SIGIR 2008 Workshop

Learning to Rank for Information Retrieval

held in conjunction with the 31th Annual International ACM SIGIR Conference

24 July 2008, Singapore

Organizers

Hang Li
Microsoft Research Asia
Tie-Yan Liu
Microsoft Research Asia
Chengxiang Zhai
University of Illinois at Urbana-Champaign

http://research.microsoft.com/users/LR4IR-2008/

Preface

The 2008 International Workshop on Learning to Rank for Information Retrieval (LR4IR 2008) is the second in a series of workshops on this topic held in conjunction with the Annual ACM SIGIR International Conference on Research and Development in Information Retrieval.

The main purpose of this workshop is to bring together information retrieval researchers and machine learning researchers working on or interested in the technologies of learning to rank, and let them share their latest research results, express their opinions on the related issues, and discuss future research directions.

Our call for papers this year has attracted many submissions. All submitted papers were thoroughly reviewed by the program committee. The program committee finally accepted 8 papers. The 8 accepted papers were divided into 3 sessions: "Learning to Rank Algorithms - I," "Learning to Rank Algorithms - II," and "Benchmark Dataset for Learning to Rank". In addition, we have a novel session named "Opinions on Learning to Rank" this year, to encourage researchers in this field to share their opinions and viewpoints on the current status and future directions of learning to rank. The LR4IR 2008 program also includes two invited talks, with one on "The Optimisation of Evaluation Metrics" by Dr. Stephen Robertson, from Microsoft Research Cambridge, and the other on "A Structured Learning Framework for Learning to Rank in Web Search" by Prof. Hongyuan Zha, from Georgia Institute of Technology.

We are grateful to the program committee members for carefully reviewing all the submissions. We also would like to thank the SIGIR 2008 main program committee for their support of this workshop and all the authors for their contributions.

Hang Li
Tie-Yan Liu
Chengxiang Zhai
Program Committee Co-Chairs

Table of Contents

Session 2: Learning to Rank Algorithms - II

Chair: Chengxiang Zhai *(University of Illinois at Urbana-Champaign)*

Session 3: Benchmark Dataset for Learning to Rank

Chair: Hang Li *(Microsoft Research Asia)*

Session 4: Opinions on Learning to Rank

Chair: Hang Li *(Microsoft Research Asia)*

LR4IR 2008 Organization

Program Co-Chair:
Hang Li, *(Microsoft Research Asia)*
Tie-Yan Liu, *(Microsoft Research Asia)*
Chengxiang Zhai, *(University of Illinois at Urbana-Champaign)*

Program Committee:
Alekh Agarwal, *(University of California at Berkeley)*
Djoerd Hiemstra, *(University of Twente)*
Donald Metzler, *(Yahoo! Research)*
Einat Minkov, *(Carnegie Mellon University)*
Filip Radlinski, *(Cornell University)*
Guirong Xue, *(Shanghai Jiao-Tong University)*
Guy Lebanon, *(Prudue University)*
Hongyuan Zha, *(Georgia Institute of Technology)*
Hsin-Hsi Chen, *(National University of Taiwan)*
Irina Matveeva, *(University of Chicago)*
Javed Aslam , *(Northeastern University)*
John Guiver, *(Microsoft Research Cambridge)*
Jun Xu, *(Microsoft Research Asia)*
Kai Yu, *(NEC Research Institute)*
Michael Taylor, *(Microsoft Research Cambridge)*
Olivier Chapelle, *(Yahoo Research)*
Ping Li, *(Cornell University)*
Quoc Le, *(Australian National University)*
Ralph Herbrich, *(Microsoft Research Cambridge)*
Ravi Kumar, *(Yahoo Research)*
Tao Qin, *(Tsinghua University)*
Yisong Yue, *(Cornell university)*
Zhaohui Zheng, *(Yahoo Research)*
Soumen Chakrabarti, *(Indian Institute of Technology)*

Keynote Address

On the Optimisation of Evaluation Metrics

Stephen Robertson
Microsoft Research Cambridge
7 JJ Thomson Avenue Cambridge, U.K.
ser@microsoft.com

Abstract

The usual approach to optimisation, of ranking algorithms for search and in many other contexts, is to obtain some training set of labeled data and optimise the algorithm on this training set, then apply the resulting model (with the chosen optimal parameter set) to the live environment. (There may be an intermediate test stage, but this does not affect the present argument.) This approach involves the choice of a metric, in this context normally some particular IR effectiveness metric. It is commonly assumed, overtly or tacitly, that if we want to optimise a particular evaluation metric M for a live environment, we should try to optimise exactly the metric M on the training set (even though in practice we often use an approximation or other substitute measure). When the assumption is stated explicitly, it is sometimes presented as self-evident. In this paper I will explore some reasons why the assumption might not be a good general rule.

Bio

Stephen Robertson is a researcher at the Microsoft Research Laboratory in Cambridge, UK. He retains a part-time professorship in the Department of Information Science of the City University. He was full-time at City University from 1978 to 1998, and started the Centre for Interactive Systems Research. His research interests are in theories and models for information retrieval and the design and evaluation of IR systems. In 1976, he was the author (with Karen Sparck Jones) of a moderately influential probabilistic theory of relevance weighting; further work (with Stephen Walker, on the Okapi system) led to the BM25 function for term weighting and document scoring, now used by many other groups. He is a Fellow of Girton College, Cambridge; he was awarded the Tony Kent Strix award in 1998 and SIGIR's Gerard Salton award in 2000.

Keynote Address

A Structured Learning Framework for Learning to Rank in Web Search

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

Abstract

Evaluation metrics are an essential part of a ranking system for Web search. We consider an evaluation metric as a form of utility function which reflects the degree of satisfaction of the users when presented a list of ranked documents in response to a user query. It is critical for Web search that we design evaluation metrics that can accurately capture the quality of search result sets in terms of relevancy, diversity and novelty. We argue that the evaluation metrics themselves should be learned from judgment data and user interaction data; and learning to rank algorithms should seek to optimize the evaluation metrics through structured learning methodology. We will illustrate the above ideas using preference learning from click data and learning the gain values and discount factors in NDCG as examples.

Bio

Hongyuan Zha received his B.S. degree in mathematics from Fudan University in Shanghai in 1984, and his Ph.D. in scientific computing from Stanford University in 1993. He was a faculty member of the Department of Computer Science and Engineering at Pennsylvania State University from 1992 to 2006, and he also worked from 1999 to 2001 at Inktomi Corporation. He is currently a professor in College of Computing at Georgia Institute of Technology. His research interests include Web search and machine learning applications.

SortNet: Learning To Rank By a Neural-Based Sorting Algorithm

Leonardo Rigutini, Tiziano Papini, Marco Maggini, Franco Scarselli
Dipartimento di Ingegneria dell'Informazione
via Roma 56, Siena, Italy
{rigutini,papinit,maggini,franco}@dii.unisi.it

ABSTRACT

The problem of relevance ranking consists of sorting a set of objects with respect to a given criterion. Since users may prefer different relevance criteria, the ranking algorithms should be adaptable to the user needs. Two main approaches exist in literature for the task of learning to rank: 1) a score function, learned by examples, which evaluates the properties of each object yielding an absolute relevance value that can be used to order the objects or 2) a pairwise approach, where a "preference function" is learned using pairs of objects to define which one has to be ranked first. In this paper, we present SortNet, an adaptive ranking algorithm which orders objects using a neural network as a comparator. The neural network training set provides examples of the desired ordering between pairs of items and it is constructed by an iterative procedure which, at each iteration, adds the most informative training examples. Moreover, the comparator adopts a connectionist architecture that is particularly suited for implementing a preference function. We also prove that such an architecture has the universal approximation property and can implement a wide class of functions. Finally, the proposed algorithm is evaluated on the LETOR dataset showing promising performances in comparison with other state of the art algorithms.

1. INTRODUCTION

The standard classification or regression tasks do not include all the supervised learning problems. Some applications require to focus on other computed properties of the items, rather than values or classes. For instance, in ranking tasks, the score value assigned to each object is less important than the ordering induced on the set of items by the scores. In other cases, the main goal is to retrieve the top k objects without considering the ordering for the remaining items. The differences among these classes of problems influences the properties of that we would like to predict, the representation of the patterns and the type of the available supervision. For example, when an user indicates that an object is to be preferred with respect to another, or that two objects should be in the same class, he/she does not assign a value to the objects themselves. In these cases, the given examples are in the form of relationships on pairs of objects and the supervision values are the result of a preference or similarity function applied on the pair of items. Two of these peculiar supervised learning tasks are *preference learning* and *learning to rank*. In the machine learning literature, preference learning problems can be categorized into two specific cases, the *Learning Objects Preference* and the *Learning Labels Preference* formulations as reviewed in [7]. In the learning objects preferences scenario, it is supposed to have a collection of instances x_i with associated a total or partial ordering. The goal of the training is to learn a function that, given a pair of objects, correctly predicts the associated preference as provided by the available ordering. In this approach, the training examples consist of preferences between pairwise instances, while the supervision label consist of the preference expressed by the user on the given pair: $x_i \succ x_j$ if x_i is to be preferred to x_j, $x_i \prec x_j$ vice versa. This approach is known as *pairwise preference learning* since it is based on pairs of objects.

On the other hand, the task of relevance ranking consists of sorting a set of objects with respect to a given criterion. In learning to rank, the criterion is not predefined, but it has to be adapted to the users' needs. The two research areas of preference learning and learning to rank have shown many interactions. In particular, the approach of Herbrich et al. in [8], which is based on a binary classifier, is considered the first work on preference learning and learning to rank. Recently, an increasing number of new algorithms have been proposed to learn a scoring function for ranking objects. Freund et al. [5] proposed RankBoost, an algorithm based on a collaborative filtering approach. Burges et al. [3] used a neural network to model the underlying ranking function (RankNet). Similarly to the approach proposed in this paper, it uses a gradient descent technique to optimize a probabilistic cost function, the cross entropy. The neural network is trained on pairs of training examples using a modified backpropagation algorithm. It differs from the method proposed in this paper for the weight-sharing scheme and for the training set construction procedure. In [1], the authors use a pairwise learning approach to train a SVM model (SVMRank), while AdaRank [13] uses an AdaBoost-based scheme to learn the preference function for ranking. Finally, Zhe Cao et al. proposed ListNet [2], that, for the first time, extends the pairwise approach to a listwise approach. In the latter approach, lists of objects are used as instances for learning.

In this paper we propose SortNet, a ranking algorithm that orders objects using a neural network as a "comparator". The neural network is trained by examples to learn a comparison function that specifies for each pair of objects which is the preferred one. The network is embedded into a sorting algorithm to provide the ranking of a set of objects.

The comparator adopts a particular neural architecture that allows us to implement the symmetries naturally present in a preference function. The approximation capability of this architecture has been studied proving that the comparator is an universal approximator and it can implement a wide class of functions. The comparator is trained by an iterative procedure, which aims at selecting the most informative patterns in the training set. In particular, at each iteration, the neural network is trained only a subset of the original training set. This subset is enlarged at each step by including the miss-classified patterns. The procedures selects the comparator that obtains the best performance on the validation set during the learning procedure.

The proposed approach is evaluated using the LETOR (LEarning TO Rank) dataset [10], which is a standard benchmark for the task of learning to rank. The comparison considers several state-of-the-art ranking algorithms, such as RankSVM [1], RankBoost [5], FRank [12], ListNet [2], and AdaRank [13]. The paper is organized as follows. In the next section, we introduce the neural network model along with a brief mathematical description of its properties. In Section 3, the whole SortNet algorithm based on the comparator is defined. In Section 4, we present the experimental setup, the LETOR dataset, and some comparative experimental results. Finally, in Section 5, some conclusions are drawn.

2. THE NEURAL COMPARATOR

In the following, S is a set of objects described by a vector of features. We assume that a preference relationship exists between the objects and it is represented by the symbols $\succ$ and $\prec$. Thus, $x \succ y$ means that x is preferred to y while $x \prec y$ that y is preferred to x. The purpose of the proposed method is to learn by examples the partial order specified by the preference relationship. The main idea is that of designing a neural network N that processes a representation of two objects x, y and produces an estimate of $P(x \succ y)$ and $P(x \prec y)$. We will refer to the network as the "comparator". More formally, we will have

$$\begin{aligned} N_{\succ}(<x,y>) &\approx P(x \succ y) \\ N_{\prec}(<x,y>) &\approx P(x \prec y), \end{aligned}$$

where $<x,y>=[x_1,\ldots,x_d,y_1,\ldots,y_d]$ is the concatenation of the feature vectors of objects x, y and $N_{\succ}, N_{\prec}$ denote the two network outputs. Since the neural network approximates a preference function, it is naturally to enforce the following constraints on the outputs:

$$N_{\succ}(<x,y>) = N_{\prec}(<y,x>)\ . \tag{1}$$

Equation (1) suggests that the outputs $N_{\succ}$ and $N_{\prec}$ must be symmetrical with respect to the order of the examples in the input pair. The comparator consists of a feedforward neural network with one hidden layer, two outputs, implementing $N_{\succ}$ and $N_{\prec}$, respectively, and $2d$ input neurons, where d is the dimension of the object feature vectors (see Figure 1). Let us assume that $v_{x_k,i}$ ($v_{y_k,i}$) denotes the weight of the connection from the input node x_k (y_k) [1] to the i-th hidden node, $w_{i,\succ}$, $w_{i,\prec}$ represent the weights of the connections from the i-th hidden to the output nodes, b_i is the bias of i-th hidden and $b_{\succ}, b_{\prec}$ are the output biases. The network

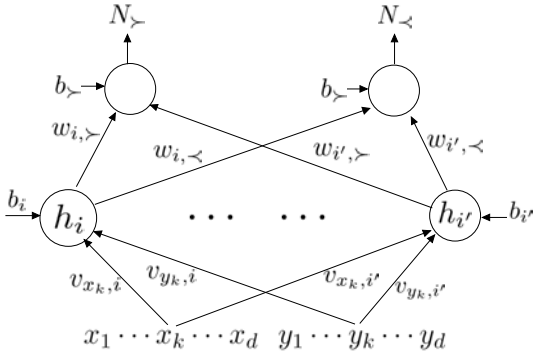


Figure 1: The comparator network architecture

architecture adopts a weight sharing mechanism in order to ensure that the constraint (1) holds. For each hidden neuron i, a dual neuron i' exists whose weights are shared with i according to the following schema:

1. $v_{x_k,i'} = v_{y_k,i}$ and $v_{y_k,i'} = v_{x_k,i}$ hold, i.e., the weights from x_k, y_k to i are inverted in the connections to i';
2. $w_{i',\succ} = w_{i,\prec}$ and $w_{i',\prec} = w_{i,\succ}$ hold, i.e., the weights of the connections from hidden i to outputs $\succ, \prec$ are inverted in the connections leaving from i';
3. $b_i = b_{i'}$ and $b_{\succ} = b_{\prec}$ hold, i.e., the biases are shared between the dual hiddens i and i' and between the outputs $\succ$ and $\prec$.

In order to study the properties of the above described architecture, let us denote by $h_i(<x,y>)$ the output of the i-th hidden neuron when the network is feeded on the pair $<x,y>$. Then, using the weight–sharing rule in point 1, we have

$$\begin{aligned} h_i(<x,y>) &= \sigma\left(\sum_k (v_{x_k,i}\, x_k + v_{y_k,i}\, y_k) + b_i\right) \\ &= \sigma\left(\sum_k (v_{x_k,i'}\, y_k + v_{y_k,i'}\, x_k) + b_i\right) \\ &= h_{i'}(<y,x>) \end{aligned}$$

where σ is the activation function of hidden units. Moreover, let $N_{\succ}(<x,y>)$ and $N_{\prec}(<x,y>)$ represent the network outputs. Then, by the rules in points 2 and 4, it follows

$$\begin{aligned} & N_{\succ}(<x,y>) = \\ & = \sigma\left(\sum_{i,i'} (w_{i,\succ}\, h_i(<x,y>) + w_{i',\succ}\, h_{i'}(<x,y>) + b_{\succ}\right) \\ & = \sigma\left(\sum_{i,i'} (w_{i',\prec}\, h_{i'}(<y,x>) + w_{i,\prec}\, h_i(<y,x>) + b_{\prec}\right) \\ & = N_{\prec}(<y,x>)\,, \end{aligned}$$

[1] Here, with an abuse of notation, x_k represents the node that is feeded with the k-th feature of x.

where we applied the fact that $h_i(<x,y>) = h_{i'}(<y,x>)$ as shown before. Thus, $N_\succ(<x,y>) = N_\prec(<y,x>)$ holds, which proves that the constraint of equation (1) is fulfilled by the network output.

Approximation capability

It has been proved that three layered networks are universal approximators [9, 6, 4]. Similarly, it can be shown that the neural comparator described in this paper can approximate up to any degree of precision most of the practically useful functions that satisfy the constraint (1). Formally, let $\mathcal{F}$ be a set of functions $f : R^n \to R^m$, $\|\cdot\|$ be a norm on $\mathcal{F}$ and σ be an activation function. The universal approximation property, proved for three layered networks, states that for any function $f \in \mathcal{F}$ and any real $\varepsilon > 0$, there exists a network that implements a function $N : R^n \to R^m$ such that $\|f - N\| \leq \varepsilon$ holds. Different versions of this property have been proved, according to the adopted function set $\mathcal{F}$ (e.g., the set of the continuous or the measurable functions), the norm $\|\cdot\|$ (e.g., the infinity norm or a probability norm) and the activation function σ (e.g., a sigmoidal or a threshold activation function) [11]. The following theorem demonstrates that the network with the proposed weight sharing mantains the universal approximation property provided that we restrict the attention to the functions that satisfy the constraint (1).

THEOREM 2.1. *Let $\mathcal{F}$ be a set of functions $f : R^{2d} \to R^2$, $\|\cdot\|$ be a norm on $\mathcal{F}$ and σ be an activation function such that the corresponding three layered neural network class has the universal approximation property on $\mathcal{F}$. Let us denote by $\overline{\mathcal{F}}$ the set of the functions k that belongs to $\mathcal{F}$ and, for any $<x,y>$, fulfill*

$$k_\succ(<x,y>) = k_\prec(<y,x>) \tag{2}$$

where $k_\succ, k_\prec$ denote the two components of the outputs of k. Then, for any function $f \in \overline{\mathcal{F}}$ and any real $\varepsilon > 0$, there exists a three layered neural network satisfying the weight sharing schema defined in points 1-4 such that

$$\|f - h\| \leq \varepsilon$$

holds, where $h : R^{2d} \to R^2$ is the function implemented by the neural network.

PROOF. (sketch)
By the universal approximation hypothesis, there exists a three layered neural network $\mathcal{A}$ that implements a function $r : R^{2d} \to R^2$ such $\|\frac{f}{2} - r\| \leq \frac{\varepsilon}{2}$, where f and ε are the function and the real of the hypothesis, respectively. Then, we can construct another network $\mathcal{B}$ that has twice the hidden nodes of $\mathcal{A}$. The indexes of hidden nodes of $\mathcal{B}$ are partitioned into pairs i, i'. The neurons with index i are connected to the input and to the outputs with the same weights as in $\mathcal{A}$: in other words, $\mathcal{B}$ contains $\mathcal{A}$ as a sub–network. On the other hand, the weights of the hidden neurons with index i' are defined following the weight sharing rules in points 1,2, and 3. Finally, the biases of the outputs nodes in $\mathcal{B}$ are set to twice the values of the biases in $\mathcal{A}$.

Notice that, by construction, the network $\mathcal{B}$ satisfies all the rules of points 1-4 and it is a good candidate to be the network of the thesis. Moreover, $\mathcal{B}$ is composed by two sub–networks, the sub-network identified by the hiddens i and the sub–network identified by the hiddens i'. Let p^1, p^2 represent the functions that define the contribution to the output by the former and the latter sub-networks, respectively. Then, we can easily prove that p_1 produces a contribution to output which is equal to $r(<x,y>)$, i.e., $p^1_\succ(<x,y>) = r_\succ(<x,y>)$ and $p^1_\prec(<x,y>) = r_\prec(<x,y>)$. On the other hand, p^2 has a symmetrical behaviour with respect to r due to the weight–sharing schema, i.e., $p^2_\succ(<x,y>) = r_\prec(<y,x>)$ and $p^2_\prec(<x,y>) = r_\succ(<y,x>)$. Since, the output function h implemented by $\mathcal{B}$ is given by the sum of the two components, then

$$\begin{aligned} h_\succ(<x,y>) &= p^1_\succ(<x,y>) + p^2_\succ(<x,y>) = \\ &= r_\succ(<x,y>) + r_\prec(<y,x>) = 2r_\succ(<x,y>) \end{aligned}$$

$$\begin{aligned} h_\prec(<x,y>) &= p^1_\prec(<x,y>) + p^2_\prec(<x,y>) = \\ &= r_\prec(<x,y>) + r_\succ(<y,x>) = 2r_\prec(<x,y>) \end{aligned}$$

where we have used $r_\succ(<x,y>) = r_\prec(<y,x>)$ that holds by definition of r and f. Then, the thesis follows straightforwardly by

$$\|f - h\| = 2\left\|\frac{f}{2} - \frac{h}{2}\right\| = 2\left\|\frac{f}{2} - r\right\| \leq \varepsilon$$

□

The training and the test phase

To train the comparator, a learning algorithm based on gradient descent is used. For each pair of inputs $<x,y>$, the assigned target is

$$t = \begin{cases} [1\ 0] & \text{if } x \succ y \\ [0\ 1] & \text{if } x \prec y \end{cases} . \tag{3}$$

and the error is measured by the squared error function

$$E(<x,y>) = (t_1 - N_\succ(<x,y>))^2 + (t_2 - N_\prec(<x,y>))^2 .$$

After training, the comparator can be used to predict the preference relationship between any pair of objects. Formally, we can define $\succ, \prec$ by

$$\begin{aligned} x \succ y \quad &\text{if} \quad N_\succ(<x,y>) > N_\prec(<x,y>) \\ x \prec y \quad &\text{if} \quad N_\prec(<x,y>) > N_\succ(<x,y>) . \end{aligned}$$

Notice that this approach cannot ensure that the predicted relationship defines a total order, since the transitivity property may not hold

$$\text{Transitivity: } x \succ y \text{ and } y \succ z \Longrightarrow x \succ z .$$

However, the experimental results show that the neural network can easily learn the transitivity relation provided that the training set supports its validity.

3. THE SORTING ALGORITHM

The neural comparator is employed to provide a ranking of a set of objects. In particular, the objects are ranked by a common sorting algorithm that exploits the neural network as a comparison function. In this case, the time computational cost of the ranking is mainly due to the sorting algorithm, so that the objects can be ranked in $O(n \log n)$.

It is worth mentioning that the obtained ranking may depend on the initial order of the objects and on the adopted sorting algorithm, since we cannot ensure that the relation $\succ$ is transitive. However, in our experiments, the same sorting algorithm was applied to different shufflings of the same objects and different sorting algorithms were used on the same set of objects. The obtained orderings were the same in most of the cases and seldom they differed only by a very small number of object positions ($1-5$ over 1000).

3.1 The incremental learning procedure

Let us assume that a function *RankQuality* is available to measure the quality of a given ranking. Some possible choices for this measure are described in the following subsection. It is worth noticing that the quality of a given ranked list roughly depends on how many pairs of objects are correctly ordered. In fact, the comparator neural network is trained using the square error function E, that forces the network outputs to be close to the desired targets. When the comparator produces a perfect classification of any input pair, also the ranking algorithm yields a perfect sorting of the objects. However, in general, the optimization of the square error does not necessarily correspond to a good ranking if the example pairs in the training set are not properly chosen.

In order to optimize the selection of the training pairs with the aim of improving the ranking performance of the algorithm by using a minimal number of training examples, we defined an incremental learning procedure to train the comparator. This procedure aims at constructing the training set incrementally by selecting only the pairs of objects that are likely to be relevant for the goal of obtaining a good ranking. This technique allows us to optimize the size of the training set avoiding the need to consider all the possible pairs of the available objects. This method is somehow related to active learning algorithms, where the learning agent is allowed to prompt the supervisor for providing the most promising examples.

Given the set of objects T and V, that can be used as training and validation sets, respectively, at each iteration i a comparator C^i is trained using two subsets $TP \subset T \times T \times \{\prec, \succ\}$, $VP \subset V \times V \times \{\prec, \succ\}$ that contain the current training and validation pairs of the form $x \succ y$ or $x \prec y$. In particular TP contains the pairs that are used to train the comparator, while VP is used as validation set for the neural network training procedure. The trained comparator neural network C^{i+1} is used to sort the objects in T and V and the objects pairs that have been mis-compared by the comparator when producing the R^i_T are added to the subsets TP, VP obtaining the training and validation sets for the next iteration. The procedure is repeated until a maximum number of iterations is reached or until there is no difference in the sets TP and VP between two consecutive iterations. The output of the incremental training is the comparator network C^* that yields the best performance on the validation set during the iterations.

The algorithm is formally described in figure 3.1. At the beginning, the neural comparator is randomly initialized and TP and VP are empty. At each iteration i, the whole labeled object set T is ranked using the comparator C^i (line 7). The sorting algorithm returns R^i_T, that is the ranking of the training examples and TP_i, the set of the objects pairs that have been mis-compared by the comparator used in the sorting algorithm to produce R^i_T. In detail, the sorting algorithm employs C^i in $|T| \times \log(|T|)$ pairwise comparisons to produce R^i_T, where $|T|$ indicates the size of T: the objects pairs for which the C^i output differs from the known correct order are inserted in TP_i. The known relative position for the two objects in the pair is available in the set T by exploiting the fact that relevant objects should precede not relevant objects. Subsequently, in a similar way, C^i is also used to rank the validation set (line 8), producing a ranking R^i_V and a set of misclassified pairs VP_i. Then, the set of misclassified pairs TP_i are added to the current learning set TP, removing the eventual duplicates (line 14). Similarly, the pairs in the set VP_i are inserted into VP (line 15). A new neural network comparator C^{i+1} is trained at each iteration using the current training and validation sets, TP and VP (line 16). More precisely, the neural network training lasts for a predefined number of epochs on the training set TP. The selected comparator is the one that achieves the best result, over all the epochs, on the validation set VP.

Algorithm 3.1 The SortNet algorithm

```
1: T ← Set of training objects
2: V ← Set of validation objects
3: C^0 ← randomInit();
4: TP ← {};
5: VP ← {};
6: for i = 0 to max_iter do
7:    [TP_i , R^i_T] ← Sort(C^i, T);
8:    [VP_i , R^i_V] ← Sort(C^i, V);
9:    score ← RankQuality(R^i_V);
10:   if score > best_score then
11:      best_score ← score;
12:      C* ← C^i;
13:   end if
14:   if TP_i ⊆ TP and VP_i ⊆ VP then
15:      return C*;
16:   end if
17:   TP ← TP ∪ TP_i;
18:   VP ← VP ∪ VP_i;
19:   C^{i+1} ← TrainAndValidate(TP, VP);
20: end for
21: return C*;
```

3.2 Measures of ranking quality

At each iteration, the quality of the ranking R^i_V over the validation set is evaluated by the *RankQuality* function and the model achieving the best performance (C^*) is stored (lines $9-13$). In this work, the following three ranking measures, proposed in the LETOR report [10], were used.

- **Precision at position n (P@n)** — This value measures the relevance of the top n results of the ranking list with respect to a given query.

$$P@n = \frac{\text{relevant docs in top } n \text{ results}}{n}$$

- **Mean average precision (MAP)** — Given a query

q, the average precision is

$$AP_q = \frac{\sum_{n=1}^{N_q} P@n \cdot rel(n)}{\text{total relevant docs for } q}$$

where N_q is the number of documents in the result set of q and $rel(n)$ is 1 if the n-th document in the ordering is relevant and 0 otherwise. Thus, AP_q averages the values of $P@n$ over the positions n of the relevant documents. Finally, the MAP value is computed as the mean of AP_q over the set of all queries.

- **Normalized discount cumulative gain (NDCG@n)** — This measure exploits an esplicit rating of the documents in the list. The NDCG value of a ranking list at position n is calculated as

$$NDCG@n \equiv Z_n \sum_{j=1}^{n} \frac{2^{r_j} - 1}{log(1+j)}$$

where r_j is the rating of the j-th document[2], and Z_n is a normalization factor chosen such that the ideal ordering (the DGC-maximizing one) gets a NDCG@n score of 1.

The measure can be choosen according to the user requirements. For example, if an information retrieval system displays only the best k documents, then $P@k$ might be the preferred measure. If, on the other hand, the user is interested on an overall precision of the results, then MAP might be the best choice. Obviously, the selection of a measure to implement *RankQuality* affects the performance of the model on the test set: for example, by selecting the $P@k$ measure, we will obtain a model producing high values of $P@k$ but not optimal values of MAP.

4. EXPERIMENTAL RESULTS

The proposed approach has been validated on two benchmarks, TD2003 and TD2004, which are included in the LETOR dataset [10] (LEarning TO Rank)[3]. The benchmarks contain query-document pairs collected from TREC (Text REtrieval Conference). TD2003 consists of 50 sets of documents, each one containing 1000 documents returned in response to a query. Similarly, TD2004 contains 75 sets of documents corresponding to different queries. The datasets are partitioned into five subsets in order to allow a comparison of different approaches by 5-fold cross-validation: in each experiment, one of the subsets is used for the test, one for validation purposes and three subsets for the training procedure. Each query-document pair is represented by 44 values which include several features commonly used in information retrieval. The features also contain a label that specifies whether the document is relevant R or not relevant NR with respect to the query. For all queries, the relevant documents are roughly 1% of whole set of documents.

In the reported experiments, the training set was built by considering all the pairs $<x, y>$ of documents, where x belongs to one of the two relevance classes (R and NR) and y belongs to the other class. The corresponding targets were

$$t = \begin{cases} [1\ 0] \text{ if } x \in R \text{ and } y \in NR \\ [0\ 1] \text{ if } y \in R \text{ and } x \in NR \end{cases}$$

The features were normalized to the range $[-1, +1]$ with zero mean. More precisely, let N_i be the number of documents returned in response to the i-th query, and let us denote by $\hat{x}^i_{j,r}$ the normalized r-th feature of the j-th document of the i-th query, then,

$$\hat{x}^i_{j,r} = \frac{x^i_{j,r} - \mu^i_r}{max_{s=1,\ldots,n_i} |x^i_{s,r}|}\,, \tag{4}$$

where $x^i_{j,r}$ is the original document feature and $\mu^i_r = \frac{\sum_{s=1}^{N_i} x^i_{s,r}}{N_i}$.

In the set of experiments, the documents of the datasets were ranked using the SortNet algorithm. In this setting, a good ranking is characterized by the presence of the relevant documents in the top positions and it was evaluated by the three measures P@n, MAP and NDCG@n. The results obtained by our approach were compared to those achieved by the methods reported in [10], i.e., RankSVM [1], RankBoost [5], FRank [12], ListNet [2] and AdaRank [13].

As first trial, we tested the SortNet algorithm varying the number of hidden neurons in $[10, 20, 30]$ to select the best architecture for the TD2003 and for the TD2004 datasets. In this first set of experiments, the MAP score was used as *RankQuality* function (line 9 in algorithm 3.1). Table 1 reports the performances of the three architectures on the test set. The results on the validation set reflect the performances reported in Table 1 on the test set. Thus, for the subsequent experiments, we selected a 10-hidden comparator for the TD2003 dataset and a 20-hidden comparator for the TD2004 dataset.

After the selection of the best neural network architecture of the comparator, we performed a set of experiments to compare the performances of SortNet with the methods reported in [10]. In particular, we ran the algorithm on the TD2003 and TD2004 datasets setting $max\_iter = 20$ and using MAP and $P@10$ as *RankQuality* function. Figure 2 and the Tables 2 (a-b) show that, on the TD2004 dataset, the SortNet algorithm clearly outperforms all the other methods. On TD2003, the SortNet method reports values of MAP and $P@n$ similar to the results of the AdaRank and RankBoost. In the figure 3, we report the plot of the MAP value on the validation set at each training iteration: this plot clearly shows that convergence is reached before the maximum number of iterations.

During the experiments on the TD2003 dataset, however, we noticed a particular behaviour of the algorithm: at each iteration, the performances on the validation set sligthly differed from the performances on the training set and test set. In particular, when the algorithm reported high values for MAP and $P@n$ for the validation set, the MAP and $P@n$ on the test set and the training set were low, and viceversa. This could mean that the distribution of data in the validation clearly differs from the test set and from the training set. This observation is also supported by the fact that in the TD2003 experiments, the neural architecture reporting the best performances has a smaller number of neurons than

[2]Here, it is assumed that $r_j \geq 0$ and smaller values indicate less relevance.

[3]The dataset was released by Microsoft Research Asia and is available on line at http://research.microsoft.com/users/LETOR/

(a) NDCG@n

TD2003	**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
SortNet 10 Hiddens	**0,34**	**0,36**	**0,35**	**0,33**	**0,33**	**0,33**	**0,34**	**0,34**	**0,33**	**0,34**
SortNet 20 Hiddens	0,3	0,31	0,3	0,3	0,29	0,28	0,28	0,27	0,27	0,27
SortNet 30 Hiddens	0,36	0,29	0,29	0,28	0,27	0,27	0,28	0,27	0,27	0,28
TD2004	**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
SortNet 10 Hiddens	0,47	0,49	0,47	0,47	0,47	0,47	0,48	0,48	0,49	0,49
SortNet 20 Hiddens	**0,47**	**0,54**	**0,54**	**0,52**	**0,52**	**0,53**	**0,53**	**0,54**	**0,54**	**0,55**
SortNet 30 Hiddens	0,47	0,5	0,5	0,49	0,48	0,49	0,49	0,5	0,5	0,5

(b) P@n

TD2003	**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
SortNet 10 Hiddens	**0,34**	**0,34**	**0,29**	**0,25**	**0,24**	**0,23**	**0,23**	**0,21**	**0,2**	**0,19**
SortNet 20 Hiddens	0,3	0,3	0,27	0,25	0,22	0,2	0,19	0,17	0,16	0,15
SortNet 30 Hiddens	0,36	0,28	0,27	0,23	0,21	0,19	0,19	0,18	0,17	0,17
TD2004	**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
SortNet 10 Hiddens	0,47	0,45	0,39	0,37	0,35	0,33	0,32	0,3	0,28	0,27
SortNet 20 Hiddens	**0,47**	**0,5**	**0,46**	**0,42**	**0,38**	**0,38**	**0,35**	**0,33**	**0,32**	**0,3**
SortNet 30 Hiddens	0,47	0,47	0,42	0,38	0,35	0,34	0,32	0,31	0,28	0,26

(c) MAP

TD2003	**MAP**
SortNet 10 Hiddens	**0,23**
SortNet 20 Hiddens	0,2
SortNet 30 Hiddens	0,21
TD2004	**MAP**
SortNet 10 Hiddens	0,41
SortNet 20 Hiddens	**0,45**
SortNet 30 Hiddens	0,41

Table 1: The results achieved on TREC2003 and TD2004 varying the hidden neuron number: (a) NDCG@n, (b) P@n and (c) MAP. The algorithm uses MAP as $RankQuality$ function.

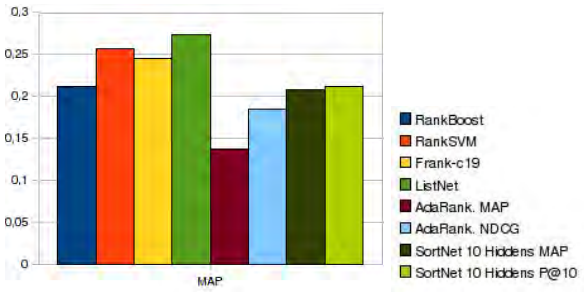
(a) TREC2003

NDCG	**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
RankBoost	0,26	0,28	0,27	0,27	0,28	0,28	0,29	0,28	0,28	0,29
RankSVM	0,42	0,37	0,38	0,36	0,35	0,34	0,34	0,34	0,34	0,34
Frank-c19.0	0,44	0,39	0,37	0,34	0,33	0,33	0,33	0,33	0,34	0,34
ListNet	0,46	0,43	0,41	0,39	0,38	0,39	0,38	0,37	0,38	0,37
AdaRank. MAP	0,42	0,32	0,29	0,27	0,24	0,23	0,22	0,21	0,2	0,19
AdaRank. NDCG	0,52	0,41	0,37	0,35	0,33	0,31	0,3	0,29	0,28	0,27
SortNet 10 Hiddens MAP	0,38	0,3	0,28	0,29	0,29	0,3	0,29	0,29	0,29	0,28
SortNet 10 Hiddens P@10	0,32	0,32	0,31	0,31	0,3	0,3	0,29	0,3	0,31	0,31
P@n	**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
RankBoost	0,26	0,27	0,24	0,23	0,22	0,21	0,21	0,19	0,18	0,18
RankSVM	0,42	0,35	0,34	0,3	0,26	0,24	0,23	0,23	0,22	0,21
Frank-c19	0,44	0,37	0,32	0,26	0,23	0,22	0,21	0,21	0,2	0,19
ListNet	0,46	0,42	0,36	0,31	0,29	0,28	0,26	0,24	0,23	0,22
AdaRank. MAP	0,42	0,31	0,27	0,23	0,19	0,16	0,14	0,13	0,11	0,1
AdaRank. NDCG	0,52	0,4	0,35	0,31	0,27	0,24	0,21	0,19	0,17	0,16
SortNet 10 Hiddens MAP	0,38	0,29	0,25	0,25	0,24	0,24	0,21	0,2	0,18	0,17
SortNet 10 Hiddens P@10	0,32	0,31	0,29	0,27	0,25	0,23	0,21	0,21	0,2	0,2

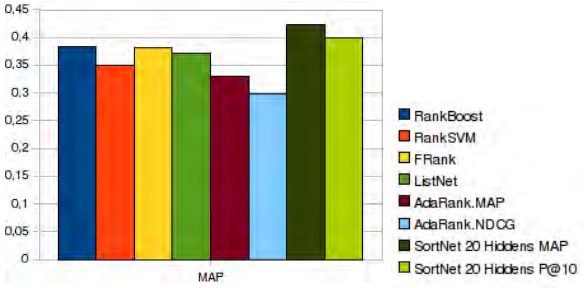
(b) TREC2004

NDCG	**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
RankBoost	0,48	0,47	0,46	0,44	0,44	0,45	0,46	0,46	0,46	0,47
RankSVM	0,44	0,43	0,41	0,41	0,39	0,4	0,41	0,41	0,41	0,42
FRank	0,44	0,47	0,45	0,43	0,44	0,45	0,46	0,45	0,46	0,47
ListNet	0,44	0,43	0,44	0,42	0,42	0,42	0,43	0,45	0,46	0,46
AdaRank.MAP	0,41	0,39	0,4	0,39	0,39	0,4	0,4	0,4	0,4	0,41
AdaRank.NDCG	0,36	0,36	0,38	0,38	0,38	0,38	0,38	0,38	0,39	0,39
SortNet 20 Hiddens MAP	**0,48**	**0,52**	**0,53**	**0,5**	**0,5**	**0,5**	**0,49**	**0,5**	**0,51**	**0,51**
SortNet 20 Hiddens P@10	0,43	0,5	0,47	0,47	0,46	0,47	0,47	0,48	0,48	0,49
P@n	**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
RankBoost	0,48	0,45	0,4	0,35	0,32	0,3	0,29	0,28	0,26	0,25
RankSVM	0,44	0,41	0,35	0,33	0,29	0,27	0,26	0,25	0,24	0,23
FRank	0,44	0,43	0,39	0,34	0,32	0,31	0,3	0,27	0,26	0,26
ListNet	0,44	0,41	0,4	0,36	0,33	0,31	0,3	0,29	0,28	0,26
AdaRank.MAP	0,41	0,35	0,34	0,3	0,29	0,28	0,26	0,24	0,23	0,22
AdaRank.NDCG	0,36	0,32	0,33	0,3	0,28	0,26	0,24	0,23	0,22	0,21
SortNet 20 Hiddens MAP	**0,48**	**0,49**	**0,46**	**0,4**	**0,36**	**0,34**	**0,3**	**0,29**	**0,28**	**0,27**
SortNet 20 Hiddens P@10	0,43	0,46	0,39	0,37	0,34	0,32	0,29	0,29	0,28	0,27

Table 2: The results achieved on (a) TREC2003 and (b) TD2004.



(a) TREC2003



(b) TREC2004

Figure 2: MAP on TREC2003 and TD2004.

the best one in the TD2004.

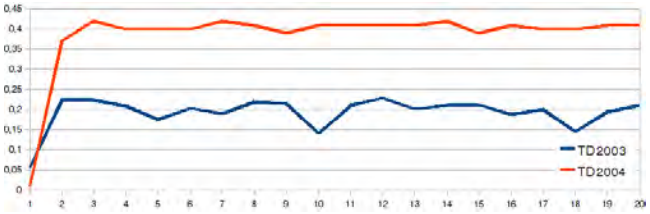


Figure 3: Trend of MAP on the validation set during the training process for the TREC2003 and TD2004 datasets.

5. CONCLUSIONS

In this paper a neural-based learning-to-rank algorithm has been proposed. A neural network is trained by examples to decide which of two objects is preferable. The learning set is selected by an iterative procedure which aims to maximize the quality of the ranking. The network adopts a weight sharing schema to ensure that the outputs satisfy logical symmetries which are desirable in an ordering relationship. Moreover, we proved that such a connectionist architecture is a universal approximator.

In order to evaluate the performances of the proposed algorithm, experiments were performed using the datasets TD2003 and TD2004. The results show that our approach outperforms the current state of the art methods on TD2004 and is comparable to other techniques on TD2003. It is also argued that a difference on the distribution of the patterns between the validation and the training and the test set of TD2003 may be responsible for some limitations on the performance of our and other algorithms.

Matters of future research include a wider experimentation of the approach and the study of different learning procedures to improve the performance of the method.

6. REFERENCES

[1] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon. Adapting ranking SVM to document retrieval. In *Proceedings of ACM SIGIR 2006*, pages 186–193, New York, NY, USA, 2006. ACM.

[2] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of ICML 2007*, pages 129–136, New York, NY, USA, 2007. ACM.

[3] E. A. M. N. C.Burges, T.Shaked and G.Hullender. Learning to rank using gradient descent. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 89–96, Madison, US, 2005.

[4] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 3:303–314, 1989.

[5] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. In J. W. Shavlik, editor, *Proceedings of ICML'98*, pages 170–178. Morgan Kaufmann Publishers, San Francisco, USA, 1998.

[6] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2:183–192, 1989.

[7] J. Furnkranz and E. Hullermeier. Preference learning. *Kunstliche Intelligenz*, 2005.

[8] R. Herbrich, T. Graepel, P. Bollmann-Sdorra, and K. Obermayer. Learning a preference relation for information retrieval. In *In Proceedings of the AAAI Workshop Text Categorization and Machine Learning*, 1998.

[9] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2:359–366, 1989.

[10] T.-Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. LETOR: Benchmarking learning to rank for information retrieval. In *SIGIR 2007 – Workshop on Learning to Rank for Information Retrieval*, Amsterdam, The Netherlands, 2007.

[11] F. Scarselli and A. C. Tsoi. Universal approximation using feedforward neural networks: a survey of some existing methods, and some new results. *Neural networks*, pages 15–37, 1998.

[12] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma. FRank: a ranking method with fidelity loss. In *Proceedings of the ACM SIGIR 2007*, pages 383–390, New York, NY, USA, 2007. ACM.

[13] J. Xu and H. Li. AdaRank: a boosting algorithm for information retrieval. In *Proceedings of ACM SIGIR 2007*, pages 391–398, New York, NY, USA, 2007. ACM.

Query-Level Learning to Rank Using Isotonic Regression

Zhaohui Zheng† Hongyuan Zha⋆ Gordon Sun†

†Yahoo! Inc.
701 First Avene
Sunnyvale, CA 94089
zhaohui,gzsun@yahoo-inc.com

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

ABSTRACT

In an IR system, 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. Most existing learning to rank methods, however, ignore the query boundaries, thus 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 the state-of-the-art methods. We demonstrate the effectiveness of the proposed method using publicly available benchmark 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, isotonic regression

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 [21, 20, 4]. 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 [19].

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 [16, 17, 27]. 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 [16, 17, 22]. 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 [26], and it is also in the same spirit as some of the recently proposed listwise learning to rank methods [8].

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 learning problem and specify an iterative method for its solution based on a minimum effort principle. 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 [12, 13, 14]. 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 [12, 13, 14]. 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 [9].

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 [16, 17, 18]. 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 [6]. RankBoost discussed in [10], 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 [7, 23, 25]; learning from pairwise preference data using gradient boosting framework [27, 28]; and extending pairwise preference learning to list learning [8]. The workshop *learning to rank for information retrieval* at SIGIR 2007 summarizes many of the recent advances in this field [19].

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, $h(x_i) < h(x_j)$. We modify the functions values at x_i from

$$h(x_i) \Rightarrow h(x_i) + \delta_i, \quad i = 1, \ldots, N$$

so that the new set of values $h(x_i) + \delta_i$ are consistent with $\mathcal{T}$, i.e.,

$$h(x_i) + \delta_i \geq h(x_j) + \delta_j, (i, j) \in \mathcal{S}.$$

We then find $g_m(x) \in \mathcal{H}$ so that $g_m(x_i) \approx \delta_i, i = 1, \ldots, N$ in the least squares sense, for example. This least square fitting can be done by using the gradient boosting trees [11].

4.2 Computing updates using isotonic regression

Generally, there are many ways to make the values $h(x_i)+\delta_i$ be consistent with $\mathcal{T}$. But large values of δ_i 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 [5]). Our basic idea is to achieve consistency with $\mathcal{T}$ with as small as possible a set of δ_i.

Recall that the set of preferences are always among documents for the same query, documents are not comparable across queries. Therefore, the computation of δ_i 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 $h_m(x_i) + \delta_i$ and compute the δ_i by solving the following optimization problem,

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

subject to

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

This quadratic programming problem is known as *isotonic regression* in the statistic literature [2]. 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

$$h_m(x_1) + \delta_1 \geq h_m(x_2) + \delta_2 \geq \cdots \geq h_m(x_n) + \delta_n,$$

(2) can be solved with computational complexity $O(n)$ using the so-called Pool-Adjacent-Violator (PAV) Algorithm [2]. 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 $O(n^2)$ [3].

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,

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

subject to

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

$$\zeta \geq 0.$$

Here ΔG_{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 ζ as a slack variable allowing softening the constraints imposed by G_{ij}; λ is the regularization parameter balancing the two requirements in the objective function. We suspect that methods in [3] 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 $O(n^3)$ [5]. 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 [11]. Once δ_i are computed, we fit a regression tree $g_m(x)$ to minimize $\sum_{i=1}^{N}(g_m(x_i) - \delta_i)^2$ [11]. 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 $\delta_i, i = 1, \ldots, N$ by solving the isotonic regression problem (3).
2. Fit a regression tree $g_m(x)$ so that $g_m(x_i) \approx \delta_i$.
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 η. 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 η 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.

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).

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

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 LETOR, which is a publicly available benchmark data collection used for comparing learning to rank algorithms [19].

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 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.

5.1 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[2]. 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.

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

5.2 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.

Precision. Given the binary relevance judgment, the precision of a ranked list is measured by the fraction of the retrieved documents that are relevant. We use precision at position n (P@n) is used to measure the quality of the top n results of the ranking list.

$$\text{P@n} = \frac{\text{No. of relevant docs in top } n \text{ results}}{n} \tag{4}$$

Mean Average Precision. The average precision of a query is the average of the precision scores after each relevant document retrieved. Formally, average precision (AP) is defined as follows,

$$\text{AP} = \frac{\sum_i \text{P@i} \times \text{rel}_i}{\text{No. of relevant documents}} \tag{5}$$

where rel_i is the indicator function whether the i-th document of the ranking list is relevant to the query. Mean Average Precision (MAP) is obtained by the mean of the average precision over a set of queries. Compared with P@n measure, the MAP score is sensitive to the entire ranking list and contains the aspects of recall as well as precision.

Normalized Discount Cumulative Gain. Since P@n and MAP are defined based on binary judgements: relevant and irrelevant. In the case of multiple levels of judgements, a more sophisticated evaluation measure called Normalized Discount Cumulative Gain (NDCG) is used [15]. Unlike P@n and MAP, NDCG has the capability to deal with multiple levels of relevance. The NDCG value of a ranking list is calculated by the following equation:

$$\text{NDCG@n} = Z_n \sum_{i=1}^{n} (2^{r_i} - 1)/\log(i+1) \tag{6}$$

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

Table 1: NDCG, MAP, and Precision at position n on OHSUMED 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	0.498	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

Table 2: 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

IsoRank on the validation set. For IsoRank, we simply set the regularization parameter λ in (4) to be 10 without much tuning.

Table 2-4 list the experimental results for IsoRank, QBrank, and other seven methods on OHSUMED, TD2003 and TD2004 respectively.[3] Overall speaking, IsoRank outperforms QBrank in more cases. Both are comparable with the other existing models.

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,

$$\min_{\delta_i} \sum_{i=1}^{n} \delta_i^2 + \lambda_1 n \zeta_1^2 + \lambda_2 n \zeta_2^2 \qquad (7)$$

subject to

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

$$|h(x_i) + \delta_i - h(x_j) - \delta_j| \leq \zeta_2, \quad (i, j) \in V_q.$$

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

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] R.E. Barlow, D.J. Bartholomew, J.M. Bremner, H.D. Brunk. *Statistical inference under order restrictions.* Wiley, New York, 1972.

[3] 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]The performance of MHR-BC on TREC2003 and TREC2004 data is missing from the Letor package.

Table 3: 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

[4] A. Berger. *Statistical machine learning for information retrieval*. Ph.D. Thesis, CMU, 2001.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

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

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

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

[14] 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.

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

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

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

[18] 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.

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

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

A Meta-Learning Approach for Robust Rank Learning

Vitor R. Carvalho, Jonathan L. Elsas, William W. Cohen and Jaime G. Carbonell
Language Technologies Institute
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA
{vitor,jelsas,wcohen,jgc}@cs.cmu.edu

ABSTRACT

Learning effective feature-based ranking functions is a fundamental task for search engines, and has recently become an active area of research [10, 3, 2]. Many of these recent algorithms are based on the pairwise preference framework, in which instead of taking documents in isolation, document pairs are used as instances in the learning process. One disadvantage of this process is that a noisy relevance judgment on a single document can lead to a large number of mis-labeled document pairs. This can jeopardize robustness and deteriorate overall ranking performance. In this paper we study the effects of outlying pairs in rank learning with pairwise preferences and introduce a new meta-learning algorithm capable of suppressing these undesirable effects. This algorithm works as a second optimization step in which any linear baseline ranker can be used as input. Experiments on eight different ranking datasets show that this optimization step produces statistically significant performance gains over various state-of-the-art baseline rankers.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Retrieval models; H.3.4 [**Systems and Software**]: Performance evaluation (efficiency and effectiveness)

General Terms

Algorithms, Design, Experimentation

Keywords

Learning to rank, Empirical Risk

1. INTRODUCTION

The "Learning to Rank" problem has gained much attention from the information retrieval research and industry communities recently. This is the problem of using queries previously submitted to a search engine and relevance information on the retrieved results to improve performance of a document ranking algorithm. One goal of this machine learning task is to automatically adjust the parameters in the ranking algorithm to return more relevant documents higher in the results list for future queries.

Much of the recent research in rank learning has focused on document ranking, but Learning to Rank (LETOR) is a widely applicable machine learning task. Ranking (or re-ranking) has long been important in machine translation [20], named-entity extraction [7] and expert finding [6] to name just a few areas. In all of these tasks, objects (named entity labels, target-language translations, candidate experts) are ranked in response to some "query" (text span, source-language sentences, keyword query).

One popular approach to learning ranking functions is to learn a *preference function* over pairs of documents given a query. This preference function indicates to which degree one document is expected to be more relevant than another with respect to the query. When these preference functions are transitive, as is typically the case, the document collection can be ranked in descending order of preference.

There is evidence that assessment of pairwise preferences is easier for assessors and yields higher inter-annotator agreement [5]. There are also many practical advantages in adopting a pairwise preference approach for automatic learning of feature-based ranking functions. First, most classification methods can be easily adapted to this formulation of the ranking problem. Second, this framework can be generalized to any graded relevance levels (e.g. definitely relevant, somewhat relevant, non-relevant). Third, in many scenarios it is easier to obtain large amounts of pairwise preference data [14].

Using pairwise preferences, however, does pose some risks. In the presence of labeling errors or other "noise" in the document relevance information, creating a training set by pairing documents causes a quadratic increase in the number of noisy outlier observations. As we will show, this can have a strong negative impact on the quality of the learned ranking function.

In this paper, we propose a two-stage optimization strategy for learning ranking functions that is robust to outliers and applicable to any method that learns a linear ranking function. This meta-ranker is computationally economical and, although developed for document ranking, this method generalizes across many ranking tasks. Experimental results on eight different ranking collections show consistent and significant improvements over a range of baseline ranking functions.

SIGIR 2008 LR4IR - Workshop on Learning to Rank for Information Retrieval Singapore

2. LEARNING TO RANK

2.1 Related Work

Learning feature-based ranking functions has become an active area of research, with the recent introduction of several new ranking algorithms. RankNet [2] learns a ranking function by minimizing the number of mis-ranked pairs via gradient descent on a probabilistic loss function. RankBoost [11] and AdaRank [25] are examples of algorithms that use the boosting framework to learn feature-based ranking functions. Various methods based on the Perceptron algorithm have also been proposed to learn ranking functions for applications such as information retrieval [12, 10], reranking for named-entity extraction [7] and machine translation [20].

One limitation of these algorithms is that they do not directly maximize information retrieval performance metrics, such as NDCG (Normalized Discounted Cumulative Gain) [13] or MAP (Mean Average Precision) [1]. Instead, different approximations for these metrics have been used as proxies in the learning procedure. Matveeva et al. [17] and Taylor et al. [22] present two distinct ways to tailor RankNet to optimize NDCG. The first approach iteratively reranks smaller and smaller portions of an originally produced ranked list of documents in order to focus on the portion of the ranked list that has a higher likelihood of being relevant. The second adapts RankNet's cost function to directly optimize parameters in the BM25 ranking function while maximizing NDCG on a held-out set. In recent work, Yue et al. [26] presented a method for adapting RankSVM to maximize an approximation of average precision. More recently, Taylor et al.[21] described SoftRank, a method that optimizes "softNDCG", an approximation of the NDCG metric.

Other common approaches to optimizing specific retrieval performance metrics include grid-search, coordinate ascent and line-search [18]. These methods perform heuristic exploration of ranking function parameters (the hypothesis space), evaluating sampled hypotheses against the target performance measure. Although these methods directly optimize any given performance metric on the training set, they are generally very expensive when applied to ranking functions with more than just a few parameters.

2.2 Pairwise-preference Ranking

Many of the recently proposed approaches to learn feature-based ranking functions take a pairwise preference approach [2, 3, 14, 10, 26]. In the pairwise framework, instead of taking documents in isolation, document pairs are taken as instances in the learning process. The goal in this setting is to learn a *preference function* over document pairs, where the output of the learned function indicates the degree to which one document is preferred over another for a given query.

This approach is appealing for several reasons. First, learning a preference function on pairs of documents reduces the ranking problem to a binary classification problem: a correct (or incorrect) classification corresponds to correctly (or incorrectly) ordering a document pair. Many classification algorithms have been adapted to this task, including support vector machines [14], perceptron algorithms [10, 12] as well as gradient descent algorithms [2].

Second, this approach imposes very few assumptions on the structure of the training data — only that preferences among documents are somehow expressed. Explicit document preferences assessment [5] can clearly be used with this learning approach. Additionally, traditional absolute relevance judgements (binary or graded relevance) can be easily converted to a pairwise preference training set by taking all pairs of document with differing relevance levels. Clickthrough data has also been used by assuming that a clicked-on document expresses a preference for that document over documents occurring higher in the document ranking [14].

We represent our learning setting as follow: a ranking dataset consists of a set of queries $q \in Q$, and a set of documents for each query $d_i \in D_q$ with some associated relevance judgement of document d for query q, y_{qi}. Our training set for a single query is then $S_q = \{(d_{q1}, y_{q1}), (d_{q2}, y_{q2}), ...\}$. The relevance judgements y are discrete and ordered, with values such as {*Probably Relevant*, *Possibly Relevant*, *Not Relevant*}, and a total ordering $\triangleright$ exists between relevance levels, e.g. *Probably Relevant* $\triangleright$ *Possibly Relevant* $\triangleright$ *Not Relevant*.

Documents are represented by a vector of query-dependent feature weights f_k. For instance, document d_i given query q is represented as:

$$d_{qi} = [f_0(d_i, q), f_1(d_i, q), ..., f_m(d_i, q)] \quad (1)$$

where each feature scoring functions f_k represents some measure of similarity between the document and query. These can be derived from low-level features typically used in information retrieval systems (such as query term frequency or inverse document frequency), higher-level features such as the score assigned by a baseline ranking algorithm for document d_i on query q (such as BM25), or even query independent document quality measurements (such as PageRank).

The goal of the learning procedure in the pairwise framework is to induce a document score function $s(\bullet)$ such that

$$y_{qi} \triangleright y_{qj} \iff s(d_{qi}) > s(d_{qj}) \quad (2)$$

i.e., whenever document d_{qi} is preferred over ($\triangleright$) d_{qj}, the scoring function s will return a larger value for d_{qi} than for d_{qj}. This formulation makes clear the connection between pairwise-preference learning and binary classification: given the preference relationship on the left in Equation 2 a correct (or incorrect) classification corresponds to maintaining (or violating) the inequality on the right.

It is often useful to explicitly model this task as binary classification. In this view, one can build a new "paired" dataset S' for each query q by creating document pairs from documents with different relevant levels $(d_{qi}, d_{qj})_l$ and associated preference labels z_l. That is, $S'_q = \{((d_{qi}, d_{qj})_l, z_{ql}) \,|y_{qi} \neq y_{qj}, \forall i, j\}$ and

$$z_{ql} = \begin{cases} +1 & \text{if } y_{qi} \triangleright y_{qj}; \\ -1 & \text{if } y_{qi} \triangleleft y_{qj}. \end{cases}$$

Analogously, we can write Equation 2 with this notation, letting P_l be the *pairwise score* of document pair $(d_{qi}, d_{qj})_l$

$$P_l = z_{ql} \times (s(d_{iq}) - s(d_{jq})) > 0. \quad (3)$$

Minimizing the number of *misranks*, i.e. incorrectly ordered document pairs, is in principle a good criterion for rank optimization. It has been shown that minimizing the number of misranks is equivalent to maximizing a lower-bound on various information retrieval performance metrics, such as average precision and reciprocal rank [10]. However, the direct optimization of the number of misranks is an NP-hard problem [14], so approximations are necessary. We will

show below several approaches to approximating this minimization.

In this paper we are concerned with linear score functions $s(\bullet)$ that can be parameterized by a single weight vector $w = [w_1, w_2, ..., w_m]$. Thus the learning algorithms output scoring functions can be expressed as $s(d_{qi}) = \langle d_{qi}, w \rangle$, where $\langle \bullet, \bullet \rangle$ is the inner product operation. After learning this score function, the final document rankings can be derived from this function by ranking in descending order according to their score[1].

2.3 Outliers in Pairwise Preference Ranking

Creating pairwise training data from absolute judgements may have some undesirable consequences. Specifically, mislabeling of a single document's absolute judgement will lead to many "mis-labeled" document pair preferences. When using graded relevance levels, confusion or inconsistencies between different relevance levels may make mis-labeling a common problem. If we consider each mis-labeled document a noisy observation or outlier, the process of pairing each document with all others of differing relevance levels yields a quadratic increase in the number of outliers in the training data. This increase can have a serious detrimental effect on performance, as we will show below.

Mis-labeling the absolute relevance level of a document is not the only source of outliers. Due to the nature of keyword search, we have an extremely impoverished view of the information need — typically only 2-3 terms per query. For this reason, the query-document features (f_i in Equation 1) may not be expressive enough to truly distinguish relevant from non-relevant documents. This may result in many non-relevant documents "looking similar" to relevant in the query-document feature space. These non-relevant documents can also be considered outliers, and similar to misjudged documents, yield a quadratic increase in the number of pairwise outliers.

To illustrate the effect of outliers on rank learning, we trained a RankSVM model (see Section 3.1) on SEAL-1, a subset of the Set Expansion ranking dataset described in Section 4.1. Given the model learned w, we calculated the pairwise decision scores P_l (Equation 3) for all training data instances and constructed a histogram, as shown in the top of Figure 1. Most pairwise instances had positive scores P_l (top right in the figure), showing that the learned ranking model correctly ordered most of the training instances. Some instances, however, had negative scores and the few having the most negative scores may be outliers (top left, Figure 1).

In order to measure the previously mentioned outlier effect, we then retrained our model on a smaller training set after removing increasing numbers of outlier instances. That is, we trained the same RankSVM model excluding from the training data a few instances whose scores were below a cutoff value, P_l', and then evaluated the learned model on the same test set. The bottom of Figure 1 shows test MAP results when training is performed excluding outliers with pairwise score below a threshold from the training data. In this figure, the dashed horizontal line shows performance when all instances are used for training. The leftmost point shows the performance when instances with score below -15 were removed from training. As the removal cuttoff increases

[1]Notice that the score function $s(\bullet)$ takes a single document as argument, and not the document pair.

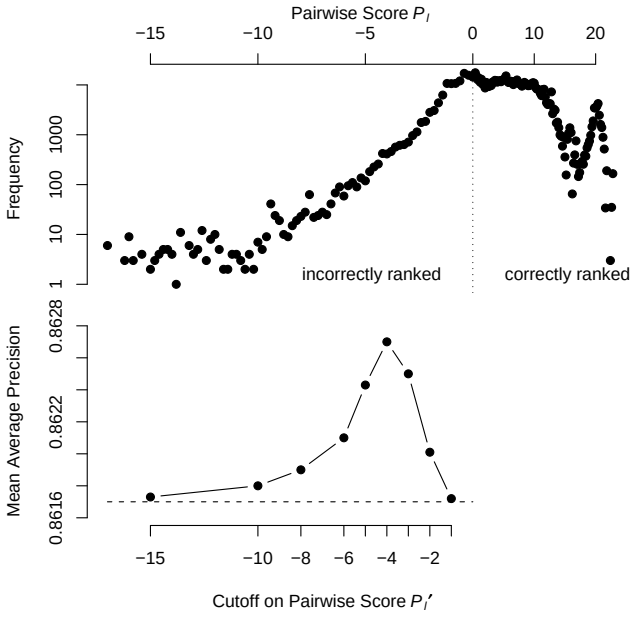


Figure 1: Example of outliers in pairwise Ranking. (top) Histogram of pairwise scores. (bottom) Mean Average Precision on the same test collection when excluding training instances whose scores were below cutoff.

up to -4, performance goes up, indicating that the removal of outliers improves the ranker's performance. For larger cutoffs, this effect is curtailed by the larger numbers of instances being discarded and performance drops.

Empirical evidence from several studies also suggests that performance of pairwise learning algorithms can be improved by removing or down-weighting these outliers. In perceptron-based learning algorithms, outliers were identified as document pairs that were consistently mis-ranked in several iterations through the training data, and removal of these document-pairs improved the performance and stability of the learned ranking function [10, 12]. This technique, known as the α-bound [15], limits the influence of potential outlier observations on the final learned hypothesis.

Although the α-bound is reported to work well with perceptron-based learners, it is unclear how it generalizes to other learning algorithms. In this work we develop a general mechanism to down-weight the influence of these outliers in pairwise preference learning and apply this technique to a variety of learning algorithms. Results show significantly improved performance of learned ranking functions across a variety of ranking tasks.

3. ROBUST PAIRWISE RANKING

In this section we propose a new learning algorithm to counteract the effect of outliers in pairwise rank learning. The algorithm takes as input the linear model learned by any linear ranker (a base model), and uses a non-convex optimization procedure to output a more robust and effective final linear ranking model. To help explain the algorithm, we start with a brief explanation of RankSVM.

3.1 RankSVM

One of the most successful algorithms for classification, Support Vector Machines (SVM), has recently been successfully adapted to ranking using the pairwise framework [14]. Given a binary paired dataset S' from a set of queries, an SVM classifier can be naturally adapted to model this problem. The SVM model will attempt to solve the following quadratic optimization problem:

$$\min_w \frac{1}{2} \|w\|^2 + C \sum_{q,l} \xi_{ql} \tag{4}$$

subject to

$$\xi_{ql} > 0, \quad z_{ql} \langle w, d_{qi} - d_{qj} \rangle \geq 1 - \xi_{ql} \qquad \forall q, l$$

where non-negative slack variables ξ_{ql} were introduced, and the tradeoff between margin size and training error [14] is controlled by the parameter C.

The optimization problem in equation 4 is equivalent to:

$$\min_w \lambda \|w\|^2 + \sum_{q,l} [1 - z_{ql} \langle w, d_{qi} - d_{qj} \rangle]_+ \tag{5}$$

where $\lambda = \frac{1}{2C}$ and $[\ ]_+$ is *hinge* operator:

$$[x]_+ = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

The first term in Equation 5 is a regularization term, and the second is frequently referred to as the *hinge loss*[3]. The hinge loss, a convex function, is an approximation to the empirical 0/1 loss of minimizing all misranks. Minimization of the hinge loss places an upper bound on the number of misranks in the paired dataset [14]. An illustration of the hinge loss function can be seen in Figure 2, the dashed line.

3.2 Sigmoid Approximation

One of the disadvantages of the hinge loss function is its sensitivity to outliers. Outlier points produce large negative scores (the far left of the score range in Figure 2). Because the hinge loss linearly increases with larger negative scores, these outliers have a strong contribution to the global loss. This large loss contribution in turn gives these outliers an important role in determining the final learned hypothesis.

To address this problem, we propose to approximate the number of misranks (the empirical 0/1 loss) using a non-linear sigmoidal function. This function can be expressed as $g(\sigma, P_l) = 1 - sigmoid(\sigma, P_l)$, where P_l is the pairwise score (Equation 3), and σ is a parameter that determines the steepness of the sigmoid function. The sigmoid function is defined as:

$$sigmoid(\sigma, x) = \frac{1}{1 + e^{-\sigma x}}.$$

The sigmoid loss with several values of σ is illustrated as the solid lines in Figure 2.

There are at least two advantages in using this particular loss function. First, this non-linear penalty suppresses the effect of outliers, i.e., not giving larger loss values to instances with very large negative pairwise scores. Second, this penalty can arbitrarily approximate the empirical 0/1 loss by increasing the σ parameter.

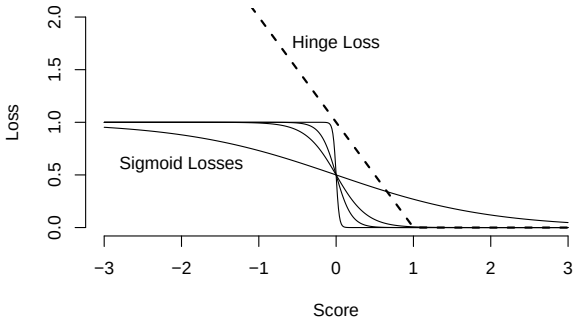


Figure 2: Loss Functions

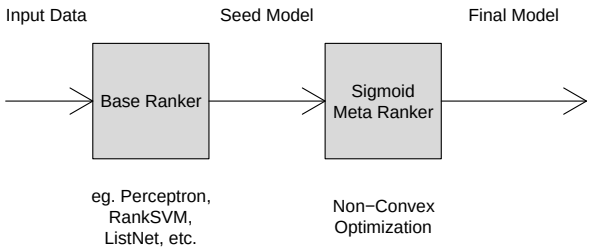


Figure 3: Meta ranking scheme: non-convex optimization procedure is seeded with the output of a base ranking model.

Similar to equation 5, the optimization problem with the sigmoid-based loss function can then be expressed as:

$$\min_w L(w) = \lambda \|w\|^2 + \sum_{q,l} [1 - sigmoid(\sigma, z_{ql} \langle w, d_{qi} - d_{qj} \rangle)]. \tag{7}$$

The sigmoid loss function is not convex, thus the learning procedure is only guaranteed to reach a local maximum. To avoid learning poor locally optimal solutions, the sigmoid ranker is used as a second optimization step, refining the hypothesis produced by another ranker. Specifically, sigmoid-based optimization is seeded with the hypothesis learned from a base ranker, such as RankSVM, and then it converges to a local optimum close to the (presumably good) seed hypothesis. The complete meta learning scheme is illustrated in Figure 3.

This sigmoidal meta-ranker is closely related to the two-stage optimization scheme proposed by Perez-Cruz et al. [19]. These researchers presented different loss functions, including a sigmoidal one, to be applied for classification tasks after being seeded by a traditional SVM classifier model. Their main goal was to better approximate the empirical classification loss (number of classification mistakes), and not to suppress outliers[19].

Tsai et. al. [23] also found that substituting a sub-linear fidelity loss function for RankNet's asymptotically linear cross-entropy loss improved ranking performance. In this regard, their algorithm, FRank, bears some similarity to the sigmoid meta-ranker proposed here. Although these two algorithms share a similar motivation, the algorithm presented here acts as a general-purpose meta-ranker for any linear seed ranker and provides added flexibility of the σ pa-

rameter, controlling the steepness of the loss. Additionally, the FRank algorithm uses a boosting framework to optimize the fidelity loss, whereas the algorithm presented here is optimized through gradient descent (see below). Boosting tends to have slower convergence properties, taking several hundred iterations through the dataset to converge [23]. As we describe below, even when seeded with a weak base learner, the gradient descent approach typically converges much faster.

3.3 Learning

We utilized a gradient descent technique to learn the final ranking model w. Specifically, we can differentiate the sigmoid-based loss function (Equation 7) with respect to the parameter vector w to obtain:

$$\frac{\partial L(w)}{\partial w} = 2w\lambda - \sum_{q,l} \sigma F(\sigma, q, l)\left[1 - F(\sigma, q, l)\right] \qquad (8)$$

where $F(\sigma, q, l) = sigmoid(\sigma, z_{ql}\langle w, d_{qi} - d_{qj}\rangle)$.

The gradient descent algorithm can then be written as:

$$w^{(k+1)} = w^{(k)} - \eta_k \frac{\partial L(w^{(k)})}{\partial w} \qquad (9)$$

where, the index k defines the number of iterations (epochs) and the step size η_k in principle can be chosen based on a line search along the descent direction, i.e.,

$$\eta_k = \underset{\eta \geq 0}{\operatorname{argmin}}\, L(w^{(k)} - \eta \frac{\partial L(w^{(k)})}{\partial w}).$$

In practice, however, we used a step-size-halving heuristic for η, initially setting $\eta = 0.05$. Whenever η was too large to yield a decrease in the loss function, η was set to $\eta/2$, and learning stops when the relative decrease in loss was less than 10^{-8}.

4. EXPERIMENTS

4.1 Datasets

We performed experiments on eight different ranking datasets. The first three ranking datasets are part of the Learning to Rank (LETOR) Benchmark dataset [16]. This dataset attempts to provide a standard set of document-query features over several test collections. These features were extracted from all the query-document pairs in the OHSUMED collection and the .GOV test collection using the queries and judgments from the TREC 2003 and 2004 web track topic distillation tasks [9, 8]. The relevance judgments in the TREC collections are binary and in the OHSUMED collection are graded in three levels: "definitely relevant", "possibly relevant" and "not relevant". The LETOR dataset also contains standardized train/validation/test splits for 5-fold cross validation. The OHSUMED collection contains 106 queries and 25 features, TREC 2003 has 50 documents and 44 features, and there are 75 queries and 44 features in the TREC 2004 collection. Please refer to the original reference [16] for a detailed explanation of the feature sets. In our experiments, the query-document feature values were normalized on a per query basis to the $[0, 1]$ interval using the linear scaling suggested by the producers of the LETOR dataset and no additional feature selection or processing was done.

The next two ranking datasets were collected from the *Recipient Recommendation* task [6], where the goal is to find persons who are potential recipients of an email message under composition given its current contents and its previously-specified recipients. This is a ranking task in a sense that, compared to traditional document retrieval, the email under composition is the equivalent of a query, and the email addresses in the address book are the analogous of documents.

The *TOCCBCC* prediction subtask aims at predicting all recipients of a message being composed, while the *CCBCC* subtask ranks all recipients inserted in the CC or BCC fields of the message under composition. Thus the CCBCC task, in addition to the text, can use information extracted from the recipients already specified in the TO field of the email. The collection contains more than 44000 queries from 36 different users, where an average of 1267 queries per user are used for training. For testing, the TOCCBCC tasks uses an average of 144 queries per user, while 20 queries per user are used for CCBCC testing. The number of documents (email addresses) to be ranked averages 377 per user [6]. The TOCCBCC task utilizes four different features for ranking (derived from frequency, recency and summarized textual scores). The CCBCC task dataset contains 7 features: the same 4 features from the TOCCBCC task, and three additional co-occurrence features (derived from the already-specified addresses in the TO field of the message).

The last three ranking datasets were derived from SEAL, a Set Expander for Any Language system [24]. Set Expansion is the task of expanding an initial set of objects into a larger and more complete set[2] of objects of the same type. More specifically, SEAL expands textual seeds (such as "California", "Colorado" and "Florida") by automatically finding semi-structured web pages having lists of items, and then aggregating these lists and ranking the "most promising" items higher. The ranking then considers different set of features such as the ones derived from proximity metrics, suffixes and prefixes extracted from wrappers, and similarity scores calculated from random walks in an entity graph [24]. Our sample dataset from SEAL contains queries in three different languages. Each language contained approximately 60 queries, and each document (entity) was represented with 18 features. Experiments were carried out with a 3-fold cross-validation split with two of the languages used for training, and the remaining language use for testing.

4.2 Performance

In this section we describe experiments conducted with the sigmoid ranker using three baseline rankers: RankSVM, the averaged ranking perceptron [10] and ListNet [4].

The averaged perceptron ranking algorithm [10] is a simple and fast online ranking algorithm that scales linearly with the number of training examples. Although recent results suggested that this algorithm may require thousands of iterations to produce reasonable performance [10], in this paper we trained it with five iterations only. By crippling the algorithm we produced a low quality input model to the meta ranker, and investigated how the meta ranker responds to a weak initialization.

ListNet is a recent feature-based ranking algorithm [4] that instead of learning by minimizing a document pair loss

[2]Google Sets is a well-known example of a set expansion system on the web.

functions, it minimizes a probabilistic listwise loss function. That is, it utilizes document lists, instead of document pairs, as instances in the learning procedure. Although it is not a pairwise ranking algorithm, ListNet outputs a linear ranking model that can be used as input in the sigmoid optimization. Hence, not only can we investigate how ListNet compares with other pairwise baseline learners, but also study if the sigmoid meta ranker can improve a non-pairwise base ranking model.

Unless otherwise noted, in all experiments the sigmoid σ parameter was set to 1.0, and the regularization parameter C for RankSVM was selected from a search within the discrete set $C \in \left\{10^{-5}, 10^{-4}, .., 10^{1}\right\}$ using a holdout set.

We start with experimental results from the largest ranking collection, the two recipient recommendation tasks. Performance results for these ranking tasks are illustrated in Figures 4, showing AUC (Area Under the ROC Curve), R-Precision and Mean Average Precision results for both TOCCBCC and CCBCC ranking tasks.

It is noticeable from these figures the large performance gains that the sigmoid optimization achieves with the perceptron algorithm baseline. On both tasks, the meta ranker produced significantly better results than the averaged perceptron ranker. There are also visible performance gains for sigmoid ranker applied to RankSVM, although more modest. The sigmoid optimization applied after ListNet did not seem to improve performance on the TOCCBCC tasks, even though it boosted results for the CCBCC task.

Performance on the LETOR collections are illustrated in Figure 5, showing MAP for each test collection and base learner. Mean average precision results are shown for each one of the LETOR collections (TREC-04, TREC-03 and OHSUMED) and for each ranker. In all tasks, the sigmoid optimization significantly improved results for the averaged perceptron ranker. For RankSVM, the sigmoid ranker produced improvements in all collections, with the largest gain for TREC-03. The ListNet + sigmoid ranker, on the other hand, experienced its largest performance improvement on the TREC-04 collection, although a small gain was also observed in TREC-03 as well.

Experimental results on the Set Expansion ranking collections are pictured in Figure 6. Again, visible MAP improvements in all three datasets can be observed for the sigmoid ranker on the top of the averaged perceptron. More surprising perhaps are the even larger performance gains obtained on the top of ListNet for all three datasets. Although smaller in magnitude, the sigmoid ranker also produced visible performance gains for all three SEAL datasets when applied to RankSVM.

Full results for Mean Average Precision are given in Table 1. Statistical significance tests of the "+sigmoid" columns over the values on the previous columns are indicated with * or ** (for paired t-test with $p < 0.05$ or 0.01, respectively) and † or †† (for the Wilcoxon Matched-Pairs Signed-Ranks test with $p < 0.05$ or 0.01, respectively).

Improvements provided by the sigmoid ranker were statistically significant for all base learners on all three SEAL ranking datasets. The sigmoid optimization also increases average perceptron in all ranking problems. MAP values obtained by ListNet+Sigmoid were also significantly better for the TREC-04 and CCBCC ranking tasks. Additionally, the meta ranker significantly improved RankSVM on the TOCCBCC ranking task.

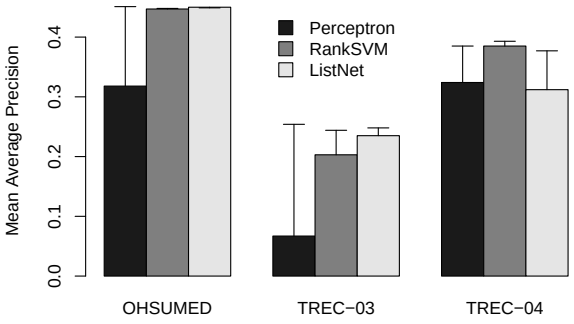


Figure 5: Performance (MAP) on LETOR Dataset. Whisker shows baseline + sigmoid.

It is interesting to note how significantly the perceptron ranker can be improved by the meta ranker. Its final performance numbers were comparable, and sometimes slightly better, than those obtained using the sigmoid optimization on top of the stronger base rankers. Although the sigmoid meta-ranker is only guaranteed to find a local optima, this local optima is sometimes better when the learner is seeded with a relatively weak ranking model. This may be an indication that initially using a method that is sensitive to outliers can lead the learner astray, yielding a seed model that is too strongly influenced by those outliers. The perceptron learner, however, was intentionally crippled, only making a small number of passes through the data. This training process doesn't allow the outliers to have such a strong influence on the seed model, potentially yielding a better final model.

Overall, the sigmoid meta ranker significantly improved ranking performances for most test cases in Table 1. In the LETOR datasets, however, this was not the case — although the meta ranker improved performance on average, these improvements were not statistically significant. Because the LETOR collections have a relatively larger number of features and a smaller number of queries, we speculate that these ranking models are overfitting the training data. In fact, we observed that very small changes in the RankSVM regularization parameter C produced very different ranking performance on these three collections.

These results also highlight that the sigmoid ranker is in fact a general purpose linear meta ranker. Not only can it improve pairwise ranking functions, but also fine-tune any linear ranking model — as attested by the ListNet + sigmoid performance.

4.3 Learning Curve

Typical sigmoid ranker learning curves can be seen in Figure 7. This curve illustrates training set AUC (i.e., performance on the training set in terms of Area Under the ROC Curve) versus the number of sigmoid gradient descent iterations for a particular CCBCC prediction task [3].

The initial points (epoch=0) in Figure 7 show the AUC values obtained by the base rankers. This is the starting point of the sigmoid rank optimization. In this particular example, RankSVM provides a higher initial AUC than ListNet, which in turn outperforms the averaged perceptron.

[3]The training set AUC was shown here because it corresponds directly to minimizing the number of misranks in the training set [26].

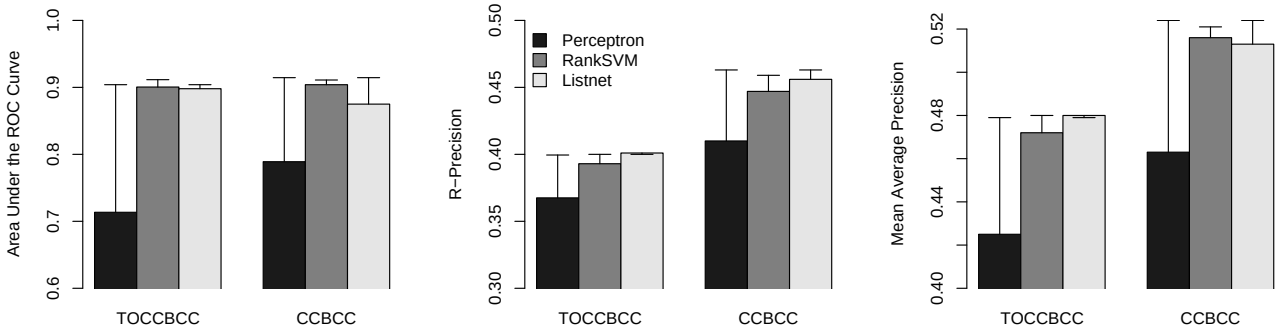


Figure 4: Performance for the recipient recommendation ranking tasks. Whisker shows baseline + sigmoid.

Collection	Perceptron	+Sigmoid	RankSVM	+Sigmoid	ListNet	+Sigmoid
OHSUMED	0.318	$0.451^{\star\star\dagger\dagger}$	0.447	0.448	0.450	0.449
TREC-03	0.067	$0.254^{\star\star\dagger\dagger}$	0.203	0.244	0.235	0.248
TREC-04	0.324	$0.385^{\star\dagger}$	0.385	0.393	0.312	$0.377^{\star\star\dagger\dagger}$
SEAL-1	0.851	$0.866^{\star\star\dagger\dagger}$	0.862	$0.866^{\dagger\dagger}$	0.843	$0.866^{\star\star\dagger\dagger}$
SEAL-2	0.869	$0.893^{\star\star\dagger\dagger}$	0.890	$0.894^{\dagger\dagger}$	0.864	$0.893^{\star\star\dagger\dagger}$
SEAL-3	0.906	$0.924^{\star\star\dagger\dagger}$	0.916	$0.920^{\star\dagger}$	0.901	$0.923^{\star\star\dagger\dagger}$
TOCCBCC	0.425	$0.479^{\star\star\dagger\dagger}$	0.472	$0.480^{\star\star\dagger\dagger}$	0.480	0.479
CCBCC	0.463	$0.524^{\star\star\dagger\dagger}$	0.516	0.521	0.513	$0.524^{\star\star\dagger\dagger}$

Table 1: Mean Average Precision values for experiments in all collections. Statistical significance tests over the values on the previous column are indicated with * or ** (for paired t-test with $p < 0.05$ or 0.01, respectively) and † or †† (for the Wilcoxon Matched-Pairs Signed-Ranks test with $p < 0.05$ or 0.01, respectively).

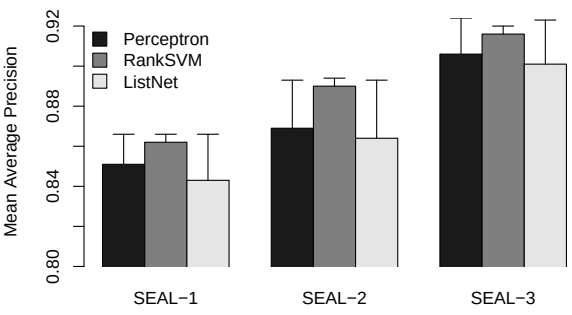


Figure 6: Performance (MAP) on Set Expansion Experiments. Whisker shows baseline + sigmoid.

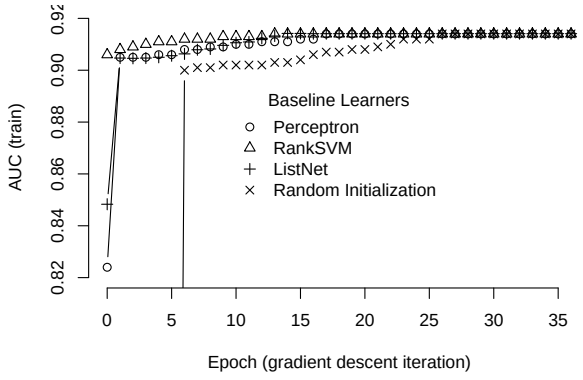


Figure 7: Learning curve of sigmoid ranker for several baseline algorithms.

The RankSVM+Sigmoid optimization then proceeds smoothly, with performance values reaching a plateau around 13 gradient descent iterations. ListNet+Sigmoid and Perceptron+Sigmoid start from different hypotheses, but are able to reach relatively high performance levels in less than three gradient descent iterations, and then converge to approximately the same plateau in less than 17 iterations. For comparison, one more curve was included in Figure 7: a sigmoid ranker with a random initial model. As expected, it takes considerably longer to reach reasonable AUC values, and converges to plateau levels in less than 30 iterations.

Figure 7 illustrates two reasons why the sigmoid meta ranker can provide robust pairwise ranking with a small extra computational cost. First, the number iterations necessary for convergence in the sigmoid ranker was usually small, since the starting point (the output of base learner) was already a well-tuned model. Second, the first few gradient steps were usually responsible for most of the performance gains observed. It is also important to note that this optimization step is powerful even as a stand-alone rank learner — the performance with a random initialization approaches the maximum performance of the best seeded rankers in less than 30 gradient descent steps.

4.4 Sigma Parameter

The steepness of the sigmoid function is controlled by the parameter σ. In principle, one can arbitrarily approximate the true 0/1 empirical loss function by increasing the values for this parameter. Experiments below showed, however, that increasing values of σ do not correspond to better overall ranking performance.

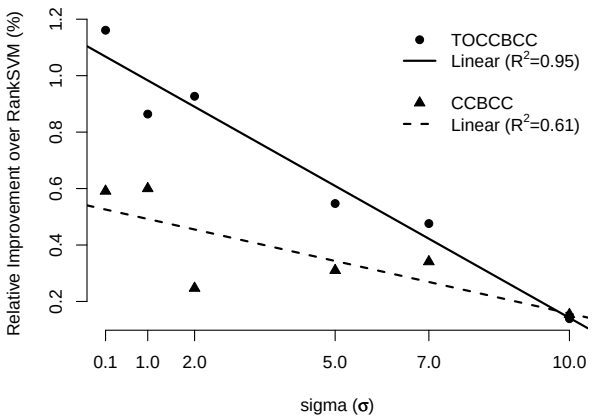


Figure 8: Relative Improvements in MAP over RankSVM for different sigma (σ) values.

Figure 8 shows, for both recipient prediction tasks, the relative improvements in MAP over RankSVM obtained by the sigmoid optimization, versus different values for the σ parameter in the sigmoid function. The values of σ considered were $\{0.1, 1, 2, 5, 7, 10\}$. Figure 8 clearly shows a trend that smaller values of σ produce better ranking performance for both ranking tasks.

Arbitrarily increasing the σ parameter generates steeper loss curves whose gradient information is largely concentrated around the decision region. We speculate that, for large σ values, this lack or reduction of gradient information from other regions of the loss function is responsible for the observed lower performance.

5. CONCLUSIONS

We considered the effects of outliers in learning pairwise feature-based ranking functions. In pairwise ranking, pairs of documents with different label levels are taken as instances in the learning process. While this process is known to bring advantages to rank learning, it can also produce many outliers, particularly when arbitrary label level judgments are used or simply when human labelers make mistakes. Outliers in the learning procedure can compromise ranking function robustness, and consequently deteriorate ranking performance.

We illustrated the effects of outliers in pairwise ranking functions, and then introduced a new meta-learning algorithm able to suppress the undesirable outlier effects. The algorithm is a non-convex optimization procedure using a sigmoid loss, in which any linear baseline ranking function can be used as input. Experiments on several different ranking datasets showed that this meta ranker produced statistically significant performance gains over various state-of-the-art baseline rankers.

This sigmoid meta-learning algorithm provided consistent and significant performance improvements when seeded by a weak rank learner, the average perceptron. When seeded by strong baseline rankers, RankSVM and ListNet, the meta-learning algorithm improved performance 88% of the time, with 64% of those performance gains statistically significant.

6. REFERENCES

[1] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[2] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 89–96, New York, NY, USA, 2005. ACM Press.

[3] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon. Adapting ranking svm to document retrieval. In *SIGIR*, pages 186–193, New York, NY, USA, 2006.

[4] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *ICML*. ACM, 2007.

[5] B. Carterette, P. N. Bennett, D. M. Chickering, and S. T. Dumais. Here or there: Preference judgments for relevance. In *European Conference on Information Retrieval*, 2008.

[6] V. R. Carvalho and W. W. Cohen. Ranking users for intelligent message addressing. In *European Conference on Information Retrieval*, 2008.

[7] M. Collins. Ranking algorithms for named-entity extraction: boosting and the voted perceptron. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 489–496, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

[8] N. Craswell and D. Hawking. Overview of the trec 2004 web track. In *13th Text REtrieval Conference (TREC 2004)*, November 2004.

[9] N. Craswell, D. Hawking, R. Wilkinson, and M. Wu. Overview of the trec 2003 web track. In *12th Text REtrieval Conference (TREC 2003)*, November 2003.

[10] J. Elsas, V. R. Carvalho, and J. G. Carbonell. Fast learning of document ranking functions with the committee perceptron. In *ACM International Conference on Web Search and Data Mining*, 2008.

[11] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, 2003.

[12] J. Gao, H. Qi, X. Xia, and J.-Y. Nie. Linear discriminant model for information retrieval. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 290–297, New York, NY, USA, 2005. ACM Press.

[13] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[14] T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, New York, NY, USA, 2002. ACM Press.

[15] R. Khardon and G. Wachman. Noise tolerant variants of the perceptron algorithm. *Journal of Machine Learning Research*, 8:227–248, 2007.

[16] T.-Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. Letor: Benchmark dataset for research on learning to rank for information retrieval. In *SIGIR '07: Proceedings of*

the Learning to Rank Workshop, 2007.

[17] I. Matveeva, C. Burges, T. Burkard, A. Laucius, and L. Wong. High accuracy retrieval with multiple nested ranker. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 437–444, New York, NY, USA, 2006. ACM Press.

[18] D. Metzler and B. W. Croft. Linear feature-based models for information retrieval. *Information Retrieval*, 10(3):257–274, June 2007.

[19] F. Perez-Cruz, A. Navia-Vazquez, A. R. Figueiras-Vidal, and A. Artes-Rodriguez. Empirical risk minimization for support vector classifiers. *IEEE Transactions on Neural Networks*, 14:296–303, Mar 2003.

[20] L. Shen and A. K. Joshi. Ranking and reranking with perceptron. *Mach. Learn.*, 60(1-3):73–96, 2005.

[21] M. Taylor, J. Guiver, S. Robertson, and T. Minka. Softrank: Optimizing non-smooth rank metrics. In *ACM WSDM*, 2008.

[22] M. Taylor, H. Zaragoza, N. Craswell, S. Robertson, and C. Burges. Optimisation methods for ranking functions with multiple parameters. In *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 585–593, New York, NY, USA, 2006. ACM Press.

[23] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma. FRank: a ranking method with fidelity loss. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 383–390, New York, NY, USA, 2007. ACM Press.

[24] R. C. Wang and W. W. Cohen. Language-independent set expansion of named entities using the web. In *IEEE International Conference on Data Mining (ICDM)*, 2007.

[25] J. Xu and H. Li. Adarank: a boosting algorithm for information retrieval. In *SIGIR 2007: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Amsterdam, The Netherlands, July 23-27, 2007*. ACM, 2007.

[26] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 271–278, New York, NY, USA, 2007. ACM Press.

A Decision Theoretic Framework for Ranking using Implicit Feedback

Onno Zoeter Michael Taylor Ed Snelson John Guiver Nick Craswell Martin Szummer
Microsoft Research Cambridge
7 J J Thomson Avenue
Cambridge, United Kingdom
{onnoz,mitaylor,esnelson,joguiver,nickcr,szummer}@microsoft.com

ABSTRACT

This paper presents a decision theoretic ranking system that incorporates both explicit and implicit feedback. The system has a model that predicts, given all available data at query time, different interactions a person might have with search results. Possible interactions include relevance labelling and clicking. We define a utility function that takes as input the outputs of the interaction model to provide a real valued score to the user's session. The optimal ranking is the list of documents that, in expectation under the model, maximizes the utility for a user session.

The system presented is based on a simple example utility function that combines both click behavior and labelling. The click prediction model is a Bayesian generalized linear model. Its notable characteristic is that it incorporates both weights for explanatory features and weights for each query-document pair. This allows the model to generalize to unseen queries but makes it at the same time flexible enough to keep in a 'memory' where the model should deviate from its feature based prediction. Such a click-predicting model could be particularly useful in an application such as enterprise search, allowing on-site adaptation to local documents and user behaviour. The example utility function has a parameter that controls the tradeoff between optimizing for clicks and optimizing for labels. Experimental results in the context of enterprise search show that a balance in the tradeoff leads to the best NDCG and good (predicted) clickthrough.

Categories and Subject Descriptors

H.3.3 [**Information Systems Applications**]:

Keywords

clickthrough, learning, ranking, metrics

1. INTRODUCTION

This paper presents a system for learning to rank in a decision-theoretic framework. In such a framework each potential top-k ranking is thought of as an action that could be made by the search engine. Then retrieval is a decision procedure, of choosing an optimal action according to a given utility function.

The decision theoretic view of IR has a long-standing tradition (see e.g. [12, 4, 8] for succesful uses). In this paper we explore the idea of using it to learn a ranker based on multiple streams of feedback. The utility function is then not only based on judge labels, but also on characteristics of a user's session. A model is learned on historic data to predict the user's interaction with a result list. Although many characteristics of the user's session could be incorporated in such a utility function, we will mainly concentrate on one particular and important one, namely clicks.

The reason to consider both labels and clicks in the utility function is that each provides a different sort of relevance information:

- Quantity and cost. Click information is available at zero cost as long as the system has some users, and the quantity depends on the level of user activity. Relevance judges are usually paid, so the quantity of labels depends on budget.
- Explicitness. Judges give explicit relevance labels. With clicks, dwell-time, and abandonment, relevance information must be inferred.
- Real user population. Clicks come from the true user population, so may reflect real relevance. Relevance judges in laboratory conditions may disagree with the real users.
- Deep/negative judgments. Relevance judges can be paid to label a large pool of documents per query, including many bad documents. Clicks tend to happen only on top-ranked documents, and gathering negative click information has a detrimental effect on users, because the bad documents must be retrieved near the top.

The question is how to build a model that works well, incorporating explicit and implicit relevance information. One approach (Figure 1a) is to choose an evaluation measure as the gold standard for relevance, such as the label-based metric DCG [6], and build a model to optimize it such as LambdaRank [2] or SoftRank [13]. The inputs may be features characterizing the quality of the query-document

Presented at LR4IR Workshop at SIGIR 2008, Singapore

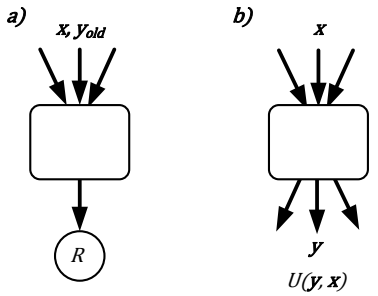


Figure 1: Two different approaches to the incorporation of implicit feedback into a ranker; a) uses historic user behavior as input to predict a single relevance score R. Approach b), proposed in this paper, constructs the best possible model to explain outputs y from inputs x and separately defines a utility function U that puts a preference ordering on possible explicit and implicit behaviors.

match. Historical implicit feedback can be incorporated as additional input features [1]. The output of the model gives a scalar-valued score by which documents are ranked via a sort. Note here that the value of an individual document's score has no practical interpretation.

Our approach (Figure 1b) is different and based on an extension of the decision theoretic framework for IR, as described in [14, 8]. The inputs and outputs of the model are all observable: inputs are query-document features and outputs are implicit/explicit relevance information. The sole task of the model is predicting outputs. We then define a utility U which is a function of these predicted outputs, namely both implicit and explicit judgments and behaviors. Ranking is then a decision procedure, to find the results list with maximum utility.

The specific contributions of this paper are as follows.

- We propose to construct rankers that combine many sources of information using the decision theoretic framework for IR. We discuss what an ideal setup would look like and how it would add diversity to result lists, correctly incorporate real world characteristics such as position bias, balance authoritiveness and popularity, and more.
- As an initial implementation of the approach, we present a Bayesian logistic regression model that predicts both relevance judgments and click rates. The model has one weight per query-document pair that acts as a "memory" of the historic click rate that is not already explained by the other features. We combine it with a crude utility function that is far from the ideal sketched setting, but already introduces many of the potential benefits the combination of two datastreams can bring.
- We evaluate the decision theoretic system in an enterprise search scenario, demonstrating that the click-predicting part of the model can adapt to a new enterprise.

2. RANKING AS A DECISION THEORY PROBLEM

Decision theory is a very well established field which dates back at least to the works of Daniel Bernoulli in the 18th century. The information retrieval problem of presenting a list of results given a specific user query has been interpreted as a decision theory problem in several studies in the past. The probability relevance principle [12] for instance can be motivated from such a view. Interesting and successful applications can also be found in amongst others [8, 4]. In this section we first review the abstract decision problem in its general form, and then move on to describe how it can be applied to incorporate both explicit and implicit feedback in a common framework.

At the basis of the decision theoretic view is a *utility function*. It represents user satisfaction in a single scalar, larger being better. Formally it is a mapping of all relevant quantities of interest (searcher charteristics, query, clicks, dwell time, etc.) to the real line. In the remaining we will make a distinction between two sets of information: *inputs* and *outputs*. Inputs are those quantities that are available *before* a result list needs to be compiled, outputs are those quantities that have become available in the user session *after* the result list is presented to the user. This includes clicks, dwell time, click backs, etc., but also explicit labels if we ask the user to act as a human judge.

The ideal utility function could be very complex incorporating detailed characteristics of a user, intent of the query, etc. It would increase if interesting results were found, decrease as more and more effort is needed to find them. We will discuss some of the potential properties of an ideal utility function in Section 2.1. In real world use we will have to make simplifications, such as is done in Section 2.2.

If we would know ahead of time exactly how a user would interact with a particular search result list it would be easy to select the optimal one. It would simply be *that* result list that maximizes user satisfaction. Since at query time we do not know the user's response, we need to construct a model that predicts user behavior. The optimal decision (the optimal list) is then the list that *in expectation* under the model maximizes the users utility.

In summary and formally we can represent the decision theoretic view of IR as follows. Given a set of inputs (query-document features) $\mathbf{x} \in \mathcal{X}$ the ranker is asked to select an action (result list) $a \in \mathcal{A}$. After performing the action we observe outputs (judgments, user behaviour etc.) $y \in \mathcal{Y}$. A utility function $U : \mathcal{X} \times \mathcal{A} \times \mathcal{Y} \mapsto \mathbb{R}$ assigns a scalar utility to the observed $\mathbf{x}, a, y$-triple[1]. The outputs y in general do not follow deterministically from the inputs $\mathbf{x}$ and action a. A model $p(y|\mathbf{x}, a)$ gives the probability of observing y after selecting a when $\mathbf{x}$ is observed. The optimal action a^* is the action that leads to the maximal expected utility

$$a^* = \operatorname*{argmax}_a \mathbb{E}_{p(y|\mathbf{x},a)} \left[U\left(\mathbf{x}, a, y\right) \right] . \tag{1}$$

We propose to use the traditional decision theoretic interpretation of IR to combine multiple sources of data in a principled way. We treat the different sources of implicit feedback as extra dimensions in the output vector y.

2.1 Utility functions

The utility function gives a real valued score to a user ses-

[1]Note that alternatively we could include $\mathbf{x}$ and a into the observation y, but this notation emphasizes the flexibility of the approach.

sion that represents his satisfaction. Thinking about what the ideal utility function would look like can easily be dazzling. For a well defined navigational query such as "What is the next train connection between Cambridge and London Kings Cross?" we might argue that finding the answer gives a fixed utility and any work that needs to be done to get to that point (reading snippets, clicking on potential answer pages, clicking back, etc.) will lead to deductions. But what about informational queries? What is the utility for one, two, or three interesting documents in the result list. Do three interesting documents have three times as much utility as a single one, or is there a law of diminishing returns? What is the "cost" of a misleading snippet? Some sources are very authoritive, some have very fresh content. How should these two properties be traded-off? Should that be done in the same way in all contexts? A small time spent thinking about these things leads easily to an extremely complex function.

Even coming up with a procedure of going about constructing a utility function is a difficult problem. Here we discuss briefly two approaches. A first approach would be to conduct lab experiments with users where they are asked to explicitly score their satisfaction with a session. The experiments in [5] form a fascinating approach in that direction for instance. Assume for simplicity that we have a binary satisfaction signal

$$t \in \{\text{thumbs up}, \text{thumbs down}\} ,$$

and a simple utility function

$$U(t = \text{thumbs up}) = 1 \quad \text{and} \quad U(t = \text{thumbs down}) = 0 .$$

In daily use the explicit satisfaction scores t are not available. To overcome this we could learn, based on $\{t, y\}$-pairs, a special model $p(t|y)$ (not to be confused with the output prediction model) and work with a "learned utility"

$$\tilde{U}(x, y, a) = \mathbb{E}_{p(t|\mathbf{x},y,a)} \left[U(t) \right] . \tag{2}$$

Combined with an output prediction model $p(y|x, a)$ we could then use Equation (1) for ranking.

In a second approach we ask experts to craft a simple utility $U(x, y, a)$ and iteratively improve it. Perhaps in the first version only a few sources of feedback are modelled in y and this is expanded in the next, or we find that certain tradeoffs looked good on paper but used in practice leads to complaints from real users.

In both approaches constructing the model $p(y|x, a)$ is a classical machine learning problem. Using historical $\{x, y, a\}$-triplets we can train and select the appropriate user behavior prediction model. An important benefit in practice is then that the problem of designing a reasonable utility function and constructing a good prediction model can be decoupled. The prediction can be tested on historic data. Adjusting and tuning the utility function can be done incrementally over time without the need of retraining the model with each new attempt.

To summarize: constructing a utility function is a very difficult problem and can leave one with the awkward feeling that a golden standard or ground truth is not available. We would argue that the IR problem simply is this complex. Any choice in a real world system will make some approximation and is likely to require changes and improvements over time.

In the Section 2.1.3 we introduce what arguably is the simplest possible utility function that combines both a signal stream of explicit label feedbacks and a stream of implicit user clicks. It is a simple convex combination of a label based utility and a click based utility intoduced in Sections 2.1.1 and 2.1.2 respectively.

2.1.1 Discounted Cumulative Gain

In some approaches to ranking the aim is to maximize a function of the labels in the result set. It is easy to see that these approaches form a special case of the framework considered here. If we look at the discounted cumulative gain (DCG) [6] for instance we see that it is an example of a utility function that only takes into account the human relevance judgments at every position. It is based on a discount function $d(p)$ over positions $p \in \{1, \ldots, n\}$, and a gain function $g(s)$ over human relevance judgments, e.g. $s \in \{1, \ldots, 5\}$. The position discount function is monotonically decreasing from the top position $p = 1$, to the bottom position $p = n$: $d(1) > d(2) > \cdots d(n)$, and a gain function $g(s)$ that is increasing for better relevance judgments: $g(1) \leq g(2) \leq \cdots \leq g(5)$. If $s[1], \ldots, s[n]$ are the scores received for the documents selected by a, the discounted cumulative gain is given by

$$DCG\left(s[1], \ldots, s[n]\right) = \sum_{p=1}^{n} d(p) g(s[p]) . \tag{3}$$

To maximize the DCG we would select and rank such that the expected DCG is highest. The expectation is then with respect to the observation model $p(y|\mathbf{x}, a) = p(s[1], \ldots, s[n]|\mathbf{x}, a)$ which represents the best estimate of the human relevance judgments for the documents selected by a given $\mathbf{x}$

$$a^* = \arg\max_a \mathbb{E}_{p(s[1],\ldots,s[n]|\mathbf{x},a)} \left[\sum_{p=1}^{n} d(p) g(s[p]) \right] .$$

Different choices of $g(s)$ lead to different ranking principles (decision rules). If $g(s)$ is convex in s the resulting principle is *risk seeking*: for two documents with the same expected judgment but different variances the document with the larger variance is preferred. This is because a larger than expected judgment leads to a bigger rise in utility than the decrease in utility that results if a lower than expected judgment is encountered. We could say that such a convex gain function leads to a "going for the jackpot" effect. The often used exponential function $g(s) = 2^s - 1$ has this effect. It is important to realize that this is not a conservative ranking principle.

If we have a linear gain $g(s) = s$, the expected utility only involves the expectations of judgments:

$$\begin{aligned} a^* &= \arg\max_a \mathbb{E}_{p(s[1],\ldots,s[n]|\mathbf{x},a)} \left[\sum_{p=1}^{n} d(p) g(s[p]) \right] \\ &= \arg\max_a \sum_{p=1}^{n} d(p) \mathbb{E}_{p(s[1],\ldots,s[n]|\mathbf{x},a)} \left[s[p] \right] . \end{aligned}$$

hence we get a ranking principle that simply orders documents according to their expected human relevance judgment:

$$a^* = \arg\max_a \sum_{p=1}^{n} d(p) \mathbb{E}_{p(s[p]|\mathbf{x},a)} \left[s[p] \right] .$$

This utility function is an example where the optimal action can be found in $\mathcal{O}(|\mathcal{D}|)$ time, where $|\mathcal{D}|$ is the number of documents in the corpus. This is despite the fact that the space of all possible selections and rankings is $|\mathcal{D}|^n$. This is due to the fact that the judgment probability $p(s[p]|\mathbf{x}, a)$ is not explicitly a function of position (the judge is presented with each document independently). This means that the expected judgment can be computed for each document and the documents simply sorted to obtain the optimal ranking. There are many interesting utility functions that lead to $\mathcal{O}(|\mathcal{D}|)$ ranking principles, but in general approximations might be necessary.

Note that, since there is no element in the utility function that encourages diversity in the results, we need to explicitly add the constraint that links to documents cannot be replicated. Otherwise a^* would be n duplications of the link with the highest expected relevance judgments.

2.1.2 Clicks

An analogous utility function that only takes into account whether or not a user clicked on a document could be given by a "click-DCG" utility

$$U_{\text{clicks}}(c[1], \ldots, c[n]) = \sum_{p=1}^{n} d(p)c[p]\,. \tag{4}$$

If $p(c[p] = 1|\mathbf{x}, a)$ (the probability of a click on the document that was put in position p by a) is modeled based on a link specific and position specific contribution it will in general not simplify to an $\mathcal{O}(|\mathcal{D}|)$ ordering rule. This is because now $p(c[p]|\mathbf{x}, a)$ is explicitly a function of p — any given document will be clicked with a different probability depending on where it is placed. It can be that position and link effects combine in complex non-linear ways. However there are suitable heuristics for ordering in $\mathcal{O}(|\mathcal{D}|)$, e.g. compute the probability a document will be clicked if it were placed in position 1, and order by that.

This click-DCG assigns a positive utility to the act of clicking itself. Philosophically this is not really sound, since the act of clicking is actually a nuisance, and only from the actual reading of an interesting document is utility obtained. So in order to motivate (4) we need to appeal to an argument along the lines of the learned utility in (2): because we have established in the past that the act of clicking on an interesting link leads to an interesting page we can assign an (expected) utility to the act of clicking. However, from a more practical point of view (4) then still has problems. If we motivate the value of a click from an apparent interest in the result page, we assume that all interesting snippets point to interesting landing pages. This will unfortunately not always be the case in practice. To overcome this the utility can be extended by incorporating a minimal dwell time as proxy for an endorsement of the landing page.

To encourage diversity, one simple approach would be to introduce a concave function f of the simple DCG-like sum of clicks:

$$U_{\text{clicks page}}(c[1], \ldots, c[n]) = f\left(\sum_{p=1}^{n} d(p)c[p]\right)\,. \tag{5}$$

This captures the notion that the step from 0 clicks to 1 click on a page is bigger than that from 1 to 2. The transformed utility would penalize systems with click-DCG near zero. For an ambiguous query with several types of result, a ranking optimized to avoid zero click-DCG could potentially present results of each type, hedging its bets by giving a more diverse results list.

To take advantage of this type of diversity-encouraging utility, one must combine it with a model that can capture *correlations* between click events on different documents on a page. For instance, for ambiguous queries, clicks on links to two different interpretations will in general be anti-correlated: someone clicking on a link of one type will be less likely to also click on a link of the other type, presuming they have one interpretation of the query in mind when searching. To do this requires a model for the joint distribution $p(c[1], \ldots, c[n]|\mathbf{x}, a)$, which is in general a difficult modeling task. An independence assumption $p(c[1], \ldots, c[n]|\mathbf{x}, a) = \prod_{p=1}^{n} p(c[p]|\mathbf{x}, a)$ does not capture these correlations, but is a reasonable simplifying modeling assumption if one is using the more straight-forward click-DCG utility of (4).

2.1.3 Combinations of basic utilities

The decision theoretic framework allows for a principled trade-off between desired behavior of the searcher and relevance cues from a selected set of human judges. In general the utility function should depend on both. A straightforward scheme is to take a weighted combination of the basic utility functions presented in Section 2.1.1 and 2.1.2.

2.2 Properties of the basic click-label utility

In the experiments in Section 3 we will use a utility function as sketched in Section 2.1.3:

$$U(y) = (1 - \lambda)U_{\text{DCG}}(y) + \lambda U_{\text{clicks}}(y)\,. \tag{6}$$

The parameter λ is still a design choice in this parametric form.

As argued in Section 2.1.2 the click part in the utility function (6) is only weakly motivated by the guiding principles, but it forms a good starting point since it captures already a few interesting characteristics from the two data streams.

- If there is noise in the labeled set, or if the model makes poor label predictions for a query, a suboptimal ordering can be corrected by clicks.
- If the model correctly predicts labels but there are ties, a top three of only good documents say, users effectively vote with their mouse which one they prefer.
- Since the framework consists of a model that *predicts* clicks based on features, an improvement in the ranking for popular queries also extends to unseen queries. For instance if Excel documents prove to be popular in a particular search context, they can be boosted for all queries in that context.
- Effectively the click based component in the utility will boost results that are predicted to be popular. If judges are instructed to label according to authority, the λ parameter allows us to trade-off popularity and authority. For instance in experiments with web-search data we found for instance that for the query "adobe" the url `www.adobe.com` is predicted to get the highest label, but the acrobat reader download page is the most popular. One could argue that the ideal result list has `www.adobe.com` at the top position and the link to the reader as the second link. This was the list returned in our experiments with $\lambda = 0.5$.

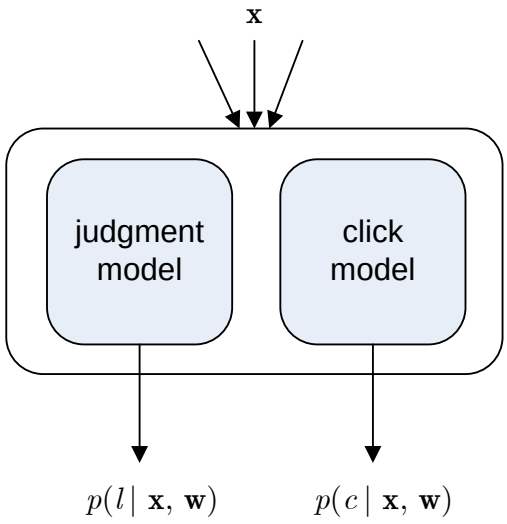


Figure 2: The model implemented in this paper sets out to predict two things: namely the probability of a click event $p(y_c|\mathbf{x}, \mathbf{w})$ and the probability of a particular relevance judgment $p(y_j|\mathbf{x}, \mathbf{w})$. The GLM model implies that the two sub-models factorize, and thus can be learned independently.

- By having the position of a document as part of the inputs x and fitting appropriate weights in the model, a position bias is automatically accounted for.
- If x contains characteristics of the user, the ranker automatically gives a personalized result.
- If the model is sophisticated enough such that it captures the interaction between documents, e.g. predicting that the probability of being clicked for near duplicate documents is anti-correlated, the ranker will, with a click-utility component from (5) automatically diversify the result list.

3. ON-SITE ADAPTATION OF INTRANET SEARCH SYSTEMS

An interesting application of the decision theoretic framework is in enterprise search. When a search system is installed out-of-the-box its ranker is based on a generic training set. Since intranets and their user bases can be quite diverse, it makes sense to use implicit feedback to adapt the ranker to the specific site for which it will be used.

It is generally difficult to obtain judged queries complete with clickthrough data from external organizations. Hence for this work, we were obliged to test the adaptation framework using an artificial corpus split created from data obtained from the Intranet of a single large multinational software corporation.

To reflect a significant change from the train set to the adaptation set, we created a split of our queries. For training, we use all queries and documents concerning the general areas of administration and marketing. For the adaptation set, simulating a potentially very different Intranet site, we use queries and documents of a technical nature.

The admin/marketing dataset used to train the out-of-the-box model consists of 546 queries. For each query, about 100 documents from the top of a ranked list from a baseline ranker were judged, and some of them had click information. The click-prediction part of the model is further trained using the adaptation query set, consisting of 201 technical queries. This simulates the on-site adaptation of the system to the user's clicks in the enterprise. In this case, the explicit judgments are not used for adapting the model, but instead used for evaluation only.

The click data we use is noteworthy in the following sense. We record not only the clicked documents, but also the documents that are skipped, or passed over, on the way to a click. In this work, inspired by [7], we assume a sequential scan of the result list, and as a consequence, that any document that is above the last click on the list is *examined*. In this way, we can aggregate the number of clicks and the number of examinations for a given query-document pair: a document which is clicked each time it is examined is intuitively good, and a document that is rarely clicked *having been examined* is probably a poor result. Importantly, we cannot infer much about the relevance of documents that have few examinations. This can happen if a result is either low in the ranking, or near the top yet just below a very good result.

3.1 A Bayesian generalized linear model

In this first illustration we use a generalized linear model (GLM) [9] for $p(y|\mathbf{x}, a)$. A GLM consists of a likelihood model $p(y|\theta)$, a linear combination of inputs $\mathbf{x}$ and model weights $\mathbf{w}$: $\mathbf{x}^\top\mathbf{w}$, and a link function $g(\theta)$ that maps the parameter θ to the real line. In this section we will use building blocks that have a binomial likelihood model and a probit link function. In a generative model interpretation the *inverse* probit link function

$$g^{-1}(s) = \Phi\left(s; 0, \frac{1}{\pi}\right)$$

plays a central role. This inverse link function is the well known cumulative normal function that maps the outcome of the inner product $\mathbf{x}^\top\mathbf{w} \in \mathbb{R}$ to the $[0, 1]$ space of the success parameter θ in the binomial. The inverse precision π can be set to an arbitrary number to fix the scale of the s-space. Here we will put a Gamma prior on π and integrate it out to obtain a robust model. If we have N examples in our training set for which the inputs have value $\mathbf{x}$, and we observe c positive outcomes, the likelihood becomes:

$$p(c|\mathbf{x}, \mathbf{w}) = Bin\left(c; g^{-1}\left(\mathbf{x}^\top\mathbf{w}\right), N\right). \quad (7)$$

In Figure 2 we show a more detailed version of Figure 1b, where we are more explicit about what we set out to predict with the model. In this initial implementation, the output y in the model describes for each position $p = 1, \ldots, n$ a single implicit feedback: the click event y_c, and a single explicit feedback: the relevance judgment y_j.

Figure 4 shows the ordinal regression submodel $p(l|x, w)$ which is a generalization of the click model. Instead of one of two outcomes it has one of five possible label values it can output. Along with the other weights we therefore also learn four boundary values $b_1, \ldots, b_4$ that mark the edges in s-space of the five categories. Each has a Gaussian prior. The IsBetween factor in the figure represents two stepfunctions that bound the interval for label l. Added to the sum

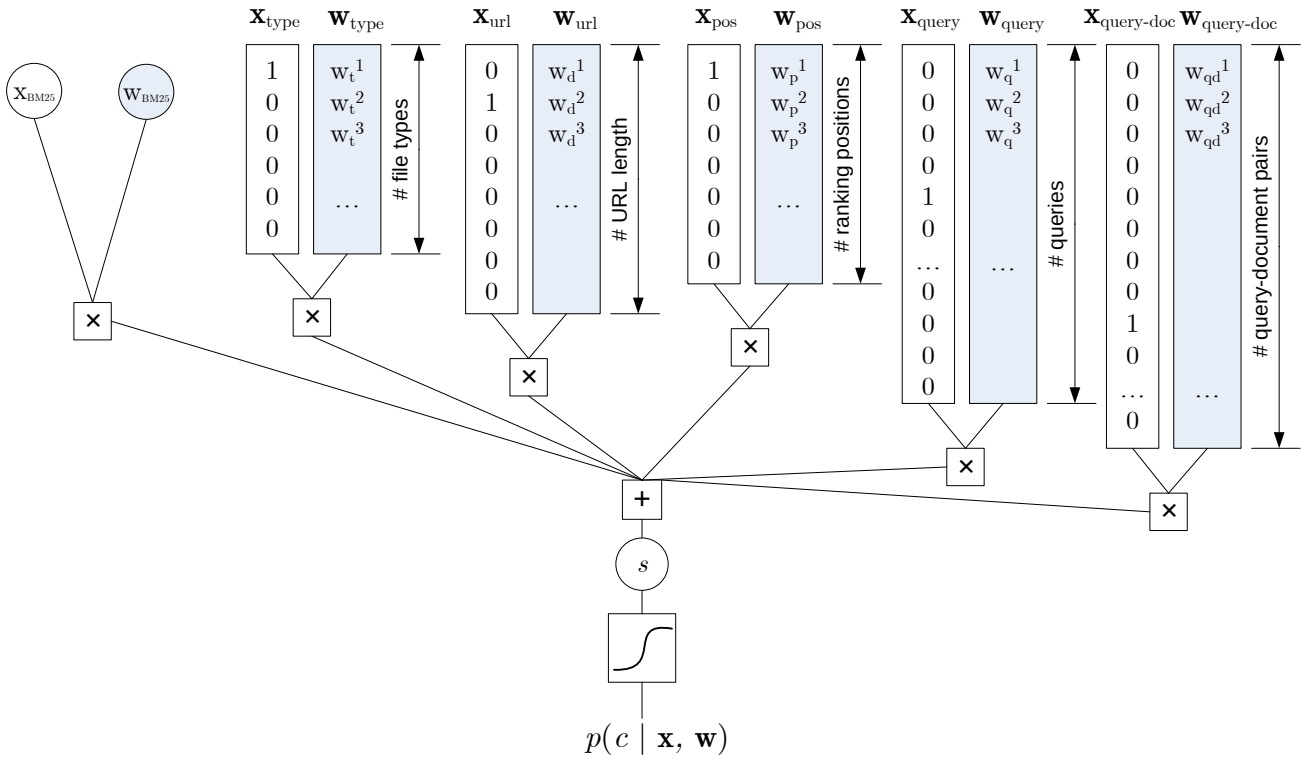


Figure 3: Indicator binary features and the GLM. This is a specific example of the click model shown on the right in Figure 2. Here the inputs x are made explicit as one real-valued feature (BM25) and five bags of binary features. The output is the predicted probability of click.

s is a Gaussian disturbance with inverse precision π. This disturbance can be interpreted as a softening of the step-function such that some noise in the label is supported by the model. It is the direct analog of the choice of probit link function instead of a hard step function in the discrete click prediction case.

3.2 Features

Figure 3 takes a more detailed look at the inputs (features) used for just the click model GLM. The input $\mathbf{x}$ contains parts that are query specific, parts that are document specific, and parts that are derived from the query-document pair. A BM25F score [11] is used as a general input that indicates the match between query and document.

Document specific features include the document file type (e.g. Html, Pdf, Excel etc) and the length of the url. Apart from these basic descriptive features, the vector $\mathbf{x}$ includes binary indicators for the query ID and the query-document ID, and also the rank (position) of the document in the list for which the click event was observed or is to be predicted.

The descriptive features give the model the ability to generalize between queries and documents, and the identifier (ID) weights effectively serve as an instance-specific memory. For frequently seen documents for popular queries the model can store, using the identifier weights, very accurate click predictions, even if they are far from the general trend predicted by the descriptive features. A bias term that is always 1 is included to capture a grand average.

3.3 Training

To learn the distributions of $\mathbf{w}$ we use the approximate Bayesian inference procedure from [15] with a factorized Gaussian prior. The ordinal regression part is treated as in [3] with the difference that here we do not resort to an ML-II approximation of π but integrate it out.

The main benefit of the Bayesian procedure is that with each individual weight in $\mathbf{w}$ a notion of the uncertainty about its optimal value is maintained. This results in a learning algorithm that correctly updates imprecisely determined weights more than precisely determined ones, which is essential for our model. The weights for descriptive features effectively see a lot more data than the query and document specific identifier weights. The Bayesian update rules ensure that each get updated at the right rate — in particular, a small number of examinations will not change the weight distributions nearly as much as a large number. This is something that could not easily be handled in for example maximum likelihood approaches.

3.4 Results

Before any implicit feedback data is available the ranker is based on a model that predicts clicks and labels. The utility we used in the experiments is a simple weighted combination between DCG and click-DCG as given in Equation 6. The specific setting of λ in this utility is a design choice. The dotted line in Figure 5 shows, for the out-of-the-box model, the NDCG@10 on the adaptation set as a function of λ. We see that using the click utility ($\lambda = 1$) actually reduces the NDCG@10 score. This is to be expected, since the NDCG score does not depend on observed clicks. The utility in (6)

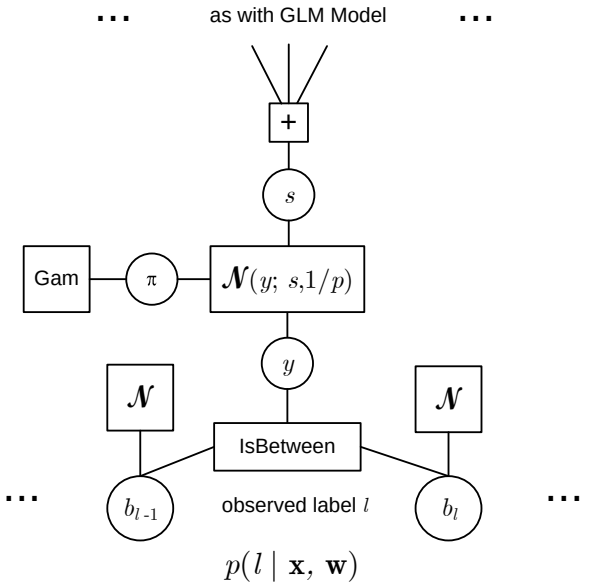


Figure 4: This is a specific example of the label model shown on the left in Figure 2. The inputs are the same as for the click model. However the output is handled differently. First noise is added to the variable s; the result is then constrained to lie between the two threshold variables which correspond to the observed label. Thresholds and noise precision are learnt in addition to the weights.

with $\lambda = 0$ is equivalent to DCG, and setting λ to another value encourages the ranker to optimize a different metric than NDCG@10 shown on the y-axis in Figure 5.

If we use two months of adaptation data, i.e. the site specific click feedback, we get the analogous solid/crossed curve in Figure 5. Here we see that incorporating clicks leads to an improvement of NDCG@10. A value for λ other than 0 and 1 leads effectively to the combination of the two datasets (the train set and the adaptation set). This improves performance, even if we measure the performance of the resulting system with NDCG, an evaluation metric that does not reward clicks.

The lower dashed line in Figure 5 represents a BM25F baseline, with no click data. We note that it is a horizontal line since ranking by BM25F does not involve a λ parameter. We see a 1 point NDCG@10 gain from the features alone ($\lambda = 0$) and an additional 2 points gain if we set $\lambda = 0.5$).

3.4.1 A proxy for a click-metric

The NDCG metric reported in Figure 5 is a well-known metric, but if $\lambda \neq 0$ it is strictly speaking not the metric that the ranker seeks to optimize. If it is decided that (6) with a particular value for λ is the utility that represents end user satisfaction the best, then *that* utility should be the final evaluation metric. Ideally we would like to test a ranker in an on-line setting where it can control the results lists. In such a setting we could monitor the accumulated utility by the ranker. However, since we only have historic

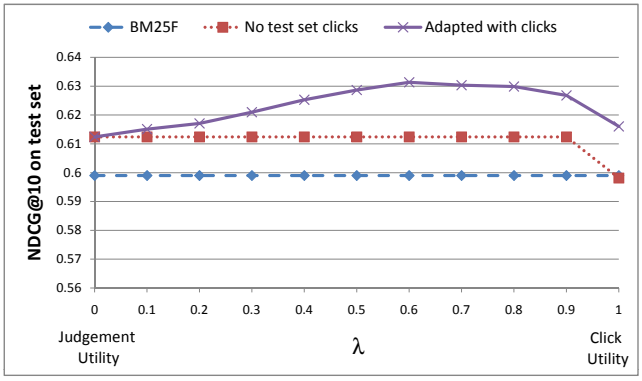


Figure 5: NDCG@10 scores for the different rankers as function of λ, the relative weight given to the click-part in a combined utility function.

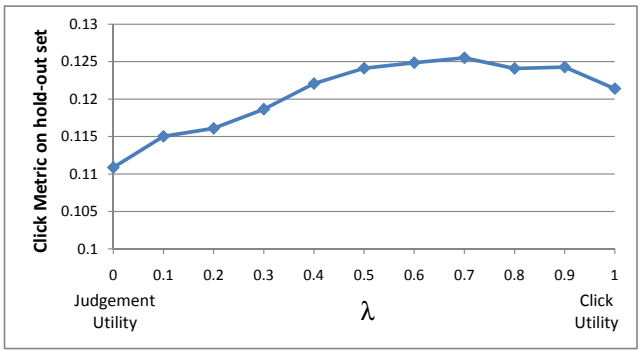


Figure 6: The click based scores from Equation (8) for the different rankers as function of λ.

data available we use the following proxy click metric:

$$S_{clicks} = \frac{\sum_{p=1}^{n} d(p) N_c(p)}{\sum_{p=1}^{n} N_e(p)} \tag{8}$$

where we denote the total number of clicks for the document on position p with $N_c(p)$ and the number of examinations with $N_e(p)$. The numerator in (8) uses the same discount function $d(p)$ as used in (3). Extra in this proxy evaluation metric is the normalization represented by the denominator. This ensures that documents that were not shown to users in the dataset (and hence have 0 clicks and 0 examinations) are properly disregarded. The score above is for a single query, and the total score would be the average over all queries.

Figure 6 shows a plot analogous to Figure 5, but now with the click-based evaluation metric from (8). We note that this new metric gets better with increasing λ. This is to be expected: a ranking formed from a utility based predominantly upon predicted click rate should do better when evaluated with a click-based metric. This provides further orthogonal evidence that combining implicit and explicit feedback improves search results.

To get a feel for the qualitative changes that the different choices of utility function imply it is instructive to look at

Relevance Judgments Utility (DCG)
1.: `http://vsts`
2.: `http://develop/vs2005field`
3.: `http://msdnprod/vstudio`
Click Utility
1.: `http://devdiv`
2.: `http://msdnprod/vstudio`
3.: `http://infoweb/c16/visualstudiodotnet`
Mixed Utility ($\lambda = 0.5$)
1.: `http://msdnprod/vstudio`
2.: `http://vsts`
3.: `http://devdiv`

Table 1: Reorderings of the top-ranked positions for the "Visual Studio" query

a specific example. Table 1 shows the top three results for the query "Visual studio" for $\lambda = 0$ (DCG ranking), $\lambda = 1$ (click ranking) and $\lambda = 0.5$ (balanced ranking). In this example the DCG based top three are all documents that could claim to be a definitive result for searchers interested in using the Visual Studio product. They were all labeled "good" by human assessors. If the ranker is using the click-only utility ($\lambda = 1$) we see that the top three changes. Of the three "good" results in the DCG list, the `msdnprod` link and snippet is apparently the most appealing to the users in the adaptation phase, containing technical information. The other two documents that have entered the top three reflect different interpretations of the query "Visual studio": the devdiv page gives information about the team that creates Visual studio, and the infoweb provides marketing data.

This example demonstrates that there is no unique definition of relevance. If we deem the most popular page to be the most relevant, we should pick the click utility. However, if we want the result list to be more authoritative, a utility based upon explicit judgments might promote pages that are more likely to have been overlooked in a straight snippet-based popularity contest. This advantage of an increased reliability of explicit judgment usually comes with the disadvantage of a single user interpretation of relevance: there is a natural tradeoff between judgment accuracy and result diversity. As Table 1 shows a mixed utility allows us to find a balance between these two extremes.

Including click feedback has had two qualitative effects for the Visual Studio query: (i) a rearrangement of, from the external perspective equally good, documents according to local preferences, and (ii) a promotion of alternative interpretations of the query that are common at the specific intranet. Although we present a single example here, we have seen these effects in many other queries, together with a third major effect: (iii) the correction of erroneous human judgments.

4. SUMMARY

In this paper we have explored the decision theoretic framework for IR and studied how it can be used to combine several sources of feedback into a single ranker. The approach is based on a utility function that describeses the user satisfaction after a search session, and a model that predicts user actions such as clicking and labelling based on known quantities at query time. Constructing a model and formulating a utility function are both difficult problems. But we observe that in the case of label and click stream data the simplest possible utility function and a reasonable prediction model already give many of the potential benefits that the combination of the two streams can bring.

In experiments in an enterprise search setting, we see that the approach leads to increased performance. Qualitatively we see that mislabelled queries get filtered out, result lists for ambiguous queries change to better reflect the most often intended interpretation by users, and tie breaking of identically labelled results is done according to population preference. In terms of NDCG@10 including the click stream in the decision theoretic ranker leads to a two point gain. This is despite the fact that the ranker does not aim to optimize this metric. Given the qualitative results we expect that end user experience improves even more than the two point NDCG gain indicates.

5. REFERENCES

[1] E. Agichtein, E. Brill, and S. Dumais. Improving web search ranking by incorporating user behavior information. In *SIGIR*, 2006.

[2] C. Burges, R. Ragno, and Q. V. L. Le. Learning to rank with nonsmooth cost functions. In *NIPS*, 2006.

[3] W. Chu and Z. Ghahramani. Gaussian processes for ordinal regression. *JMLR*, 6:1019–1041, 2005.

[4] I. J. Cox, M. L. Miller, T. P. Minka, T. Papathomas, and P. N. Yianilos. The Bayesian image retrieval system, pichunter: Theory, implementation and psychophysical experiments. *IEEE Transactions on Image Processing, Special Issue on Image and Video Processing for Digital Libraries*, 9(1):20–37, 2000.

[5] S. Fox, K. Karnawat, M. Mydland, S. T. Dumais, and T. White. Evaluating implicit measures to improve the search experience. *ACM Transactions on Information Systems*, 23:147–168, 2005.

[6] Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *SIGIR*, 2000.

[7] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of Knowledge Discovery in Databases*, 2002.

[8] J. Lafferty and C. Zhai. Document language models, query models, and risk minimization for information retrieval. In *SIGIR*, pages 111–119, 2001.

[9] P. McCullagh and J. A. Nelder. *Generalized Linear Models*. CRC Press, 2nd edition, 1990.

[10] S. Robertson, H. Zaragoza, and M. Taylor. A simple BM 25 extension to multiple weighted fields. In *CIKM*, pages 42–29, 2004.

[11] S. E. Robertson. The probability ranking principle in IR. *Journal of Documentation*, 33:294–304, 1977.

[12] M. Taylor, J. Guiver, S. Robertson, and T. Minka. SoftRank: optimizing non-smooth rank metrics. In *WSDM '08*, pages 77–86. ACM, 2008.

[13] S. K. M. Wong, P. Bollmann, and Y. Y. Yao. Information retrieval based on axiomatic decision theory. *General Systems, 19(2)*, 23(2):101–117, 1991.

[14] O. Zoeter. Bayesian generalized linear models in a terabyte world. In *IEEE Conference on Image and Signal Processing and Analysis*, 2007.

A Framework for Unsupervised Rank Aggregation

Alexandre Klementiev, Dan Roth, and Kevin Small
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
{klementi,danr,ksmall}@uiuc.edu

ABSTRACT

The need to meaningfully combine sets of rankings often comes up when one deals with ranked data. Although a number of heuristic and supervised learning approaches to rank aggregation exist, they generally require either domain knowledge or supervised ranked data, both of which are expensive to acquire. To address these limitations, we propose[1] a mathematical and algorithmic framework for learning to aggregate (partial) rankings in an unsupervised setting, and instantiate it for the cases of combining permutations and combining top-k lists. Furthermore, we also derive an unsupervised learning algorithm for rank aggregation (ULARA), which approximates the behavior of this framework by directly optimizing the weighted Borda count. We experimentally demonstrate the effectiveness of both approaches on the data fusion task.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Retrieval models

General Terms

Algorithms, Theory

Keywords

Ranking, Rank Aggregation, Distance-Based Models

1. INTRODUCTION

Consider the scenario where each member of a panel of judges independently generates a (partial) ranking over a set of items while attempting to reproduce a true underlying ranking according to their level of expertise. This setting motivates a fundamental machine learning and information retrieval (IR) problem - the necessity to meaningfully aggregate preference rankings into a joint ranking. The IR community refers to this as *data fusion*, where a joint ranking is derived from the outputs of multiple retrieval systems, possibly from several heterogeneous sources. A canonical data fusion task is *meta-search* where the aim is to aggregate Web search query results from several engines into a more accurate ranking.

One impediment to solving *rank aggregation* tasks is the high cost associated with acquiring full or partial preference information, making supervised approaches (e.g. [22, 23]) of limited utility. For data fusion, efforts to overcome this difficulty include applying domain specific heuristics [26] or collecting such preference information indirectly (e.g. using clickthrough data [15]). In order to address this limitation, we consider the task of learning to aggregate (partial) rankings *without supervision*.

Analyzing ranked data is an extensively studied problem in statistics [25], economics [2], information retrieval [26], and machine learning literature [1]. Mallows [24] introduced a distance-based model for fully ranked data and investigated its use with Kendall's and Spearman's metrics. The model was later generalized to other distance functions and for use with partially ranked data [4]. [20] proposed a multi-parameter extension, where multiple modal rankings (e.g. expert opinions) are available and use their formalism for supervised ensemble learning; they also analyzed their model for partially ranked data [21].

The first key contribution of our work is the derivation of an EM-based algorithm for learning the parameters of the extended Mallows model without supervision. We instantiate the model with appropriate distance functions for two important scenarios: combining permutations and combining top-k lists. In the context of defining distances between rankings, various metrics have been proposed and analyzed [4, 9]. Distances over top-k lists, i.e. rankings over the k most preferable objects, receive particular attention in the IR community [10]. [12] show that a class of distance functions between full rankings, such as Kendall's and Cayley's metrics, decompose into a sum of independent components allowing for efficient parameter estimation of the standard Mallows model.

The second key contribution of our work is the derivation of a novel decomposable distance function for top-k lists. We show it to be a generalization of the Kendall metric and demonstrate that it can be decomposed, enabling us to estimate the parameters of the extended Mallows model efficiently.

[1]This paper unifies and extends work from [18] and [19]

SIGIR'08 LR4IR Workshop, July 24, 2008, Singapore.

The third contribution is the derivation of an unsupervised learning algorithm for rank aggregation (ULARA), which approximates the learning of the parameters of this model by directly learning the parameters of a weighted Borda count through an optimization procedure.

The remainder of the paper is organized as follows: section 2 formalizes distance-based ranking models and introduces relevant notation. Section 3 derives our EM-based algorithm for learning model parameters and specifies the requirements for efficient learning and inference. Section 4 instantiates the framework for two common scenarios: permutations (full rankings) and top-k lists. Section 5 describes our approximation method based on an optimization of the weighted Borda count. Section 6 experimentally demonstrates the model's effectiveness in both cases. Finally, section 7 concludes the work and gives ideas for future directions.

2. DISTANCE-BASED RANKING MODELS

2.1 Notation and Definitions

Let $\{x_1, \ldots, x_n\}$ be a set of objects to be ranked, i.e. assigned rank-positions $1, \ldots, n$, by a judge. We denote the resulting permutation $\pi = (\pi(1), \ldots, \pi(n))$, where $\pi(i)$ is the rank assigned to object x_i. Correspondingly, we use $\pi^{-1}(j)$ to denote the index of the object assigned to rank j.

Let $\mathcal{S}_n$ be the set of all $n!$ permutations over n items, and let $d : \mathcal{S}_n \times \mathcal{S}_n \rightarrow \mathbb{R}$ be a distance function between two permutations. We will require $d(\cdot, \cdot)$ to be a *right-invariant metric* [6]: in addition to the usual properties of a metric, we will also require that the value of $d(\cdot, \cdot)$ does not depend on how the set of objects is indexed. In other words, $d(\pi, \sigma) = d(\pi\tau, \sigma\tau)\ \forall \pi, \sigma, \tau \in \mathcal{S}_n$, where $\pi\tau$ is defined by $\pi\tau(i) = \pi(\tau(i))$.

In particular, note that the right-invariance property implies that $d(\pi, \sigma) = d(\pi\pi^{-1}, \sigma\pi^{-1}) = d(e, \sigma\pi^{-1})$, where $e = (1, \ldots, n)$ is the identity permutation. That is, the value of d does not change if we re-index the objects such that one of the permutations becomes e and the other $\nu = \sigma\pi^{-1}$. Borrowing the notation from [12] we abbreviate $d(e, \nu)$ as $D(\nu)$. In a later section, when we define ν as a random variable, we may treat $D(\nu) = D$ as a random variable as well: whether it is a distance function or a r.v. will be clear from the context.

2.2 Mallows Models

While a large body of work on ranking models exists in statistics literature, of particular interest to us are the distance based conditional models first introduced in [24]. Let us give a brief review of the formalism and elucidate some of the its properties relevant to our work. The model generates a judge's rankings according to:

$$p(\pi|\theta, \sigma) = \frac{1}{Z(\theta, \sigma)} \exp(\theta\ d(\pi, \sigma)) \tag{1}$$

where $Z(\theta, \sigma) = \sum_{\pi \in \mathcal{S}_n} \exp(\theta\ d(\pi, \sigma))$ is a normalizing constant. The parameters of the model are $\theta \in \mathbb{R}$, $\theta \leq 0$ and $\sigma \in \mathcal{S}_n$, referred to as the dispersion and the location parameters, respectively. The distribution's single mode is the modal ranking σ; the probability of ranking π decreases exponentially with distance from σ. When $\theta = 0$, the distribution is uniform, and it becomes more concentrated at σ as θ decreases.

One property of (1) is that the normalizing constant $Z(\theta, \sigma)$ does not depend on σ due to the right invariance of the distance function:

$$Z(\theta, \sigma) = Z(\theta) \tag{2}$$

Let us denote the moment generating function of D under (1) as $M_{D,\theta}(t)$, and as $M_{D,0}(t)$ under the uniform distribution ($\theta = 0$). Since (1) is an exponential family,

$$M_{D,\theta}(t) = \frac{M_{D,0}(t + \theta)}{M_{D,0}(\theta)}$$

Therefore,

$$\begin{aligned} E_\theta(D) &= \frac{1}{M_{D,0}(\theta)} \frac{dM_{D,0}(t+\theta)}{dt}\bigg|_{t=0} \\ &= \frac{d \ln(M_{D,0}(t))}{dt}\bigg|_{t=\theta} \end{aligned} \tag{3}$$

[12] note that if a distance function can be expressed as $D(\pi) = \sum_{i=1}^{m} V_i(\pi)$, where $V_i(\pi)$ are independent (with π uniformly distributed) with m.-g.f. $M_i(t)$, then $M_{D,0}(t) = \prod_{i=1}^{m} M_i(t)$. Consequently, (3) gives:

$$E_\theta(D) = \frac{d}{dt} \sum_{i=1}^{m} \ln M_i(t)\bigg|_{t=\theta} \tag{4}$$

We will call such distance functions *decomposable* and will later use (4) in section 4 in order to estimate θ efficiently.

2.3 Extended Mallows Models

[20] propose a natural generalization of the Mallows model to the following conditional model:

$$p(\pi|\boldsymbol{\theta}, \boldsymbol{\sigma}) = \frac{1}{Z(\boldsymbol{\theta}, \boldsymbol{\sigma})} p(\pi) \exp\left(\sum_{i=1}^{K} \theta_i\ d(\pi, \sigma_i)\right) \tag{5}$$

where $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_K) \in \mathcal{S}_n^K$, $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_K) \in \mathbb{R}^K$, $\boldsymbol{\theta} \leq \mathbf{0}$, $p(\pi)$ is a prior, and normalizing constant $Z(\boldsymbol{\theta}, \boldsymbol{\sigma}) = \sum_{\pi \in \mathcal{S}_n} p(\pi) \exp(\sum_{i=1}^{K} \theta_i\ d(\pi, \sigma_i))$.

The rankings σ_i may be thought of as votes of K individual judges, e.g. rankings returned by multiple search engines for a particular query in the meta-search setting. The free parameters θ_i represent the degree of expertise of the individual judges: the closer the value of θ_i to zero, the less the vote of the i-th judge affects the assignment of probability.

Under the right-invariance assumption on d, we can use property (2) to derive the following generative story underlying the extended Mallows model:

$$p(\pi, \boldsymbol{\sigma}|\boldsymbol{\theta}) = p(\pi) \prod_{i=1}^{K} p(\sigma_i|\theta_i, \pi) \tag{6}$$

That is, π is first drawn from prior $p(\pi)$. $\boldsymbol{\sigma}$ is then made up by drawing $\sigma_1 \ldots \sigma_K$ *independently* from K Mallows models $p(\sigma_i|\theta_i, \pi)$ with the *same* location parameter π.

It is straightforward to generalize both Mallows models [4], and the extended Mallows models to *partial rankings* by constructing appropriate distance functions. We will assume this more general setting in the following section.

3. LEARNING AND INFERENCE

In this section, we derive the general formulation of Expectation Maximization algorithm for parameter estimation of the extended Mallows models (5), and suggest a class of distance functions for which learning can be done efficiently. We then describe an inference procedure for the model.

3.1 EM Background and Notation

Let us start with a brief overview of Expectation Maximization (EM) [5], mostly to introduce some notation. EM is a general method of finding maximum likelihood estimate of parameters of models which depend on unobserved variables. The EM procedure iterates between:

E step: estimate the expected value of complete data log-likelihood with respect to unknown data $\mathcal{Y}$, observed data $\mathcal{X}$, and current parameter estimates θ':

$$T(\theta, \theta') \quad = \quad E[\log p(\mathcal{X}, \mathcal{Y}|\theta)|\mathcal{X}, \theta']$$

M step: choose parameters that maximize the expectation computed in the E step:

$$\theta' \leftarrow \operatorname*{argmax}_{\theta} T(\theta, \theta')$$

In our setting, the $K > 2$ experts generate votes $\boldsymbol{\sigma}$ corresponding to the unobserved true ranking π. We will see multiple instances of $\boldsymbol{\sigma}$ so the observed data we get are ranking vectors $\mathcal{X} = \{\boldsymbol{\sigma}^{(j)}\}_{j=1}^{Q}$ with the corresponding true (unobserved) rankings $\mathcal{Y} = \{\pi^{(j)}\}_{j=1}^{Q}$.

In the meta-search example, $\sigma_i^{(j)}$ is the ranking of the i-th (of the total of K) search engine for the j-th (of the total of Q) query. The (unknown) true ranking corresponding to the j-th query is denoted as $\pi^{(j)}$.

3.2 EM Derivation

We now use the generative story (6) to derive the following propositions (proofs omitted due to space constraints):

PROPOSITION 1. *The expected value of the complete data log-likelihood under* (5) *is:*

$$T(\boldsymbol{\theta}, \boldsymbol{\theta'}) = \sum_{(\pi^{(1)},\ldots,\pi^{(Q)}) \in \mathcal{S}_n^Q} \mathcal{L}_{\boldsymbol{\theta}} \; \mathcal{U}_{\boldsymbol{\theta'}} \qquad (7)$$

where the complete data log-likelihood $\mathcal{L}_{\boldsymbol{\theta}}$ *is:*

$$\mathcal{L}_{\boldsymbol{\theta}} = \sum_{j=1}^{Q} \log p(\pi^{(j)}) - Q \sum_{i=1}^{K} \log Z(\theta_i) + \sum_{j=1}^{Q} \sum_{i=1}^{K} \theta_i \; d(\pi^{(j)}, \sigma_i^{(j)})$$

and the marginal distribution of the unobserved data $\mathcal{U}_{\boldsymbol{\theta'}}$ *is:*

$$\mathcal{U}_{\boldsymbol{\theta'}} = \prod_{j=1}^{Q} p\left(\pi^{(j)}|\boldsymbol{\theta'}, \boldsymbol{\sigma}^{(j)}\right)$$

PROPOSITION 2. $T(\boldsymbol{\theta}, \boldsymbol{\theta'})$ *is maximized by* $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_K)$ *such that:*

$$E_{\theta_i}(D) = \sum_{\substack{(\pi^{(1)},\ldots,\pi^{(Q)}) \\ \in \mathcal{S}_n^Q}} \left(\frac{1}{Q} \sum_{q=1}^{Q} d(\pi^{(q)}, \sigma_i^{(q)}) \right) \mathcal{U}_{\boldsymbol{\theta'}} \qquad (8)$$

That is, on each iteration of EM, we need to evaluate the right-hand side (RHS) of (8) and solve the LHS for θ_i for each of the K components.

3.3 Model Learning and Inference

At first, both evaluating the RHS of (8) and solving the LHS for θ_i seem quite expensive ($> n!$). While true in general, we can make the learning tractable for a certain type of distance functions.

In particular, if a distance function can be decomposed into a sum of independent components under the uniform distribution of π (see section 2.2), property (4) may enable us to make the estimation of the LHS efficient. In Section 4, we show two examples of such distance functions (for permutations and top-k lists).

In order to estimate the RHS, we use the Metropolis algorithm [14] to sample from (5). The chain proceeds as follows: denoting the most recent value sampled as π_t, two indices $i, j \in \{1, \ldots, n\}$ are chosen at random and the objects $\pi_t^{-1}(i)$ and $\pi_t^{-1}(j)$ are transposed forming π'_t. If $a = p(\pi'_t|\boldsymbol{\theta}, \boldsymbol{\sigma})/p(\pi_t|\boldsymbol{\theta}, \boldsymbol{\sigma}) \geq 1$ the chain moves to π'_t. If $a < 1$, the chain moves to π'_t with probability a; otherwise, it stays at π_t. [7] demonstrates rapid convergence for Mallows model with Cayley's distance. While no convergence results are known for the extended Mallows model with arbitrary distance, we found experimentally that the MC chain converges rapidly with the two distance functions used in this work ($10n$ steps in experiments of Section 6). As the chain proceeds, we update the distance value with the incremental change due to a single transposition, instead of recomputing it from scratch, resulting in substantial savings in computation.

Alternatively, we also found (Section 6.1) that combining rankings σ_i with the Borda count weighted by $\exp(-\theta_i)$ provides a reasonable and quick estimate for evaluating the RHS.

Sampling or the suggested alternative RHS estimation used during training is also used for model inference.

4. MODEL APPLICATION

Overcoming the remaining hurdle (the LHS estimation) in learning the model efficiently depends on the definition of a distance function. We now consider two particular types of (partial) rankings: permutations, and top-k lists. The latter is the case when each judge specifies a ranking over k most preferable objects out of n. For instance, a top-10 list may be associated with the 10 items on the first page of results returned by a web search engine. For both permutations and top-k lists, we show distance functions which satisfy the decomposability property (Section 2.2), which, in turn, allows us to estimate the LHS of (8) efficiently.

4.1 Combining Permutations

Kendall's tau distance [16] between permutations π and σ is a right-invariant metric defined as the minimum number of pairwise adjacent transpositions needed to turn one permutation into the other. Assuming that one of the permutations, say σ, is the identity permutation e (we can always turn one of the permutations into e by re-indexing the objects without changing the value of the distance, see Section 2.1), it can be written as:

$$D_K(\pi) = \sum_{i=1}^{n-1} V_i(\pi)$$

where[2] $V_i(\pi) = \sum_{j>i} I(\pi^{-1}(i) - \pi^{-1}(j))$. V_i are independent and uniform over integers $[0, n-i]$ [11] with m.-g.f. $M_i(t) = \frac{1}{n-i+1}\sum_{k=0}^{n-i} e^{tk}$. Following [12], equation (4) gives:

$$E_\theta(D_K) = \frac{ne^\theta}{1-e^\theta} - \sum_{j=1}^{n} \frac{je^{\theta j}}{1-e^{\theta j}} \quad (9)$$

$E_\theta(D_K)$ is monotone decreasing, so line search for θ will converge quickly.

4.2 Combining Top-*k* Lists

We now propose an extension of the Kendall's tau distance to top-k lists, i.e. the case where π and σ indicate preferences over different (possibly, overlapping) subsets of $k \leq n$ objects.

Let us denote by F_π and F_σ the elements in π and σ respectively, noting that $|F_\pi| = |F_\sigma| = k$. We define $Z = F_\pi \cap F_\sigma$, $|Z| = z$, $P = F_\pi \setminus F_\sigma$, and $S = F_\sigma \setminus F_\pi$ (note that $|P| = |S| = k - z = r$). We treat π and σ as rankings, which in our case means that the smallest index will indicate the top, i.e. contain the most preferred object. For notational convenience, let us now define the *augmented ranking* $\tilde{\pi}$ as π augmented with the elements of S assigned the same index $(k+1)$, one past the bottom of the ranking as shown on Figure 1 ($\tilde{\sigma}$ is defined similarly). We will slightly abuse our notation and denote $\tilde{\pi}^{-1}(k+1)$ to be the set of elements in position $(k+1)$.

Kendall's tau distance D_K is naturally extended from permutations to augmented rankings.

DEFINITION 1. *Distance $\tilde{D}_K(\tilde{\pi}, \tilde{\sigma})$ between augmented rankings $\tilde{\pi}$ and $\tilde{\sigma}$ is the minimum number of adjacent transpositions needed to turn $\tilde{\pi}$ into $\tilde{\sigma}$.*

It can be shown that $\tilde{D}_K(\tilde{\pi}, \tilde{\sigma})$ is a right-invariant metric, thus we will again simplify the notation denoting it as $\tilde{D}_K(\tilde{\pi})$. This distance can be decomposed as:

$$\tilde{D}_K(\tilde{\pi}) = \sum_{\substack{i=1 \\ \tilde{\pi}^{-1}(i)\in Z}}^{k} \tilde{V}_i(\tilde{\pi}) + \sum_{\substack{i=1 \\ \tilde{\pi}^{-1}(i)\notin Z}}^{k} \tilde{U}_i(\tilde{\pi}) + \frac{r(r+1)}{2}$$

where

$$\tilde{V}_i(\tilde{\pi}) = \sum_{\substack{j=i \\ \tilde{\pi}^{-1}(j)\in Z}}^{k} I(\tilde{\pi}^{-1}(i) - \tilde{\pi}^{-1}(j)) + \sum_{j\in\tilde{\pi}^{-1}(k+1)} I(\tilde{\pi}^{-1}(i) - j)$$

$$\tilde{U}_i(\tilde{\pi}) = \sum_{\substack{j=i \\ \tilde{\pi}^{-1}(j)\in Z}}^{k} 1$$

[2] $I(x) = 1$ if the variable $x > 0$ or a predicate x is true, and 0 otherwise.

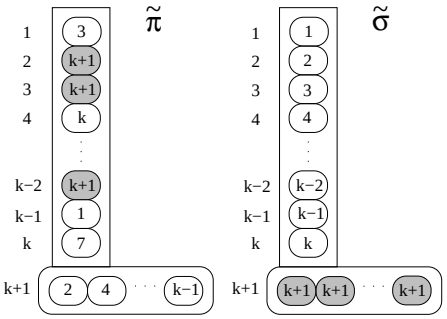


Figure 1: An example of augmented permutations $\tilde{\pi}$ (left) and identity augmented permutation $\tilde{\sigma}$ (right, in natural order). Grey boxes are objects in π but not in σ. $\tilde{D}_K(\tilde{\pi})$ is the minimum number of adjacent transpositions needed to turn $\tilde{\pi}$ into $\tilde{\sigma}$: namely, bring all grey boxes into the position $k+1$ and put the remaining k objects in their natural order.

Decomposing $\tilde{D}_K(\tilde{\pi})$, the second term is the minimum number of adjacent transpositions necessary to bring the r elements not in Z (grey boxes on Figure 1) to the bottom of the ranking. The third term is the minimum number of adjacent transpositions needed to switch them with the elements in $\tilde{\pi}^{-1}(k+1)$, which would then appear in the correct order in the bottom r positions. Finally, the first term is the adjacent transpositions necessary to put the k elements now in the list in the natural order.

It can be shown that the random variable summands comprising $\tilde{D}_K(\tilde{\pi})$ are independent when $\tilde{\pi}$ is uniformly distributed. Furthermore, $\tilde{V}_i$ and $\tilde{U}_j$ are uniform over integers $[0, k-i]$ and $[0, z]$, with moment generating functions $\frac{1}{k-i+1}\sum_{j=0}^{k-i} e^{tj}$ and $\frac{1}{z+1}\sum_{j=0}^{z} e^{tj}$, respectively. Assuming $z > 0$, and $r > 0$ equation (4) gives:

$$E_\theta(\tilde{D}_K) = \frac{ke^\theta}{1-e^\theta} - \sum_{j=r+1}^{k} \frac{je^{j\theta}}{1-e^{j\theta}} + \frac{r(r+1)}{2} - r(z+1)\frac{e^{\theta(z+1)}}{1-e^{\theta(z+1)}} \quad (10)$$

If $r = 0$ (i.e. the augmented rankings are over the same objects), both the distance and the expected value reduce to the Kendall distance results. Also, if $z = 0$ (i.e. the augmented rankings have no objects in common), $\tilde{D}_K = E_\theta(\tilde{D}_K) = k(k+1)/2$, which is the smallest number of adjacent transpositions needed to move the $r = k$ objects in $\tilde{\pi}^{-1}(k+1)$ into the top k positions.

$E_\theta(\tilde{D}_K)$ decreases monotonically, so we can again use line search to find the value of θ. Notice that the expected value depends on the value of z (the number of common elements between the two permutations). We will compute the average value of z as we estimate the RHS of (8) and use it to solve the LHS for θ.

5. DIRECTLY OPTIMIZING THE WEIGHTED BORDA COUNT

A common approach to aggregate a *single* set of votes is to find a permutation with the minimum average Kendall tau distance to those votes. Computing such a ranking,

Algorithm 1 Training

1: **Input:** $\{\boldsymbol{\sigma}^{(q)}\}_{q=1}^{Q}, \{\kappa_i\}_{i=1}^{K}, \lambda, \nu$
2: $\mathbf{w} \leftarrow \mathbf{0}$
3: $t \leftarrow 1$
4: **for** $q \leftarrow 1, \ldots, Q$ **do**
5: **for** $j \leftarrow 1, \ldots, n$ **do**
6: **if** $K_j^{(q)} \geq \nu$ **then**
7: $\mu^{(q)}(j) = \frac{\sum_{i \in \mathcal{K}_j^{(q)}} \sigma_i^{(q)}(j)}{K_j^{(q)}}$
8: **for** $i \leftarrow 1, \ldots, K$ **do**
9: **if** $\sigma_i^{(q)}(j) \leq \kappa_i$ **then**
10: $\nabla_i \leftarrow \left[\sigma_i^{(q)}(j) - \mu^{(q)}(j)\right]^2$
11: **else**
12: $\nabla_i \leftarrow \left[\kappa_i + 1 - \mu^{(q)}(j)\right]^2$
13: $w_i^t \leftarrow w_i^{t-1} + \lambda \cdot \nabla_i$
14: $t \leftarrow t + 1$
15: NORMALIZE(w)
16: **Output:** $\boldsymbol{w} \in [0,1]^K$

Figure 2: An unsupervised algorithm for rank aggregation: Training.

known as the Kemeny-optimal aggregation, is known to be NP-hard [8]. However, the well known Borda count method provides a good approximation [3] and is known to minimize the average Spearman's distance (11) to the constituent rankings.

$$d_S(\sigma, \pi) = \sum_{i=1}^{n} (\sigma(i) - \pi(i))^2 \tag{11}$$

Empirically, we found (see Section 6.1) the Borda count method *augmented* with weights representing relative ranker quality to be a good alternative in the inference step of Section 3.3. Here, we explore this idea further and propose a simple unsupervised algorithm (ULARA) to learn these weights directly by minimizing the empirical average (weighted) Spearman's distance between the votes of the constituent rankers and a surrogate true ranking. As before, we extend the algorithm to handle the top-k setting.

More formally, for an item x_j and a given query, let $\rho_B(j) = \sum_{i=1}^{K} \sigma_i(j); \forall j = 1, \ldots, n$ be the Borda count yielding a predicted true ranking $\hat{\pi}_B = \text{argsort}_{j=1,\ldots,n}\, \rho_B(j)$. Correspondingly, let $\rho_W(j) = \sum_{i=1}^{K} w_i \cdot \sigma_i(j); \forall j = 1, \ldots, n$ be the weighted Borda count yielding a predicted true ranking $\hat{\pi}_W = \text{argsort}_{j=1,\ldots,n}\, \rho_W(j)$. Furthermore, denote $\mathcal{K}_j$ as the set of rankers which placed the item x_j above their respective set threshold values κ_i, and $|\mathcal{K}_j| = K_j = \sum_{i=1}^{K} I(\sigma_i(j) \leq \kappa_i)$. Finally, let $\mu(j) = \frac{\sum_{i \in \mathcal{K}_j} \sigma_i(j)}{K_j}$ denote the mean ranking of x_j. The values $\mu = (\mu(1), \ldots, \mu(n))$ will be used in computing the distance (11) and will play the role of a surrogate true ranking. Slightly abusing the notation, we will use μ in place of a permutation when computing distance $d_S(\sigma, \mu)$.

We aim to learn the weights $\boldsymbol{w}$ minimizing the average weighted Spearman's distance with the additional restriction that they are positive and add up to one, i.e.

Algorithm 2 Evaluation

1: **Input:** $\boldsymbol{w}, \boldsymbol{\sigma}, \{\kappa_i\}_{i=1}^{K}$
2: **for** $j \leftarrow 1, \ldots, n$ **do**
3: $\rho_W(j) \leftarrow 0$
4: **for** $i \leftarrow 1, \ldots, K$ **do**
5: **if** $\sigma_i(j) \leq \kappa_i$ **then**
6: $\rho_W(j) \leftarrow \rho_W(j) + w_i \cdot \sigma_i(j)$
7: **else**
8: $\rho_W(j) \leftarrow \rho_W(j) + w_i \cdot (\kappa_i + 1)$
9: $\hat{\pi}_W \leftarrow \text{argsort}_{j=1,\ldots,n}\, \rho_W(j)$
10: **Output:** $\hat{\pi}_W$

Figure 3: An unsupervised algorithm for rank aggregation: Evaluation.

$$\underset{\boldsymbol{w}}{\text{argmin}} \quad \sum_{q=1}^{Q} \sum_{i=1}^{K} w_i \; d_S(\sigma_i^{(q)}, \mu^{(q)}) \tag{12}$$

$$\text{s.t.} \quad \sum_{i=1}^{K} w_i = 1; \forall i, w_i \geq 0. \tag{13}$$

Informally, the weights should be small for rankers which tend to disagree with the others, and vice versa.

5.1 The algorithm

As opposed to optimizing this problem directly, we use iterative gradient descent [17] to derive an online learning algorithm 1. It takes as input a set of rankings for each query $\{\boldsymbol{\sigma}^{(q)}\}_{q=1}^{Q}$ along with the associated ranking function threshold values $\{\kappa_i\}_{i=1}^{K}$, a learning rate λ, and a significance threshold value ν, all discussed in greater detail below. For each query q and item x_j, the rankings $(\sigma_1^{(q)}(j), \ldots, \sigma_K^{(q)}(j))$ are used to calculate the mean $\mu^{(q)}(j)$ (line 7), the gradient is determined (line 10), and the weight update is made (line 13). Once all of these updates are completed, the weight vector is normalized (line 15) to generate a probability vector for evaluation in algorithm 2. The remaining discussion entails algorithmic details for practical situations:

- Missing Rankings (κ_i) - For most settings, there are more items in the instance space than the individual ranking functions will return. In the top-k setting, for instance, systems return rankings over a subset of documents and most corpus documents remain unranked. We denote this threshold value as κ_i, noting that rankers may have different thresholds. If an item does not appear in a ranking, we substitute $\kappa_i + 1$ for update calculations (line 12), assuming unranked items are ranked just below the last ranked item.

- Variable Number of Rankers (ν) - Some items may only appear in the rankings of a subset of judges. If less than ν rankers, as defined by the user, rank an item, no updates are made for this item (line 6).

6. EXPERIMENTAL EVALUATION

We demonstrate the effectiveness of our approach for permutations and top-k lists considered in Section 4, as well the performance of the alternative algorithm proposed in Section 5.

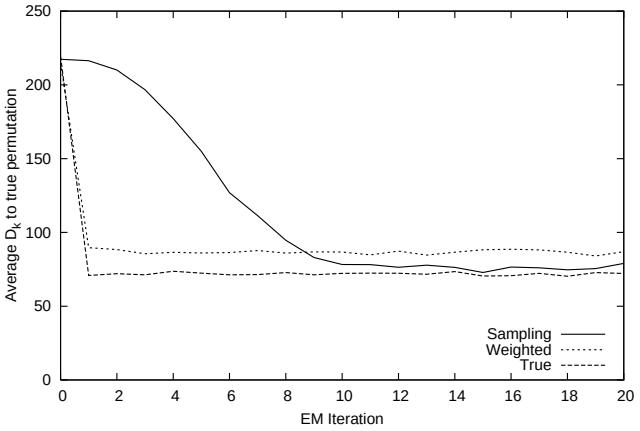


Figure 4: Permutations: learning performance of the model (averaged over 5 runs) when RHS is estimated using sampling (Sampling), the proposed weighted Borda count approximation (Weighted), or the true permutation (True). As expected, the model trained with the sampling method achieves better performance, but the approximation method performs quite well too and converges faster.

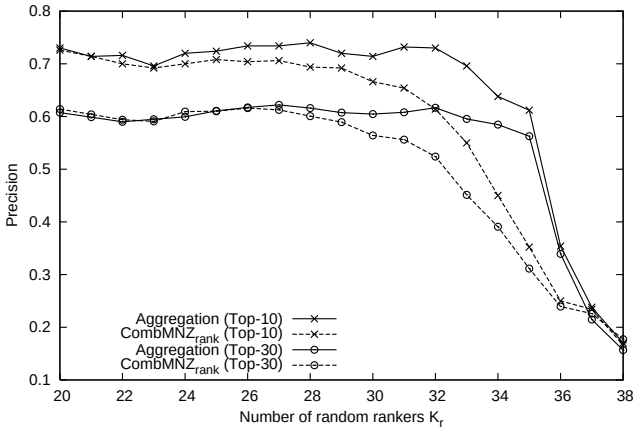


Figure 5: Top-k lists: precision of the aggregate ranker as a function of the number of random component rankers K_r in top 10 and top 30 documents. Our algorithm learns to discount the random components without supervision substantially improving over $CombMNZ_{rank}$.

6.1 Permutations

We first consider the scenario of aggregating permutations. For this set of experiments, the votes of $K = 10$ individual experts were produced by sampling standard Mallows models (1), with the same location parameter $\sigma^* = e$ (an identity permutation over $n = 30$ objects), and concentration parameters $\theta^*_{1,2} = -1.0$, $\theta^*_{3,..,9} = -0.05$, and $\theta^*_{10} = 0$ (the latter generating all permutations uniformly randomly). The models were sampled 10 times, resulting in $Q = 10$ lists of permutations (one for each "query"), which constituted the training data.

In addition to the sampling procedure described in Section 3.3 to estimate the RHS of (8), we also tried the following approximation. For each "query" q, K constituent votes were combined into a single permutation $\hat{\sigma}_q$ with the weighted Borda method (defined in Section 5). The weights are computed using the current values of the model parameters as $\exp(-\theta_i)$. The rationale is that the smaller the absolute value of θ_i, the lower the relative quality of the ranker, and the less it should contribute to the aggregate vote. Finally, the RHS for the i-th component is computed as the distance from its vote to $\hat{\sigma}_q$ averaged over all Q queries.

We also tried using the true permutation σ^* in place of $\hat{\sigma}_q$ to see how well the learning procedure can do.

At the end of each EM iteration, we sampled the current model (5), and computed the Kendall's tau distance between the generated permutation to the true σ^*. Figure 4 shows the model performance when sampling and the proposed approximation are used to estimate the RHS. Although the convergence is much faster with the approximation, the model trained with the sampling method achieves better performance approaching the case when the true permutation is known.

6.2 Top-*k* lists

In order to estimate the model's performance in the top-k list combination scenario, we performed data fusion experiments using the data from the ad-hoc retrieval shared task of the TREC-3 conference [13]. Our goal here is to examine the behavior of our approach as we introduce poor judges into the constituent ranker pool. In this shared task, 40 participants submitted top-1000 ranking over a large document collection for each of the 50 queries. For our experiments, we used top-100 ($k = 100$) rankings from $K = 38$ of the participants (two of the participants generated shorter rankings for some of the queries and were not used) for all $Q = 50$ queries. We replaced a specific number $K_r \in [0, K]$ of the participants with random rankers (drawing permutations of k documents from the set of documents returned by all participants for a given query uniformly randomly). We then used our algorithm to combine top-k lists from K_r random rankers and $(K - K_r)$ participants chosen at random.

We measure performance using the precision in top-$\{10, 30\}$ documents as computed by *trec_eval*[3] from the TREC conference series. As a baseline, we use $CombMNZ_{rank}$, a variant of a commonly used $CombMNZ$ [26]. Given a query q for each document x_j in the collection it computes a score $\rho_{MNZ}(j) = K_j \cdot \sum_{i=1}^{K}(k+1-\sigma_i^{(q)}(j))$, where $\sigma_i^{(q)}(j) = (k+1)$ if the document doesn't appear in the ranking. K_j is the total number of participants which place x_j in their top-k rankings. The aggregate ranking is obtained by sorting documents according to their scores in descending order. Essentially, $CombMNZ_{rank}$ is a weighted Borda count method where the weighting is determined by the number of judges that rank the given document. Intuitively, the more judges rank a document highly, the higher it appears in the aggregate ranking.

Figure 5 shows that our algorithm learns to discount the random components *without supervision* substantially improving over the baseline as $K_r \to K$.

6.3 Model Dispersion Parameters

In order to demonstrate the relationship between the learned dispersion parameters of the model, $\boldsymbol{\theta}$, and the relative per-

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

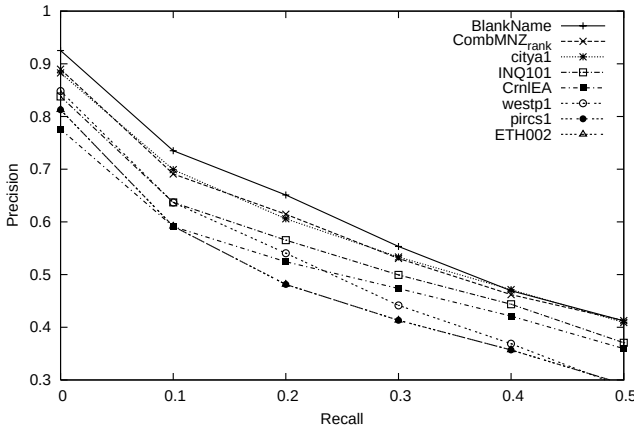


Figure 6: Experimental results for data fusion of the retrieval systems submitted to the TREC-3 shared task. While $CombMNZ_{rank}$ only negligibly outperforms the top system, ULARA performs significantly better than any component system at multiple recall levels.

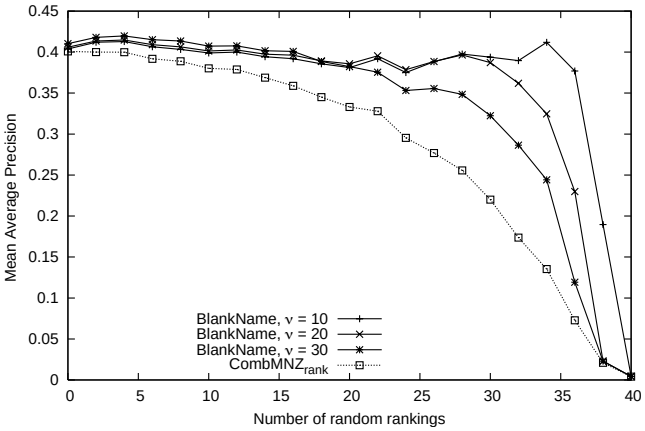


Figure 7: Experimental results where a number of TREC-3 systems are replaced with random rankings, demonstrating robustness of ULARA. While the performance of the $CombMNZ_{rank}$ algorithm deteriorates rapidly, ULARA performs well even when more than half of the systems are replaced.

Table 1: $MRPR$ of the four search engines and their corresponding model parameters; the results suggest a correlation between the magnitude of the dispersion parameters and the relative system performance.

	$S1$	$S2$	$S3$	$S4$
θ	-0.065	0.0	-0.066	-0.049
$MRPR$	0.86	0.43	0.82	0.78

formance of the constituent rankers, we also conducted a meta-search experiment. First, we generated $Q = 50$ queries which result in an unambiguous most relevant document and submitted them to $K = 4$ commercial search engines. For each engine, we kept the 100 highest ranked documents (10 pages of 10 documents each) after removing duplicates, and unified URL formatting differences between engines. We measure performance with Mean Reciprocal Page Rank ($MRPR$), which we define as mean reciprocal rank of the page number on which the correct document appears.

Table 1 shows $MRPR$ of the four search engines and their corresponding model parameters. As expected, the results suggest a correlation between the magnitude of the dispersion parameters and the relative system performance, implying that their values may also be used for unsupervised search engine evaluation. Finally, our model achieves $MRPR = 0.92$ beating all of the constituent rankers.

6.4 Top-*k* lists with ULARA

We also studied the ad-hoc retrieval shared task of the TREC-3 conference with the alternative algorithm we proposed in Sect. 5, demonstrating competitive behavior with significantly faster running times. In this experiment, we used the top-1000 rankings for each of the 50 queries from all $K = 40$ participants, setting $\kappa = 1000$. ULARA was used to combine the rankings of the individual research groups into an aggregate ranking $\hat{\pi}_W$. As previously, performance is quantified by the precision/recall curves and mean average precision metric as provided by the software (`trec_eval`) from the TREC conference series.

Figure 6 shows the results of the top individual submissions, $CombMNZ_{rank}$, and ULARA for the data fusion task. We observe that ULARA outperforms all component ranking functions as well as $CombMNZ_{rank}$. More significantly, while $CombMNZ_{rank}$ performs slightly better than the top system, ULARA achieves a relative increase in average precision of 4.0% at the top ranking, 6.4% at 0.1 recall, and 6.0% at 0.2 recall over $CombMNZ_{rank}$.

In the second experiment, as in Sec. 6.2, we demonstrate the robustness properties of ULARA by adding poor judges to constituent ranker pool. As before, we replaced a specified number of the $K = 40$ systems with a rankings drawn uniformly from all documents returned by all systems for a given query, denoted as *random rankings*. As figure 7 shows, the mean average precision of ULARA versus $CombMNZ_{rank}$ is consistently superior, becoming more pronounced as the number of random rankings is increased. To further explore this effect, we varied ν and observe that as more noise is added, ν must be lowered to accommodate the lack of agreement between rankers. Even under relatively extreme cicumstances, ULARA produces an aggregate ranking competitive with a noise free system; however, unlike the approach in Section 4, it does require manual setting of additional parameters.

7. CONCLUSIONS AND FUTURE WORK

We propose a formal mathematical and algorithmic framework for aggregating (partial) rankings without supervision. We derive an EM-based algorithm for the extended Mallows model and show that it can be made efficient for the right-invariant decomposable distance functions. We instantiate the framework and experimentally demonstrate its effectiveness for the important cases of combining permutations and combining top-k lists. In the latter case, we introduce the notion of augmented permutation and a novel decomposable distance function for efficient learning. In addition, we present an unsupervised algorithm for rank aggregation (ULARA) which approximates the mathematical framework

by directly optimizing a weighted Borda count.

A natural extension of the current work is to instantiate our framework for other types of partial rankings, as well as to cases where ranking data is not of the same type. The latter is of practical significance since often preference information available is expressed differently by different judges (e.g. top-k rankings of different lengths).

Another direction for future work is to extend the rank aggregation model to accommodate position dependence. In IR, more importance is generally given to results appearing higher in the rankings. Within our framework one may be able to design a distance function reflecting this requirement. Additionally, the quality of votes produced by individual components may depend on the rank, e.g. in the top-k scenario some rankers may be better at choosing few most relevant objects, while others may tend to have more relevant objects in the k selected but may not rank them well relative to one another. This case may be modeled by adding a dependency on rank to the dispersion parameters of the model.

Acknowledgments

We would like to thank Ming-Wei Chang, Sariel Har-Peled, Vivek Srikumar, and the anonymous reviewers for their valuable suggestions. This work is supported by NSF grant ITR IIS-0428472, DARPA funding under the Bootstrap Learning Program and by MIAS, a DHS-IDS Center for Multimodal Information Access and Synthesis at UIUC.

8. REFERENCES

[1] W. W. Cohen, R. E. Schapire, and Y. Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 10:243–270, 1999.

[2] V. Conitzer. *Computational Aspects of Preference Aggregation*. PhD thesis, Carnegie Mellon University, 2006.

[3] D. Coppersmith, L. Fleischer, and A. Rudra. Ordering by weighted number of wins gives a good ranking for weighted tournaments. In *Proc of the Annual ACM Symposium on Discrete Algorithms*, pages 776–782, 2006.

[4] D. E. Critchlow. *Metric Methods for Analyzing Partially Ranked Data*, volume 34 of *Lecture Notes in Statistics*. Springer-Verlag, 1985.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39:1–38, 1977.

[6] P. Diaconis and R. L. Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society*, 39:262–268, 1977.

[7] P. Diaconis and L. Saloff-Coste. What do we know about the Metropolis algorithm? *Journal of Computer and System Sciences*, 57:20–36, 1998.

[8] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proc. of the International World Wide Web Conference (WWW)*, pages 613–622, 2001.

[9] V. Estivill-Castro, H. Mannila, and D. Wood. Right invariant metrics and measures of presortedness. *Discrete Applied Mathematics*, 42:1–16, 1993.

[10] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. *SIAM Journal on Discrete Mathematics*, 17:134–160, 2003.

[11] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. John Wiley and Sons, Inc., 1968.

[12] M. A. Fligner and J. S. Verducci. Distance based ranking models. *Journal of the Royal Statistical Society*, 48:359–369, 1986.

[13] D. Harman. Overview of the third Text REtrieval Conference (TREC-3), 1994.

[14] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, April 1970.

[15] T. Joachims. Unbiased evaluation of retrieval quality using clickthrough data. In *SIGIR Workshop on Mathematical/Formal Methods in Information Retrieval*, 2002.

[16] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, Jun. 1938.

[17] J. Kivinen and M. K. Warmuth. Additive versus exponentiated gradient updates for linear prediction. In *Proc, of the Annual ACM Symposium on Theory of Computing*, pages 209–218, 1995.

[18] A. Klementiev, D. Roth, , and K. Small. An unsupervised learning algorithm for rank aggregation. In *Proc. of the European Conference on Machine Learning (ECML)*, pages 616–623, 2007.

[19] A. Klementiev, D. Roth, , and K. Small. Unsupervised rank aggregation with distance-based models. In *Proc. of the International Conference on Machine Learning (ICML)*, 2008.

[20] G. Lebanon and J. Lafferty. Cranking: Combining rankings using conditional probability models on permutations. In *Proc. of the International Conference on Machine Learning (ICML)*, 2002.

[21] G. Lebanon and J. Lafferty. Conditional models on the ranking poset. In *The Conference on Advances in Neural Information Processing Systems (NIPS)*, volume 15, pages 431–438, 2003.

[22] D. Lillis, F. Toolan, R. Collier, and J. Dunnion. Probfuse: A probabilistic approach to data fusion. In *Proc. of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 139–146, 2006.

[23] Y.-T. Liu, T.-Y. Liu, T. Qin, Z.-M. Ma, and H. Li. Supervised rank aggregation. In *Proc. of the International World Wide Web Conference (WWW)*, 2007.

[24] C. L. Mallows. Non-null ranking models. *Biometrika*, 44:114–130, 1957.

[25] J. I. Marden. *Analyzing and Modeling Rank Data*. CRC Press, 1995.

[26] J. A. Shaw and E. A. Fox. Combination of multiple searches. In *Text REtrieval Conference (TREC)*, pages 243–252, 1994.

Machine Learned Sentence Selection Strategies for Query-Biased Summarization

Donald Metzler
metzler@yahoo-inc.com
Yahoo! Research
2821 Mission College Blvd.
Santa Clara, CA 95054

Tapas Kanungo
kanungo@yahoo-inc.com
Yahoo! Labs
2821 Mission College Blvd.
Santa Clara, CA 95054

ABSTRACT

It has become standard for search engines to augment result lists with document summaries. Each document summary consists of a title, abstract, and a URL. In this work, we focus on the task of selecting relevant sentences for inclusion in the abstract. In particular, we investigate how machine learning-based approaches can effectively be applied to the problem. We analyze and evaluate several learning to rank approaches, such as ranking support vector machines (SVMs), support vector regression (SVR), and gradient boosted decision trees (GBDTs). Our work is the first to evaluate SVR and GBDTs for the sentence selection task. Using standard TREC test collections, we rigorously evaluate various aspects of the sentence selection problem. Our results show that the effectiveness of the machine learning approaches varies across collections with different characteristics. Furthermore, the results show that GBDTs provide a robust and powerful framework for the sentence selection task and significantly outperform SVR and ranking SVMs on several data sets.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Experimentation, Theory

Keywords

sentence selection, learning to rank, gradient boosted decision trees

1. INTRODUCTION

Search engines have become popular and are widely used for many different tasks, such as web search, desktop search, enterprise search, and various domain-specific verticals. It has become almost standard for search engines to augment result lists with document summaries. It is important to produce high quality summaries, since the summaries can bias the perceived relevance of a document. For example, if the summary for a highly relevant document is poorly constructed, the user may perceive the document as non-relevant and may never view the document. Since users implicitly infer relevance from these summaries, it is important to construct high quality summaries that align the user's perceived relevance of the document with the actual relevance of the document.

Document summaries can either be query independent [8, 15] or query dependent [26, 29]. A query independent summary conveys general information about the document, and typically includes a title, static abstract, and URL, if applicable. Here, titles and static abstracts can either be extracted from the document, manually constructed, or automatically generated. These types of summaries can be computed offline and cached for fast access. The main problem with query independent summaries is that the summary for a document never changes across queries. This is one of the problems that query dependent summarization algorithms attempt to address, by biasing the summary towards the query. These summaries typically consist of a title, dynamic abstract, and URL. Since these summaries are dynamically generated, they are typically constructed at query time.

In this paper, we focus on the task of automatically generating abstracts for query dependent summarization. Given a query and a document, a query dependent abstract is generated as follows. First, relevant (with respect to the query) sentences or passages within the document must be identified. This is referred to as the *sentence selection* problem. After the relevant sentences have been identified, the *composition* phase begins. When composing an abstract, it is important to take into account how many sentences to include, and how to compress the sentences to fit within a fixed bounding box [14]. Furthermore, notions of readability and novelty also play a role during composition. Since the composition process can quickly become overly complex due factors involving presentation, user interaction, and novelty, we focus on the sentence selection problem in the remainder of this paper and leave composition as future work.

We propose using machine learning techniques to solve the sentence selection problem. There are several benefits to using such techniques. First, they provide an easy means of incorporating a wide range of features. It is often difficult

SIGIR 2008 LR4IR Workshop, Singapore.

to incorporate arbitrary features into standard information retrieval models, such as language modeling and BM25. Second, depending on the machine learning technique used, the model can be learned over a rich function space. Manually constructing such a function would require a great deal of effort. Finally, machine learning techniques provide a mechanism for learning from explicit (e.g., human judgments) or implicit (e.g., click data) training data. Of course, this may also be one of the biggest disadvantages to using machine learning, as well, as it is often difficult or expensive to obtain training data. All of the techniques we explore are fully supervised, and therefore require some form of training data.

A recent study has shown the feasibility of using machine learning approaches for sentence selection [29]. Wang *et al.* showed that ranking support vector machines (SVMs) outperform SVM classifiers and BM25 on a very small test collection of only 10 queries. In this paper, we augment the observations presented by Wang *et al.* and undertake a more comprehensive view of the problem from multiple perspectives.

Our work has four key contributions. First, we propose using regression-based models, such as support vector regression (SVR) and gradient boosted decision trees (GBDTs) for the sentence selection problem. Regression models, and GBDTs, in particular, have recently been shown to be highly effective for learning ranking functions for web search [17, 32]. We hypothesize the same will be true for sentence selection. Second, we carry out a rigorous set of experiments over three TREC data sets that, combined, have 200 queries associated with them. The results of these experiments provide unique insights into the applicability of the various learning techniques to data sets with different characteristics. Third, we show that performance is quite sensitive to how sentences are actually selected by comparing and contrasting the effectiveness of retrieving a fixed number of sentences per query/document pair versus using a global score threshold. While this topic is often ignored, it is important when using these algorithms in practice. Finally, we plan to release our data set for possible inclusion in the LETOR benchmark suite. This would allow researchers to explore various aspects of learning to rank in the context of an interesting application that has many unique characteristics that differentiates it from *ad hoc* retrieval and web search.

The remainder of this paper is laid out as follows. First, in Section 2, we detail related work in both summarization and machine learning approaches to ranking. Then, in Section 3, we describe the three machine learning models used, the features used with the models, and the different strategies for choosing the number of sentences to select. In Section 4 we describe our experimental evaluation. Section 5 discusses miscellaneous sentence selection issues. Finally, in Section 6, we conclude and describe possible areas of future work.

2. RELATED WORK

Automatic text summarization has been explored in many research areas including artificial intelligence, natural language processing, and information retrieval [21, 19]. While research in AI and NLP has focused on analyzing well-written text and generating large summaries (5-10 sentences), web search and information retrieval has focused on generating very small summaries. In fact, in web search, the role of the summary is to give an idea to the user whether or not the destination page is relevant for the user's query. Since a search result page typically has 10 or more URLs, the summary associated with an individual URL can not exceed more than 2-3 lines.

Kupiec, Pedersen and Chen [15] first addressed the sentence selection problem by using a binary Naïve Bayes classifier that learned if a given sentence should be part of the summary or not. The features used within the model were query independent, and therefore the goal of the model was to generate static abstracts. The model was trained using a corpus where human judges selected sentences that they thought should be part of the summary.

Tombros and Sanderson [26] conducted user studies using query-biased summaries. Their experiments suggest that query-biased summaries improve the ability of users to accurately judge relevance. Clarke *et al.* [3] studied the correlation of various attributes of summaries with click behavior. Goldstein *et al.* proposed various features and scored sentences in newspaper articles according to a specific scoring function. The function itself was not learned, however. In addition, Turpin *et al.* recently described techniques for efficiently compressing summaries [27]. However, this work does not take quality/relevance of the summary into account, which is the primary focus of our work.

More recently, initiatives, such as the Document Understanding Conference (DUC) and the Text Retrieval Conference (TREC) have conducted quantitative evaluations of various summarization algorithms and sentence retrieval tasks. In particular, the TREC Novelty Track, which ran from 2002 to 2004 included a sentence retrieval sub-task that required participants to retrieve relevant sentences, rather than relevant documents. A majority of the groups participating used standard information retrieval models for the task, such as language modeling and BM25. It is important to note that sentence selection is very closely related to sentence retrieval. The primary difference is that in sentence retrieval, a ranked list of sentences is returned for a *set* of documents, whereas the sentence selection task only returns a ranked list of sentences for a single document. Since the two tasks are so similar, it is likely that techniques developed for sentence retrieval will also work well for sentence selection, and vice versa.

The problem of learning to rank for information retrieval has become a topic of great interest in recent years. Many different techniques have been proposed, including logistic regression [7], SVMs [2, 12, 20], neural networks [1], and perceptrons [6]. These techniques have been adapted to optimize information retrieval-specific metrics, such as precision at K [13], mean average precision [30], nDCG [16], among others. While benchmark data sets exist for *ad hoc* and web retrieval [18], none currently exist for sentence selection.

Regression-based models, and in particular, gradient boosted decision trees, have recently been explored for learning to rank and and have been shown to be highly effective for web search [32, 17]. In this work, we apply regression-based techniques to the sentence selection problem, which has very different characteristics than web search, both in terms of features and in terms of context.

The work done by Wang *et al.* [29] is the most closely related to ours. The authors propose using SVMs and ranking SVMs to model the relevance of sentences to queries. Their results show that ranking SVMs outperformed standard SVMs on a small test collection of 10 queries. In their

experiments, they do not have a methodology for selecting the number of sentences. Instead, they always retrieve three sentences per document. In our work, we use ranking SVMs as a baseline against which we compare regression-based models, such as SVR and GBDTs. In addition, we analyze two different strategies for choosing the number of sentences to retrieve and carry out experiments out on three different test collections with over 200 queries, which allows us to draw inferences about the influence of various data set characteristics on effectiveness.

3. SENTENCE SELECTION USING MACHINE LEARNING

In this section, we describe the three machine learning techniques that we use for sentence selection, the set of features that we consider, and strategies for automatically choosing the number of sentences to return.

3.1 Models

There are numerous approaches to estimating the value of a categorical or continuous *response* variable (the human judgments) from measurements of *explanatory* variables (the extracted features). This problem has been studied under the names of statistical inference [28], pattern recognition [11] and more recently statistical machine learning [10]. Logistic regression, support vector machines, neural networks, and decision trees are some of the popular techniques. In this section, we briefly describe the three machine learning algorithms that we use for sentence selection.

3.1.1 Ranking SVMs

Ranking SVMs are a generalization of the classical SVM formulation that learns over *pairwise preferences*, rather than binary labeled data [12]. The motivation behind ranking SVMs is that for ranking problems, it is inappropriate to learn a classification model, since it does not take the structure of the problem into account. Instead, pairwise preferences can implicitly encode the structure of ranking problems, and therefore learning an SVM over such pairwise preferences is typically more effective when used for ranking since its objective function tends to be more in line with standard information retrieval metrics, such as precision, mean average precision, and F1.

Formally, the ranking SVM is formulated as a quadratic programming problem that has the following form:

$$\begin{aligned} \min \quad & \tfrac{1}{2}||w||^2 + C\sum_{i,j}\xi_{i,j} \\ s.t. \quad & (w\cdot x_i - w\cdot x_j) \geq 1-\xi_{i,j} \quad \forall (i,j)\in\mathcal{P} \\ & \xi_{i,j}\geq 0 \quad \forall (i,j)\in\mathcal{P} \end{aligned} \tag{1}$$

where w is the weight vector being fit, $\mathcal{P}$ is the set of pairwise preferences used for training, and C is a tunable parameter that penalizes misclassified input pairs. Once a weight vector w is learned, we can score unseen sentences by computing $w\cdot x_s$, where x_S is the feature vector for the sentence. These scores can then be used to rank sentences.

Ranking SVMs have been shown to significantly outperform standard SVMs for the sentence selection task and are currently the state of the art [29].

3.1.2 Support Vector Regression

Another generalization of the classical SVM formulation is support vector regression, which attempts to learn a regression model, rather than a classification or pairwise preference classification model. In our work, we fit a regression model directly to the human judgments, which typically corresponds to a target of +1 for relevant documents and -1 for non-relevant documents.

Support vector regression is formulated as follows:

$$\begin{aligned} \min \quad & \tfrac{1}{2}||w||^2 + C_+\sum_{i:y_i=1}(\xi_i+\xi_i^*) + C_-\sum_{i:y_i=-1}(\xi_i+\xi_i^*) \\ s.t. \quad & \\ & y_i - w\cdot x_i - b \leq \epsilon + \xi_i \\ & w\cdot x_i + b - y_i \leq \epsilon + \xi_i^* \\ & \xi_i, \xi_i^* \geq 0 \end{aligned} \tag{2}$$

where w is the weight vector being fit, C_- controls the cost associated with errors on non-relevant documents, C_+ controls the cost associated with errors on relevant documents, and ϵ is a free parameter controlling the amount of error tolerated for each input. In our experiments, we use $\epsilon = 0.1$.

Notice that the formulation we use allows for different costs for the relevant (+1 target) and non-relevant (-1 target) inputs. This is very important, since there are typically many more non-relevant sentences than there are relevant sentences. Therefore, it typically makes sense to ensure that the ratio of C_+ to C_- is greater than 1 in order to learn an effective model in the presence of such an imbalance.

3.1.3 Gradient Boosted Decision Trees

Gradient boosted decision trees are another technique that can be used for estimating a regression model [4]. Here, we use the stochastic variant of GBDTs [5]. GBDTs are a promising new machine learning approach that computes a function approximation by performing a numerical optimization in the function space instead of the parameter space. We provide a brief overview of the the GBDT algorithm and the parameters that influence the algorithm.

A basic regression tree $f(x)$, $x\in R^N$, partitions the space of explanatory variable values into disjoint regions R_j, $j = 1,2,\ldots,J$ associated with the terminal nodes of the tree. Each region is assigned a value ϕ_j such that $f(x) = \phi_j$ if $x\in R_j$. Thus the complete tree is represented as:

$$T(x;\Theta) = \sum_{j=1}^{J}\phi_j I(x\in R_j), \tag{3}$$

where $\Theta = \{R_j,\phi_j\}_1^J$, and I is the indicator function. For a given loss function $L(y_i,\phi_j)$ the parameters are estimated by minimizing the the total loss:

$$\hat{\Theta} = \arg\min_{\Theta}\sum_{j=1}^{J}\sum_{x_i\in R_j} L(y_i,\phi_j). \tag{4}$$

Numerous heuristics are used to solve the above minimization problem.

A boosted tree is an aggregate of such trees, each of which is computed in a sequence of stages. That is,

$$f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m), \tag{5}$$

where at each stage m, Θ_m is estimated to fit the *residuals* from the $m-1$th stage:

$$\hat{\Theta}_m = \arg\min_{\Theta_m}\sum_{i=1}^{N} L(y_i, f_{m-1}(x_i)+\phi_{j_m}). \tag{6}$$

In practice, instead of adding $f_m(x)$ at the mth stage, one adds $\rho f_m(x)$ where ρ is the *learning rate*. This is similar to a "line search" where one moves in the direction of the gradient, but the step size need not be equal to the gradient. In the *stochastic* version of GBDT, instead of using the entire data set to compute to loss function, one sub-samples the data and then finds the function values ϕ_j such that the loss on the test set is minimized. The stochastic variant minimizes overfitting issues.

The depth of the trees in each stage is another algorithm parameter of importance. Interestingly, making the trees in each stage very shallow while increasing the number of boosted trees tends to yield good function approximations. In fact, even with depth 1 trees, often called stubs, it possible to achieve good results. Interaction amongst explanatory variables is modeled by trees of depth greater than 1.

Finally, the GBDT algorithm also provides what is called *feature importance* [4]. The importance is computed by keeping track of the reduction in the loss function at each feature variable split and then computing the total reduction of loss function along each explanatory feature variable. The importance is useful for analyzing which features contribute most to the model.

3.2 Features

Features play an important role in any machine learning algorithm. Since it is not the goal of this paper to undertake a comprehensive exploration of features for sentence selection, we use a relatively simple, yet representative set of features in our models. The features that we consider can be divided into those that are query dependent and those that are query independent. We now briefly describe how each is computed.

3.2.1 Query Dependent Features

Query dependent features attempt to capture how relevant a given sentence S is to the query Q. We use four different query dependent features that model relevance at different levels of granularity and expressiveness.

The first feature is *exact match*. It is a binary feature that returns 1 if there is an exact lexical match of the query string within the sentence. It is computed as:

$$f_{EXACT}(Q,S) = I(Q \text{ substring of } S) \tag{7}$$

where I is the indicator function that returns 1 if its argument is satisfied.

The next feature is *overlap*, which is simply the fraction of query terms that occur, after stopping and stemming, in the sentence. Mathematically, it is computed as:

$$f_{OVERLAP}(Q,S) = \frac{\sum_{w \in Q} I(w \in S)}{|Q|} \tag{8}$$

where $|Q|$ is the number of non-stopword terms that occur in Q.

The next feature, *overlap-syn*, generalizes the overlap feature by also considering synonyms of query terms. It is computed as the fraction of query terms that either match Q or have a synonym that matches Q. It is computed as:

$$f_{OVERLAP-SYN}(Q,S) = \frac{\sum_{w \in Q} I(SYN(w) \in S)}{|Q|} \tag{9}$$

where $SYN(w)$ denotes the set of synonyms of w. Note that $SYN(w)$ also includes w itself.

The last query dependent feature, *LM*, is based on the language modeling approach to information retrieval [25]. It is computed as the log likelihood of the query being generated from the sentence. The sentence language model is smoothed using Dirichlet smoothing [31]. The feature is computed as:

$$f_{LM}(Q,S) = \sum_{w \in Q} tf_{w,Q} \log \frac{tf_{w,S} + \mu P(w|C)}{|S| + \mu} \tag{10}$$

where $tf_{w,Q}$ is the number of times that w occurs in the query, $tf_{w,S}$ is the number of times w occurs in the sentence, $|S|$ is the number of terms in the sentence, $P(w|C)$ is the background language model, and μ is a tunable smoothing parameter.

Although approaches such as language modeling and BM25 are well known to be highly effective text retrieval models, we include all of the simpler query dependent features because they may provide additional useful information to the classifier when learning a sentence selection model. In fact, the features do end up playing an important role, as we will show in Section 5.

3.2.2 Query Independent Features

The goal of query independent features is to encode any *a prior* knowledge we have about individual sentences. Here, we use two very simple query independent features.

We expect that very short sentences and, possibly, very long sentences are less likely to be relevant, therefore our first query independent feature is *length*, which is the total number of terms in the sentence after stopping. It is computed as:

$$f_{LENGTH}(S) = |S| \tag{11}$$

The other query independent feature we consider is *location*, which is the relative location of the sentence within the document. The feature is computed according to:

$$f_{LOCATION}(S,D) = \frac{sentnum_D(S)}{\max_{S'} sentnum_D(S')} \tag{12}$$

where $sentnum_D(S)$ is the sentence number for S in D and $\max_{S'} sentnum_D(S')$ is the total number of sentences in D.

Although not explored here, other query independent features are possible, such as readability, formatting, among others.

3.3 Result Set Filtering

Each of the machine learning methods described produce a real-valued score for every query/sentence pair. For a given document, these scores can be used to produce a ranked list of the sentences within the document. However, when constructing a summary, we only want to consider the most relevant sentences in the document. This requires using a decision mechanism that filters the ranked list of sentences, eliminating the least relevant sentences, and keeping the most relevant ones. We now briefly describe two solutions to this problem that have been used in the past. In our evaluation, we compare the effectiveness of the two approaches.

3.3.1 Fixed Depth

Perhaps the most simple and straightforward way of filtering the result set is to only return the top k ranked sentences for every document. Filtering in this way is useful if the un-

	N2002	N2003	N2004
Query / Doc. Pairs	597	1187	1214
Avg. Sentences per Pair	52.1	31.9	30.5
Avg. Relevant Sentences per Pair	2.3	13.1	6.9

Table 1: Overview of the TREC Novelty Track data sets used in the experimental evaluation.

derlying summary construction algorithm requires a fixed number of sentences as input.

However, fixed depth filtering has several disadvantages. For example, if $k = 5$, but the document only contains a single relevant sentence, then we would end up returning four non-relevant sentences. At the opposite end of the spectrum, if the document contained ten relevant sentences, then we would miss out on returning five of them. Therefore, the fixed depth filtering scheme is very rigid and fails to adapt to documents with very few or very many relevant sentences.

3.3.2 Global Score Threshold

One way to overcome the rigid nature of fixed depth filtering is to filter based on the *scores* of the sentences. If we assume that the scores returned by the machine learning algorithm are reasonable, then it is fair to believe that sentences with higher scores will be more likely to be relevant than those with lower scores. Therefore, in order to filter, we can set a global score threshold, where sentences with scores above the threshold are returned, and sentences with scores below the threshold are not returned.

Of course, there are also issues concerned with global thresholding, such as the fact that scores may not be comparable across queries. However, our evaluation shows that this may actually not be an issue for the machine learning techniques used here. In fact, we will show that the optimal global score threshold, particularly for SVR and GBDTs, is not only comparable across queries, but also across data sets, meaning that it is very easy to choose such a threshold.

4. EXPERIMENTAL RESULTS

In this section, we evaluate the effectiveness of ranking SVMs, SVR, and GBDTs. We also analyze the effectiveness of two result set filtering techniques just described. All of our experiments are carried out on the 2002, 2003, and 2004 TREC Novelty Track data sets. These data sets include human relevance judgments for query / sentence pairs, and therefore can be used to evaluate sentence selection algorithms. Note that we do not use any of the novelty-related judgments associated with these data sets, only the relevance judgments. For more information on the details of these data sets, please refer to the TREC Novelty Track overview papers [9, 24, 23]. For our purposes, we throw out all query / document pairs that have no relevant sentences associated with them, since these are uninteresting from a learning and evaluation perspective. Summary statistics for the data sets are provided in Table 1. Notice that the characteristics of the data sets are quite varied, both in terms of average number of sentences per pair, as well as the proportion of relevant sentences per document. This allows us to analyze how the various learning algorithms perform over a range of data sets.

Algorithm 1 Evaluation Algorithm

```
for i = 1 to 5 do
  (TRAIN, VALIDATE) ← split(TRAIN_i, p)
  utility_max ← −∞
  for θ ∈ Θ do
    model ← train(TRAIN; θ)
    utility ← eval(model, VALIDATE)
    if utility > utility_max then
      utility_max ← utility
      model_max ← model
    end if
  end for
  output rank(TEST_i, model_max)
end for
```

We use 5-folds cross validation for evaluation. All of the learning techniques have a number of hyperparameters that control various aspects of the learning algorithm. In order to properly tune the models, we must consider all reasonable settings of these hyperparameters. Therefore, we use a slightly modified version of 5-folds cross validation. The details of our evaluation algorithm are provided in Algorithm 1. In the algorithm, Θ is the set of hyperparameters that we will train over and *eval* is some evaluation measure that we are trying to maximize, such as precision, recall, or F1. As we see, during each training fold, the algorithm attempts to find the setting of the hyperparameters that maximizes the metric of interest by doing a brute force sweep over all reasonable settings. In order to control for overfitting, the effectiveness is measured on a held-out validation set.

For ranking SVMs, our algorithm sweeps over values for C (misclassification cost) and γ (RBF kernel variance). For SVR, we try various values for C_- and C_+ (misclassification costs), as well as γ. Finally, for GBDTs, we sweep over various weight values for the positive instances and tree depths (1, 2, 3). Additionally, for ranking SVMs and SVR, we only report results using the radial basis kernel, which provided the best results. Results for other kernels are omitted due to space constraints.

We use the SVMlight[1] implementation of ranking SVMs and SVR. We construct $\mathcal{P}$, the set of pairwise preferences used for training the ranking SVM, as the cross product of the relevant sentences and the non-relevant sentences for each query / document pair. For GBDTs, we use the GBM package for R [22].

Although none of the algorithms considered here directly maximize the metrics of interest to us, such as precision, recall, or F1, by training in this way we are implicitly optimizing for these measures by choosing the setting of the hyperparameters that maximizes the final measure we are interested in.

4.1 Sentence Selection

We now evaluate the effectiveness of the various machine learning algorithms for the sentence selection task within our experimental framework. There are many different ways to measure retrieval effectiveness, but most of the standard measures commonly used are inappropriate for the sentence selection task. From our perspective, R-Precision, computed

[1] `http://svmlight.joachims.org/`

	N2002	N2003	N2004
LM	.2602	.5566	.3944
Ranking SVM	$.3792^{\alpha}$	$.6904^{\alpha}$	$.4771^{\alpha}$
SVR	$.3587^{\alpha}$	$.7005^{\alpha\beta}$	$.4757^{\alpha}$
GBDT	$.4047^{\alpha\beta\delta}$	$.7060^{\alpha\beta}$	$.4806^{\alpha}$

Table 2: R-Precision for each data set and sentence selection approach. The α, β, and δ subscripts indicate a statistically significant improvement over language modeling, ranking SVMs, and SVR, respectively, according to a one-tailed pair t-test with $p < 0.05$.

over query/document pairs is one of the most meaningful measures. For a given query / document pair, R-Precision is computed as the precision at rank R, where R is the total number of relevant sentences in the document. This measure is appropriate because we ideally would like to return only the relevant sentences. Therefore, if a document has 10 relevant sentences, but we only retrieve 3 relevant sentences in the top 10, we have done a bad job for that document. Measures, such as precision at rank 10 do not take the number of relevant items per document into account, and therefore are not appropriate here.

Table 2 lists the R-Precision values of each learning method on each data set. For the sake of comparison, we also compare against language modeling (LM), which is a state of the art "bag of words" information retrieval model. The superscripts in the table indicate statistically significant improvements in R-Precision, as described in the caption.

The results indicate that all of the machine learned methods are better than language modeling, which is not surprising, since the language modeling score is a feature used by the learning algorithms that only considers term occurrences. This suggests that the other features we consider add considerable value.

Furthermore, we see that SVR is significantly better than ranking SVMs on the 2003 data set (1.5% improvement), and that GBDTs are significantly better than ranking SVMs on the 2002 (6.7% improvement) and 2003 (2.3% improvement) data sets. Therefore, the regression-based techniques are more effective than ranking SVMs, which is the current state of the art for sentence selection [29]. Lastly, we note that GBDTs are significantly better than SVR on the 2002 (12.8% improvement) data set. Thus, for the sentence selection problem, GBDTs are robust and highly effective across the different collections.

Interestingly, as the data set size grows, the effectiveness of ranking SVMs, SVR, and GBDTs seems to converge. This suggests that GBDTs, and SVR to a lesser extent, generalize better when the training data is sparse. It would be interesting to see if this behavior would persist if a larger feature set was used, as it would take more training examples to learn a good fit. This is an interesting area for future investigation.

4.2 Fixed Depth vs. Threshold Filtering

In our previous experiments, we always retrieved R sentences per query/document pair. While this was useful for comparing the effectiveness of the various techniques, it is not something that can be done in practice, since we do not know, *a priori*, how many sentences are relevant. Therefore, we must use one of the filtering techniques described earlier. When using these techniques, it is possible to retrieve a variable number of results per query/document pair. Therefore, R-Precision is no longer an appropriate measure. Instead, we use the F1 measure, which is the harmonic mean of precision and recall. This measure emphasizes the importance of both precision and recall and is comparable across query/document pairs that return different numbers of sentences.

In order to compare the effectiveness of fixed depth filtering and threshold filtering, we conduct an "upper bound" experiment. For each data set, we find the depth that results in the best F1, as well as the threshold setting that results in the best F1. We then can compare these two numbers to see which filtering technique, in the best case, would result in the best effectiveness. The results of this experiment are given in Table 3.

The results show that using fixed depth filtering is more effective on the 2002 data and threshold filtering yields better results on the 2003 and 2004 data sets. These results indicate that when there are very few relevant sentences per document, as is the case for the 2002 data set, a very shallow fixed depth filtering is better than using a global score threshold. Conversely, when there are many relevant sentences per document, as with the 2003 and 2004 data sets, fixed depth filtering is much worse than global thresholding.

In addition, the results show that GBDT have the most potential in terms of real world applicability, since the technique outperforms the others for both fixed depth and threshold filtering in a majority of cases. However, as we indicated, these results are upper bounds on the actual effectiveness that can be achieved using these filtering techniques.

In practice, one would have to either automatically learn the correct depth or threshold to use or use some robust "default" setting. Typically, it is difficult to define one setting that will work well across multiple data sets. However, as Figure 1 shows, the optimal threshold setting for GBDTs is very stable across the collections, more so than for ranking SVMs, and SVR. In fact, choosing -0.55 as a "default" threshold for GBDT yields an F1 that is within 2% of the optimal F1 for all three data sets.

Therefore, based on the sentence selection and filtering results, GBDTs appear to be the best choice of models to use out of the three that we explored. When using GBDTs, we recommend using fixed depth filtering for tasks with few relevant sentences per document, and that threshold filtering be used for tasks with many relevant sentences per document. Furthermore, if a threshold setting can not be reasonably estimated for the given task, then empirical evidence suggests that using -0.55 is a reasonable "default".

5. DISCUSSION

We now briefly discuss miscellaneous issues concerned with the approaches we explored in this paper.

5.1 Loss Functions

One theoretically interesting aspect of our work is the fact that regression-based models do not directly maximize the retrieval metric under consideration. Instead, they try to find a model that best fits the target labels. Ranking SVMs do not directly maximize general metrics, either, but they at least take the structure of the problem into account, more so, it seems, than simple regression models. However, as our results and the results of others indicate [17, 32], using

	N2002				N2003				N2004			
	Depth	F1	Thresh.	F1	Depth	F1	Thresh.	F1	Depth	F1	Thresh.	F1
Ranking SVM	2	.3411	-0.9	.2474	22	.5794	1.2	.6330	11	.4416	1.0	.4736
SVR	2	.3350	-0.9	.2880	22	.5791	-0.9	.6503	8	.4407	-0.2	.4637
GBDT	2	.3576	-0.55	.3302	20	.5771	-0.2	.6691	11	.4389	-0.5	.4745

Table 3: Comparison of result set filtering methods. For each data set, the optimal F1 measure for each technique is reported. The optimal depth and threshold settings are also reported.

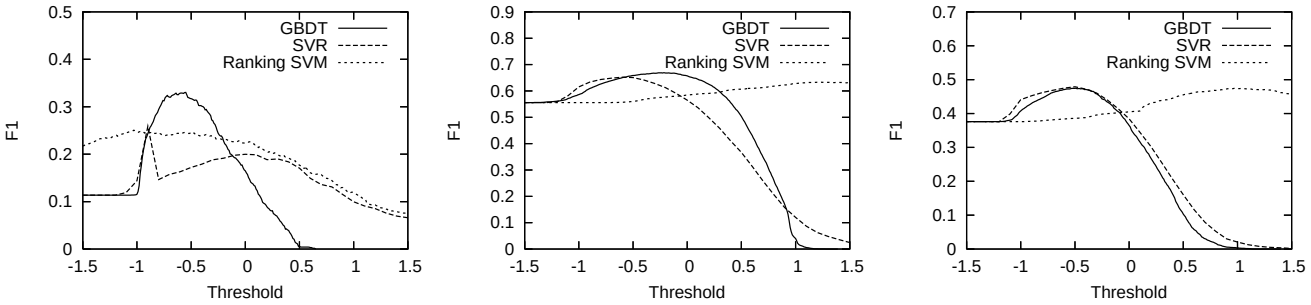


Figure 1: Effectiveness, measured in terms of F1, as a function of threshold value for the TREC 2002, 2003, and 2004 Novelty data sets (left to right).

these GBDT models prove to be highly effective. However, it is not yet clear as to exactly why this is the case. It would be interesting, as part of future work, to compare the effectiveness of regression-based techniques with those that directly optimize the metric of interest for this task.

5.2 Feature Importance

As discussed in Section 3, GBDTs provide a mechanism for reporting the relative importance of each feature. By analyzing the relative importances, we can gain insights into the importance of each feature for a given data set.

As an example, Figure 2 plots the relative feature importances of the features for the 2002 (top) and 2003 (bottom) data sets. It is interesting to note that the ordering of the importances is different for the two data sets. The two features with the highest importance for the 2002 data set are $overlap - syn$ (query/sentence overlap with synonyms) and $overlap$ (query/sentence overlap), whereas the two features with the highest importance for the 2003 data set are $length$ (sentence length) and lm (language modeling score). The ordering is also different for the 2004 data set, which indicates it may be difficult to manually construct a heuristic rule-based method that works well for all data sets. Although such rule-based methods may work well for a single task, such as web search, we are primarily interested in developing approaches that work well across a wide range of application domains.

5.3 Efficiency

Although our primary focus in this work is on effectiveness, we briefly describe our general observations on the efficiency of the three machine learning approaches explored here. First, the ranking SVM model was the least efficient of the techniques. This is due to the fact that the model is trained over pairwise preferences, which are inherently quadratic in nature. The SVR did not suffer from this problem, however. Second, SVR and ranking SVM models took even longer to train when the RBF kernel was used. Train-

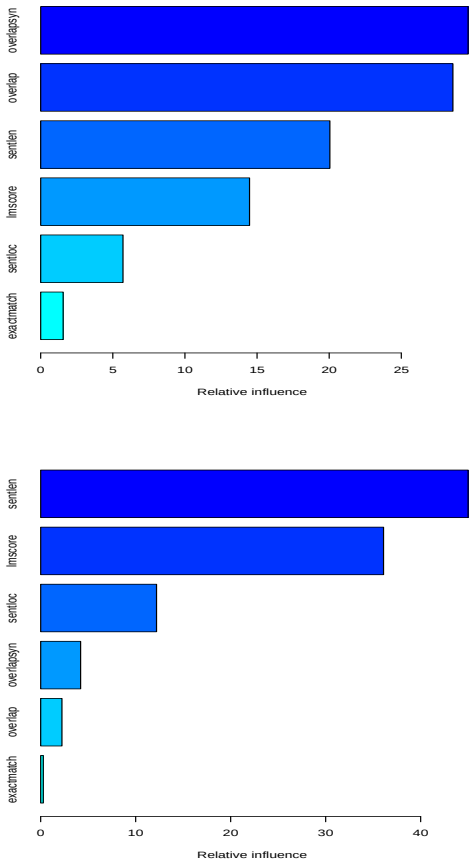


Figure 2: Relative feature importances, as computed by gradient boosted decision trees, for the Novelty 2002 (top) and 2003 (bottom) data sets.

ing time was significantly reduced when the "linear" kernel was used instead, but effectiveness was reduced. Finally, the GBDTs took significantly less time to train than the SVR and ranking SVMs (with and without kernels). The GBDTs were boosted for up to 1500 iterations. However, retrospective analysis shows that the optimal number of trees (iterations) for a model was always less than 200, which means that the training time could have been sped up even more. Therefore, in addition to the advantages GBDTs provide with respect to effectiveness, they also provide a number of benefits in terms of efficiency, as well.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed using regression-based machine learning techniques, such as support vector regression (SVR) and gradient boosted decision trees (GBDTs), for the sentence selection task, which is an important sub-task of constructing query-biased abstracts and summaries.

Our experimental results showed that SVR and GBDTs significantly outperform a simple language modeling baseline and ranking SVMs, which are considered to be the current state of the art. Our results also show that GBDTs are very robust and achieve strong effectiveness across three data sets of varying characteristics.

We also investigated two result set filtering techniques, including fixed depth and global score threshold filtering. Our results showed that fixed depth filtering is effective when there are few relevant sentences per document and that threshold filtering is more effective when there are many relevant sentences per document. Furthermore, our results indicated that threshold-based filtering for GBDTs is much more stable across data sets than ranking SVMs or SVR.

As part of future work, we plan to compare SVR and GBDTs to methods that directly maximize R-Precision or F1 to better understand the impact of the underlying loss function. We would also like to investigate set-based ranking algorithms in order to incorporate notions of novelty and sub-topic coverage. In addition, we would like to make our feature sets available as part of the growing LETOR benchmark [18] so that other researchers can develop and evaluate learning to rank techniques for the sentence selection task, which has very different characteristics than the typical *ad hoc* and web retrieval tasks.

7. REFERENCES

[1] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proc. 22nd Proc. Intl. Conference on Machine Learning*, pages 89–96, 2005.

[2] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon. Adapting ranking svm to document retrieval. In *Proc. 29th Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 186–193, 2006.

[3] C. Clarke, E. Agchtein, S. Dumais, and R. White. The influence of caption features on clickthrough patterns in web search. In *Proc. of SIGIR*, 2007.

[4] J. H. Friedman. Greedy function approximation: A graidient boosting machine. *Annals of Statistics*, 29:1189–1232, 2001.

[5] J. H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38:367–378, 2001.

[6] J. Gao, H. Qi, X. Xia, and J. Nie. Linear discriminant model for information retrieval. In *Proc. 28th Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 290–297, 2005.

[7] F. Gey. Inferring probability of relevance using the method of logistic regression. In *Proc. 17th Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 1994.

[8] J. Goldstein, M. Kantrowitz, V. Mittal, and J. Carbonell. Summarizing text documents: sentence selection and evaluation metrics. In *Proc. 22nd Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 121–128, 1999.

[9] D. Harman. Overview of the trec 2002 novelty track. In *Proc. 11th Text REtrieval Conference*, 2002.

[10] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Sringer-Verlag, New York, NY, 2001.

[11] A. K. Jain, R. P. W. Duin, and J. Mao. Statistical pattern recogntion: A review. *IEEE Transactions on Pattern Analysis and Machine Learning*, 22:4–37, 2000.

[12] T. Joachims. Optimizing search engines using clickthrough data. In *Proc. 8th Ann. Intl. ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 133–142, 2002.

[13] T. Joachims. A support vector method for multivariate performance measures. In *Proc. 22nd Proc. Intl. Conference on Machine Learning*, pages 377–384, 2005.

[14] K. Knight and D. Marcu. Statistics-based summarization — step one: Sentence compression. In *Proc. of AAAI*, 2000.

[15] J. Kupiec, J. Pedersen, and F. Chen. A trainable document summarizer. In *Proc. 18th Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 68–73, 1995.

[16] Q. Le and A. Smola. Direct optimization of ranking measures. http://www.citebase.org/abstract?id=oai:arXiv.org:0704.3359, 2007.

[17] P. Li, C. J. Burges, and Q. Wu. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Proc. 21st Proc. of Advances in Neural Information Processing Systems*, 2007.

[18] T.-Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. LETOR: Benchmark dataset for research on learning to rank for information retrieval. In *SIGIR 2007 workshop: Learning to Rank for Information Retrieval*, 2007.

[19] I. Mani and M. T. Maybury. *Advances in Automatic Text Summarization*. MIT Press, Cambridge, MA, 1999.

[20] R. Nallapati. Discriminative models for information retrieval. In *Proc. 27th Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 64–71, 2004.

[21] D. R. Radev and E. Hovy. *Intelligent Text Summarization*. AAAI, 1998.

[22] G. Ridgeway. The state of boosting. *Computing Science and Statistics*, 31:172–181, 1999.

[23] I. Soboroff. Overview of the trec 2004 novelty track. In *Proc. 13th Text REtrieval Conference*, 2004.

[24] I. Soboroff and D. Harman. Overview of the trec 2003 novelty track. In *Proc. 12th Text REtrieval Conference*, 2003.

[25] F. Song and W. B. Croft. A general language model for information retrieval. In *Proc. 8th Intl. Conf. on Information and Knowledge Management*, pages 316–321, 1999.

[26] A. Tombros and M. Sanderson. Advantages of query biased summaries in information retrieval. In *Proc. 21st Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 2–10, 1998.

[27] A. Turpin, Y. Tsegay, D. Hawking, and H. E. Williams. Fast generation of result snippets in web search. In *Proc. 30th Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 127–134, 2007.

[28] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Sringer-Verlag, New York, NY, 2002.

[29] C. Wang, F. Jing, L. Zhang, and H.-J. Zhang. Learning query-biased web page summarization. In *Proc. 16th Intl. Conf. on Information and Knowledge Management*, pages 555–562, 2007.

[30] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *Proc. 30th Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, page To appear, 2007.

[31] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proc. 24th Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 334–342, 2001.

[32] 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 *Proc. 21st Proc. of Advances in Neural Information Processing Systems*, 2007.

Selection bias in the LETOR datasets

Tom Minka
Microsoft Research
7 JJ Thomson Avenue
Cambridge, U.K.
minka@microsoft.com

Stephen Robertson
Microsoft Research
7 JJ Thomson Avenue
Cambridge, U.K.
ser@microsoft.com

ABSTRACT

The LETOR datasets consist of data extracted from traditional IR test corpora. For each of a number of test topics, a set of documents has been extracted, in the form of features of each document-query pair, for use by a ranker. An examination of the ways in which documents were selected for each topic shows that the selection has (for each of the three corpora) a particular bias or skewness. This has some unexpected effects which may considerably influence any learning-to-rank exercise conducted on these datasets. The problems may be resolvable by modifying the datasets.

1. INTRODUCTION

For the Learning to Rank workshop at SIGIR 2007, a dataset (actually a group of three datasets) was released for experimental purposes [6]. The intention was to provide some set of standard benchmarks, and to encourage participants to conduct comparable experiments (comparable to the benchmarks and to each other). The benchmark results provided in the cited paper are for Ranking SVM and RankBoost.

The three LETOR datasets were extracted from two TREC datasets (TDT2003, TDT2004) and from the OHSUMED corpus. Each dataset consists of a set of topics, together with extracted features for each of a set of query-document pairs, and associated relevance judgements. A number of different features are provided, both low-level and relatively high-level (features which might themselves serve as simple ranking algorithms). For the purpose of this paper, we note that one of the provided features is the BM25 score, which is a well-established ranking algorithm in its own right [7]. Although there is clearly no guarantee to this effect, we might expect the BM25 score on its own to give at least a reasonably effective ranking.

The set of documents associated with each topic is a selection, not the whole original corpus. There are obvious practical reasons for this procedure; however, the selection methods have given rise to a skew in the judgements which calls into question the validity of at least some of the results obtained on this dataset. This selection bias is present not only in the LETOR training data but also the test data. Therefore the algorithms which give the best test results are the ones which output rankings consistent with this bias. As a result, performance on the LETOR datasets is not an accurate guide for choosing a ranking algorithm for a real-world problem.

There are two issues here. On the one hand, we are concerned with ranking algorithms, and with evaluating such algorithms. On the other, we are concerned with learning algorithms, and their use in learning ranking algorithms. We discuss the effects of the skewed selection methods on both tasks. We note also that the LETOR datasets come with pre-defined training and test splits, and evaluation scripts. These scripts evaluate a ranker (possibly but not necessarily one trained or learnt using the training data) on the test data. They encode assumptions about (for example) how to deal with unjudged documents in evaluation.

1.1 Related work

The issue raised in the present paper can be seen as relating to the issue of evaluation with incomplete judgements, which has been the subject of much recent work (e.g. [8]). The traditional way to deal with incomplete judgements has been to regard unjudged documents as not relevant; this is probably a fair assumption if the original judgements were obtained from complete assessment of large pools, obtained from a wide variety of systems/runs. Some proposals involve leaving out the unjudged documents altogether. Recent work has addressed the issue of evaluation where this assumption is not good, and also the case where documents can be selected for judgement (so the task is to select for judgement those documents that are most likely to be informative, e.g. [3]). Some work has also addressed the possible bias in judgements (e.g. [2]). The approach in the paper just cited, as in [1], is to estimate the relevance of the unjudged documents, and include them in the evaluation.

Generally this work has not yet addressed the question of learning or training with incomplete or biased judgements (a recent exception is [4]). In constructing the LETOR datasets, a prior selection of documents has been made, with the effect that some assumptions about appropriate ways to deal with incomplete or biased judgements have been built into the datasets. The particular selection methods, and therefore the built-in assumptions, differ between the different LETOR datasets.

We note also that there has been some work in the machine learning literature on learning with 'imbalanced' or 'skewed' datasets. However, this is a different problem: typically learning a binary classifier in the case where one of the two classes occurs very much more frequently than the other (again typically, both in training-and-test datasets and in the real world). The problem discussed in the present paper has to do with the selection of data, for training-and-test, from real world data with different characteristics. (One rather obvious form of solution to the problem, that may be discovered in the machine learning literature, is discussed in section 4.1.)

2. THE DATASETS

2.1 TDT

In the LETOR TDT dataset, the documents selected for each topic include (a) the top 1000 documents ranked by BM25, with relevance judgements where these are available, plus (b) any other documents judged relevant. An immediate bias is evident: documents with high BM25 scores are selected anyway, irrespective of relevance; but documents with low BM25 scores are selected *only if they are relevant*.

The effect of this selection policy on the apparent effectiveness of BM25 as a ranking algorithm/feature is dramatic. It means that within these extractions, BM25 is negatively correlated with relevance. If using BM25 on its own as a ranking algorithm, it is best to select documents with low BM25 over documents with high BM25 scores. A ranking in reverse BM25 order is not only more effective than a ranking in positive BM25 order, it is also more effective at early ranks than either of the other benchmarks (Ranking SVM or RankBoost) in [6] – see table 1 for the TDT2003 dataset. All the tables and results below are as reported by the LETOR evaluation scripts, and in particular on the test split of the datasets.

	Reverse BM25	RankBoost	RankSVM	BM25
P@1	0.52	0.26	0.42	0.12
P@2	0.40	0.27	0.35	0.13
P@3	0.35	0.24	0.34	0.16
P@5	0.27	0.22	0.26	0.15
NDCG@5	0.326	0.279	0.347	0.183
MAP	0.185	0.212	0.256	0.126

Table 1: Results for ranking in reverse BM25 order, compared to baselines, on the LETOR TDT2003 dataset

Any learning algorithm which chooses to rank in reverse order of BM25 is therefore being rewarded in the LETOR evaluation. In general, we have no way of knowing whether a learning algorithm acquired this bias from the training data or whether it is inherent to the algorithm. If the bias is inherent to the algorithm, then applying the same learning algorithm to real data may give very different results than those observed on LETOR.

Another way of reading the results in the table is as follows: if all the relevant documents are contained in the top 1000, then it is very unlikely that the 1000th is relevant; thus a topic with this condition probably contributes zero to the P@1 figure for reverse BM25. From which it follows that up to 52% of the test topics have at least one relevant outside the top 1000. The P@2 row suggests that a much smaller proportion, probably around 28% (because 40% is the average of 52% and 28%), have at least two relevant outside the top 1000. In fact, these figures are exactly correct: 26 out of 50 topics have at least 1001 documents, while 14 out of 50 have at least 1002. For NDCG@5, Reverse BM25 does less well than RankingSVM (although still better than RankBoost); this probably has to do with the distribution of total numbers of relevant documents per topic, which correlates differently with effectiveness for the different methods. Reverse BM25 does worse than either baseline on MAP, which takes account of the entire curve, although its early-rank performance is enough to keep it above BM25 itself.

In the case of TDT2004, regular BM25 does much better and Reverse BM25 not nearly so well. The incidence of extra relevant documents outside the top 1000 (by BM25) is very much lower: the proportion of test topics with at least 1001 documents is $14/75 = 19\%$, and for 1002 is $6/75 = 8\%$ (average 13%). Again, these figures are reflected exactly in the early-rank results for Reverse BM25 – see table 2. The effect is still enough to give Reverse BM25 a non-negligible early-rank precision.

	Reverse BM25	BM25
P@1	0.19	0.31
P@2	0.13	0.29
P@3	0.10	0.26
P@5	0.06	0.23
NDCG@5	0.097	0.319
MAP	0.060	0.282

Table 2: Results for ranking in reverse BM25 order, compared to baselines, on the LETOR TDT2004 dataset

2.2 OHSUMED

The OHSUMED dataset presents different issues. The documents selected for each topic were just those for which relevance judgements were available. When the original OHSUMED dataset was constructed [5], expert searchers conducted the searches for each topic, using a traditional Boolean search system. Subsequently, relevance assessment was done by another set of expert physicians. The pools of documents provided for assessment were constructed from those items viewed by the expert searchers, together with those items retrieved by searchers' final refined Boolean search statements.

This selection meant that documents in the pool had a high chance of being relevant. This is evident from the proportion of non-relevants among the selected documents – nine topics have less than 40% in the non-relevant category. Furthermore, every document in the pool matched some version of the query very well. Thus these non-relevant documents are highly atypical. Examples of the wider range of non-relevant documents that clearly exist in the full collection are missing in the dataset. In particular, there are likely to be many other documents that could be scored relatively highly by a ranking algorithm (which one would particularly like the algorithm to learn to distinguish).

3. LEARNING

As indicated, the LETOR datasets contain a number of features for each topic-document pair. The objective is to allow a learning method to learn how these features should be combined in order to provide optimal ranking according to some measure of search effectiveness. Such combination might for example be a linear function with learnt weights, or some more complex combination. In this section we discuss the impact of the skewed judgements on learning.

Suppose that, in addition to the BM25 feature, we add the log of BM25 to the TDT dataset as a separate feature (this feature is already present in the OHSUMED data for example). This means that even a simple linear model can actually learn a class of non-linear functions of BM25, by

combining these two features linearly. We note that BM25 itself with a positive weight, combined with log BM25 with a negative weight, can yield a U-shaped function where very low as well as high BM25 scores are rewarded. We have indeed found this kind of effect when adding log BM25 as a new feature and then fitting a linear model on the TDT datasets. It seems clear that the effect is an artefact of the dataset.

This artefact can arise even without explicitly adding non-linear functions of BM25. This is because many standard IR features are correlated with BM25. In the TDT dataset, the features "sitemap based score propagation" and "sitemap based feature propagation" have a correlation coefficient with BM25 exceeding 0.98. Like BM25, these features perform best when given negative weights on TDT2003. However, a linear combination of BM25 with these features, using weights of opposite signs, provides increased performance due to the effective nonlinearity. Language modelling functions would presumably also have a high correlation with BM25, although in the TDT dataset these were not included at the whole-document level.

It may be argued that the LETOR dataset is intended to compare learning algorithms, and that it may serve this purpose even if the resulting learnt ranker is not a useful one. In this sense, it may serve a similar purpose to a purely artificial dataset, for which one has no guarantee that it reflects any real-world data. However, this seems a very limited aspiration for LETOR, and one that would indeed be better served by generating purely artificial data from known distributions. The fact that some attempt has been made to draw LETOR data from realistic datasets should be one of its advantages. We note also that it is difficult to draw useful conclusions about the value of learning algorithms in discovering good rankers for search, if what the learning algorithm learns is so dominated by selection biases in the dataset.

4. POSSIBLE SOLUTIONS

It is clear that in order to learn how to rank for real, or even to test ideas about learning properly, we need more realistic datasets. Results on the present LETOR datasets cannot be relied upon to yield believable research conclusions.

But the challenge of designing really good datasets for the learning to rank task is not simple. We may be able to suggest some modifications to the LETOR datasets which have some chance of making them more useful, but we also suggest that some serious investigation of the validity of results from any proposed dataset is required. The danger (as revealed above) is that a learning system will succeed only in learning artefactual characteristics of the dataset.

4.1 TDT

A simple way to remove the bias in the TDT datasets is to remove the relevant documents outside the top 1000 of BM25. This redefines the learning task as 'learning to rank within the results returned by another search engine', defined for these purposes as the top 1000 retrieved by a BM25 search engine. The effect of this on reverse BM25 is dramatic: its P@5 and NDCG@5 drop to zero and its MAP is nearly zero. Regular BM25 has its NDCG and MAP slightly increased. One problem with this approach is that it still gives special status to BM25, and indeed by extension to any ranking algorithm that is highly correlated with BM25.

Rather than remove relevant documents, we could also add more non-relevants from the original TDT collection. How many such documents should be included, and where should they be sampled from? For the number, we might assume that precision declines with rank in the BM25 ranking (again, this seems to be loading too much onto BM25, but again it's hard to see an alternative). This assumption would imply that (for example) the total number of non-relevant beyond rank 1000 should be fixed to ensure that the precision of this set alone is less than (say) the precision of ranks 900-1000 alone.

In order for the declining-precision argument to apply at any rank, it would be necessary to generate a much larger ranking in BM25 order, locate the relevant documents within it, and sample non-relevants from each interval between relevant documents. Such a procedure could probably be worked out, but would have to deal with some special cases:

- two relevant documents occurring close together in the ranking, or even tied;
- relevant documents with zero BM25.

The latter case does indeed occur in the LETOR datasets.

4.2 OHSUMED

The OHSUMED dataset is somewhat more tricky, since we have very little idea (certainly no formal definition) of how the included documents were selected for judgement in the first place. It might be better to replicate the TDT procedure, and introduce an algorithmic ranking such as BM25 as the basis for selection. Once again, we would have to sample in some systematic way in the gaps between the selected documents. In this case the selection includes non-relevant documents already; we would assume that the additional random documents are also non-relevant. This is perhaps a questionable assumption in the case of the OHSUMED data.

4.3 Redefining the test set

One of the issues that led to the construction of the LETOR datasets, and in particular the selection of documents for each topic, is that training on a full-size corpus is often not feasible. Many learning algorithms from the machine learning domain would be impossible to scale to operate on complete collections of documents (even TREC collections, let alone the web). However, this constraint does not apply to testing/evaluation. Most reasonable ranking algorithms could without difficulty be applied to a full-sized TREC collection.

This suggests that we should have different kinds of training and test collection: the test collection should be the entire original document set (TDT or OHSUMED as appropriate). This would ensure that a learning algorithm would not be rewarded for learning the biases of the selection process (as is currently the case). On the contrary, there would be benefit to be gained by designing a learning algorithm which could take proper account of these biases in training, and thereby produce a ranking algorithm which would work well with unselected data. Furthermore, the tests would have similar validity to many current experiments on TREC-like test collections outside the LETOR context, which they do not currently have.

5. CONCLUSIONS

We have shown that the LETOR datasets exhibit skewed judgements which cast doubt on any results derived from them. The TDT datasets can be fixed in a simple way by excluding relevant documents outside the top 1000. It may also be possible to improve all the datasets by including some additional sampled documents, assumed non-relevant, in the per-topic extractions. A more radical suggestion is to redefine the test part of LETOR to match much more closely the way in which ranking algorithms are normally tested on TREC-like corpora, using the entire corpus.

6. REFERENCES

[1] J. A. Aslam and E. Yilmaz. Inferring document relevance from incomplete information. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 633–642, New York, NY, USA, 2007. ACM.

[2] S. Buttcher, C. L. A. Clarke, P. C. K. Yeung, and I. Soboroff. Reliable information retrieval evaluation with incomplete and biased judgements. In *SIGIR 2007*, pages 63–70. ACM Press, 2007.

[3] B. Carterette, J. Allan, and R. Sitaraman. Minimal test collections for retrieval evaluation. In E. N. Efthimiadis, S. T. Dumais, D. Hawking, and K. Järvelin, editors, *SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 268–275, New York, 2006. ACM Press.

[4] B. He, C. Macdonald, and I. Ounis. Retrieval sensitivity under training using different measures. 2008. To appear in SIGIR 2008.

[5] W. R. Hersh, C. Buckley, T. J. Leone, and D. H. Hickam. OHSUMED: an interactive retrieval evaluation and new large test collection for research. In W. B. Croft and C. J. van Rijsbergen, editors, *SIGIR '94: Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 192–201. Springer-Verlag, 1994.

[6] T.-Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. Letor: Benchmark dataset for research on learning to rank for information retrieval. Technical report, 2007. LR4IR 2007, in conjunction with SIGIR 2007.

[7] K. Sparck Jones, S. Walker, and S. E. Robertson. A probabilistic model of information retrieval: development and comparative experiments. *Information Processing and Management*, 36:779–808 (Part 1) and 809–840 (Part 2), 2000. `http://www.soi.city.ac.uk/~ser/blockbuster.html`.

[8] E. Yilmaz and J. A. Aslam. Estimating average precision with incomplete and imperfect judgements. In P. S. Yu, V. J. Tsotras, E. A. Fox, and B. Liu, editors, *CIKM 2006: Proceedings of the 13th ACM Conference on Information and Knowledge Management*, pages 102–111, New York, 2006. ACM Press.

How to Make LETOR More Useful and Reliable

Tao Qin, Tie-Yan Liu, Jun Xu, Hang Li
Microsoft Research Asia
No.49 Zhichun Road, Haidian District
Beijing 100190, P.R. China
tsintao@gmail.com,
{tyliu, junxu, hangli}@microsoft.com

ABSTRACT

Learning to rank has attracted great attention recently in both information retrieval and machine learning communities. However, the lack of public dataset had stood in its way until the LETOR benchmark dataset (actually a group of three datasets) was released in the SIGIR 2007 workshop on Learning to Rank for Information Retrieval (LR4IR 2007). Since then, this dataset has been widely used in many learning to rank papers, and has greatly speeded up the corresponding research. In this paper, we discuss how to further improve LETOR to make it more useful and reliable. First, we notice that some low-level information, such as the term frequency in each stream (title, body, url, anchor, etc.) and the stream length, are missing in the current feature set of LETOR. We propose adding the information to LETOR, so as to enable the reproduction or optimization of models like BM25. Second, we find that the sampling of documents associated with each query in LETOR was somehow biased. We therefore propose a new document sampling strategy to reduce the bias. Third, the scale (less than 100 queries) of LETOR is relatively small for real world ranking applications. We propose adding more queries to the current datasets in LETOR, and/or building even larger datasets by leveraging the effort of the entire information retrieval community.

1. INTRODUCTION

Ranking is the central problem for many information retrieval (IR) applications, including document retrieval, collaborative filtering, key term extraction, definition finding, important email routing, sentiment analysis, product rating, and anti web spam. In the task, given a set of objects, a ranking model (function) is used to calculate a score for each object and the objects are sorted in the descending order of the scores.

Learning to rank has attracted great attention recently in both information retrieval and machine learning communities. Many algorithms have been proposed for learning to rank, such as the pointwise approach [12], the pairwise approach [8, 9, 5, 2] and the listwise approach [3, 17]. The lack of public dataset had stood in the way of research on learning to rank until the LETOR benchkmar dataset was released in the SIGIR 2007 Workshop on Learning to Rank for Information Retrieval (LR4IR 2007). Since then, this dataset has been widely used in many learning to rank papers [16, 21, 4, 10, 25, 23, 7, 6].

Although the release of LETOR has greatly speeded up the research on learning to rank, we also notice several issues in it. To make LETOR more useful and reliable, in this paper, we discuss how to improve it from three aspects:

(1) Some low-level information is missing in the current feature set of LETOR, such as the term frequency in each stream (title, body, url, anchor et. al.) and the stream length. This makes it difficult to reproduce and/or optimize models like BM25. We suggest adding these low-level features to the new version of LETOR to solve the problem.

(2) The sampling of documents associated with each query in LETOR was somehow biased because of the specific document selection strategy. We consider three new document selection strategies in this paper to avoid or reduce the bias, and discuss their feasibility and costs.

(3) The scale (less than 100 queries) of LETOR is not large enough for real-world ranking applications. We propose adding more queries to the current datasets in LETOR. We also suggest leveraging the efforts of the entire IR community to collect larger-scale data for learning to rank research.

The remaining part of this paper is organized as follows: a brief introduction to LETOR is given in Section 2; the details of the proposed ways of improving the existing datasets in LETOR are discussed in Section 3; In Section 4, we propose creating larger-scale training data; and conclusions are given in the last section.

2. ANALYSIS ON THE LETOR DATASET

2.1 Overview of LETOR

LETOR was built based on two widely-used data collections in information retrieval: the OHSUMED collection used in the information filtering task of TREC 2000, and the ".gov" collection used in the topic distillation tasks of TREC 2003 and 2004. Accordingly, there are three sub datasets in LETOR, namely OHSUMED, TD2003, and TD2004.

Appearing in the proceedings of SIGIR 2008 Workshop on Learning to Rank for Information Retrieval (LR4IR 2008).

Table 1: Statistics of three datasets in LETOR

Dataset	#Query	#Doc	#Doc/#Query	#Feature
OHSUMED	106	16140	152.26	25
TD2003	50	49171	983.42	44
TD2004	75	74170	988.93	44

Both 'low level' and 'high level' features have been extracted for each query-document pair in the OHSUMED collection. Low-level features include term frequency (tf), inverse document frequency (idf), document length (dl) and their combinations [1]. High-level features include the outputs of BM25 [18] and language model for IR [26].

Four kinds of features have been extracted for each query-document pair in the ".gov" collections: low-level content features, high-level content features, hyperlink features, and hybrid features. The low-level and high-level content features are similar to that in the OHSUMED dataset. Hyperlink features include PageRank [14], HITS [11], and their variations (HostRank [24], topical PageRank and topical HITS [13]). Hybrid features refer to those features containing both content and hyperlink information, including "hyperlink-based relevance propagation" [19] and "sitemap-based relevance propagation" [15].

Since there are too many documents in the OHSUMED and ".gov" collections, when building LETOR, documents are sampled for each query to facilitate the experiments on ranking, according to the following strategy. For the OHSUMED dataset, all judged documents were selected, and all un-judged documents were neglected. In this way, about 150 documents per query were eventually selected, as shown in Table 1. The sampling strategy for the ".gov" collection is a little different. First, BM25 was used as the model to rank all the documents with respect to each query, and then the top 1000 documents (if there are such number of documents containing the query) were selected. Considering that some relevant documents may not appear in the top 1000 results, all the relevant documents for each query were also added to the selected document pool. The statistics on the TD2003 and TD2004 datasets can also be found in Table 1.

The value of the LETOR dataset lies in the following two aspects:

(1) LETOR contains a set of standard features, and so researchers can directly use them to test the effectiveness of their ranking algorithms, without the necessity of creating a dataset by their own. Because the dataset creation is very costly (including collection hunting, query selection, feature extraction, etc.), LETOR has greatly reduced the barrier of the research on learning to rank.

(2) LETOR makes the fair comparison among different learning to rank algorithms possible. Before the release of LETOR, different datasets (i.e. different query sets, different document collections, different features, or different evaluation tools) were used in different papers. As a result, it is not easy to get conclusive observations on the effectiveness of the learning to rank algorithms, by comparing the performances reported in different papers. LETOR has made it possible to compare the algorithms, since a standard dataset and a set of standard evaluation tools are used. Furthermore, in the current version of LETOR (version 2.0), several state-of-the-art baselines have been included, which even further ease the algorithm comparison: researchers even do not need to implement some of the baselines to be compared by their own.

2.2 Aspects of LETOR to Be Improved

Although LETOR has achieved great success, there are also problems with it. While using the LETOR dataset, we find that there are several aspects that should be improved, in order to make it more useful and reliable.

First, some "low-level" information is missing in the feature extraction process of LETOR.

One of the goals of LETOR is to provide standard features so that researchers using LETOR do not need to deal with the raw documents. However, the current features provided in LETOR may limit people's research. For example, a BM25 score is provided which was computed with default parameters. If one wants to further tune this model and get a more effective feature, he/she will find that there is no sufficient information to perform the task.

To better understand the problem, let us look at one of the most prominent instantiations of BM25 as follows.

Given a query Q, containing keywords $q_1, ..., q_t$, the BM25 score of a document D is computed as:

$$BM25(D,Q) = \sum_{i=1}^{t} idf(q_i) \cdot \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})},$$

where $f(q_i, D)$ is the occurrences of q_i in the document D, $|D|$ is the length of the document D (i.e., the number of words), and $avgdl$ is the average document length in the entire document collection from which documents are drawn. k_1 and b are free parameters. $idf(q_i)$ is the IDF (inverse document frequency) weight of the query term q_i. It is usually computed as:

$$idf(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5},$$

where N is the total number of documents in the collection, and $n(q_i)$ is the number of documents containing q_i.

As can be seen, to compute the BM25 score, we need to know the term frequency $f(q_i, D)$ in document D and the document frequency $n(q_i)$ in the entire collection containing each query term q_i. Besides, we also need to know the document length $|D|$ and the average document length of the entire collection. In the current version of LETOR, however, only $|D|$ is provided as a feature. $f(q_i, D)$ and $n(q_i)$ are missing[1]. This prevents researchers from tuning the parameters of BM25 (e.g. k_1 and b) and/or creating other strong features based on such low level information.

Second, the document sampling strategy used in LETOR is somehow biased.

For the OHSUMED dataset, only judged query-document pairs were selected. The judgments are "highly relevant", "partially relevant" or "irrelevant". However, this setting is not consistent with real world applications. For a IR system, when a user issues a query, the system does not know which

[1] $\sum_{i=1}^{t} f(q_i, D)$ and $\sum_{i=1}^{t} n(q_i)$ are provides in LETOR2.0 instead. One cannot use the sum of term frequency and document frequency to reproduce the BM25 score.

document is judged or not. That is, we cannot get only judged documents for ranking.

For TD2003 and TD2004 datasets, for each query, both top 1000 documents based on their BM25 scores and the relevant documents which are not ranked in top 1000 positions were selected. It is clear that such a sampling strategy is not consistent with the ranking process in real search scenarios either. Given a query, we can get its top 1000 documents by BM25, but we do not know which documents ranked out of top 1000 are relevant.

Third, the scale of the dataset in LETOR is not large enough.

Some statistics of the three datasets are listed in Table 1. As can be seen, there are only 50 and 75 queries in TD2003 and TD2004 separately, and about 100 queries in OHSUMED. After the 5-fold partitioning, each validation (testing) set only contains 10/15/21 queries in TD2003/TD2004 /OHSUMED. The small number of queries may make the experimental results got on these datasets not reliable. Failing in one single query may lead to significant performance drop.

From the view of machine learning, a larger-scale dataset will be better for a learning algorithm. From the view of real world ranking applications (e.g. web search), a ranking algorithm should also handle large-scale dataset. Considering these, it is important and necessary to add large datasets to LETOR.

3. IMPROVING EXISTING DATASETS IN LETOR

In this section, we discuss how to improve the existing datasets in LETOR. We mainly focus on the first and second aspects mentioned in the previous section.

3.1 Adding More Raw Features

For the OHSUMED dataset, besides the 25 features in the current version of LETOR, we suggest adding more features for learning. Specifically, we suggest extracting 20 new features from the fields of 'title', 'abstract', and 'title + abstract': 10 'low level features' from 'title + abstract', 5 'high level' features from 'title', and 5 'high level' features from 'abstract'. The entire feature set can be found in Table 3.

we also suggest providing 'meta features' on the OHSUMED dataset. Term frequency and document frequency of each query term, and document lengths etc. are important meta features. Some statistics on the corpus, such as the average document length, the total number of documents in the corpus etc., are also important and should be included. These meta features may not be directly used by a learning algorithm. However, based on these meta features, people can construct their own features, reproduce and tune BM25 and other strong features. Table 3 lists all of the proposed meta features.

For TD2003 and TD2004, besides the 44 features in the current version of LETOR, we suggest adding in-link number, out-link number, length of URL, number of slashes in the URL, etc. as new features. Also, we suggest extracting those existing features in all streams (URL, title, anchor and body), while features in some streams are missing in the current version of LETOR. Overall, there will be 64 features (Table. 4) which can be directly used by learning algorithms.

We also propose adding the term frequency in each stream

Table 2: Features for OHSUMED

ID	Feature Description
1	$\sum_{q_i \in q \cap d} c(q_i, d)$ in 'title'
2	$\sum_{q_i \in q \cap d} \log\left(c(q_i, d) + 1\right)$ in 'title'
3	$\sum_{q_i \in q \cap d} \frac{c(q_i,d)}{\lvert d \rvert}$ in 'title'
4	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i,d)}{\lvert d \rvert} + 1\right)$ in 'title'
5	$\sum_{q_i \in q \cap d} \log\left(\frac{\lvert C \rvert}{df(q_i)}\right)$ in 'title'
6	$\sum_{q_i \in q \cap d} \log\left(\log\left(\frac{\lvert C \rvert}{df(q_i)}\right)\right)$ in 'title'
7	$\sum_{q_i \in q \cap d} \log\left(\frac{\lvert C \rvert}{c(q_i,C)} + 1\right)$ in 'title'
8	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i,d)}{\lvert d \rvert} \cdot \log\left(\frac{\lvert C \rvert}{df(q_i)}\right) + 1\right)$ in 'title'
9	$\sum_{q_i \in q \cap d} c(q_i, d) \cdot \log\left(\frac{\lvert C \rvert}{df(q_i)}\right)$ in 'title'
10	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i,d)}{\lvert d \rvert} \cdot \frac{\lvert C \rvert}{c(q_i,C)} + 1\right)$ in 'title'
11	BM25 score in 'title'
12	log(BM25 score) in 'title'
13	LMIR with DIR smoothing in 'title'
14	LMIR with JM smoothing in 'title'
15	LMIR with ABS smoothing in 'title'
16	$\sum_{q_i \in q \cap d} c(q_i, d)$ in 'abstract'
17	$\sum_{q_i \in q \cap d} \log\left(c(q_i, d) + 1\right)$ in 'abstract'
18	$\sum_{q_i \in q \cap d} \frac{c(q_i,d)}{\lvert d \rvert}$ in 'abstract'
19	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i,d)}{\lvert d \rvert} + 1\right)$ in 'abstract'
20	$\sum_{q_i \in q \cap d} \log\left(\frac{\lvert C \rvert}{df(q_i)}\right)$ in 'abstract'
21	$\sum_{q_i \in q \cap d} \log\left(\log\left(\frac{\lvert C \rvert}{df(q_i)}\right)\right)$ in 'abstract'
22	$\sum_{q_i \in q \cap d} \log\left(\frac{\lvert C \rvert}{c(q_i,C)} + 1\right)$ in 'abstract'
23	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i,d)}{\lvert d \rvert} \cdot \log\left(\frac{\lvert C \rvert}{df(q_i)}\right) + 1\right)$ in 'abstract'
24	$\sum_{q_i \in q \cap d} c(q_i, d) \cdot \log\left(\frac{\lvert C \rvert}{df(q_i)}\right)$ in 'abstract'
25	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i,d)}{\lvert d \rvert} \cdot \frac{\lvert C \rvert}{c(q_i,C)} + 1\right)$ in 'abstract'
26	BM25 score in 'abstract'
27	log(BM25 score) 'abstract'
28	LMIR with DIR smoothing in 'abstract'
29	LMIR with JM smoothing in 'abstract'
30	LMIR with ABS smoothing in 'abstract'
31	$\sum_{q_i \in q \cap d} c(q_i, d)$ in 'title + abstract'
32	$\sum_{q_i \in q \cap d} \log\left(c(q_i, d) + 1\right)$ in 'title + abstract'
33	$\sum_{q_i \in q \cap d} \frac{c(q_i,d)}{\lvert d \rvert}$ in 'title + abstract'
34	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i,d)}{\lvert d \rvert} + 1\right)$ in 'title + abstract'
35	$\sum_{q_i \in q \cap d} \log\left(\frac{\lvert C \rvert}{df(q_i)}\right)$ in 'title + abstract'
36	$\sum_{q_i \in q \cap d} \log\left(\log\left(\frac{\lvert C \rvert}{df(q_i)}\right)\right)$ in 'title + abstract'
37	$\sum_{q_i \in q \cap d} \log\left(\frac{\lvert C \rvert}{c(q_i,C)} + 1\right)$ in 'title + abstract'
38	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i,d)}{\lvert d \rvert} \cdot \log\left(\frac{\lvert C \rvert}{df(q_i)}\right) + 1\right)$ in 'title + abstract'
39	$\sum_{q_i \in q \cap d} c(q_i, d) \cdot \log\left(\frac{\lvert C \rvert}{df(q_i)}\right)$ in 'title + abstract'
40	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i,d)}{\lvert d \rvert} \cdot \frac{\lvert C \rvert}{c(q_i,C)} + 1\right)$ in 'title + abstract'
41	BM25 score in 'title + abstract'
42	log(BM25 score) 'title + abstract'
43	LMIR with DIR smoothing 'title + abstract'
44	LMIR with JM smoothing in 'title + abstract'
45	LMIR with ABS smoothing in 'title + abstract'

Table 3: Meta Features for OHSUMED

ID	Feature Description
1	Term frequency (TF) of a single query term in 'title'
2	Term frequency (TF) of a single query term in 'abstract'
3	Term frequency (TF) of a single query term in 'title + abstract'
4	Document frequency (DF) of a single query term in terms of 'title'
5	Document frequency (DF) of a single query term in terms of 'abstract'
6	Document frequency (DF) of a single query term in terms of 'title + abstract'
7	Document length in terms of 'title'
8	Document length in terms of 'abstract'
9	Document length in terms of 'title + abstract'
10	Average document length in terms of 'title'
11	Average document length in terms of 'abstract'
12	Average document length in terms of 'title + abstract'
13	Corpus size

(body, anchor, title and url) and document frequency of each query term as meta features. Note that different from OHSUMED, the stream length of each document is already included in LETOR 2.0. The meta features are listed in Table. 5. They can be used to derive new strong features or tuning the parameters in existing features like BM25.

3.2 Adopting New Sampling Strategy

Here we discuss how to better sample documents for a query from the original document collection to make LETOR more consistent with real world applications.

As we have pointed out, for the OHSUMED dataset, it might not be reasonable to only look at the small number of judged documents. A possible solution is that for a query, we include all the documents containing at least one query term. By doing so, we can signicantly increase the number of documents to be ranked. However, there are also some concerns with this method because we have introduced a number of unjudgment document into the dataset. First, in training, it is not clear how to use these un-judged documents. Simply regarding them as irrelevant may or may not be reasonable. In fact, this is a research problem by itself: how to make good use of unlabeled documents in learning. Second, in testing, it is difficult to evaluate the accuracy of a ranking algorithm. At least the current evaluation measures used in LETOR, MAP and NDCG, cannot be computed in this case. Considering the two issues, we suggest keeping the sampling of the current version of OHSUMED dataset unchanged.

For the TD2003 and TD2004 datasets, it is a different story. The judgments in these datasets only include the "relevant" documents. In other words, all unjudged documents are regarded as irrelevant [22, 17, 3, 20]. In this case, if we introduce more documents, there are no such confusions on the labels as for the OHSUMED dataset. Note that, in TD2003 and TD2004 datasets, all the relevant documents were added to the datasets, while no enough unjudged documents were added at the same time. As a result, the document distribution is changed significantly as compared to the original collection. To solve the problem, we suggest making one of the following changes.

Table 4: Features for TD2003 and TD2004

ID	Feature Description
1	Term frequency (TF) of body
2	TF of anchor
3	TF of title
4	TF of URL
5	TF of whole document
6	Inverse document frequency (IDF) of body
7	IDF of anchor
8	IDF of title
9	IDF of URL
10	IDF of whole document
11	TD*IDF of body
12	TD*IDF of anchor
13	TD*IDF of title
14	TD*IDF of URL
15	TD*IDF of whole document
16	Document length (DL) of body
17	DL of anchor
18	DL of title
19	DL of URL
20	DL of whole document
21	BM25 of body
22	BM25 of anchor
23	BM25 of title
24	BM25 of URL
25	BM25 of whole document
26	LMIR.ABS of body
27	LMIR.ABS of anchor
28	LMIR.ABS of title
29	LMIR.ABS of URL
30	LMIR.ABS of whole document
31	LMIR.DIR of body
32	LMIR.DIR of anchor
33	LMIR.DIR of title
34	LMIR.DIR of URL
35	LMIR.DIR of whole document
36	LMIR.JM of body
37	LMIR.JM of anchor
38	LMIR.JM of title
39	LMIR.JM of URL
40	LMIR.JM of whole document
41	Sitemap based feature propagation
42	Sitemap based score propagation
43	Hyperlink base score propagation: weighted in-link
44	Hyperlink base score propagation: weighted out-link
45	Hyperlink base score propagation: uniform out-link
46	Hyperlink base feature propagation: weighted in-link
47	Hyperlink base feature propagation: weighted out-link
48	Hyperlink base feature propagation: uniform out-link
49	HITS authority
50	HITS hub
51	PageRank
52	HostRank
53	Topical PageRank
54	Topical HITS authority
55	Topical HITS hub
56	Inlink number
57	Outlink number
58	Number of slash in URL
59	Length of URL
60	Number of child page
61	BM25 of extracted title
62	LMIR.ABS of extracted title
63	LMIR.DIR of extracted title
64	LMIR.JM of extracted title

Table 5: Meta Features for TREC

ID	Feature Description
1	Term frequency (TF) of a single query term in body
2	TF of a single query term in anchor
3	TF of a single query term in title
4	TF of a single query term in URL
5	TF of a single query term in whole document
6	Inverse document frequency (IDF) of a single query term in body
7	IDF of a single query term in anchor
8	IDF of a single query term in title
9	IDF of a single query term in URL
10	IDF of a single query term in whole document
11	TD*IDF of a single query term in body
12	TD*IDF of a single query term in anchor
13	TD*IDF of a single query term in title
14	TD*IDF of a single query term in URL
15	TD*IDF of a single query term in whole document
16	Average length of body
17	Average length of anchor
18	Average length of title
19	Average length of URL
20	Average length of whole document
21	Corpus size

(1) For each query, we can create a document pool with all documents containing at least one query term in it. The problem is that, however, there are too many documents in the pool for a query to rank. Table 6 shows the number of documents containing at least one query term for each query ($|D(q)|$). As can be seen, there are about 5 millon documents for TD2003[2] which need to be considered for feature extraction. Note that the size (with zip compression) of LETOR 2.0 with about 50000 documents in TD2003 dataset and 75000 documents in TD2004 dataset is about 44M bytes. The number of documents w.r.t. this new sampling strategy is about 100 times the number in the current version of TD2003 and TD2004. Then the size of the new TD2003 and TD2004 datasets will be about 4G bytes without considering the new features added. Such a dataset would be too large to download and to use[3].

(2) For a query, we first create a pool with all the documents containing at least one query term. Then we rank the documents in this pool in the descending order of their BM25 scores. Let $p = \max(pos(rd))$ be the rank position of the last relevant document. We select $p + 1000$ documents for this query for feature extraction. In this way, we will get about 0.6 million documents for TD2003[4] as shown in Table 6. This is about 10 times the current version of TD2003 and TD2004.

(3) We only change the current version of LETOR slightly. We remove all the relevant documents that are ranked out of the top 1000 positions according to their BM25 scores. In this way, the data distribution will not be as biased as before either.

[2]There are about 10 millon documents for TD2004, which are not listed here.

[3]Someone may argue that 4.4G is not very large for download. However, it would take a lot of training time for a learning algorithm.

[4]And about 0.8 million documents for TD2004.

4. CREATING LARGER SCALE DATASETS

There are only about (or less than) 100 queries in each dataset of LETOR. These datasets are relatively small. Large scale datasets are desired for the research on learning to rank. Here we suggest three possible solutions.

(1) Note that there are three tasks in TREC web track 2003 and 2004: topic distillation (TD), homepage finding (HP) and named page finding (NP). In LETOR 2.0, only topic distillation task was used for data creation. The first solution we suggest is to add the queries of the other two tasks into LETOR. There are two benefits to include all the three tasks. First, by doing so, we can enlarge the data to hundreds of queries, as shown in Table 7. Second, since there are three different types of queries, some other research topics can be conducted based on LETOR, such as query classification and query dependent ranking.

(2) Considering that hundreds of queries may be still not large enough, another possible data source is the Million Query Track at TREC 2007[5], which contains about 10000 queries (about 1700 queries are labeled in the track last year, and more queries are expected to be labeled this year.). The documents associated with these queries are retrieved from the ".gov2" collection. It would be great to create a large dataset based on this source. The creation process can be very similar to that of the current version of LETOR: indexing the ".gov2" dataset, extracting standard features, and making the data files publicly available.

(3) In addition, we make a even wilder proposal on leveraging the efforts of the entire IR community (and even web users) for data collection. The basic idea is to develop a meta search engine, which mixes the results from multiple search engines. We will collect the search log of this meta search engine to mine the ground truth. Of course, the one who uses this engine should be aware that we are recording the search log for research purpose. After mining the ground truth from the search log, we can extract features for query-document pairs and release the dataset.

The advantages of using this solution are as follows. First, it is easy for data collection. One only need to use the engine like some other search engines without any other effort. Second, the number of available queries can increase along with time, and thus can be extremely large eventually. Third, the documents are up-to-date webpages indexed by the search engines, while the documents in the ".gov" and ".gov2" collections are pretty old.

There are also concerns with this solution. First, we need to filter the logs to delete some privacy related queries. Second, the ground truth mining from log

[5]We would like to thank the reviewer for the recommendation on this data source.

Table 6: Statistics on sampled documents of TD2003 dataset

QueryID	$\|D(q)\|$	$max(pos(rd))$
1	47360	16435
2	11263	1247
3	38852	22
4	196651	1917
5	220930	3685
6	81253	193
7	115902	66641
8	117876	12
9	301335	286648
10	101552	512
11	45090	255
12	68896	166
13	1010	2
14	87993	560
15	139505	560
16	219321	1896
17	525	3
18	533	5
19	109707	316
20	79393	44112
21	1619	737
22	106215	532
23	7536	268
24	181882	7
25	46414	116
26	141552	1676
27	112917	783
28	182231	1798
29	3149	45
30	41912	409
31	68912	526
32	61599	20485
33	148548	1078
34	6694	444
35	63765	47773
36	72125	2657
37	125920	1250
38	134569	19881
39	536840	37821
40	111526	253
41	52267	1351
42	6035	250
43	64461	5340
44	28141	3004
45	19445	517
46	19856	1058
47	132448	53441
48	7609	529
49	179907	69
50	2885	80
total	4653926	629365

Table 7: Number of queries in TREC web track

Task	TREC2003	TREC2004
TD	50	75
HP+NP	300	150

data may not be very easy because it is still a research topic by its own. Alternatively, we can also release the log data directly for research.

5. CONCLUSIONS

LETOR is a public dataset for learning to rank, which eases the research in this direction. Since benchmark dataset is always very important for a research direction, continuous discussions and efforts should be given to the construction of such datasets. In this paper, we discussed how to make LETOR more useful and reliable from three aspects, i.e., adding more raw features, adopting new document sampling strategies, and creating larger scale datasets. We hope with the joint efforts of the learning to rank research community, a better benchmark dataset can be built shortly and our research work can benefit from it.

6. REFERENCES

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, May 1999.

[2] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 89–96, New York, NY, USA, 2005. ACM Press.

[3] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 129–136, New York, NY, USA, 2007. ACM Press.

[4] J. Elsas, V. Carvalho, and J. Carbonell. Fast learning of document ranking functions with the committee perceptron. *Proceedings of the international conference on Web search and web data mining*, pages 55–64, 2008.

[5] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, 2003.

[6] X. Geng, T.-Y. Liu, T. Qin, A. Arnold, H. Li, and H.-Y. Shum. Query dependent ranking using k-nearest neighbor. In *SIGIR '08: Proceedings of the 31th annual international ACM SIGIR conference on Research and development in information retrieval*, New York, NY, USA, 2008. ACM.

[7] J. Guiver and E. Snelson. Learning to rank with softrank and gaussian processes. In *SIGIR '08: Proceedings of the 31th annual international ACM SIGIR conference on Research and development in information retrieval*, New York, NY, USA, 2008. ACM Press.

[8] R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. In *ICANN1999*, pages 97–102, 1999.

[9] R. Herbrich, T. Graepel, and K. Obermayer. *Large margin rank boundaries for ordinal regression*. MIT Press, Cambridge, MA, 2000.

[10] R. Jin, H. Valizadegan, and H. Li. Ranking refinement and its application to information retrieval. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 397–406, New York, NY, USA, 2008. ACM.

[11] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.

[12] P. Li, C. Burges, and Q. Wu. Mcrank: Learning to rank using multiple classification and gradient boosting. In *NIPS2007*, 2007.

[13] L. Nie, B. D. Davison, and X. Qi. Topical link analysis for web search. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 91–98, New York, NY, USA, 2006. ACM.

[14] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[15] T. Qin, T.-Y. Liu, X.-D. Zhang, Z. Chen, and W.-Y. Ma. A study of relevance propagation for web search. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 408–415, New York, NY, USA, 2005. ACM.

[16] T. Qin, T.-Y. Liu, X.-D. Zhang, D.-S. Wang, W.-Y. Xiong, and H. Li. Learning to rank relational objects and its application to web search. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 407–416, New York, NY, USA, 2008. ACM.

[17] T. Qin, X.-D. Zhang, M.-F. Tsai, D.-S. Wang, T.-Y. Liu, and H. Li. Query-level loss functions for information retrieval. *Information Processing & Management*, 2007.

[18] S. J. M. M. H.-B. M. G. S. E. Robertson, S. Walker. Okapi at TREC-3. In *Text REtrieval Conference*, pages 109–126, 1994.

[19] A. Shakery and C. Zhai. Relevance propagation for topic distillation uiuc trec 2003 web track experiments. In *TREC*, pages 673–677, 2003.

[20] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma. Frank: a ranking method with fidelity loss. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 383–390, New York, NY, USA, 2007. ACM.

[21] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank - theory and algorithm. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, New York, NY, USA, 2008. ACM Press.

[22] J. Xu and H. Li. Adarank: a boosting algorithm for information retrieval.

[23] J. Xu, T.-Y. Liu, M. Lu, H. Li, and W.-Y. Ma. Directly optimizing evaluation measures in learning to rank. In *SIGIR '08: Proceedings of the 31th annual international ACM SIGIR conference on Research and development in information retrieval*, New York, NY, USA, 2008. ACM Press.

[24] G.-R. Xue, Q. Yang, H.-J. Zeng, Y. Yu, and Z. Chen. Exploiting the hierarchical structure for link analysis. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 186–193, New York, NY, USA, 2005. ACM.

[25] J. Yeh, J. Lin, H. Ke, and W. Yang. Learning to rank for information retrieval using genetic programming. In *Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, 2007.

[26] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, 22(2):179–214, 2004.