1) absolute relevance judgments assessing the degree of relevance of a document with respect to a query. This type of labeled data are usually obtained from explicit relevance assessment by human editors, where labels or grades indicating degree of relevance are assigned to documents with respect to a query. For example, a judge can assign a label to a document from the ordinal set *Perfect, Excellent, Good, Fair, Bad*; and 2) relative relevance judgments, also known as pairwise preference data, indicating that a document is more relevant than another with respect to a query [15, 16, 26]. Collecting the first type of data is labor-intensive while the second type of data can be generated from potentially unlimited supplies of user clickthrough data and they also have the advantage of capturing user searching behaviors and preferences in a more timely manner [15, 16, 21]. Moreover, it is also easy to convert labeled data into pairwise preference data.

In this paper, we focus on novel machine learning methods for learning ranking functions from the two types of relevance judgments. Unlike most existing learning to rank

methods, we emphasize the importance of appropriately treating the training data within a query as a whole rather than ignoring the query boundaries. This is a point similar to that of exploring query difference in learning to rank discussed in [25], and it is also in the same spirit as some of the recently proposed listwise learning to rank methods [7].

In particular, we consider either the labeled or preference training data within a query as a set of constraints on the ranking function to be learned. Starting with an arbitrary ranking function, some of the constraints within this set will be violated, we need to modify the ranking function to conform to this set of constraints as much as possible. There are many possible ways to achieve this goal. In this paper, we propose the minimum effort optimization principle: at each iteration, we should spend the least amount of effort to perturbed the current ranking function so as to satisfy this set of constraints within a query as much as possible. As we will see, this formulation not only leads to a tractable quadratic optimization problem, it also has convergence ramifications for the overall optimization approach for learning the ranking functions. It turns out that the minimum effort optimization at each iterative step can be computed by solving an *isotonic regression* problem. Furthermore, the associated quadratic programming problem takes into account the entire preference data within a query. More importantly, the proposed approach also delivers comparable or significantly improved performance of the learned ranking functions over existing state-of-the-art methods. This will be illustrated in the experimental study using data from a commercial search engine as well as data from publicly available sources.

The rest of the paper is organized as follows: in section 2, we review previous work on the topic of learning ranking functions especially its applications in learning ranking functions for information retrieval and Web search. In section 4, we give the precise formulation of the problem of learning from pairwise preference data. We then argue that minimizing the total number of contradicting pairs is a natural way to learn a ranking function and outline a functional iterative method for its solution. We introduce isotonic regression as a means for computing the updates in the iterative method. In section 5, we describe experimental studies using publicly available data as well as data from a commercial search engine. In the last section, we conclude and point out directions for further investigations.

## 2. RELATED WORK

The notion of learning ranking functions in information retrieval can be traced back to the work of Fuhr and coworkers [11, 12, 13]. They proposed the use of *feature-oriented* methods for probabilistic indexing and retrieval whereby features of query-document pairs such as the number of query terms, length of the document text, term frequencies for the terms in the query, are extracted, and least-squares regression methods and decision-trees are used for learning the ranking functions based on a set of query-document pairs represented as feature vectors with relevance assessment [11, 12, 13]. In a related work, Cooper and coworkers have developed similar approaches and used logistic regression to build the ranking functions and experimented with several retrieval tasks in TREC [8].

With the advance of the World Wide Web, learning ranking functions has emerged as a rapidly growing area of research in the information retrieval, Web search as well as machine learning communities. Earlier works in this active area include: RankSVM based on linear SVM for learning ranking functions [15, 16, 17]. RankNet, developed by a group from Microsoft Research, proposed an optimization approach using an objective function based on Bradley-Terry models for paired comparisons and explored neural networks for learning the ranking functions [5]. RankBoost discussed in [9], using ideas of Adaboost for learning ranking functions based weak learners that can handle preference data.

Most recently, there is an explosion of research in the general area of learning ranking functions and its applications in information retrieval and Web search: machine learning algorithms for a variety of objective functions that more closely match the metrics used in information retrieval and Web search [6, 22, 24]; learning from pairwise preference data using gradient boosting framework [26, 27]; and extending pairwise preference learning to list learning [7]. The workshop *learning to rank for information retrieval* at SIGIR 2007 summarizes many of the recent advances in this field [18].

## 3. TRAINING DATA FORMATS

Before we discuss learning to rank in more detail, we first describe the formats of the training data we will use for the learning process. We represent each query-document pair $(q, d)$ by a feature vector, generically denoted by $x$, and in Section 5 we discuss the details on extraction of query-document features. For query $q$, we have several associated documents $d_1, \ldots, d_n$, and the corresponding relevance judgments either in the form of preference data or labeled data.

First, to describe the setting for the preference data more precisely, let $\mathcal{S}_q$ be a subset of the index set

$$\mathcal{P}_n \equiv \{(i, j),\ \ i, j = 1, \ldots, n\}.$$

We assume $(i, j) \in \mathcal{S}_q$ represents the preference data stating that $d_i$ is more relevant than $d_j$ with respect to the query $q$. Let $x_i$ be the feature vector for $(q, d_i)$, we represent the above preference dat as $x_i \succ x_j, (i, j) \in \mathcal{S}_q$, i.e., document $d_i$ should be ranked higher than document $d_j$ with respect to the query $q$. Second, we will convert labeled data into preference data in the following way: given a query $q$ and two documents $d_x i$ and $d_j$. Let the feature vectors for $(q, d_i)$ and $(q, d_j)$ be $x_i$ and $x_j$, respectively. If $d_i$ has a higher (or better) grade than $d_j$, we include the preference $x_i \succ x_j$ while if $d_j$ has a higher grade than $d_i$, we include the preference $x_j \succ x_i$.

The training data involve a set of queries $\mathcal{Q} = \{q_1, \ldots, q_Q\}$, their associated documents and their relevance judgments. We use $x_1, \ldots, x_N$ to represent the feature vectors for all the query-document pairs in the training set, and denote the associated set of preferences (or converted preferences) as a subset $\mathcal{S} \subset \mathcal{P}_N$. We write the training set concisely as

$$\mathcal{T} = \{\langle x_i, x_j \rangle \mid x_i \succ x_j, (i, j) \in \mathcal{S}\}, \tag{1}$$

which can also be written as $\mathcal{T} = \cup_{i=1}^{Q} \mathcal{S}_{q_i}$. Notice that each preference involves two query-document pairs corresponding to the *same* query.

# 4. MINIMUM EFFORT OPTIMIZATION

Given a query $q$ and the associated $d_1, \ldots, d_n$, a ranking function ranks those documents according to the functions values $h(x_1), \ldots, h(x_n)$, say, $d_i$ should be ranked higher than $d_j$ if $h(x_i) \geq h(x_j)$. For a ranking function $h$, how do learn such a ranking function from the training set $\mathcal{T}$?

## 4.1 Functional iterative methods

Our strategy for learning to rank from $\mathcal{T}$ is based on functional iterative methods. We assume we have a function class $\mathcal{H}$ which is closed under summation. We start with an initial guess $h_0(x) \in \mathcal{H}$, and at each step $m = 1, 2, \ldots$, we compute an update $g_m(x) \in \mathcal{H}$ to obtain the next iterate

$$h_{m+1} = h_m(x) + g_m(x).$$

The basic idea for computing $g_m(x)$ is the following: for the current iterate $h_m(x)$, when considering all the query $q \in \mathcal{Q}$, some of the pairs in $\mathcal{S}$ are consistent, i.e., $h_m(x_i) \geq h_m(x_j)$, and the rest becomes contradicting pairs, i.e, $h_m(x_i) < h_m(x_j)$. We modify the functions values at $x_i$ from $h_m(x_i) \Rightarrow h_m(x_i) + \delta_i^{(m)}, \quad i = 1, \ldots, N$ so that the new set of values $h_m(x_i) + \delta_i^{(m)}$ are consistent with $\mathcal{T}$, i.e., $h_m(x_i) + \delta_i^{(m)} \geq h_m(x_j) + \delta_j^{(m)}, (i,j) \in \mathcal{S}$. We then find $g_m(x) \in \mathcal{H}$ so that $g_m(x_i) \approx \delta_i^{(m)}, \quad i = 1, \ldots, N$ in the least squares sense, for example. This least square fitting can be done by using the gradient boosting trees [10].

## 4.2 Computing updates using isotonic regression

Generally, there are many ways to make the values $h_m(x_i) + \delta_i^{(m)}$ be consistent with $\mathcal{T}$. But large values of $\delta_i^{(m)}$ may give rise to $g_m(x)$ that result in problems in the convergence of the functional iterative algorithm (generally, one needs to control the step size at each iteration in an iterative algorithm in order for the algorithm to converge [4]). Our basic idea is to achieve consistency with $\mathcal{T}$ with as small as possible a set of $\delta_i^{(m)}$.

Recall that the set of preferences are always among documents for the same query, documents are not comparable across queries. Therefore, the computation of $\delta_i^{(m)}$ decouples into several subproblems each for a single query in $\mathcal{Q}$, i.e., each based on one $\mathcal{S}_q$. Without loss of generality, let $x_1, \ldots, x_n$ belong to a single query $q$, and $x_i \succ x_j$ where $(i,j) \in \mathcal{S}_q$ and $\mathcal{S}_q$ is a subset of $\mathcal{P}_n$. Given the current iterate $h_m(x)$, we update $h_m(x_i)$ to $h_m(x_i) + \delta_i^{(m)}$ and compute the $\delta_i^{(m)}$ by solving the following optimization problem,

$$\min_{\delta_i^{(m)}} \sum_{i=1}^{n} (\delta_i^{(m)})^2 \tag{2}$$

subject to

$$h_m(x_i) + \delta_i^{(m)} \geq h_m(x_j) + \delta_j^{(m)} \quad (i,j) \in \mathcal{S}_q.$$

This quadratic programming problem is known as *isotonic regression* in the statistic literature. It is generally used for computing isotonic regression functions. Several special numerical methods have been proposed for solving (2), in particular, when $\mathcal{S}_q \equiv \mathcal{P}_n$, i.e., we have constraints such as

$$h_m(x_1) + \delta_1^{(m)} \geq h_m(x_2) + \delta_2^{(m)} \geq \cdots \geq h_m(x_n) + \delta_n^{(m)},$$

(2) can be solved with computational complexity $O(n)$ using the so-called Pool-Adjacent-Violator (PAV) Algorithm. This is important because for the preference data converted from labeled data, the constraints for each query is of the above form (see section 4.1.1). For general $S_q$, (2) can be solved with computational complexity $O(n^2)$ [2].

## 4.3 Incorporating margins

In case the grade difference for each preference pair is available, we can use it as margin to enhance the constraints in (2). We now have the following optimization problem,

$$\min_{\delta_i^{(m)}} \sum_{i=1}^{n} (\delta_i^{(m)})^2 + \lambda n \zeta^2 \tag{3}$$

subject to

$$h_m(x_i) + \delta_i^{(m)} \geq h_m(x_j) + \delta_j^{(m)} + \Delta G_{ij}(1 - \zeta), \quad (i,j) \in \mathcal{S}_q.$$

$$\zeta \geq 0.$$

Here $\Delta G_{ij}$ is the margin, set to be the grade difference between $x_i$ and $x_j$ when we have the corresponding labels and simply 1.0 otherwise;[1] We also use $\zeta$ as a slack variable allowing softening the constraints imposed by $G_{ij}$; $\lambda$ is the regularization parameter balancing the two requirements in the objective function. We suspect that methods in [2] can be extended to solve (3) with complexity $O(n^2)$, but for the present we treat (3) as a convex quadratic programming problem which can be solved with complexity $O(n^3)$ [4]. Fortunately, in our context $n$ is relatively small and all the quadratic programming problems across the queries can be solved in parallel.

## 4.4 IsoRank

We choose the function class $\mathcal{H}$ to be sums of regression trees which has been widely used in gradient boosting methods [10]. Once $\delta_i^{(m)}$ are computed, we fit a regression tree $g_m(x)$ to minimize $\sum_{i=1}^{N} (g_m(x_i) - \delta_i^{(m)})^2$ [10]. We call the overall algorithm *ranking with isotonic regression* (IsoRank) and summarize it in the following

**Algorithm 1** IsoRank

**Input:** A set of pairwise preference data $\mathcal{T}$ in (1).
**Output:** A ranking function $h_{\max}(x)$.
Start with an initial guess $h_0$, for $m = 1, 2, \ldots, m_{\max}$,

1. Compute $\delta_i^{(m)}, i = 1, \ldots, N$ by solving the isotonic regression problem (3).
2. Fit a regression tree $g_m(x)$ so that $g_m(x_i) \approx \delta_i^{(m)}$.
3. Update $h_{m+1} = h_m(x) + \eta g_m(x)$.

There are mainly three parameters in this algorithm: the number of trees $m_{\max}$, the number of leaf nodes for each regression tree, and the shrinkage factor $\eta$. The number of leaf nodes is related to number of features to use in each regression and is usually set to be a small integer number around 5-20. The shrinkage factor $\eta$ controls the step size along the direction $g_m(x)$ in the iterative step and is set to be small real number around 0.05-0.1. The iteration number $m_{\max}$ is computed by cross-validation.

[1] For example, we can map the set of labels {*Perfect, Excellent, Good, Fair, Bad*} to the set of grades $\{5, 4, 3, 2, 1\}$.

REMARK. A theoretical analysis of the convergence behavior of IsoRank is out of the scope of the current paper. Intuitively, if $g_m(x)$ fits the data $(x_i, \delta_i), i = 1, \ldots, N$ with high accuracy, then $h_{m+1}(x)$ will be consistent with many of the pairs in $\mathcal{T}$. Empirically, we have also observed almost monotonic decreasing of the total number of contradicting pairs on the training set as $m$ increases (see section 4.3).

# 5. EXPERIMENTAL RESULTS

In this section, we describe the results of an experimental study. We carry out several experiments illustrating the properties and effectiveness of IsoRank. We also compare its performance with some existing algorithms for learning ranking functions. We use two data collections for the experiments: one data collection is from a commercial search engine and the other is LETOR, which is a publicly available benchmark data collection used for comparing learning to rank algorithms [18]. We now describe the results for those two data collections.

## 5.1 Experiments on a commercial search engine data collection

As we mentioned before, each query-document pair is represented by a feature vector. For the search engine data, a feature vector is generated for each query-document pair and the features can be grouped into the following three categories:

1) Query features: those dependent on the query only and have constant values across all the documents within that query, for example, the number of terms in the query, and whether or not the query is a navigational query.

2) Document features: those dependent on the document only and have constant values across all the queries, for example, the number of inbound links pointing to the document, and the spam score for the document.

3) Query-document features: those dependent on the relation of the query with respect to the document, for example, the number of times each term in the query appears in the document, and the number of times each term in the query appears in the anchor-texts of the document.

For this search engine data set, we extracted about 200 features in total. The data are generated as follows: a set of queries are first sampled from the search engine query logs, and a certain number of query-document pairs are labeled according to their relevance judged by human editors, we assign a label to each query-document from the set {*perfect, excellent, good, fair, bad*}. In total the data set contains 4,372 queries and 115,278 query-document pairs, among which 768 are perfect, 4,288 are excellent, 30,325 are good, 42,571 are fair, and 37,326 are bad.

We then use the above labeled data to generate a set of preference data as follows: given a query $q$ and two documents $d_x$ and $d_y$. Let the feature vectors for $(q, d_x)$ and $(q, d_y)$ be $x$ and $y$, respectively. If $d_x$ has a higher grade than $d_y$, we include the preference $x \succ y$ while if $d_y$ has a higher grade than $d_x$, we include the preference $y \succ x$. For each query, we consider all pairs of documents within the search results except those with equal grades. This way, we generate around 1.2 million preferences in total. We randomly split the data across queries into *five folds*, use four folds as training the remaining fold as testing data and report results by averaging the metrics over the five folds. When we examine the performance of the algorithms on the training data, we show that using all the data.

**Table 1: Precision at $K\%$ (average over 5 test folds) for IsoRank, QBrank, and RankSVM**

| %K | IsoRank | QBrank | RankSVM |
|---|---|---|---|
| 20% | 0.9852 | 0.9785 | 0.9543 |
| 40% | 0.9224 | 0.9123 | 0.8900 |
| 60% | 0.8519 | 0.8427 | 0.8222 |
| 80% | 0.7927 | 0.7858 | 0.7656 |
| 100% | 0.7413 | 0.7372 | 0.7185 |

### 5.1.1 Evaluation Metrics

Based on a ranking function $h$, the document $x$ is ranked higher than the document $y$ if $h(x) > h(y)$, and we call this order of $x$ and $y$ the predicted preference based on $h$. We use the following metrics to evaluate the performance of a ranking function $h$ with respect to a given set of preferences (those expressed in the training data discussed above) which we considered as the true preferences.

1) Number of contradicting pairs: the number of predicted preferences inconsistent with the true preferences.

2) Percentage of matched pairs: the percentage of predicted preferences consistent with the true preferences.

3) Precision at $K\%$: for two documents $x$ and $y$ (with respect to the same query), it is reasonable to assume that it is easy to compare $x$ and $y$ if $|h(x) - h(y)|$ is large, and $x$ and $y$ should have about the same rank if $h(x)$ is close to $h(y)$. Base on this, we sort all the document pairs $\langle x, y \rangle$ according to $|h(x) - h(y)|$. We call *precision at K%*, the fraction of matched pairs in the top $K\%$ of the sorted list. Precision at 100% is equivalent to % of matched pairs.

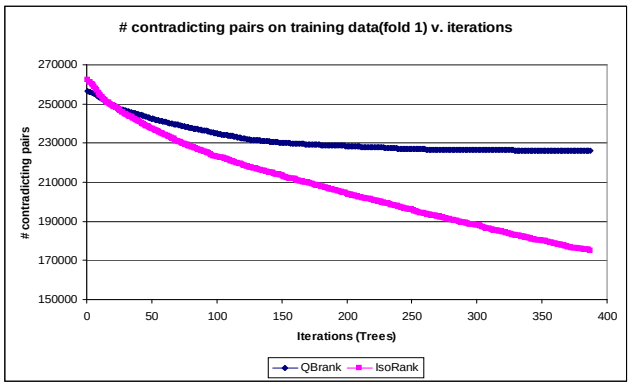
4) Discounted Cumulative Gain (DCG): DCG has been widely used to assess relevance in the context of search engines [14]. For a ranked list of $N$ documents ($N$ is set to be 5 in our experiments), we use the following variation of DCG,

$$\mathrm{DCG}_N = \sum_{i=1}^{N} G_i / \log_2 (i + 1),$$

where $G_i$ represents the weights assigned to the label of the document at position $i$.

### 5.1.2 Experiment Design and Results

We single out QBrank for comparison with our new algorithm IsoRank because it was shown that QBrank outperforms some of the existing state of the art learning to rank algorithms. As is discussed in [26, 27], QBrank uses a squared hinge loss function which is minimized by an iterative gradient boosting method [10]. The key difference



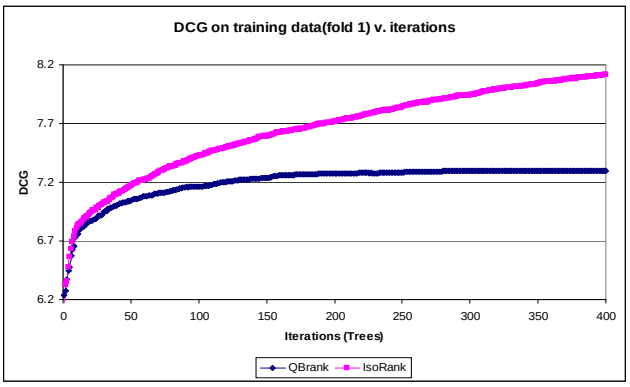
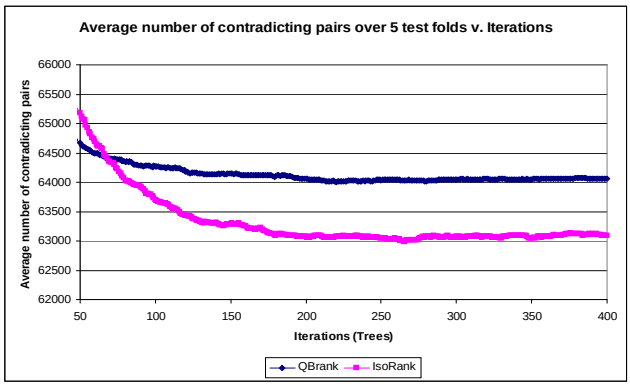




**Figure 1: Number of contradicting pairs over training data (left), DCG over training data (right)**



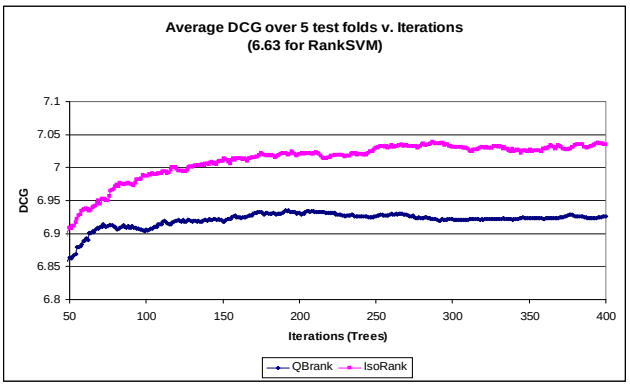




**Figure 2: Average number of contradicting pairs over 5-fold (left), Average DCG over 5-fold (right)**

between IsoRank and QBrank is: IsoRank uses the total number of contradicting pairs as the loss function which gives rise to a more global approach for updating the iterates using the preferences within each queries. QBRank uses gradient descent, and the computation of the gradients relies on the individual preferences. A url can appear in multiple preferences, in this case the average gradient was used for each distinct url. As a baseline, we also include the results for RankSVM. The main questions we want to address are: What is the convergence behavior of IsoRank? Is the total number of contradicting pairs on the training set generally monotonically decreasing as the iteration number of IsoRank increases? How do the two methods IsoRank and QBrank compare in terms of the three metrics discussed in the previous subsection?

To answer the first question, Figure 1 shows the total number of contradicting pairs and $DCG_5$ with respect to number of iterations (trees) for both QBrank and IsoRank on the training data. The parameters for base regression trees are set to be 20 leaf nodes and we 0.1 as the shrinkage factor. Both methods have a clear convergence trend: $DCG_5$ increases and the total number of contradicting pairs decreases as the iteration number increases. Figure 1 also demonstrates that IsoRank has much faster and better convergence than QBrank, this should not come as a surprise because each step of IsoRank is more expensive than QBrank. On the training set, QBrank flattened out after about 200 iterations, but IsoRank can still go on for some more iterations will be terminated around 250 iterations using crossvlaidation. For performance on the test data, we plot the five fold average number of the total number of contradicting pairs and the average $DCG_5$ against number of iterations in Figure 2. IsoRank is better than QBrank in term of both metrics.[2] As a baseline comparison, we did the same experiments for RankSVM and the average $DCG_5$ over the same five folds is 6.63, which is worse than both QBrank and IsoRank. Table 2 presents average precision at $K\%$ over the five test folds for IsoRank, QBrank, and RankSVM. As is illustrated in Figure 3, IsoRank consistently outperform both QBrank and RankSVM over all the five folds.

## 5.2 Experiments on LETOR data

LETOR was derived from the existing data sets widely used in IR, namely, OHSUMED and TREC data sets. The data contain queries, the contents of the retrieved documents, and human judgments on the relevance of the documents with respect to the queries. Various features have been extracted including both conventional features, such as term frequency, inverse document frequency, BM25 scores, and language models for IR, and features proposed recently such as HostRank, feature propagation, and topical PageRank. The package of LETOR contains the extracted features, queries, and relevance judgments. The results of several

[2]Notice that a 1% $DCG_5$ gain is considered significant on this data set for commercial search engines.

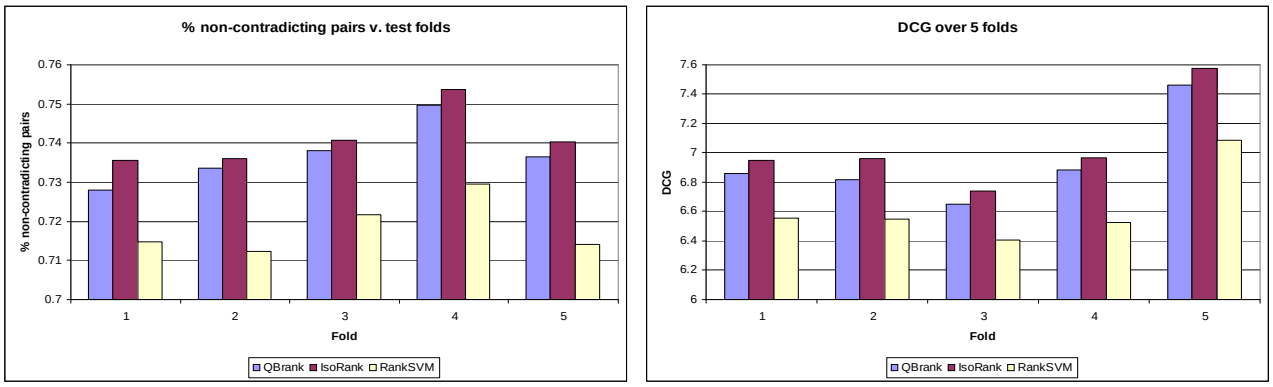



**Figure 3: Percentage of matched pairs for 5-fold cross validation (left), DCG for 5-fold cross validation (right)**

**Table 2: NDCG, MAP, and Precision at position $n$ on OHSUMED data(average over 5 folds)**

| Methods | ndcg@1 | ndcg@2 | ndcg@3 | ndcg@4 | ndcg5 | P@1 | P@2 | P@3 | P@4 | P@5 | MAP |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RankBoost | 0.498 | 0.483 | 0.473 | 0.461 | 0.450 | 0.605 | 0.595 | 0.586 | 0.562 | 0.545 | 0.440 |
| RankSVM | 0.495 | 0.476 | 0.465 | 0.459 | 0.458 | 0.634 | 0.619 | 0.592 | 0.579 | 0.577 | 0.447 |
| FRank-c4.2 | 0.545 | 0.510 | 0.499 | 0.478 | 0.469 | 0.671 | 0.619 | 0.617 | 0.581 | 0.560 | 0.446 |
| ListNet | 0.523 | 0.497 | 0.478 | 0.468 | 0.466 | 0.643 | 0.629 | 0.602 | 0.577 | 0.575 | 0.450 |
| AdaRank.MAP | 0.542 | 0.496 | 0.480 | 0.471 | 0.455 | 0.661 | 0.605 | 0.583 | 0.567 | 0.537 | 0.442 |
| AdaRank.NDCG | 0.514 | 0.474 | 0.462 | 0.456 | 0.442 | 0.633 | 0.605 | 0.570 | 0.562 | 0.533 | 0.442 |
| MHR-BC | 0.552 | 0.490 | 0.485 | 0.480 | 0.467 | 0.652 | 0.615 | 0.612 | 0.591 | 0.566 | 0.440 |
| **QBRank** | 0.563 | 0.536 | 0.483 | 0.471 | 0.463 | 0.708 | 0.676 | 0.624 | 0.589 | 0.570 | 0.452 |
| **IsoRank** | 0.565 | 0.556 | 0.520 | 0.505 | 0.488 | 0.653 | 0.676 | 0.643 | 0.617 | 0.588 | 0.457 |

state-of-the-arts learning to rank algorithms, e.g., RankSVM, RankBoost, AdaRank, Multiple hyperline ranker, FRank, and ListNet, on the data sets are also included in that package.

### 5.2.1 *Letor data collection*

**OHSUMED.** The OHSUMED data set is a subset of the MEDLINE database, which is popular in the information retrieval community. This data set contains 106 queries. The documents are manually labeled with absolute relevance judgements with respect to the queries. There are three levels of relevance judgments in the data set: *definitely relevant*, *possibly relevant* and *not relevant*. Each query-document pair is represented by a 25-dimensional feature vector. The total number of query-document pairs is 16,140, among which 11,303 are not relevant, 2585 are possibly relevant, and 2252 are definitely relevant.

**TREC2003.** This data set is extracted from the topic distillation task of TREC2003[3]. The goal of the topic distillation task is to find good websites about the query topic. There are 50 queries in this data set. For each query, the human assessors decide whether a web page is an relevant result for the query, so two levels of relevance are used: *relevant* and *not relevant*. The documents in the TREC2003 data set are crawled from the `.gov` websites, so the features extracted by link analysis are also used to represent the query-document pair in addition to the content features used in the OHSUMED data set. The total number of features are 44 and total number of query-document pairs is 49,171: 516 relevant examples and 48,655 non-relevant examples.

**TREC2004.** This data set is extracted from the data set of the topic distillation task of TREC2004, so it is very similar to the TREC2003 data set. This data set contains 75 queries and 74,170 documents (444 are relevant and 73,726 non-relevant) with 44 features.

Since TREC data have only two distinct labels with very skewed distribution, the ranking on that data is more like a binary classification problem on imbalanced data, and thus less interesting than commercial search egnine data and OHSUMED data from a ranking point of view.

### 5.2.2 *Evaluation Metrics*

To be consistent with Letor evaluation we use the three performance metrics: Precision, Mean average precision and Normalized Discount Cumulative Gain. All these evaluation measures are widely used for comparing information retrieval systems. In the case of multiple levels of judgements, the Normalized Discount Cumulative Gain (NDCG) is used [14]. The NDCG value of a ranking list is calculated by the following equation:

$$\text{NDCG@n} = Z_n \sum_{i=1}^{n} (2^{r_i} - 1) / \log(i+1)$$

[3]http://trec.nist.gov/

**Table 3: NDCG, MAP, and Precision at position $n$ on TD2003 data(average over 5 folds)**

| Methods | ndcg@1 | ndcg@2 | ndcg@3 | ndcg@4 | ndcg@5 | P@1 | P@2 | P@3 | P@4 | P@5 | MAP |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RankBoost | 0.260 | 0.280 | 0.270 | 0.272 | 0.279 | 0.260 | 0.270 | 0.240 | 0.230 | 0.220 | 0.212 |
| RankSVM | 0.420 | 0.370 | 0.379 | 0.363 | 0.347 | 0.420 | 0.350 | 0.340 | 0.300 | 0.264 | 0.256 |
| FRank-c4.2 | 0.440 | 0.390 | 0.369 | 0.342 | 0.330 | 0.440 | 0.370 | 0.320 | 0.260 | 0.232 | 0.245 |
| ListNet | 0.460 | 0.430 | 0.408 | 0.386 | 0.382 | 0.460 | 0.420 | 0.360 | 0.310 | 0.292 | 0.273 |
| AdaRank.MAP | 0.420 | 0.320 | 0.291 | 0.268 | 0.242 | 0.420 | 0.310 | 0.267 | 0.230 | 0.188 | 0.137 |
| AdaRank.NDCG | 0.520 | 0.410 | 0.374 | 0.347 | 0.326 | 0.520 | 0.400 | 0.347 | 0.305 | 0.268 | 0.185 |
| **QBRank** | 0.540 | 0.460 | 0.418 | 0.384 | 0.360 | 0.540 | 0.460 | 0.393 | 0.330 | 0.284 | 0.231 |
| **IsoRank** | 0.520 | 0.450 | 0.421 | 0.392 | 0.367 | 0.520 | 0.450 | 0.373 | 0.325 | 0.288 | 0.248 |

**Table 4: NDCG, MAP, and Precision at position $n$ on TD2004 data(average over 5 folds)**

| Methods | ndcg@1 | ndcg@2 | ndcg@3 | ndcg@4 | ndcg@5 | P@1 | P@2 | P@3 | P@4 | P@5 | MAP |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RankBoost | 0.480 | 0.473 | 0.464 | 0.439 | 0.437 | 0.480 | 0.447 | 0.404 | 0.347 | 0.323 | 0.384 |
| RankSVM | 0.440 | 0.433 | 0.409 | 0.406 | 0.393 | 0.440 | 0.407 | 0.351 | 0.327 | 0.291 | 0.350 |
| FRank-c4.2 | 0.440 | 0.467 | 0.448 | 0.435 | 0.436 | 0.440 | 0.433 | 0.387 | 0.340 | 0.323 | 0.381 |
| ListNet | 0.440 | 0.427 | 0.437 | 0.422 | 0.421 | 0.440 | 0.407 | 0.400 | 0.357 | 0.331 | 0.372 |
| AdaRank.MAP | 0.413 | 0.393 | 0.402 | 0.387 | 0.393 | 0.413 | 0.353 | 0.342 | 0.300 | 0.293 | 0.331 |
| AdaRank.NDCG | 0.360 | 0.360 | 0.384 | 0.377 | 0.377 | 0.360 | 0.320 | 0.329 | 0.300 | 0.280 | 0.299 |
| **QBRank** | 0.400 | 0.373 | 0.372 | 0.365 | 0.359 | 0.400 | 0.340 | 0.311 | 0.287 | 0.256 | 0.294 |
| **IsoRank** | 0.453 | 0.440 | 0.425 | 0.407 | 0.396 | 0.453 | 0.413 | 0.360 | 0.317 | 0.283 | 0.336 |

where $r_i$ is the grade assigned to the $i$-th document of the ranking list. In our experiments, $r_i$ takes value of 0, 1 and 2 in OHSUMED data set for not, possibly and definitely relevant documents respectively. For data sets with binary judgments, such as TREC2003 and TREC2004 data set, $r_i$ is set to 1 if the document is relevant and 0 otherwise. The constant $Z_n$ is chosen so that the perfect ranking gives an NDCG value of 1.

We apply QBrank and IsoRank to LETOR data and compare them with other state-of-the-arts learning to rank algorithms reported in LETOR package. Since this data are significantly different from the commercial search engine data in term of features, grades, etc, we re-tune the base regression tree parameters on their corresponding validation data. Unlike on the commercial search engine data where we plot DCGs for different methods against iterations (or number of trees), we tune the number of trees as well for QBrank and IsoRank on the validation set. For IsoRank, we simply set the regularization parameter $\lambda$ in (3) to be 10 without much tuning.

Table 2 presents experimental results of the nine methods on OHSUMED data, which, as mentioned earlier, are more interesting than TREC data from a ranking point of view. From the table, one can observe that 1) IsoRank outperforms QBRank at almost all metrics and 2) both are better than the remaining seven methods. The first observation demonstrates the effectiveness of "global view" and "minimum effort" featured in IsoRank while the second indicates boosting tree approaches work well for learning to ranking. It is also interesting to see that some methods especially RankBoost performs quite differently on the two TREC data sets as shown in Table 3 and 4 respectively: RankBoost is the worst in TD2003, but the best in TD2004. This could be due to the characteristics of the data (only two distinct labels with highly skewed distributions) and the method itself (ability to learn from imperfect data). In contrast, IsoRank seems more robust and performs reasonably well on both TREC data sets.

# 6. CONCLUSIONS AND FUTURE WORK

In this paper we propose a new method for learning ranking functions for information retrieval and Web search: we use the total number of contradicting pairs as the objective function and develop a novel functional iterative method to minimize the objective function. It turns out that the computation of the updates in the iterative method can incorporate all the preferences within a query using Isotonic regression. This more global approach result in improvement in the performance of the learned ranking functions. We could also include in IsoRank tied pairs, e.g. pairs of urls $\langle x_i, x_j \rangle$ with same grades. We denote the set of $(i, j)$ in those tied pairs as $V_q$. Accordingly, we would have the following optimization problem,

$$\min_{\delta_i^{(m)}} \sum_{i=1}^{n} (\delta_i^{(m)})^2 + \lambda_1 n \zeta_1^2 + \lambda_2 n \zeta_2^2 \tag{4}$$

subject to

$$h_m(x_i) + \delta_i^{(m)} \geq h_m(x_j) + \delta_j^{(m)} + \Delta G_{ij}(1 - \zeta_1), \quad (i, j) \in \mathcal{S}_q.$$

$$|h_m(x_i) + \delta_i^{(m)} - h_m(x_j) - \delta_j^{(m)}| \leq \zeta_2, \quad (i, j) \in V_q.$$

$$\zeta_1, \quad \zeta_2 \geq 0.$$

As future research directions, we plan to provide more rigorous analysis of IsoRank, characterize theoretical as well

as computational properties of the computed updates and their relations to gradient descent directions. We will also seek to provide better understanding on the characteristics of the data sets that influence the performance of various existing methods for learning ranking functions, those can include the levels of relevance judgment, the heterogeneity of the features and the noise levels of the preference data.

## 7. REFERENCES


[1] R. Atterer, M. Wunk, and A. Schmidt. Knowing the user's every move: user activity tracking for website usability evaluation and implicit interaction. *Proceedings of the 15th International Conference on World Wide Web*, 203-212, 2006.

[2] O. Burdakov, A. Grimvall and O. Sysoev. Data preordering in generalized PAV algorithm for monotonic regression. *Journal of Computational Mathematics*, 24:771-790, 2006.

[3] A. Berger. *Statistical machine learning for information retrieval*. Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, 2001.

[4] D. Bertsekas. *Nonlinear programming*. Athena Scientific, second edition, 1999.

[5] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, Ê and G. Hullender. Learning to rank using gradient descent. *Proceedings of international conference on Machine learning*, 89–96, 2005.

[6] C. Burges, R. Ragno and Q. Le. Learning to rank with nonsmooth cost functions. *Advances in Neural Information Processing Systems 19*, MIT Press, Cambridge, MA, 2007.

[7] Z. Cao, T. Qin, T-Y Liu, M-F Tsai and H. Li. Learning to rank: from pairwise to listwise approach. *Proceedings of international conference on Machine learning*, 2007.

[8] W. Cooper, F. Gey and A. Chen. Probabilistic retrieval in the TIPSTER collections: an application of staged logistic regression. *Proceedings of TREC*, 73-88, 1992.

[9] Y. Freund, R. Iyer, R. Schapire and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969, 2003.

[10] J. Friedman. Greedy function approximation: a gradient boosting machine. *Ann. Statist.*, 29:1189 - 1232, 2001.

[11] N. Fuhr. Optimum polynomial retrieval functions based on probability ranking principle. *ACM Transactions on Information Systems*, 7:183-204, 1989.

[12] N. Fuhr and C. Buckley. A probabilistic learning approach for document indexing. *ACM Transactions on Information Systems*, 9:223-248, 1991.

[13] N. Fuhr and U. Pfeifer. Probabilistic information retrieval as a combination of abstraction, inductive learning, and probablistic assumptions. *ACM Transactions on Information Systems*, 12:92-115, 1994.

[14] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20:422-446, 2002.

[15] T. Joachims. Optimizing search engines using clickthrough data. *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining*, 2002.

[16] T. Joachims. Evaluating retrieval performance using clickthrough data. *Proceedings of the SIGIR Workshop on Mathematical/Formal Methods in Information Retrieval*, 2002.

[17] T. Joachims, L. Granka, B. Pang, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2005.

[18] Learning to rank in information retrieval. SIGIR Workshop, 2007.

[19] J. Ponte and W. Croft. A language modeling approach to information retrieval. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval*, 1998.

[20] G. Salton. *Automatic Text Processing*. Addison Wesley, Reading, MA, 1989.

[21] F. Radlinski and T. Joachims. Query chains: learning to rank from implicit feedback. *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)* , 2005.

[22] M-F. Tsai, T-Y Liu, T. Qin, H-H Chen and W-Y Ma. FRank: a ranking method with fidelity loss. *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007.

[23] J. Xu and H. Li. A boosting algorithm for information retrieval. *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007.

[24] Y. Yue, T. Finley, F. Radlinksi and T. Joachims. A Support vector method for optimizing average precision. *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007.

[25] H. Zha, Z. Zheng, H. Fu and G. Sun. Incorporating query difference for learning retrieval functions in *Proceedings of the 15th ACM Conference on Information and Knowledge Management*, 2006.

[26] Z. Zheng, H. Zha, K. Chen and G. Sun. A Regression framework for learning ranking functions using relative relevance judgments. *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007.

[27] Z. Zheng, H. Zha, T. Zhang, O. Chapelle, K. Chen and G. Sun. A General boosting method and its application to learning ranking functions for Web search. In *Advances in Neural Information Processing Systems 20*, MIT Press, Cambridge, MA, 2008.