A Regression Framework for Learning Ranking Functions Using Relative Relevance Judgments

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

Hongyuan Zha
College of Computing
Georgia Institute of Technology
Atlanta, GA 30032
zha@cc.gatech.edu

Keke Chen, Gordon Sun
Yahoo! Inc.
701 First Avenue
Sunnyvale, CA 94089
kchen,gzsun@yahoo-inc.com

ABSTRACT

Effective ranking functions are an essential part of commercial search engines. We focus on developing a regression framework for learning ranking functions for improving relevance of search engines serving diverse streams of user queries. We explore supervised learning methodology from machine learning, and we distinguish two types of relevance judgments used as the training data: 1) absolute relevance judgments arising from explicit labeling of search results; and 2) relative relevance judgments extracted from user clickthroughs of search results or converted from the absolute relevance judgments. We propose a novel optimization framework emphasizing the use of relative relevance judgments. The main contribution is the development of an algorithm based on regression that can be applied to objective functions involving preference data, i.e., data indicating that a document is more relevant than another with respect to a query. Experimental results are carried out using data sets obtained from a commercial search engine. Our results show significant improvements of our proposed methods over some existing methods.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Search and Retrieval—*Retrieval functions*; H.4.m [**Information Systems**]: Miscellaneous—*Machine learning*

General Terms

Algorithms, Experimentation, Theory

Keywords

ranking function, machine learning, absolute relevance judgment, relative relevance judgment, preferences, clickthroughs, functional gradient descent, regression, gradient boosting

SIGIR'07, July 23–27, 2007, Amsterdam, The Netherlands.

1. INTRODUCTION

Research and experiments in information retrieval have produced many fundamental methodologies and algorithms enabling the technological advances in the current commercial search engines. Ranking functions are at the core of search engines and they directly influence the relevance of the search results and users' search experience. In the past, many models and methods for designing ranking functions have been proposed, including vector space models, probabilistic models and the more recently developed language modeling-based methodology [17, 16, 2]. In particular, learning ranking functions within the framework of machine learning have attracted much interest long before the recent advances of Web search [10, 6, 5, 11, 22, 19]. The trend continues to this day and several methods have been proposed incorporating many of the recent advances in machine learning such as SVM and gradient boosting [7, 4, 19].

Machine learning approaches for learning ranking functions, in particular, the supervised learning approaches, entails the generation of training data, in the form of labeled data explicitly constructed from relevance assessment by human editors. As an example, labels or grades such as perfect, good, or bad, can be assigned to documents with respect to a query indicating the degree of relevance of documents. With labels associated with query-document pairs, we are using the absolute relevance framework where judgments are made with respect to whether a document is or is not relevant to a query. Acquiring large quantities of absolute relevance judgments, however, can be very costly because it is necessary to cover a diverse set of queries in the context of Web search. An additional issue is the reliability and variability of absolute relevance judgments.

One possibility to alleviate this problem is to make use of the vast amount of data recording user interactions with the search results, in particular, user clickthroughs data [1]. Each individual user click may not be very reliable, but the aggregation of a great number of user clicks can provide a very powerful indicator of relevance preference. In this regard, Joachims and his coworkers have developed methods for extracting relative relevance judgments from user clickthroughs data [13, 14, 20, 15, 21]. Particularly, the relative relevance judgments are in the form of whether a document is more relevant than other documents with respect to a query. The benefit for using relative relevance judgments are the potential unlimited supplies of user click-

throughs data and the timeliness of user clickthroughs data for capturing user searching behaviors and preferences. The drawback for using relative relevance judgments are that user clickthroughs data tend to be quite noisy, especially we also need to deal with fraudulent clicks. Although there have been some research on how to extract relative relevance judgments from user clickthroughs data, much research is still needed to make the extraction process more effective.

Once relative relevance judgments are extracted from user clickthroughs data, the next question is how to use them for the purpose of learning a ranking function. This falls under the general framework of learning ranking functions from preference data and several algorithms have been proposed in the past: Joachims and his coworkers used RankSVM based on linear SVM for learning ranking functions. To incorporate nonlinear interactions of features in RankSVM, either more complicated features need to be devised or some kind of kernels used [13, 14, 15]. 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 [4]. The closest to our proposed method is RankBoost discussed in [8], using ideas of Adaboost for learning ranking functions for preference data. The choice for selecting weak learners for RankBoost as discussed in [8] is very limited and is less flexible to deal with the complicated features used in Web search context.

The main contribution of our work is the development of a learning framework for preference data using regression as the basic ingredient. For example, the ranking functions are represented as a combination of regression trees when we use gradient boosting for regression [9]. More interestingly, our experimental results also show that even with absolute relevant judgments, it is more advantageous to first convert them into preference data and apply our proposed methods than to treat the ranking problem with absolute relevant judgments as a regression problem.

The rest of the paper is organized as follows: section 2 develops the main algorithmic contribution of the paper; we start with a brief review of the basic idea of gradient descent in function spaces [9]. We then propose an objective function, the optimization of which will lead to the construction of the ranking function. We apply functional gradient descent methodology to the objective function and transform the problem of learning ranking functions as a sequence of problems of learning regression functions. For concreteness, we use gradient boosting regression as an illustration of the general methodology. In section 3, we present a detailed experimental study using data from a commercial search engine. In the last section, we make some concluding remarks and also point out several directions for future research.

2. A REGRESSION FRAMEWORK FOR LEARNING FROM PREFERENCE DATA

Our basic premise is that ranking and regression are fundamentally different problems, but regression can be used to solve ranking problems using preference data. Our main contribution is a framework for solving ranking problems with regression using relative judgments and the regression methods used can be be chosen to tailor to the specific applications. For concreteness, we will discuss the framework using gradient descent, and we start with a brief introduction of gradient descent in function spaces [9]. We then propose a new objective function for learning ranking functions using preference data and develop an algorithm that adapts functional gradient descent for optimizing the proposed objective function.

2.1 Functional gradient descent

We first give a brief discussion of gradient descent for unconstrained optimization of a multivariate function [3]. To this end, suppose we want to solve $\min_{x\in\mathcal{R}^d} F(x)$, where $F(x)$ is a d-variable function. The idea of gradient descent is to start with an initial guess x_0 of a minimizer, and at each step compute the gradient of the objective function F at the current iterate x_k, say $\nabla F(x_k)$, and use the negative gradient as the search direction to obtain the next iterate

$$x_{k+1} = x_k - \alpha_k \nabla F(x_k),$$

where α_k is the step size which can be chosen, for example, by a line-search.

In the context of regression, we are given a training set $\{(x_i, g_i)\}_{i=1}^N$, and we seek to find a function h such that $g_i \approx h(x_i), i = 1, \ldots, N$. For simplicity we use the square loss function, i.e., we measure the discrepancy between g_i and $h(x_i)$ by $(g_i - h(x_i))^2$. Then we need to find a function $h(x)$ to solve the following minimization problem,

$$\min_{h\in\mathcal{H}} \; L(h) \equiv \min_{h\in\mathcal{H}} \; \frac{1}{2}\sum_{i=1}^{N}(g_i - h(x_i))^2,$$

where $\mathcal{H}$ is a pre-defined function class such as the class of polynomials not to exceed certain degree. We can apply gradient descent in function space to minimize the functional $L(h)$, i.e., compute the gradient of $L(h)$ with respect to h at the current iterate $h_k(x)$ and form the next iterate as

$$h_{k+1}(x) = h_k(x) - \alpha_k \nabla L(h_k(x)).$$

The problem is that, we can not compute $\nabla L(h_k(x))$ at all x, but rather we can only compute it at a finite sample,

$$\nabla L(h_k(x_i)) = -(g_i - h_k(x_i)), \quad i = 1, \ldots, N.$$

The crucial idea of functional gradient descent is to find a function that interpolates/approximates the above sample values and therefore obtain an approximation of the negative gradient $-\nabla L(h_k(x))$ to form the next iterate. As an illustration, we explain the details of the algorithm when the interpolation/approximation is done by fitting a regression tree to the sample values, as is easily seen other regression methods can also be used here [9]. We summarize the above in the following and label it as GBT (Gradient Boosting Trees).

ALGORITHM. (Gradient Boosting Trees [9])

1. Initialize $h_0(x) = \sum_{i=1}^N g_i/N$.
2. For $k = 1, \ldots, M$: (number of trees in gradient boosting)

   (a) For $i = 1, \ldots, N$, compute the negative gradient

   $$r_{ik} = g_i - h_{k-1}(x_i)$$

   (b) Fit a regression tree to $\{r_{ik}\}_{i=1,\ldots,N}$ giving terminal regions $R_{jk}, j = 1, \ldots, J_k$.

(c) For $j = 1, \ldots, J_k$, compute

$$\gamma_{jk} = \sum_{x_i \in R_{jk}} (g_i - h_{k-1}(x_i))/|\{i : \ x_i \in R_{jk}\}|,$$

average of the residual in each terminal region.

(d) Update

$$h_k(x) = h_{k-1}(x) + \eta(\sum_{j=1}^{J_k} \gamma_{jk} I(x \in R_{jk})),$$

where η is the *shrinkage factor*, and $I(\cdot)$ is the indicator function.

There are two parameters M, the number of regression trees and η, the shrinkage factor that need to be chosen by the user. In general, we use cross-validation for choosing the two parameters.

2.2 Ranking with relative judgments

As we mentioned before, the relative relevance judgments are in the form of whether a document is more relevant than other documents with respect to a query. We encode this information as follows: given the feature vectors for two query-document pairs x and y (see Section 3 for details on extraction of query-document features), we use $x \succ y$ to mean that x is preferred over y, i.e., x should be ranked higher than y. Simply put, this means that the document represented by x is considered more relevant than that represented by y with respect to the query in question.

We denote the set of available preferences based on the relative relevance judgments as

$$\mathcal{S} = \{\langle x_i, y_i \rangle \mid x_i \succ y_i, \ i = 1, \ldots, N\}.$$

We formulate the problem of learning ranking functions as computing a ranking function $h \in \mathcal{H}$, $\mathcal{H}$ a given function class, such that h match the set of preferences, i.e., $h(x_i) \geq h(y_i)$, if $x_i \succ y_i, i = 1, \ldots, N$, as much as possible. We propose to use the following objective function to measure the risk of a ranking function h,

$$\mathcal{R}(h) = \frac{1}{2} \sum_{i=1}^{N} (\max\{0, h(y_i) - h(x_i)\})^2, \tag{1}$$

the motivation is that if for the pair $\langle x_i, y_i \rangle$, h matches the given preference, i.e., $h(x_i) \geq h(y_i)$, then h incurs no cost on the pair, otherwise the cost is given by $(h(y_i) - h(x_i))^2$. Direct optimization of the above can be difficult, the basic idea of our regression framework is to fix either one of the values $h(x_i)$ or $h(y_i)$, e.g., replace either one of the function values by its current predicted value, and solve the problem by way of regression.

REMARK. To avoid obtaining an optimal h which is constant, we actually need to optimize, for $0 < \tau \leq 1$,

$$\mathcal{R}(h, \tau) = \frac{1}{2} \sum_{i=1}^{N} (\max\{0, h(y_i) - h(x_i) + \tau\})^2 - \lambda \tau^2,$$

Our implementation in the sequel corresponds to setting τ to be a fixed constant.

To this end, we use the idea of functional gradient descent as reviewed in the previous section. We consider

$$h(x_i), \ \ h(y_i), \quad i = 1, \ldots, N,$$

as the unknowns, and compute the gradient of $\mathcal{R}(h)$ with respect to those unknowns. The components of the *negative* gradient corresponding to $h(x_i)$ and $h(y_i)$, respectively, are

$$\max\{0, h(y_i) - h(x_i)\}, \quad -\max\{0, h(y_i) - h(x_i)\}.$$

Both of the above equal to zero when h matches the pair $\langle x_i, y_i \rangle$, and therefore, in this case no modification is needed for the components corresponding to $h(x_i)$ or $h(y_i)$. On the other hand, if h does not match the pair $\langle x_i, y_i \rangle$, the components of the gradient are

$$h(y_i) - h(x_i), \quad h(x_i) - h(y_i).$$

The above tells us how to modify the difference of function values, to know how to modify the function itself we need to translate those gradient components into modification to h. We adopt the following simple approach: we set the target value for x_i as $h(y_i) + \tau$ and that for y_i as $h(x_i) - \tau$ for some fixed τ. Then, we obtain the following set of data that need to be fitted at each iteration,

$$\{(x_i, h(y_i) + \tau), (y_i, h(x_i) - \tau)\}, \tag{2}$$

where h does not match pair $\langle x_i, y_i \rangle$.

When some feature vectors x_i or y_i can appear more than once in $\mathcal{S}$, there will be several components of the negative gradient of $\mathcal{R}(h)$ that will involve x_i or y_i. When translating the gradient components to modification of h, we may end up with inconsistent requirements. One approach will be to compute an average taking into account of all the requirements. This is a local approach using information in the training data related to the feature vectors in question. A better alternative approach is to add all the different and potentially inconsistent requirements in the training set, and let the regression methods such as GBT to handle the inconsistency using more global information based on all the training data. We summarize the algorithm, again using GBT for regression as an illustration, as follows which we label as GBrank.

ALGORITHM. (GBrank) [1]
Start with an initial guess h_0, for $k = 1, 2, \ldots,$

1) using h_{k-1} as the current approximation of h, we separate $\mathcal{S}$ into two disjoint sets,

$$\mathcal{S}^+ = \{\langle x_i, y_i \rangle \in \mathcal{S} \mid h_{k-1}(x_i) \geq h_{k-1}(y_i) + \tau\}$$

and

$$\mathcal{S}^- = \{\langle x_i, y_i \rangle \in \mathcal{S} \mid h_{k-1}(x_i) < h_{k-1}(y_i) + \tau\};$$

2) fitting a regression function $g_k(x)$ using GBT and the following training data

$$\{(x_i, h_{k-1}(y_i) + \tau), \ \ (y_i, h_{k-1}(x_i) - \tau) \mid (x_i, y_i) \in \mathcal{S}^-\};$$

3) forming (with normalization of the range of h_k)

$$h_k(x) = \frac{k h_{k-1}(x) + \eta g_k(x)}{k+1},$$

where η is a shrinkage factor.

REMARK. We want to point out that the above framework is generic in the sense that it can use any application-specific regression method for learning the regression function $g_k(x)$.

[1] If the preferences $\langle x_i, y_i \rangle$ are converted from absolute relevance judgments, we multiple τ by the absolute value of grade difference between x_i and y_i

3. EXPERIMENTAL RESULTS

We carried out several experiments illustrating the properties and effectiveness of GBrank. We also compared its performance with some existing algorithms such as GBT and RankSVM.

3.1 Data Collection

We first describe how the data used in the experiments are collected.

3.1.1 Feature vectors

As we mentioned before, each query-document pair is represented by a feature vector. For query-document pair (q, d), a feature vector $x = [x^Q, x^D, x^{QD}]$ is generated and the features generally fall into the following three categories:

- Query-feature vector x^Q which comprises features dependent on the query q only and have constant values across all the documents $d \in \mathcal{D}$, for example, the number of terms in the query, whether or not the query is a person name, etc.
- Document-feature vector x^D which comprises features dependent on the document d only and have constant values across all the queries $q \in \mathcal{Q}$, for example, the number of inbound links pointing to the document, the amount of anchor-texts in bytes for the document, and the language identity of the document, etc.
- Query-document feature vector x^{QD} which comprises features dependent on the relation of the query q with respect to the document d, for example, the number of times each term in the query q appears in the document d, the number of times each term in the query q appears in the anchor-texts of the document d, etc.

The preference data for training are extracted from the follow two sources: absolute relevance judgments arising from editorial labeling, and relative relevance judgments extracted from user clickthroughs data.

3.1.2 Preference data from labeled data

A set of queries are sampled from query logs, and a certain number of query-document pairs are labeled according to their relevance judged by human editors. A 0-4 grade is assigned to each query-document pair based on the degree of relevance (perfect match, excellent match, etc), and the numerical grades are also used as the target values for GBT regression. We use a data set from a commercial search engine which contains 4,372 queries and 115,278 query-document pairs.

We 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.

As we will see later, the labeled data not only provide us with preference data, they also allow us to compare GBrank based on converted preference data and GBT regression using the labeled data.

3.1.3 Preference data from clickthroughs data

We also examined a certain amount of clickthroughs data and extracted a set of preference data as follows. For a query q, we consider two documents d_1 and d_2 among the top 10 results from Yahoo! web search. Assume that in the clickthroughs data, d_1 has c_1 clicks out of n_1 impressions, and d_2 has c_2 clicks out of n_2 impressions. We want to consider document pairs d_1 and d_2 for which either d_1 or d_2 is significantly better than the other in terms of clickthroughs rate. To this end, we assume that clicks in user sessions obey binomial distribution. Denote the binomial distribution by

$$B(k; n, p) = \left(\begin{array}{c} n \\ k \end{array} \right) p^k (1-p)^{n-k}$$

We apply likelihood ratio test (LRT) and compute,

$$\lambda \equiv \frac{B(c_1 + c_2; n_1 + n_2, (c_1 + c_2)/(n_1 + n_2))}{B(c_1; n_1, c_1/n_1) B(c_2; n_2, c_2/n_2)} \rightarrow -\chi^2.$$

We consider a pair d_1 and d_2 when the above is greater than a threshold, and we say the pair is significant. Among the significant pairs, we apply rules similar to those in [15] such as *Skip-Above* to extract preference data. In total we extracted 20,948 preferences.

3.2 Evaluation Metrics

The output of GBrank is a ranking function h which is used to rank the documents x according to $h(x)$. Therefore, document x is ranked higher than y by the ranking function h if $h(x) > h(y)$, and we call this the predicted preference. We propose the following three metrics to evaluate the performance of a ranking function with respect to a given set of preferences which we considered as the true preferences.

- Number of contradicting pairs: for a pair of documents if the predicted preference is different from the true preference, the pair is a contradicting pair.
- 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 non-contradicting pairs in the top $K\%$ of the sorted list.[2]
- Discounted Cumulative Gain (DCG): DCG has been widely used to assess relevance in the context of search engines [12]. For a ranked list of N documents (N is set to be 5 in our experiments), we use the following variation of DCG,

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

where G_i represents the weights assigned to the label of the document at position i, e.g. 10 for Perfect match, 7 for Excellent match, 3 for Good match, etc. Higher degree of relevance corresponds to higher value of the weight. We will use the symbol *dcg* to indicate the average of this value over a set of testing queries in our

[2]Notice that precision at 100% corresponds to the percentage of contradicting pairs.

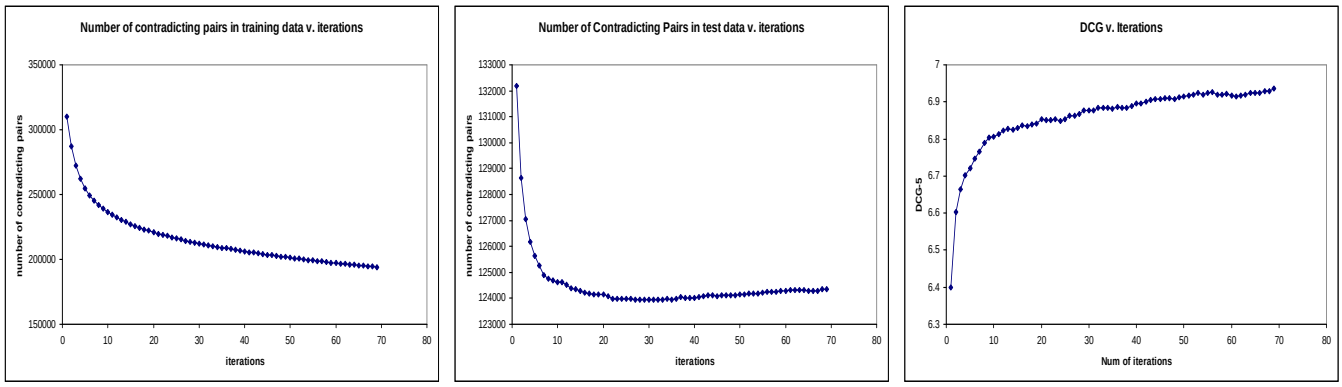

Figure 1: Number of contradicting pairs in training set against GBrank iterations (left), number of contradicting pairs in testing set against GBrank iterations (middle), DCG on testing data againt GBrank iterations (right)

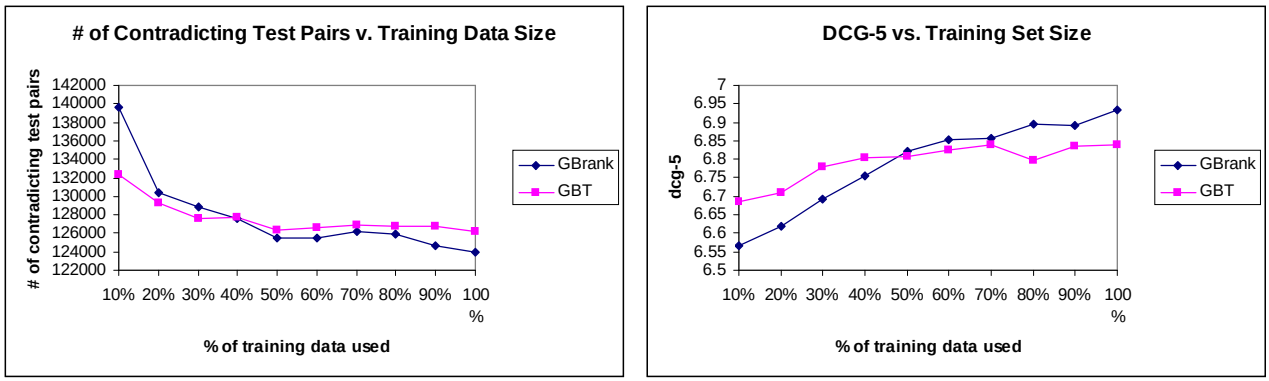

Figure 2: Number of contradicting pairs v. training data size (left), DCG v. Training Data Size (right)

experiments. In our experiments, *dcg* will be reported only when absolute relevance judgments are available.

3.3 Experiment Design and Results

The following questions guide the design of the experiments we carry out:

1. What is the convergence behavior of GBrank, will the number of contradicting pairs decrease in the training data? in the test data?
2. Quantitatively, what is the effect of the training data size on the performance of GBrank in terms of the three metrics we proposed?
3. When we have absolute relevance judgments, we can also use a regression method such as GBT to learn the ranking function based on the assigned numerical grades [7, 19]. How does GBT learned with absolute relevance judgments compare to GBrank learned with relative relevance judgments converted from the same set of absolute relevance judgments?
4. RankSVM based on linear SVM is a popular method for training ranking functions using preference data. How does GBrank compare with RankSVM?

For GBT and RankSVM, we tuned the parameters to get the best performance. For GBrank, we just used the parameters tuned for GBT: we use 100 trees each with 15 leaf nodes, and the shrinkage factor is set to be 0.05. All the experiments were conducted on a 2.4Ghz 4-cpu AMD server with 4G RAM. GBT training will take about 15 minutes while the training time for GBrank would be a few hours depending on the number of iterations and number of preferences. The testing time for GBT is only a few minutes and that for GBrank would be the number of iterations multiplied by the above time for GBT. Both training and testing time for GBrank could be significantly reduced by using less number of trees for each iteration, e.g. single tree instead of GBT. Our more recent experiments showed that GBrank using single tree with a few hundred iterations achieves comparable dcg-5 while reducing the time complexity a great deal.

3.3.1 *Experiments with labeled data*

For the first two questions, we generate the training and testing data as follows: we randomly split the labeled data described in section 3.1.1 by query into training set (60% of labeled queries, 71,338 query-documents and 753,976 preferences) and testing set (the remaining 40% queries, 43,940 query-document pairs and 465,893 preferences).

From the left panel of Figure 1, we can see that the number of contradicting pairs monotonically decreases iteration by iteration during training, which indicates the convergent trend of GBrank. As for now, we always use a validation set

Table 1: Number of contradicting pairs (CP) and precision (Prec) at K% for GBrank and GBT

%K	num pairs	GBrank		GBT	
		CP	Prec	CP	Prec
10%	46590	225	0.9952	949	0.9796
20%	93179	1095	0.9883	3313	0.9644
30%	139768	5695	0.9593	8718	0.9376
40%	186358	15324	0.9178	18340	0.9016
50%	232947	28117	0.8793	30865	0.8675
60%	279536	46334	0.8434	43776	0.8342
70%	326126	61189	0.8124	63711	0.8046
80%	372715	80726	0.7834	82976	0.7774
90%	419304	101601	0.7577	103530	0.7531
100%	465893	123939	0.7340	126188	0.7291

to decide when to stop the iteration. The middle panel of Figure 1 shows the number of contradicting pairs on the testing data first decreases and then gradually increases after a certain number of iterations. Since we also have available the absolute relevance judgments, we plot the dcgs against each iteration in the right panel of Figure 1. The dcg plot is almost a mirror image along the horizontal axis of the contradicting pairs plot on testing data. We mention that we have observed similar trends on other data sets we have tested.

In order to demonstrate the effect of training data size, we randomly sample increasing percentages of training data and generate the corresponding pairwise preference data as described in section 3.1.1. The experimental results on the same testing data are reported as follows:

The left panel of Figure 2 shows the number of contradicting pairs in the testing data decreases with the increasing size of training data. The dcg-5 for different training data size was shown in the right panel of Figure 2, which indicates a strongly positive correlation between the dcg gain and the increase of training data size.

Although our major concern is about GBrank using relative relevance judgments, it is actually rather instructive to see how it compares with GBT when trained on the same absolute relevance judgments, of course for GBrank, we will need to convert the absolute relevance judgments to relative relevance judgments. This experiment is aimed at the third question. One interesting observation is that GBrank underperforms GBT when the training data size is small. One plausible explanation for this is that to rank objects according to a set of preferences, there needs to be enough *overlaps* among the preferences, for small amount of data, the overlaps are weak and hence the poorer performance. Why would GBrank outperform GBT when there are plenty of training data? We suspect the deeper reason lies at the fundamental difference between a ranking problem and a regression problem. GBT is designed for regression problems and is therefore not necessarily the optimal choice for ranking problems.

We elaborate on this point a bit more now: ranking web search results is fundamentally a preference learning problem rather than a regression problem, i.e., we just want to rank more relevant results higher than those less relevance ones, we do not need to care about predicting the grades of the documents very accurately. Let us illustrate this using a simple example [19]. Consider two queries q_1: "harvard university" and q_2: "college of san mateo". q_1 is more popular than q_2 generating about 13 million search results while results for q_2 are two orders of magnitude less. For simplicity we consider a ranking function $h(x)$ using a single feature x which counts the number of inbound links to a document. Now let us examine the top three results $d_{i1}, d_{i2}, d_{i3}, i = 1, 2$, for each of the query. Assume d_{i1} is ranked perfect, d_{i2} is ranked excellent, and d_{i3} is ranked good, and we convert the labels to numerical values as follows,

$$0 \Leftrightarrow \text{perfect},\ 1 \Leftrightarrow \text{excellent},\ 2 \Leftrightarrow \text{good}.$$

Since q_1 is very popular, each of the top three results generate high feature values, say, $x = 100000, 80000, 50000$ while for q_2 the corresponding feature values are $x = 1000, 800, 500$. Assume x is negatively correlated with the label, i.e., small x values tend to indicate better relevance, then we will need to find a monotonically decreasing function h such that for q_1

$$h(100000) \approx 0,\ h(80000) \approx 1,\ h(50000) \approx 2$$

and for q_2,

$$h(1000) \approx 0,\ h(800) \approx 1,\ h(500) \approx 2.$$

The major issue comes from the difference of popularity of the queries. This could partially explain why ranking functions could be better learned using GBranking with preference data than GBT regression with absolute relevance judgments.

In our experiments, we exclude all tied data (pairs of documents with the same grade) when converting preference data from the absolute relevance judgments, which is a significant information loss especially when the training data are small. Adding those tied data will certainly increase the overlaps among the training preferences. Including those tied data in GBrank learning will be part of our future work.

We now turn to the precision at K% metric: Table 1 presents the number of contradicting pairs and precision at K% for GBT learned with all of training data and GBrank learned with the corresponding preference data. This again shows that GBrank outperforms GBT with respect to the precision at K% metric.

To further explore the third and the last questions, we conduct an experimental comparison among GBrank, GBT, and RankSVM in a 5-fold cross-validation setting. Again, the 5-fold splitting is on queries using the data described in section 3.1.1. Figure 3 and 4 show the results using the two metrics, dcg-5 and number of contradicting pairs, for GBrank, GBT, and RankSVM, from which we can see GBrank is the best performer and RankSVM is worse than both GBrank and GBT. Average over the 5-folds, dcg-5 for GBrank is 1.2% better than GBT with p-value equal to 0.0005, and 5.7% better than RankSVM with p-value close to zero. As a baseline comparison, the dcg-5 difference among the top search engines on this data set is about 2-3%.

3.3.2 Experiments with clickthroughs data

We also use the preference data extracted from user clickthroughs data as described in section 3.1.2. The comparison is for RankSVM and GBrank in a 5-fold cross validation setting. For this data set, we can no longer use GBT since we do not have the absolute relevance judgments. Tables

2 and 3 present the results with respect to the number of contradicting pairs metric as well as the precision at $K\%$ metric. Both tables again show that GBrank outperforms RankSVM.

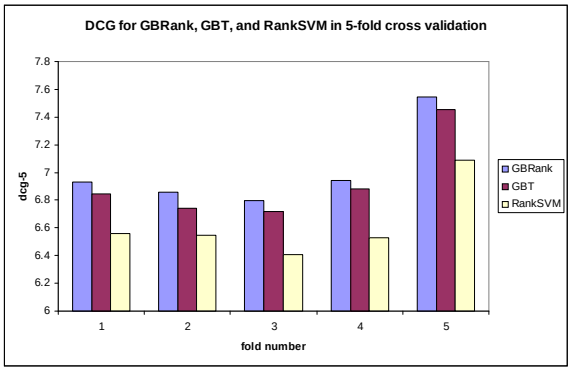

Figure 3: DCG for GBrank, GBT and RankSVM in 5-fold cv

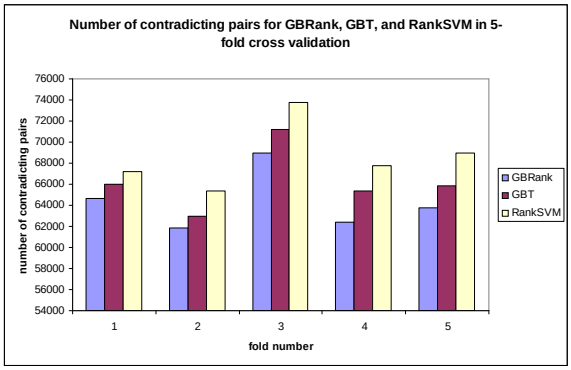

Figure 4: Number of contradicting pairs for GBrank, GBT, and RankSVM in 5-fold cv

4. CONCLUSIONS AND FUTURE WORK

In this paper we proposed a general regression framework for learning ranking functions from preference data. In particular, we developed GBrank, a specialization of our framework using gradient boosting trees as the regression method. When only preference data are available, GBrank provides a more flexible and effective solution to the problem of learning ranking functions. Even when absolute labels are available, our experiments suggest that it is preferable to first convert them into preference data and apply GBrank over them than directly apply GBT to the original aboslute labels.

There are several directions we can pursue to further enhance our approaches: 1) when converting absolute relevance data, we can overweigh the document pairs with larger grade difference. 2) weigh each error term in the loss function defined in Equation (1) with the DCG difference. Specifically, assume we have two documents: d_1 and d_2. At the current iteration, d_1 and d_2 were ranked at position i and j respectively, where $i < j$. Suppose the resulted predicted preference contradicts with the true preference. Their dcg contribution with respect to the wrong ordering would be $\frac{G(d_1)}{log_2 i+1} + \frac{G(d_2)}{log_2 j+1}$ while that for the correct ordering should be $\frac{G(d_1)}{log_2 j+1} + \frac{G(d_2)}{log_2 i+1}$. The DCG difference caused by the wrong ordering is therefore $|G(d_1) - G(d_2)|[\frac{1}{log_2 i+1} - \frac{1}{log_2 j+1}]$. During training, we can weigh each error term according to that difference. When the absolute relevance judgments are not availabe, we can just remove $|G(d_1) - G(d_2)|$. 3) We mentioned that we can also include the tied data, pairs $\langle x_i,\ y_i \rangle$ with the same grade. One way to do that is to add the following to the set in Equation (2) to construct the training set for computing the regression function at each iteration,

$$\left\{ (x_i, \frac{h_{k-1}(x_i) + h_{k-1}(y_i)}{2}),\ \ (y_i, \frac{h_{k-1}(x_i) + h_{k-1}(y_i)}{2}) \right\};$$

4) as we mentioned before, our framework including GBrank is very flexible for combining relative and absolute relevance judgments. With any query-document feature vector x_i and its grade g_i, we just need to add (x_i, g_i) to the set in Equation (2), and there is no need to modify the objective function. Such flexibility is desirable considering there are many queries having a single document with absolute relevance judgment(or documents with same absolute relevance judgment), and we could not extract any preference data from that.

Acknowledgment. We thank Tong Zhang for suggesting a modification of the objective function (1) and Alex Simma for the use of LRT.

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

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

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

[5] H. Chen. Machine Learning for information retrieval: Neural networks, symbolic learning and genetic algorithms. *JASIS*, 46:194-216, 1995.

[6] 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.

[7] D. Cossock and T. Zhang. Subset ranking using regression. *COLT*, 2006.

[8] 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.

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

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

Table 2: Number of contradicting pairs (CP) and precision (Prec) at K% for RankSVM on clickthroughs data

%K	fold 1		fold 2		fold 3		fold 4		fold 5	
	CP	Prec	CP	Prec	CP	Prec	CP	Prec	CP	Prec
10%	18	0.9143	14	0.9333	11	0.9476	16	0.9238	18	0.9143
20%	41	0.9021	30	0.9284	32	0.9236	36	0.9141	50	0.881
30%	79	0.8744	73	0.8839	73	0.8839	67	0.8935	90	0.8571
40%	141	0.8317	118	0.8592	124	0.8521	115	0.8628	146	0.8262
50%	206	0.8034	187	0.8216	186	0.8225	175	0.833	203	0.8067
60%	265	0.7892	255	0.7971	259	0.794	242	0.8075	257	0.7959
70%	344	0.7653	318	0.7831	340	0.7681	313	0.7865	342	0.7672
80%	425	0.7464	397	0.7631	424	0.747	394	0.7649	420	0.7499
90%	509	0.73	487	0.7416	507	0.731	488	0.7411	516	0.7268
100%	606	0.7106	579	0.7235	597	0.7149	582	0.7221	627	0.7011

Table 3: Number of contradicting pairs (CP) and precision (Prec) at K% for GBrank on clickthroughs data

%K	fold 1		fold 2		fold 3		fold 4		fold 5	
	CP	Prec	CP	Prec	CP	Prec	CP	Prec	CP	Prec
10%	6	0.9714	6	0.9714	6	0.9714	8	0.9619	9	0.9571
20%	24	0.9427	14	0.9666	20	0.9523	23	0.9451	30	0.9286
30%	47	0.9253	48	0.9237	52	0.9173	51	0.9189	72	0.8857
40%	95	0.8866	77	0.9081	103	0.8771	86	0.8974	112	0.8667
50%	155	0.8521	129	0.8769	164	0.8435	124	0.8817	156	0.8514
60%	218	0.8266	193	0.8465	207	0.8353	192	0.8473	211	0.8324
70%	294	0.7995	261	0.822	281	0.8083	262	0.8213	283	0.8074
80%	373	0.7774	342	0.7959	367	0.781	333	0.8013	365	0.7826
90%	466	0.7528	420	0.7772	464	0.7538	411	0.782	459	0.757
100%	542	0.7412	505	0.7588	555	0.735	509	0.7569	542	0.7417

[11] F. Gey, A. Chen, J. He and J. Meggs. Logistic regression at TREC4: probabilistic retrieval from full text document collections. *Proceedings of TREC*, 65-72, 1995.

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

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

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

[15] 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.

[16] 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.

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

[18] H. Turtle and W. B. Croft. Inference networks for document retrieval. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1-24, 1990.

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

[20] Diane Kelly and Jaime Teevan. Implicit Feedback for Inferring User Preference: A Bibliography. *SIGIR Forum*, 32:2, 2003.

[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] C. Zhai and J. Lafferty. A risk minimization framework for information retrieval , *Information Processing and Management*, 42:31–55, 2006.