Adapting Ranking Functions to User Preference

Keke Chen, Ya Zhang, Zhaohui Zheng, Hongyuan Zha, Gordon Sun

Yahoo!

{kchen,yazhang,zhaohui,zha,gzsun}@yahoo-inc.edu

Abstract— **Learning to rank has become a popular method for web search ranking. Traditionally, expert-judged examples are the major training resource for machine learned web ranking, which is expensive to get for training a satisfactory ranking function. The demands for generating specific web search ranking functions tailored for different domains, such as ranking functions for different regions, have aggravated this problem. Recently, a few methods have been proposed to extract training examples from user clickthrough log. Due to the low cost of getting user preference data, it is attractive to combine these examples in training ranking functions. However, because of the different natures of the two types of data, they may have different influences on ranking function. Therefore, it is challenging to develop methods for effectively combining them in training ranking functions. In this paper, we address the problem of adapting an existing ranking function to user preference data, and develop a framework for conveniently tuning the contribution of the user preference data in the tuned ranking function. Experimental results show that with our framework it is convenient to generate a batch of adapted ranking functions and to select functions with different tradeoffs between the base function and the user preference data.**

I. Introduction

Web search has been an important tool in our life. Today, there are billions of web pages accessible on the Internet. These web pages are highly diversified in terms of content, format and quality. It is the basic challenge for search engines to rank web pages for a given query to find the most relevant one from such a huge amount of diversified data. Recently, machine learning to rank has become a popular approach in web search ranking [1], [2], [3], where manually labeled web search results are the major source of training data.

Typically, web relevance judgers follow certain guidelines to label the relevance of web search results. These guidelines roughly measure the observed relevance between a query and the content of a web page and decide a relevance grade for the query-document pair. Such expert-judged training examples have certain advantages: the queries for judgment can be carefully selected for different purposes and the judged results can be thoroughly verified and tuned. Due to these properties, the expert-judged examples are often regarded as the "gold set" for training ranking functions. However, it is very costly to manually label and verify a large number of web search examples, which are often required for learning a satisfactory ranking function. This issue is even aggravated when we need to train multiple ranking functions for different domains. It is also arguable that the opinions of a few experts on relevance may not be representative enough for a highly diversified user population.

Recently, implicit relevance judgments derived from user clickthrough logs have become another attractive resource for machine learned ranking due to the relatively low cost to obtain them. Joachims et al. [4], [2] and Agichtein et al. [5], [6] have shown that relevance judgments or features related to user preference that are mined from user clickthrough log can also be helpful for training ranking functions. However, relevance judgments from click log are usually relative judgments, i.e., we know only document d_1 is better than d_2 for certain query q, which is different from the expert judged examples that are labeled with absolute grades. Especially, compared to expert-judged examples, user preference can be more dynamic, which may better reflect some epidemic topics, such as movies, fashion, recent events, or user-group biased searches.

Due to the different natures of the two types of data, directly combining them may not give a satisfactory ranking function. In this paper, we propose a framework based on empirical risk minimization (ERM) [7] and tree-based gradient boosting (GBT) [8] to address the problem of flexibly combining the two types of data. The GBT technique for ordinal regression (regressing to the grades) has been shown effective in modeling ranking functions [9]. Therefore, we decide to adapt GBT-based ranking functions with user preference. Concretely, this framework consists of two phases: in the first phase, the expert-judged examples [1] are used to train a nonlinear boosted tree model with regression on the target relevance grade. In the second phase (the tuning phase) the model is transformed to a *hyper linear tree model*, and the user preference examples are used to tune the transformed model. We designed three algorithms for tuning the base model, which are 1) stochastic gradient descent, 2) closed-form optimization, and 3) hinge-loss optimization.

Our framework allows us to flexibly tune the contributions of the two types of data in the final ranking model, by controlling the parameter in the tuning phase. This flexibility is important due to the varying quality and the representativeness of the two types of training data in practice. In addition, in order to make the contributions from both types of data measurable, we also design the B measure.With different parameter setting, different tuning algorithms, and the B measure, we are able to identify models with balanced performance to the two types of data in experiments.

This paper proceeds as follows. In Section II we will review some related work in learning to rank and extracting user preference data from clickthrough logs. Then, we

[1]They may also not be particularly prepared for the target domain.

give notations and definitions in Section III and present the framework including the three tuning algorithms in Section IV. Experimental results are presented in Section V.

II. Related Work

Recent research on learning to rank web search results has been focused on two aspects: developing effective learning algorithms, and mining useful information from clickthrough data to improve ranking functions. Most new learning algorithms are derived from traditional classification or regression algorithms. Nallapati [10] formalizes the information retrieval problem as a binary classification problem, i.e., class "relevant" vs. class "irrelevant" to the given query. This might be too rough for the ranking problem, however. In fact, multi-grade expert-judged examples are more widely used in training ranking functions, which naturally cast the ranking problem as an ordinal regression problem, where each query-document pair is mapped to a relevance grade. SVM technique has been extensively used to address the ordinal-regression based ranking approach, such as, Ranking SVM by Herbrich et al. [1]. PRank [11] maximizes the margin with a perceptron like algorithm. In addition, Gradient Boosting Tree [8] also works effectively [9] for nonlinear regression, which will be used in our approach to train the base model.

Moreover, there are several algorithms working on pairwise training examples. RankNet [3] applies neural network to minimize the number of reversely ordered pairs, and another "Ranking SVM" [4] tries to minimize the number of reversely ordered pairs with SVM technique. RankBoost [12] applies Ada boosting techniques to pairwise training examples.

The rest interests have been on mining relevance examples or click-related features from user implicit feedback, or clickthrough logs. There are discussions on extracting effective pairwise relevance examples from the clickthrough log [2], [**?**], [13], [14] and how to derive effective click-related features [6], [5]. The user preference examples are usually in the form of pairs, reflecting *relative relevance* between a pair of documents to a query, which are extracted with certain strategy such as "Skip Above" [2]. A few learning algorithms, as we discussed earlier, have been developed to train on such pair examples directly.

III. Notations and Definitions

In this section, we define the basic concepts and notations used in this paper. In general, training examples in learning to rank are described by the set of potential user queries $\mathcal{Q}$, the set of accessible documents $\mathcal{D}$ in web, and a small set of labels/grades $\mathcal{L}$, approximately reflecting the level of relevance. We define the two types of training data as follows.

Expert Judged Examples. This type of data is labeled by relevance experts who follow some agreed guidelines. $\mathcal{L}$ usually consists of a small number of integer grades, e.g., five grades, representing the degree of relevance level for a document d to a specific query q. Let l be the grade. These training examples are usually represented by (q, d, l). For a given query q, the ideal ranking is given by the list of examples $\{(q, d_i, l_i)\}$ ordered by their labels l_i. Often, metrics like DCG or $NDCG$ [15] are used to evaluate the quality of an arbitrary ranked list.

User Preference Examples. User preference examples are derived from clickthrough logs, which are often pairwise examples [2], [4], representing relative relevance for pairs of documents (d_1, d_2). We use (q, d_1, d_2) to represent a user preference example, where d_1 is more relevant to q than d_2. Sometimes, $d_1 >_r d_2$ is also used to represent the relative relevance between the two documents. Precision on test pairs is the commonly used metric for the ranking performance on the pairwise data.

Note that expert-judged examples can be easily converted to pairwise examples, but normally we are unable to convert the pairwise examples to examples with absolute grades. Therefore, algorithms working on the directly combined data are often pairwise learning algorithms.

Ranking Function. With sufficient amount of training examples, a ranking function H can be trained. Given a query q and a document d, the ranking function $H(q, d)$ will give a score $s \in \mathcal{R}$, where $\mathcal{R}$ is the set of real number.

$$H : \mathcal{Q} \times \mathcal{D} \rightarrow \mathcal{R}$$

with H we are able to determine a ranking for all documents D for any given query q – $\{(q, d_i, s_i)\}$ are simply ordered by the scores s_i.

In practice, we will use a set of features $\mathbf{x}_{q,d}$ to describe the query, document, and query-document relationship. Accordingly, $\mathbf{x}_{q,d}$ consists of three sets of features: features for query $\mathbf{x}^Q$, features for document $\mathbf{x}^d$, features for query-document relationship $\mathbf{x}^{QD}$.

$$\mathbf{x}_{q,d} = [\mathbf{x}^Q, \mathbf{x}^D, \mathbf{x}^{QD}]$$

We will give a more detailed description about features in experiments. With the definition of feature vector $\mathbf{x}_{q,d}$, the ranking function $H(q, d)$ is in fact represented as $H(\mathbf{x}_{q,d})$.

As a convention, we will use bold characters to represent vectors, and regular characters to represent scalars, constants or function names.

IV. Learning Ranking from Multiple Sources

Assume that we have the two sets of training data ready generated from different sources. A simple method is to convert the expert judged examples to pairwise examples and simply combining them for pairwise training. However, this simple method may not give satisfactory functions, which we will show in experiments. More importantly, we will need certain level of flexibility in combining the data and control their contributions in the final ranking function. In this section, we propose a new method based on the framework of empirical risk minimization [7].

Our method consists of two phases. The first phase will generate a base model with one type of training data $\mathcal{T}_1$, which is denoted as $H(\mathbf{w}_0, \mathbf{x})$, where $\mathbf{w}_0$ is the tunable model parameter and $\mathbf{x}$ is the feature vector for a given query-document pair (q, d). In the second phase, the model parameter

$\mathbf{w}_0$ are tuned with the other set of training data $\mathcal{T}_2$, with the following ERM framework.

$$\arg\min_{\mathbf{w}}\{\frac{1}{n}\sum_{i=1}^{n} L(H(\mathbf{x}_i,\mathbf{w}),l_i) + \lambda \parallel \mathbf{w}-\mathbf{w}_0 \parallel^2\} \quad (1)$$

L in the first item is a loss function and λ in the second item is the regularizor constraining the deviation of $\mathbf{w}$ from original model parameter $\mathbf{w}_0$.

There are several challenges in implementing this method in practice. First, we need to determine an appropriate base model, from which the optimization of Eq. 1 can be effectively done. Second, the loss function should be selected with respect to the property of the second set of training data. The two problems are tightly correlated and seem complicated.

To simplify the problems, we first decide to take the gradient-boosting-trees based ranking function as the base model, which can be transformed to a "hyper linear" base model if we allow only a subset of parameters tunable. It will be convenient to use pairwise training examples to tune the hyper linear base model with the above framework. In the following subsections, we will describe the definition of hyper linear model, then we will discuss the optimization methods for different loss functions based on the framework of Eq. 1 and the hyper linear base model.

A. *Using Hyper Linear Model as the Base*

The basic idea of "hyper linear model" is to transform the original feature vectors into another set of feature vectors so that a linear model can be learned on the transformed feature vectors. Additive models [16] are good examples of hyper linear models. An additive model consists of a set of sub-functions, h_i, $h_i \in \mathcal{H}$, $i = 1,\ldots,k$. Linearly combining them gives us a satisfactory model $H(\mathbf{w}_0,\mathbf{x})$

$$H(\mathbf{w}_0,\mathbf{x}) = \sum_{i=1}^{k} w_{0,i}h_i(\mathbf{x}) \quad (2)$$

In practice, h_i may also generate a vector rather than one value. Then, $w_{0,i}$ should be a vector too, $\mathbf{w}_{0,i}$. The above model becomes

$$H(\mathbf{w}_0,\mathbf{x}) = \sum_{i=1}^{k} \mathbf{w}_{0,i}^T h_i(\mathbf{x}) \quad (3)$$

If h_i is regarded as a feature transformation function, the function H is a linear function in terms of the transformed features. The transformation of the original feature by h_is will generate a transformed vector $\mathbf{h}_{\mathbf{x}}^T = [h_1(\mathbf{x}), h_2(\mathbf{x}),\ldots,h_k(\mathbf{x})]$. Together with the linear combination parameters $\mathbf{w}_0^T = [\mathbf{w}_{0,1},\ldots,\mathbf{w}_{0,k}]$, we get $H(\mathbf{w}_0,\mathbf{x}) = \mathbf{w}_0^T \cdot \mathbf{h}_{\mathbf{x}}$. Hyper linear model will enable us to conveniently implement several optimization methods based on the framework Eq.1 $-$ we will focus on tuning the linear combination parameters $\mathbf{w}_0$.

There are a few excellent techniques for learning an additive model. For example, gradient boosting tree (GBT) [8], is an effective way for both regression and classification. It has been successfully applied to learning ranking functions [9], recently. In our framework, we will use GBT to learn the base ranking function and transform it to a hyper linear base model. GBT progressively generates a set of additive decision trees h_i, $i = 1\ldots k$, each of which has m leaf nodes. The latter trees are trained on the residues of the prediction given by the previously trained trees. It uses gradient boosting technique to learn the tree structure and the weighting vector $\mathbf{w}_{0,i}$ for each tree h_i. Concretely, after training the tree h_i, each node corresponds to a predicate, e.g., "feature 1 ¡ threshold 1", and each leaf node e_{ij} in the tree h_i corresponds to one decision path as well as one transformation to the feature vector, with $w_{0,i,j}$ as the weight of this transformation. When the model is applied to an original feature vector $\mathbf{x}$, the vector goes through each tree h_i and always leads to only one leaf node, assuming it is e_{ij}. Then, the output of h_i to $\mathbf{x}$ can be represented as a selective vector $[e_{i1}, e_{i2},\ldots,e_{im}]$ with all entries 0 except $e_{ij} = 1$. $\mathbf{w}_{0,i}^T \cdot h_i(\mathbf{x})$ is the output of the h_i tree. The sum of all tree output gives the final ranking score. Therefore, the GBT model can be transformed to a hyper linear model in the form Eq. 3.

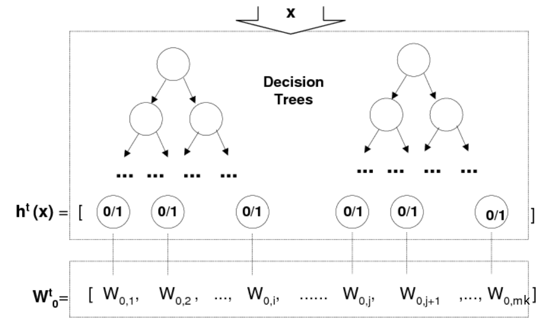
x
Decision Trees
h^t(x) = [0/1 0/1 0/1 0/1 0/1 0/1]
W^t_0= [W_{0,1}, W_{0,2}, ..., W_{0,i}, ... W_{0,j}, W_{0,j+1} ,..., W_{0,mk}]

Fig. 1. GBT generates a specified number of trees, and the weights $\mathbf{w}_0$ are associated with the leaf nodes.

Figure 1 illustrated the structure generated by GBT algorithm. GBT training algorithm will determine the structure of trees and the corresponding leaf-node weights, with the specified parameters, such as the number of trees, the number of leaf node per tree, and the learning rate [8]. As a result, GBT learns an ordinal regression model with the labeled training examples.

B. *Algorithms for Incorporating User Preference*

We have generated the hyper linear based model with gradient boosting tree training algorithm. Now we try to incorporate the user preference data with the framework Eq. 1. As we have defined, examples of user preference exist in the form of pairs (q, d_1, d_2), where d_1 is more relevant than d_2 in terms of the query q. First, we need to transform the pairwise data so that the training algorithm can take them conveniently.

Delta Transformation. Since h_i is fixed in the tuning phase, for conveniently using the pair examples (q, d_1, d_2) in optimization, we transform them to the following delta form. We define the *Delta Feature Vector* as

$$\Delta\mathbf{x}_{d_1,d_2} = \mathbf{h}_{\mathbf{x}_{d_1}} - \mathbf{h}_{\mathbf{x}_{d_2}}$$

assuming the greater the prediction score, the more relevant the document to the query. When the ranking function $H(\mathbf{w},\mathbf{x}) = \mathbf{w}^T \cdot \mathbf{h_x}$ is used, in order to keep the right order of d_1 and d_2, the following relationship should be preserved.

$$H(\mathbf{w},\mathbf{x}_{d_1}) - H(\mathbf{w},\mathbf{x}_{d_2}) = \mathbf{w}^T \cdot \Delta\mathbf{x}_{d_1,d_2} > 0$$

For clear presentation, we define the *Delta Function* f for the pair.

$$f(\mathbf{w},\Delta\mathbf{x}_{d_1,d_2}) \quad = \quad H(\mathbf{w},\mathbf{x}_{d_1}) - H(\mathbf{w},\mathbf{x}_{d_2})$$

Since only the ordering matters in pair examples, we define the *Delta Label* as

$$\Delta l_{d_1,d_2} = \begin{cases} 1 & d_1 >_r d_2 \\ -1 & d_2 >_r d_1 \end{cases} \tag{4}$$

By definition, if $f(\mathbf{w}_0,\Delta\mathbf{x}_{d_1,d_2}) > 0$, we derive the preference $d_1 >_r d_2$, otherwise $d_2 >_r d_1$. The goal is to learn such a function f so that the output preference is as consistent with the user preference as possible.

With the hyper linear base model and the delta definitions, we are ready to optimize the final ranking function based on the framework Eq. 1. The selection of loss functions and the optimization of Eq. 1 become much easier now. We discuss two types of loss functions and their optimization methods in the following discussion. One of the loss functions is “Mean Square Error (MSE) loss function”, which is normally used in regression modeling. The other is called hinge loss function, which is used in Support Vector Machine (SVM) modeling [16].

The above loss functions and the corresponding basic optimization methods are well known. But when they are combined with the framework, we need to describe the particular algorithms in more details.

1) Optimization with MSE Loss: We first try the Mean-Square-Error (MSE) loss function in the framework Eq. 1 for incorporating the user preference data. MSE loss function is defined as follows.

$$L_{MSE}(f(\mathbf{w},\Delta\mathbf{x}),\Delta l) = (f(\mathbf{w},\Delta\mathbf{x}) - \Delta l)^2$$

If there are n pairs in training dataset, with the Delta transformation, the framework can be implemented with

$$\arg\min_{\mathbf{w}}\{\frac{1}{n}\sum_{i=1}^{n}(f(\mathbf{w},\Delta\mathbf{x}_i) - \Delta l_i)^2 + \lambda \parallel \mathbf{w} - \mathbf{w}_0 \parallel^2\} \tag{5}$$

where the initial value of the weight vector $\mathbf{w}$ is set to $\mathbf{w}_0$, i.e., the base model’s weight vector.

We use two methods to optimize Eq. 5. The first method is called Stochastic Gradient Descent (SGD) [17]. SGD follows the idea of Gradient Descent, but in each iteration it randomly chooses some data sample(s) to calculate the gradient and uses this gradient to revise the weight vector $\mathbf{w}$. SGD is particularly good for noisy or large datasets. Because of its randomization nature, it is possible to escape from local minima caused by noises, and random sampling in each iteration also dramatically reduces the huge computational cost in the direct gradient descent.

Closed Form Solution (CF). If the training examples are small and not so noisy, we can also use a closed form solution to Eq. 5. Let $Q = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{f}(\mathbf{w},\Delta\mathbf{x}_i) - \Delta l_i)^2 + \lambda \parallel \mathbf{w} - \mathbf{w}_0 \parallel^2$. The closed form solution is obtained by letting

$$\frac{\partial Q}{\partial \mathbf{w}} = 0$$

It follows

$$(\sum_{i=1}^{n}\Delta\mathbf{x}_i\Delta\mathbf{x}_i^T + \lambda\mathbf{I})\mathbf{w} = \sum_{i=1}^{n}\Delta l_i\Delta\mathbf{x}_i + \lambda\mathbf{w}_0 \tag{6}$$

Solving Eq. 6, we can get $\mathbf{w}$ directly.

2) Optimization with Hinge Loss (SVML): Suppose we have a pair of documents d_1 and d_2, $d_1 >_r d_2$ (or $d_1 <_r d_2$) as derived from user preference. Because it is difficult to quantify the difference in user preference between d_1 and d_2, for the training data, the Delta label that we assigned to each training pair can be an arbitrary positive number for $d_1 >_r d_2$ and an arbitrary negative number for $d_1 <_r d_2$. With the MSE loss, the selection of Delta labels will affect the optimization, which is not what we desire. In fact, for $d_1 >_r d_2$, as long as the predicted Delta label is positive, i.e. $l_{d_1} - l_{d_2} > 0$, the ranking is correct.

Due to the above problem, we also try hinge loss based optimization. Considering the Delta label we assigned to each training pair is either $+1$ or -1, hinge loss is defined as follows.

$$L_{svm}(f(\mathbf{w},\Delta\mathbf{x}),\Delta l) = [1 - \Delta l f(\mathbf{w},\Delta\mathbf{x})]_+ \tag{7}$$

where ‘+’ means only the positive values are accounted as the penalty. With hinge loss, we are modeling the optimization problem as a two-class classification problem: for any pair (d_1, d_2), if d_1 is more “relevant” than d_2 for a certain query q, the pair falls into positive class, otherwise, negative class. This model, to some extent, bypasses the problem of quantifying the difference in user preference between the pair of documents.

Plugging Eq. 7 into the framework Eq. 1, we get

$$\arg\min_{\mathbf{w}}\{\frac{1}{n}\sum_{i=1}^{n}[1 - \Delta l_i f(\mathbf{w},\Delta\mathbf{x}_i)]_+ + \lambda \parallel \mathbf{w} - \mathbf{w}_0 \parallel^2\} \tag{8}$$

Instead of directly solving Eq. 8, we solve the following quadratic programming problem.

$$\begin{array}{ll} & \min_{\mathbf{w}} \frac{1}{2} \parallel \mathbf{w} - \mathbf{w}_0 \parallel^2 + C\sum_i \xi_i \\ s.t. & \Delta l_i f(\mathbf{w},\Delta\mathbf{x}_i) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1,\ldots,n \end{array} \tag{9}$$

It is straight-forward to prove that Eq. 8 and Eq. 9 are equivalent to each other when $C = \frac{1}{\lambda}$. This formulation closely resembles that of SVM models.

Applying the optimization procedure similar to the SVM algorithm, finally, we get the approximation of the function f based on the hinge loss function.

$$\hat{f}(\Delta\mathbf{x}) = \mathbf{w}_0^T\Delta\mathbf{x} + \sum_{i=1}^{n}\alpha_i\Delta l_i\Delta\mathbf{x}_i^T\Delta\mathbf{x}_i \tag{10}$$

Eq. 10 consists of two components: the base model and the additional component controlled by λ. Intuitively, large λ, i.e., small C, will constrain the α_i values to a small range, which in turn reduces the deviation of Eq. 10 from the base model. We refer this algorithm as "SVML" in later discussion.

C. Balancing Contributions of Training Datasets

Our framework provides the possibility tuning the contributions of the two training datasets in the final ranking function by the regularizor item λ in the framework (Eq. 1). It intuitively controls the deviation of the final model to the base model. Concretely, the smaller the λ parameter, the lower the restriction is put on the final weights $\mathbf{w}$. $\mathbf{w}$ then deviates more from $\mathbf{w}_0$, preferring the model identified by the second set of training data. However, we do not have a metric quantitatively measuring the contributions yet. In this section, we will propose one metric for this purpose.

If the two sets of data are perfectly consistent, the tuned model will hopefully perform better on both sets of data. However, due to different data quality and distributional bias, we may want to make tradeoff between the two types of data. 1) If the base model is trained with relatively small number of labeled examples and we are not confident with the generalization ability of the base model, we would like the second dataset to contribute more to the final model. In this case, we will prefer smaller λ. 2) The base model is trained with a descent amount of high-quality expert-judged examples, but we also observed that user preference is slightly different from the expert judgment. Then, we may want to fine tune the base model. In this case, we will set a larger λ to constrain the model deviation. The designed metric will help us estimate the balance of contributions between the two sets of data and help us to determine the optimal λ and the appropriate optimization algorithm.

Ideally, we want the final model to improve the performance for both types of data. This goal may be achieved if the two datasets are highly consistent. When they are not consistent, there is always a tradeoff, making the final model prefer one set of data to the other. For quantifying this tradeoff in the training, we define the B measure based on a similar form to the well-known F measure [18]. B measure defines the weighted harmonic mean of the ranking quality on the two types of data. The balanced score should be based on two metrics that are in the same range, e.g., [0, 1]. Otherwise, the result will be biased by the one with larger range. Concretely, we let μ and ν represent the two metrics that evaluate the fitness of the model to the expert judged examples and the user preference examples, respectively. We then define the B measure as follows.

$$B = \frac{2\mu \cdot \nu}{\mu + \nu}$$

Now we discuss how to select appropriate metrics as μ and ν. Since the expert-labeled data have multiple grades, we will use normalized discounted cumulative gain (NDCG) [15] to evaluate the test result on the expert judged examples, i.e., as the μ metric. NDCG is defined as follows. Let the five-grade labeling scheme be encoded with integers '4' to '0', corresponding to the most relevant documents to the most irrelevant documents to the query q. Suppose there are k documents used for testing query q and each query-document pair (q, d_i) in the test set is labeled with a grade l_i. The testing will give a list of the k documents that is sorted by the scores given by the ranking function H to each pair (q, d_i). Let $i = 1, \ldots, k$ be the order of the sorted documents. The $NDCG_k$ score is computed for the sorted list as follows.

$$NDCG_k = \frac{1}{\psi} \sum_{i=1}^{k} \frac{2^l_i - 1}{\log(i+1)}$$

where ψ is the normalization constant so that a perfect ordering of the results for the query q will receive NDCG score of 1.0. By definition, any non-perfect ordering will result in a NDCG score less than 1.0 and greater than 0. The closer to 1.0 the better the ranking is.

Pairwise user preference is not multi-graded, however. The ν metric is thus simply based on the prediction on the preference over pairs of query-documents. Again this precision metric will be on the range [0, 1]. Therefore, both μ and ν are in the same range and they are applicable to the B measure.

The above B measure equally weights the two metrics, which does not include preference on any one of the datasets. In case that we think one set is more important than the other, an unbalanced B measure can be used. Let α be the proportion of importance between ν and μ, e.g., $\alpha = 2$ weights ν twice as much as μ.

$$B_\alpha = \frac{(1+\alpha) \cdot \mu \cdot \nu}{\alpha \cdot \mu + \nu}$$

In experiments, we will use the balanced B measure ($\alpha = 1$), while it is also convenient to set different α for different preferences. Together with the λ tuning, we are able to find a model that will give what we desire in balancing the contributions of the two types of data.

V. Experimental Results

In this section, we present three sets of experiments. We will first test the inconsistency between the two types of experimental data − if they are consistent we will not need to tune λ to find the balance. The second set of experiments studies the properties of the three proposed tuning algorithms. The last set of experiments will compare the balanced ranking quality of the models generated by our method, to the models generated by Ranking SVM [4] (RSVM). RSVM is a well-known open-source ranking model that minimizes the number of reversely ordered pairs based on the same idea of linear SVM.

A. Data Collections

In this section, we first describe the features used in our modeling, the methods for collecting the two types of data in our experiments, and then the two experimental datasets.

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:

- Features modeling user query, x^Q. They are dependent on the query q only and have constant values across all the documents $d \in \mathcal{D}$. Such features may include, the number of terms in the query, frequency of the terms in the corpus, query classification (name query, adult query, or navigational query), and so on and so forth. Totally over ten query features are used in training.
- Features modeling, x^D. They are dependent on the document d only and have constant values across all the queries $q \in \mathcal{Q}$. Such features may include, the number of inbound links pointing to the document, the number of distinct anchor-texts for the document, the language/region identify of the document, document classification(such as spam page, good quality page, or product page), and etc. More than tens of such features are used in training.
- Features modeling the query-document relationship, x^{QD}, which comprise features dependent on the relation of the query q with respect to the document d. Such features may include, the number of times each query term appears in the document d, the number of matched terms in the anchor-texts of the document d, etc. There are hundreds of such features used in training.

The few mentioned concrete features are among the most important features in modeling ranking functions with GBT method.

2) Expert-judged Examples: For each query q, we collect a set of results from a commercial web search engine. The example (q, d) is labeled with five-level grades from the most relevant "Perfect Match" to the totally irrelevant "Bad Result", which are mapped to $(4, 3, 2, 1, 0)$. The relevance experts follow some guidelines to give a grade for each query-document pair.

3) User Preference Examples: We also examine certain amount of clickthrough data from the commercial search engine and extract a set of user preference data as follows. For a query q, we consider two documents d_1 and d_2 in the result set for q. Assume that in the cleaned clickthrough data, d_1 has c_1 clicks out of n_1 impressions, and d_2 has c_2 clicks out of n_2 impressions for the same query q. In the above aggregation, we also remove noisy clicks as much as possible. We want to only 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 click through rate c/d for a particular query. To this end, we assume that clicks in user sessions obey binomial distribution, which is also held by user preference study in paper [2].

A binomial test is performed to distinguish whether the two ratios c_1/d_1 and c_2/d_2 are significantly different. Among the significantly different pairs, we apply rules similar to "Skip-Above" [2] to extract user preference.

Dataset	Expert Judged	User Preference
DS1	4K	12.5K
DS2	25K	20.5K

TABLE I
DESCRIPTION OF DATASETS

4) Datasets: Two sets of real datasets are used in experiments. Each set consists of an expert-judged dataset and a user preference pair dataset extracted from the clickthrough log of a major commercial web search engine in two different regions, respectively. In Table I, the numbers under "Expert Judged" mean the number of labeled examples (q, d, l) in the expert-judged dataset, and those under "User Preference" are the number of preference examples (q, d_1, d_2), where $d_1 >_r d_2$. DS1 has more user preference pairs than expert-judged examples, while DS2 has similar size on both types of data.

B. Inconsistency Test

In the first set of experiments, we use RSVM to explore the difference between the two types of training data. RSVM takes both types of data as input, but internally the absolute judgments are converted to pairs. If the two sets of data are from the same distribution, the RSVM model trained on one set of data will have a similar precision over the other set of data.

Experimental result in Figure 2 shows that the difference, which is statistically significant. "RSVM_judge" represents the model trained on expert-judged data and "RSVM_user" represents the model trained on user preference data. Figure 2 shows the testing result on the two types of data, with five-fold cross validation. The five-fold random splitting is on queries, rather than on query-document pairs, because the ranking quality has to be evaluated by queries if absolute judgments are used. The tests on the two datasets show similar pattern. Clearly, the models trained on expert-judged data do not fit user preference data well, and vice versa. Therefore, it will be meaningful to see how to tune the training process to find a balance between the two types of data. We will describe this in next section.

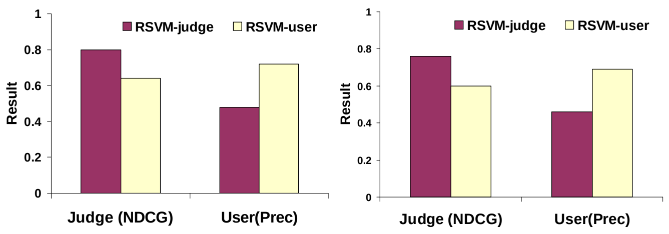

Fig. 2. Testing RSVM models (Left figure for DS1 and right for DS2)

C. Tradeoffs between Expert Judged Data and User Preference Data

With the B metric ($\alpha = 1$) as the the balance measure, we run a set of experiments to see how the three proposed algorithms perform with varying λ parameter. The following procedure is followed in practise. 1) For each algorithm, we train and cross-validate a set of models with a few candidate λ values; 2)Then, we evaluate the models with the B measure, and find the algorithm and its λ setting that gives the best B measure among all candidate models; 3)With the selected algorithm and λ we train the final model with all available data from both sources. λ can also be fine-tuned around a small range to find an even better final model.

Our experiments will focus on the first two steps with five-fold cross-validation for both the base model and the candidate tuned models. Again, the five-fold random splitting is on queries. The base model is generated with the GBT algorithm [8] on the manually labeled training data. GBT algorithm has three parameters: the number of trees, the number of leaf nodes per tree, and the learning rate, the optimal values of which are chosen through cross-validation in training the base model. The candidate tuned models are generated with the three algorithms: SGD, Closed_Form (CF), and SVM-Loss Refinement (SVML), for a list of regularization factor $\lambda = [0.01, 0.1, 1, 10, 10^2, 10^3, 10^4]$, respectively.

For clear presentation, we will only show the detailed result on DS1, while also present the summarized result on DS2. Two metrics are used to calculate the B measure: NDCG5 that evaluates the quality of ranking the top 5 URLs for each query for the labeled dataset, and order precision for predicting user preference pairs.

Figure 3 shows the result with SGD optimization. Both NDCG5 and pair precision change significantly over λ 0.01 $\sim$ 100, while stay constant for larger λ. Remember that λ controls the deviation of the tuned model to the base model. Large λ forces $\mathbf{w}$ to be very close to $\mathbf{w}_0$. Therefore, the models optimized with large λ will be almost identical to the base model with some small variation, which do not show large change on both NDCG5 and pair precision. In Figure 3, the optimal balanced model happens around $\lambda = 1$, where B measure is maximized.

Figure 4 and 5 show the results with closed-form and SVM-Loss optimization, respectively. In contrast to SDG, both closed-form and SVM-Loss stay stable with small λ. These Figures imply that with small λ the component of loss function in the framework for both CF and SVML dominates the overall risk function. In fact, we need very large λ to promote the difference between $\mathbf{w}$ and $\mathbf{w}_0$. The optimal models appear around $\lambda = 1000$ for both closed-form and SVM-Loss optimization.

Figure 6 summarizes B measure for all three algorithms and all λ settings. Overall, for DS1, SGD at $\lambda = 1$ will generate the best model and SVM Loss at $\lambda = 1000$ is very close to the best SGD. Thus we can choose SGD with $\lambda = 1$ for training the final model for DS1. Similarly, for DS2, SGD at $\lambda = 0.01$ outperforms other candidates (Figure 7). However, so far there is no clear clue whether one algorithm performs absolutely better than others in all cases.

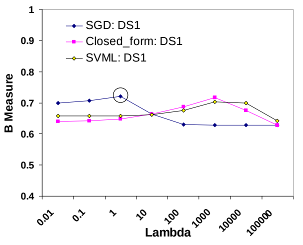

Fig. 6. Summary of B measures for different algorithms and λ settings for DS1.

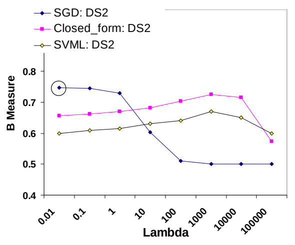

Fig. 7. Summary of B measures for different algorithms and λ settings for DS2.

D. Comparison with Ranking SVM

In Section V-B we have shown that RSVM models trained on different types of data have significantly different performance. Namely, RSVM trained on one type of data has much better performance on the same type of data, while performs not so well on the other. It is intuitive to understand that such RSVMs will not have satisfactory balanced performance on both types of data. Then, why not pool the two types of data together to train a unified model? If this is possible, will the unified model outperforms our balanced models? We are going to answer these questions in this section.

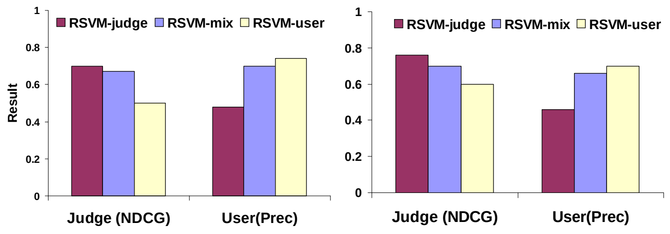

Fig. 8. Performance of RSVM_mix compared to other RSVM models

Pooling the two types of data together as one unified training set is possible for RSVM modeling. In fact, RSVM takes both types of data as input. The result of RSVM with pooled training data (RSVM_mix) matches what we expect – its performance is better balanced in terms of both types of data (Figure 8).

We then compare our balanced models to RSVM models. Figure 9 show the comparison on DS1 and DS2. “Base Model” represents the result of the GBT base model. “Our best model” represents the best result generated by our method, which is optimized by one of the three algorithms with certain λ setting. “RSVM_judge”, “RSVM_user”, “RSVM_mix” are the RSVM models described earlier. The result clearly shows that our best models have the best performance in terms of B measure among all methods and models, while “RSVM_mix” is ranked second with significantly lower B measure. Figure 10 also shows that our balanced models are able to sacrifice a little (DS1) or have almost no loss (DS2) on the performance of expert judged examples, to achieve a better balanced performance.

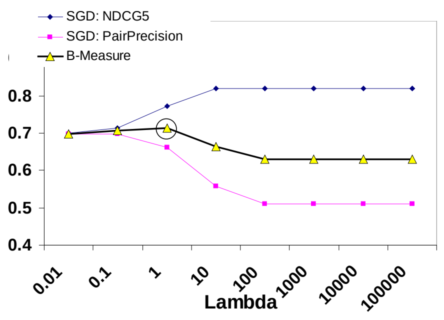

Fig. 3. Evaluation of SGD on DS1

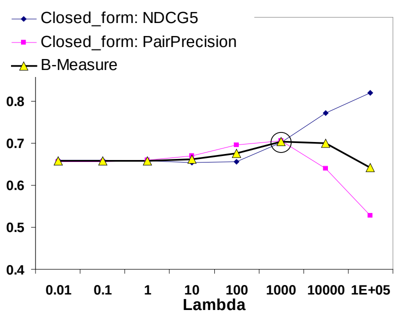

Fig. 4. Evaluation of closed-form on DS1

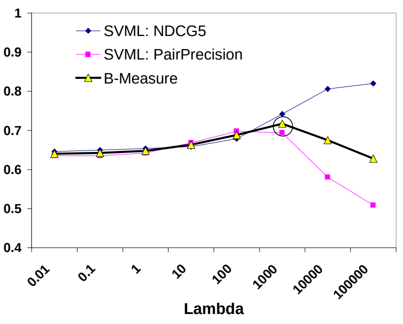

Fig. 5. Evaluation of SVM-Loss on DS1.

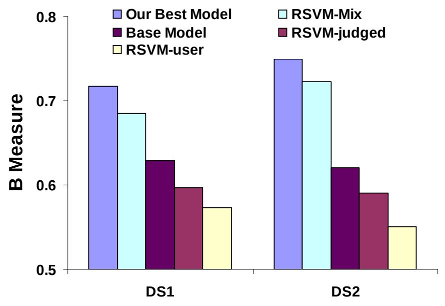

Fig. 9. B measure for all models.

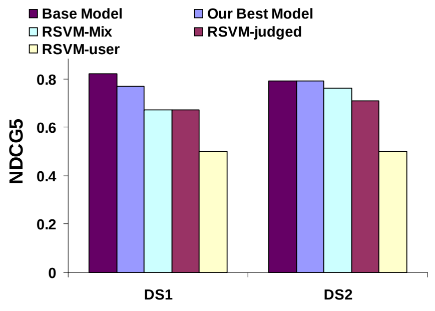

Fig. 10. NDCG5 on manually labeled examples for all models.

VI. Conclusion

Manually labeled examples and automatically extracted user preference are the two major training data sources for web search ranking. Due to the large cost of getting labeled training data, we often prefer to use user preference data as the complementary source. Due to the different natures of the two types of data, it is challenging to generate a model adapting well to both datasets. In this paper, we present a framework for flexibly combining these two types of data. Namely, in this framework a base model is trained with manually labeled data and then tuned with user preference examples. The B measure is proposed to evaluate the balanced performance on both types of data. By tuning the regularization parameter λ and the bias in the B measure, we can conveniently identify the models we want. Experiments show that with the proposed framework, we are able to find models with satisfactorily balanced performance, compared to the existing methods.

References

[1] R. Herbrich, T. Graepel, and K. Obermayer, "Large margin rank boundaries for ordinal regression," *Advances in Large Margin Classifiers*, pp. 115–132, 2000.
[2] T. Joachims, L. Granka, B. Pan, and G. Gay, "Accurately interpreting clickthough data as implicit feedback," *Proc. of ACM SIGIR Conference*, 2005.
[3] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," *Proc. of Intl. Conf. on Machine Learning (ICML)*, 2005.
[4] T. Joachims, "Optimizing search engines using clickthrough data," *Proc. of ACM SIGKDD Conference*, 2002.
[5] E. Agichtein, E. Brill, S. Dumais, and R. Ragno, "Learning user interaction models for predicting web search result preferences," *Proc. of ACM SIGIR Conference*, 2006.
[6] E. Agichtein, E. Brill, and S. Dumais, "Improving web search ranking by incorporating user behavior information," *Proc. of ACM SIGIR Conference*, 2006.
[7] V. N. Vapnik, *The Nature of Statistical Learning Theory*. Springer Science and Bussiness Media, LLC., 1999.
[8] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.
[9] Z. Zheng, K. Chen, G. Sun, and H. Zha, "A regression framework for learning ranking functions using relative relevance judgments," in *SIGIR*, 2007, pp. 287–294.
[10] R. Nallapati, "Discriminative models for information retrieval," *Proc. of ACM SIGIR Conference*, pp. 64–71, 2004.
[11] K. Crammer and Y. Singer, "Pranking with ranking," *the conference on Neural Information Processing Systems (NIPS)*, 2001.
[12] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *Journal of Machine Learning Research*, vol. 4, pp. 933–969, 2003.
[13] S. Fox, K. Karnawat, M. Mydland, S. Dumias, and T. White, "Evaluating implicit measures to improve web search," *ACM Trans. on Information Systems*, vol. 23, no. 2, pp. 147–168, 2005.
[14] F. Radlinski and T. Joachims, "Evaluating the robustness of learning from implicit feedback," *W4: Learing in Web Search, at 22nd Intl. Conf. on Machine Learning*, 2005.
[15] K. Jarvelin and J. Kekalainen, "Ir evaluation methods for retrieving highly relevant documents," *Proc. of ACM SIGIR Conference*, 2000.
[16] T. Hastie, R. Tibshirani, and J. Friedmann, *The Elements of Statistical Learning*. Springer-Verlag, 2001.
[17] W. Gardner, "Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique," *Signal Processing*, vol. 6, no. 2, pp. 113–133, 1984.
[18] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. New York City, NY: Addison Wesley, 1999.