Optimizing Search Engines using Clickthrough Data

Thorsten Joachims
Cornell University
Department of Computer Science
Ithaca, NY 14853 USA
tj@cs.cornell.edu

ABSTRACT

This paper presents an approach to automatically optimizing the retrieval quality of search engines using clickthrough data. Intuitively, a good information retrieval system should present relevant documents high in the ranking, with less relevant documents following below. While previous approaches to learning retrieval functions from examples exist, they typically require training data generated from relevance judgments by experts. This makes them difficult and expensive to apply. The goal of this paper is to develop a method that utilizes clickthrough data for training, namely the query-log of the search engine in connection with the log of links the users clicked on in the presented ranking. Such clickthrough data is available in abundance and can be recorded at very low cost. Taking a Support Vector Machine (SVM) approach, this paper presents a method for learning retrieval functions. From a theoretical perspective, this method is shown to be well-founded in a risk minimization framework. Furthermore, it is shown to be feasible even for large sets of queries and features. The theoretical results are verified in a controlled experiment. It shows that the method can effectively adapt the retrieval function of a meta-search engine to a particular group of users, outperforming Google in terms of retrieval quality after only a couple of hundred training examples.

1. INTRODUCTION

Which WWW page(s) does a user actually want to retrieve when he types some keywords into a search engine? There are typically thousands of pages that contain these words, but the user is interested in a much smaller subset. One could simply ask the user for feedback. If we knew the set of pages actually relevant to the user's query, we could use this as training data for optimizing (and even personalizing) the retrieval function.

Unfortunately, experience shows that users are only rarely willing to give explicit feedback. However, this paper argues that sufficient information is already hidden in the logfiles of WWW search engines. Since major search engines receive millions of queries per day, such data is available in abundance. Compared to explicit feedback data, which is typically elicited in laborious user studies, any information that can be extracted from logfiles is virtually free and substantially more timely.

This paper presents an approach to learning retrieval functions by analyzing which links the users click on in the presented ranking. This leads to a problem of learning with preference examples like "for query q, document d_a should be ranked higher than document d_b". More generally, I will formulate the problem of learning a ranking function over a finite domain in terms of empirical risk minimization. For this formulation, I will present a Support Vector Machine (SVM) algorithm that leads to a convex program and that can be extended to non-linear ranking functions. Experiments show that the method can successfully learn a highly effective retrieval function for a meta-search engine.

This paper is structured as follows. It starts with a definition of what clickthrough data is, how it can be recorded, and how it can be used to generate training examples in the form of preferences. Section 3 then introduces a general framework for learning retrieval functions, leading to an SVM algorithm for learning parameterized orderings in Section 4. Section 5 evaluates the method based on experimental results.

2. CLICKTHROUGH DATA IN SEARCH ENGINES

Clickthrough data in search engines can be thought of as triplets (q, r, c) consisting of the query q, the ranking r presented to the user, and the set c of links the user clicked on. Figure 1 illustrates this with an example: the user asked the query "support vector machine", received the ranking shown in Figure 1, and then clicked on the links ranked 1, 3, and 7. Since every query corresponds to one triplet, the amount of data that is potentially available is virtually unlimited.

Clearly, users do not click on links at random, but make a (somewhat) informed choice. While clickthrough data is typically noisy and clicks are not "perfect" relevance judgments, the clicks are likely to convey some information. The key question is: how can this information be extracted? Before deriving a model of how clickthrough data can be analyzed, let's first consider how it can be recorded.

2.1 Recording Clickthrough Data

Clickthrough data can be recorded with little overhead and without compromising the functionality and usefulness

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGKDD 02 Edmonton, Alberta, Canada
Copyright 2002 ACM 1-58113-567-X/02/0007 ...$5.00.

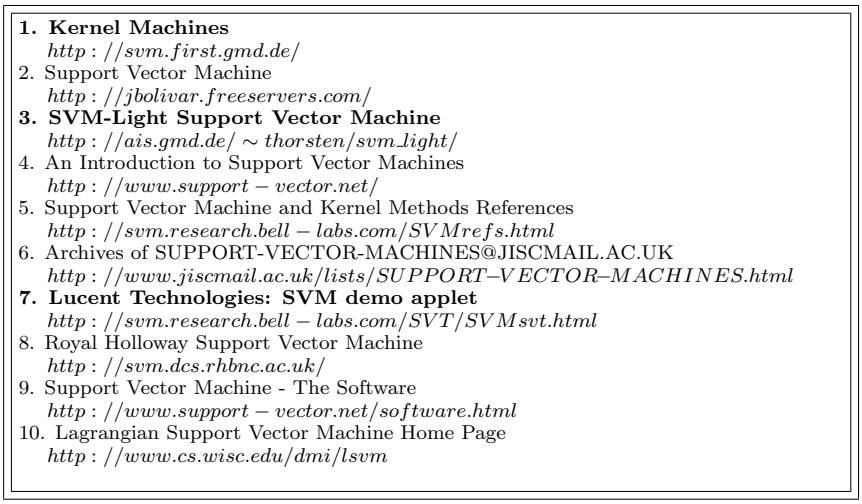

Figure 1: Ranking presented for the query "support vector machine". Marked in bold are the links the user clicked on.

of the search engine. In particular, compared to explicit user feedback, it does not add any overhead for the user. The query q and the returned ranking r can easily be recorded whenever the resulting ranking is displayed to the user. For recording the clicks, a simple proxy system can keep a logfile. For the experiments in this paper, the following system was used.

Each query is assigned a unique ID which is stored in the query-log along with the query words and the presented ranking. The links on the results-page presented to the user do not lead directly to the suggested document, but point to a proxy server. These links encode the query-ID and the URL of the suggested document. When the user clicks on the link, the proxy-server records the URL and the query-ID in the click-log. The proxy then uses the *HTTP Location* command to forward the user to the target URL. This process can be made transparent to the user and does not influence system performance.

This shows that clickthrough data can be recorded easily and at little cost. Let's now address the key question of how it can be analyzed in a principled and efficient way.

2.2 What Kind of Information does Clickthrough Data Convey?

There are strong dependencies between the three parts of (q, r, c). The presented ranking r depends on the query q as determined by the retrieval function implemented in the search engine. Furthermore, the set c of clicked-on links depends on both the query q and the presented ranking r. First, a user is more likely to click on a link, if it is relevant to q [16]. While this dependency is desirable and interesting for analysis, the dependency of the clicks on the presented ranking r muddies the water. In particular, a user is less likely to click on a link low in the ranking, independent of how relevant it is. In the extreme, the probability that the user clicks on a link at rank 10.000 is virtually zero even if it is the document most relevant to the query. No user will scroll down the ranking far enough to observe this link. Therefore, in order to get interpretable and meaningful results from clickthrough data, it is necessary to consider and model the dependencies of c on q and r appropriately.

	retrieval function		
	bxx	tfc	hand-tuned
avg. clickrank	6.26±1.14	6.18±1.33	6.04± 0.92

Table 1: Average clickrank for three retrieval functions ("bxx", "tfc" [23] , and a "hand-tuned" strategy that uses different weights according to HTML tags) implemented in LASER. Rows correspond to the retrieval method used by LASER at query time; columns hold values from subsequent evaluation with other methods. Figures reported are means and two standard errors. The data for this table is taken from [5] .

Before defining such a model, let's first consider an interpretation of clickthrough data that is not appropriate. A click on a particular link cannot be seen as an *absolute* relevance judgment. Consider the empirical data in Table 1. The data is taken from [5] and was recorded for the search engine LASER covering the WWW of the CMU School of Computer Science. The table shows the average rank of the clicks per query (e.g. 3.67 in the example in Figure 1). Each table cell contains the average clickrank for three retrieval strategies averaged over ≈ 1400 queries. The average clickrank is almost equal for all methods. However, according to subjective judgments, the three retrieval functions are substantially different in their ranking quality. The lack of difference in the observed average clickrank can be explained as follows. Since users typically scan only the first l (e.g. $l \approx 10$ [24]) links of the ranking, clicking on a link cannot be interpreted as a relevance judgment on an absolute scale. Maybe a document ranked much lower in the list was much more relevant, but the user never saw it. It appears that users click on the (relatively) most promising links in the top l, independent of their absolute relevance. How can these relative preference judgments be captured

and analyzed?

Consider again the example from Figure 1. While it is not possible to infer that the links 1, 3, and 7 are relevant on an *absolute* scale, it is much more plausible to infer that link 3 is more relevant than link 2 with probability higher than random. *Assuming that the user scanned the ranking from top to bottom, he must have observed link* 2 *before clicking on* 3*, making a decision to not click on it.* Given that the abstracts presented with the links are sufficiently informative, this gives some indication of the user's preferences. Similarly, it is possible to infer that link 7 is more relevant than links 2, 4, 5, and 6. This means that clickthrough data does not convey *absolute* relevance judgments, but partial *relative* relevance judgments for the links the user browsed through. A search engine ranking the returned links according to their relevance to q should have ranked links 3 ahead of 2, and link 7 ahead of 2, 4, 5, and 6. Denoting the ranking preferred by the user with r^*, we get partial (and potentially noisy) information of the form

$$\begin{aligned} link_3 <_{\mathrm{r}^*} link_2 \qquad & link_7 <_{\mathrm{r}^*} link_2 \\ & link_7 <_{\mathrm{r}^*} link_4 \\ & link_7 <_{\mathrm{r}^*} link_5 \\ & link_7 <_{\mathrm{r}^*} link_6 \end{aligned} \tag{1}$$

This strategy for extracting preference feedback is summarized in the following algorithm.

Algorithm 1. (Extracting Preference Feedback from Clickthrough)
For a ranking $(link_1, link_2, link_3, ...)$ *and a set* C *containing the ranks of the clicked-on links, extract a preference example*

$$link_i <_{r^*} link_j$$

for all pairs $1 \leq j < i$*, with* $i \in C$ *and* $j \notin C$*.*

Unfortunatly, this type of feedback is not suitable for standard machine learning algorithms. The following derives a new learning algorithm, so that this "weak" type of relative feedback can be used as training data.

3. A FRAMEWORK FOR LEARNING OF RETRIEVAL FUNCTIONS

The problem of information retrieval can be formalized as follows. For a query q and a document collection $D = \{\mathrm{d}_1, ..., \mathrm{d}_m\}$, the optimal retrieval system should return a ranking r^* that orders the documents in D according to their relevance to the query. While the query is often represented as merely a set of keywords, more abstractly it can also incorporate information about the user and the state of the information search.

Typically, retrieval systems do not achieve an optimal ordering r^*. Instead, an operational retrieval function f is evaluated by how closely its ordering $\mathrm{r}_{\mathrm{f(q)}}$ approximates the optimum. Formally, both r^* and $\mathrm{r}_{\mathrm{f(q)}}$ are binary relations over $D \times D$ that fulfill the properties of a weak ordering, i.e. $\mathrm{r}^* \subset D \times D$ and $\mathrm{r}_{\mathrm{f(q)}} \subset D \times D$ being asymmetric, and negatively transitive. If a document d_i is ranked higher than d_j for an ordering r, i.e. $\mathrm{d}_i <_{\mathrm{r}} \mathrm{d}_j$, then $(\mathrm{d}_i, \mathrm{d}_j) \in \mathrm{r}$, otherwise $(\mathrm{d}_i, \mathrm{d}_j) \notin \mathrm{r}$. If not stated otherwise, let's assume for simplicity that r^* and $\mathrm{r}_{\mathrm{f(q)}}$ are both strict orderings. This means that for all pairs $(\mathrm{d}_1, \mathrm{d}_2) \in D \times D$ either $\mathrm{d}_i <_{\mathrm{r}} \mathrm{d}_j$ or $\mathrm{d}_j <_{\mathrm{r}} \mathrm{d}_i$. However, it is straightforward to generalize most of the following result to r^* being a weak ordering.

What is an appropriate measure of similarity between the system ranking $\mathrm{r}_{\mathrm{f(q)}}$ and the target ranking r^*? For a binary relevance scale, Average Precision [1] is most frequently used in information retrieval. However, most information retrieval researchers agree that binary relevance is very coarse and that it is merely used as as simplifying assumption. Since the method presented in the following does not require such a simplification, we will depart from a binary relevance scheme and adapt Kendall's τ [19][21] as a performance measure. For comparing the ordinal correlation of two random variables, Kendall's τ is the most frequently used measure in statistics. For two finite strict orderings $\mathrm{r}_a \subset D \times D$ and $\mathrm{r}_b \subset D \times D$, Kendall's τ can be defined based on the number P of concordant pairs and the number Q of discordant pairs (inversions). A pair $\mathrm{d}_i \neq \mathrm{d}_j$ is concordant, if both r_a and r_b agree in how they order d_i and d_j. It is discordant if they disagree. Note, that on a finite domain D of m documents, the sum of P and Q is $\binom{m}{2}$ for strict orderings. In this case, Kendall's τ can be defined as:

$$\tau(\mathrm{r}_a, \mathrm{r}_b) = \frac{P - Q}{P + Q} = 1 - \frac{2Q}{\binom{m}{2}} \tag{2}$$

As an example, consider the two rankings r_a and r_b as follows:

$$\mathrm{d}_1 <_{\mathrm{r}_a} \mathrm{d}_2 <_{\mathrm{r}_a} \mathrm{d}_3 <_{\mathrm{r}_a} \mathrm{d}_4 <_{\mathrm{r}_a} \mathrm{d}_5 \tag{3}$$

$$\mathrm{d}_3 <_{\mathrm{r}_b} \mathrm{d}_2 <_{\mathrm{r}_b} \mathrm{d}_1 <_{\mathrm{r}_b} \mathrm{d}_4 <_{\mathrm{r}_b} \mathrm{d}_5 \tag{4}$$

The number of discordant pairs is 3 (ie. $\{\mathrm{d}_2, \mathrm{d}_3\}$, $\{\mathrm{d}_1, \mathrm{d}_2\}$, $\{\mathrm{d}_1, \mathrm{d}_3\}$), while all remaining 7 pairs are concordant. Therefore, $\tau(\mathrm{r}_a, \mathrm{r}_b) = 0.4$.

Why is this similarity measure appropriate for information retrieval? Equation (2) depends only on Q for a fixed collection. Taken as a distance measure, Q fulfills the axioms of Kemeny and Snell [18] for strict orderings. Furthermore, it is proportional to the measure of Yao [26] proposed for evaluating information retrieval systems. If applied to a binary relevance scale, it is easy to see that maximizing (2) is equivalent to minimizing the average rank of the relevant documents. And finally, $\tau(\mathrm{r}_{\mathrm{f(q)}}, \mathrm{r}^*)$ is related to Average Precision [1]. In particular, the number of inversions Q gives a lower bound on the Average Precision as follows.

$$AvgPrec(\mathrm{r}_{\mathrm{f(q)}}) \geq \frac{1}{R}\left[Q + \binom{R+1}{2}\right]^{-1}\left(\sum_{i=1}^{R}\sqrt{i}\right)^2 \tag{5}$$

R is the number of relevant documents. The proof is given in the appendix. These arguments show how $\tau(\mathrm{r}_{\mathrm{f(q)}}, \mathrm{r}^*)$ relates to retrieval quality. They demonstrate that maximizing $\tau(\mathrm{r}_{\mathrm{f(q)}}, \mathrm{r}^*)$ is connected to improved retrieval quality in multiple frameworks.

We are now in a position to define the problem of learning a ranking function. For a fixed but unknown distribution $\Pr(\mathrm{q}, \mathrm{r}^*)$ of queries and target rankings on a document collection D with m documents, the goal is to learn a retrieval function f(q) for which the expected Kendall's τ

$$\tau_P(\mathrm{f}) = \int \tau(\mathrm{r}_{\mathrm{f(q)}}, \mathrm{r}^*) d\Pr(\mathrm{q}, \mathrm{r}^*) \tag{6}$$

is maximal. Note that (6) is (proportional to) a risk functional [25] with $-\tau$ as the loss function. While the goal of learning is now defined, the question remains whether it is possible to design learning methods that optimize (6)?

4. AN SVM ALGORITHM FOR LEARNING OF RANKING FUNCTIONS

Most work on machine learning in information retrieval does not consider the formulation of above, but simplifies the task to a binary classification problem with the two classes "relevant" and "non-relevant". Such a simplification has several drawbacks. For example, due to a strong majority of "non-relevant" documents, a learner will typically achieve the maximum predictive classification accuracy, if it always responds "non-relevant", independent of where the relevant documents are ranked. But even more importantly, Section 2.2 showed that such absolute relevance judgments cannot be extracted from clickthrough data, so that they are simply not available. Therefore, the following algorithm directly addresses (6), taking an empirical risk minimization approach [25]. Given an independently and identically distributed training sample S of size n containing queries q with their target rankings r^*

$$(\mathrm{q}_1, \mathrm{r}_1^*), (\mathrm{q}_2, \mathrm{r}_2^*), ..., (\mathrm{q}_n, \mathrm{r}_n^*). \tag{7}$$

the learner $\mathcal{L}$ will select a ranking function f from a family of ranking functions F that maximizes the empirical τ

$$\tau_S(\mathrm{f}) = \frac{1}{n} \sum_{i=1}^{n} \tau(\mathrm{r}_{\mathrm{f}(\mathrm{q}_i)}, \mathrm{r}_i^*). \tag{8}$$

on the training sample. Note that this setup is analogous to e.g. classification by minimizing training error, just that the target is not a class label, but a binary ordering relation.

4.1 The Ranking SVM Algorithm

Is it possible to design an algorithm and a family of ranking functions F so that (a) finding the function $\mathrm{f} \in F$ maximizing (8) is efficient, and (b) that this function generalizes well beyond the training data. Consider the class of linear ranking functions

$$(\mathrm{d}_i, \mathrm{d}_j) \in \mathrm{f}_{\vec{w}}(\mathrm{q}) \Longleftrightarrow \vec{w}\Phi(\mathrm{q}, \mathrm{d}_i) > \vec{w}\Phi(\mathrm{q}, \mathrm{d}_j). \tag{9}$$

$\vec{w}$ is a weight vector that is adjusted by learning. $\Phi(\mathrm{q}, \mathrm{d})$ is a mapping onto features that describe the match between query q and document d like in the description-oriented retrieval approach of Fuhr et al. [10][11]. Such features are, for example, the number of words that query and document share, the number of words they share inside certain HTML tags (e.g. TITLE, H1, H2, ...), or the page-rank of d [22] (see also Section 5.2). Figure 2 illustrates how the weight vector $\vec{w}$ determines the ordering of four points in a two-dimensional example. For any weight vector $\vec{w}$, the points are ordered by their projection onto $\vec{w}$ (or, equivalently, by their signed distance to a hyperplane with normal vector $\vec{w}$). This means that for $\vec{w}_1$ the points are ordered $(1, 2, 3, 4)$, while $\vec{w}_2$ implies the ordering $(2, 3, 1, 4)$.

Instead of maximizing (8) directly, it is equivalent to minimize the number Q of discordant pairs in Equation (2). For the class of linear ranking functions (9), this is equivalent to finding the weight vector so that the maximum number of the following inequalities is fulfilled.

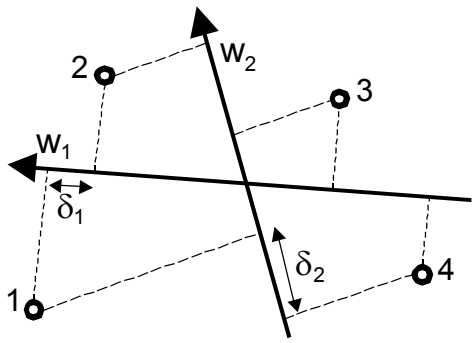

Figure 2: Example of how two weight vectors $\vec{w}_1$ and $\vec{w}_2$ rank four points.

$$\forall(\mathrm{d}_i, \mathrm{d}_j) \in \mathrm{r}_1^* : \ \vec{w}\Phi(\mathrm{q}_1, \mathrm{d}_i) > \vec{w}\Phi(\mathrm{q}_1, \mathrm{d}_j) \tag{10}$$

$$...$$

$$\forall(\mathrm{d}_i, \mathrm{d}_j) \in \mathrm{r}_n^* : \ \vec{w}\Phi(\mathrm{q}_n, \mathrm{d}_i) > \vec{w}\Phi(\mathrm{q}_n, \mathrm{d}_j) \tag{11}$$

Unfortunately, a direct generalization of the result in [13] shows that this problem is NP-hard. However, just like in classification SVMs [7], it is possible to approximate the solution by introducing (non-negative) slack variables $\xi_{i,j,k}$ and minimizing the upper bound $\sum \xi_{i,j,k}$. Adding SVM regularization for margin maximization to the objective leads to the following optimization problem, which is similar to the ordinal regression approach in [12].

Optimization Problem 1. (Ranking SVM)

$$\textit{minimize:} \qquad V(\vec{w}, \vec{\xi}) = \frac{1}{2}\, \vec{w} \cdot \vec{w} + C \sum \xi_{i,j,k} \tag{12}$$

subject to:

$$\forall(d_i, d_j) \in r_1^* : \vec{w}\Phi(q_1, d_i) \geq \vec{w}\Phi(q_1, d_j) + 1 - \xi_{i,j,1}$$

$$... \tag{13}$$

$$\forall(d_i, d_j) \in r_n^* : \vec{w}\Phi(q_n, d_i) \geq \vec{w}\Phi(q_n, d_j) + 1 - \xi_{i,j,n}$$

$$\forall i \forall j \forall k : \xi_{i,j,k} \geq 0 \tag{14}$$

C is a parameter that allows trading-off margin size against training error. Geometrically, the margin δ is the distance between the closest two projections within all target rankings. This is illustrated in Figure 2.

Optimization Problem 1 is convex and has no local optima. By rearranging the constraints (13) as

$$\vec{w}\,(\Phi(\mathrm{q}_k, \mathrm{d}_i) - \Phi(\mathrm{q}_k, \mathrm{d}_j)) \geq 1 - \xi_{i,j,k}, \tag{15}$$

it becomes apparent that the optimization problem is equivalent to that of a classification SVM on pairwise difference vectors $\Phi(\mathrm{q}_k, \mathrm{d}_i) - \Phi(\mathrm{q}_k, \mathrm{d}_j)$. Due to this similarity, it can be solved using decomposition algorithms similar to those used for SVM classification. In the following, an adaptation of the SVM^{light} algorithm [14] is used for training[1].

It can be shown that the learned retrieval function $\mathrm{f}_{\vec{w}^*}$ can always be represented as a linear combination of the feature

[1]Available at *http://svmlight.joachims.org*

vectors.

$$(\mathrm{d}_i, \mathrm{d}_j) \in \mathrm{f}_{\vec{w}^*}(\mathrm{q}) \quad (16)$$

$$\iff \vec{w}^* \Phi(\mathrm{q}, \mathrm{d}_i) > \vec{w}^* \Phi(\mathrm{q}, \mathrm{d}_j) \quad (17)$$

$$\iff \sum \alpha^*_{k,l} \Phi(\mathrm{q}_k, \mathrm{d}_l) \Phi(\mathrm{q}, \mathrm{d}_i) > \sum \alpha^*_{k,l} \Phi(\mathrm{q}_k, \mathrm{d}_l) \Phi(\mathrm{q}, \mathrm{d}_j) \quad (18)$$

This makes it possible to use Kernels [4][25] and extend the Ranking SVM algorithm to non-linear retrieval functions. The $\alpha^*_{k,l}$ can be derived from the values of the dual variables at the solution.

Most commonly, $\mathrm{f}_{\vec{w}^*}$ will be used for ranking the set of documents according to a new query q. In this case it is sufficient to sort the documents by their value of

$$rsv(\mathrm{q}, \mathrm{d}_i) = \vec{w}^* \Phi(\mathrm{q}, \mathrm{d}_i) = \sum \alpha^*_{k,l} \Phi(\mathrm{q}_k, \mathrm{d}_l) \Phi(\mathrm{q}, \mathrm{d}_j). \quad (19)$$

If Kernels are not used, this property makes the application of the learned retrieval function very efficient. Fast algorithms exists for computing rankings based on linear functions by means of inverted indices (see e.g. [1]).

4.2 Using Partial Feedback

If clickthrough logs are the source of training data, the full target ranking r^* for a query q is not observable. However, as shown in Section 2.2, a subset $\mathrm{r}' \subseteq \mathrm{r}^*$ can be inferred from the logfile. It is straightforward to adapt the Ranking SVM to the case of such partial data by replacing r^* with the observed preferences r'. Given a training set S

$$(\mathrm{q}_1, \mathrm{r}'_1), (\mathrm{q}_2, \mathrm{r}'_2), ..., (\mathrm{q}_n, \mathrm{r}'_n) \quad (20)$$

with partial information about the target ranking, this results in the following algorithm.

Optimization Problem 2. (Ranking SVM (partial))

$$\textit{minimize:} \qquad V(\vec{w}, \vec{\xi}) = \frac{1}{2} \vec{w} \cdot \vec{w} + C \sum \xi_{i,j,k} \quad (21)$$

subject to:

$$\forall (d_i, d_j) \in r'_1 : \vec{w}\Phi(q_1, d_i) > \vec{w}\Phi(q_1, d_j) + 1 - \xi_{i,j,1}$$

$$... \quad (22)$$

$$\forall (d_i, d_j) \in r'_n : \vec{w}\Phi(q_n, d_i) > \vec{w}\Phi(q_n, d_j) + 1 - \xi_{i,j,n}$$

$$\forall i \forall j \forall k : \xi_{i,j,k} \geq 0 \quad (23)$$

The resulting retrieval function is defined analogously. Using this algorithm results in finding a ranking function that has a low number of discordant pairs with respect to the observed parts of the target ranking.

5. EXPERIMENTS

The following experiments verify whether the inferences drawn from the clickthrough data are justified, and whether the Ranking SVM can successfully use such partial preference data. First, the experiment setup in the framework of a meta-search engine is described. It follow the results of an offline experiment and an online experiment. The offline experiment is designed to verify that the Ranking SVM can indeed learn a retrieval function maximizing Kendall's τ on partial preference feedback. The online experiment goes further and verifies that the learned retrieval function does improve retrieval quality as desired.

5.1 Experiment Setup: Meta-Search

To elicit data and provide a framework for testing the algorithm, I implemented a WWW meta-search engine called "Striver". Meta-search engines combine the results of several basic search engines without having a database of their own. Such a setup has several advantages. First, it is easy to implement while covering a large document collection — namely the whole WWW. Second, the basic search engines provide a basis for comparison.

The "Striver" meta-search engine works as follows. The user types a query into Striver's interface. This query is forwarded to "Google", "MSNSearch", "Excite", "Altavista", and "Hotbot". The results pages returned by these basic search engines are analyzed and the top 100 suggested links are extracted. After canonicalizing URLs, the union of these links composes the candidate set V. Striver ranks the links in V according to its learned retrieval function $f_{\vec{w}^*}$ and presents the top 50 links to the user. For each link, the system displays the title of the page along with its URL. The clicks of the user are recorded using the proxy system described in Section 2.1.

To be able to compare the quality of different retrieval functions, the method described in [16] is used. The key idea is to present two rankings at the same time. This particular form of presentation leads to a blind statistical test so that the clicks of the user demonstrate unbiased preferences. In particular, to compared two rankings A and B, they are combined into a single ranking C so that the following condition holds for any top l links of the combined ranking. The top l links of the combined ranking C contain the top k_a links from A and the top k_b links from B, with $|k_a - k_b| \leq 1$. In other words, if the user scans the links of C from top to bottom, at any point he has seen almost equally many links from the top of A as from the top of B. It is shown in [16] that such a combined ranking always exists and that it can be constructed efficiently.

An example is given in Figure 3. The results of two retrieval functions are combined into one ranking that is presented to the user. Note that the abstracts and all other aspects of the presentation are unified, so that the user cannot tell which retrieval strategy proposed a particular page. In the example, the user clicks on links 1, 3, and 7. What inference can one draw from these clicks?

In the example, the user must have seen the top 4 links from both individual rankings, since he clicked on link 7 in the combined ranking. He decided to click on 3 links in the top 4 in ranking A (namely 1, 2, and 4), but only on 1 link in ranking B (namely 1). It is reasonable to conclude, that (with probability larger than random) the top 4 links from A were judged to be better than those from B for this query.

It is straightforward to design hypothesis tests regarding the user preferences based on such a combined ranking. Roughly speaking, if a user does not have any preference regarding A or B, he will click equally often on links in the top k of each ranking. If for a sample of pairs (A_1, B_1), ... , (A_s, B_s) the user clicks on significantly more links from A than from B, then A must contain more relevant links than B in the following sense. Formalizing the assumption that

- users click by some $\epsilon > 0$ more often on a more relevant link than on a less relevant link
- and that the decision of the user to click on a link is not influenced by other factors (i.e. links from both A

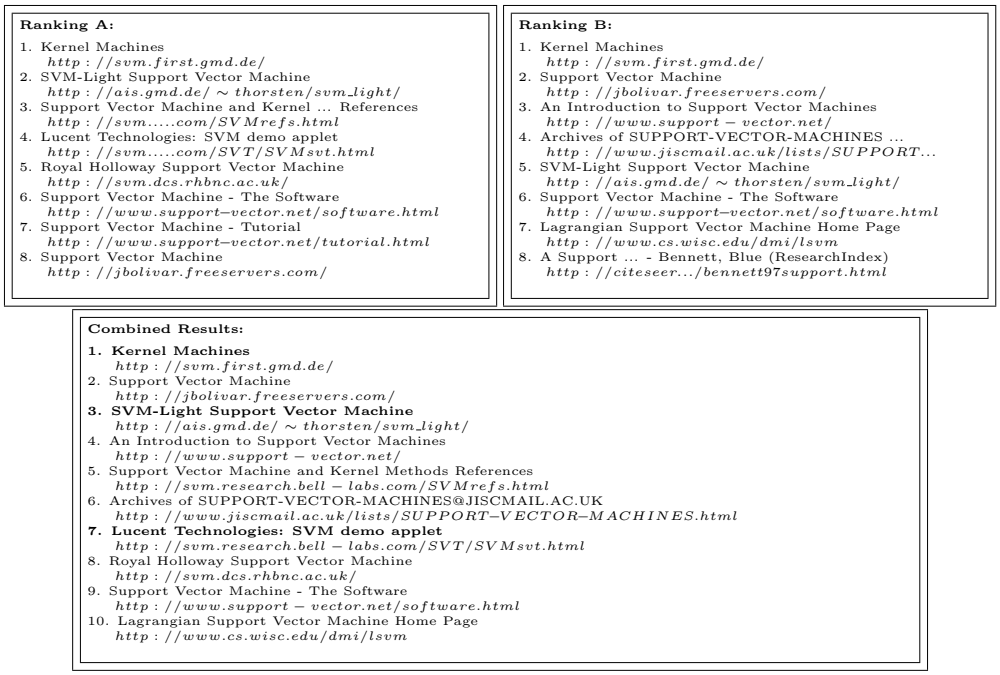

Figure 3: Example for query "support vector machine". The two upper boxes show the rankings returned by retrieval functions A and B. The lower box contains the combined ranking presented to the user. The links the user clicked on are marked in bold.

and B are presented in the same way)

it is proven and empirically verified in [16] that the conclusions drawn from this method lead to the same result as an evaluation with explicit manual relevance judgments for large s.

5.2 Offline Experiment

This experiment verifies that the Ranking SVM can indeed learn regularities using partial feedback from clickthrough data. To generate a first training set, I used the Striver search engine for all of my own queries during October, 2001. Striver displayed the results of Google and MSNSearch using the combination method from the previous section. All clickthrough triplets were recorded. This resulted in 112 queries with a non-empty set of clicks. This data provides the basis for the following offline experiment.

To learn a retrieval function using the Ranking SVM, it is necessary to design a suitable feature mapping $\Phi(\mathrm{q}, \mathrm{d})$ describing the match between a query q and a document d. The following features are used in the experiment. However, this set of features is likely to be far from optimal. While the attributes reflect some of my intuition about what could be important for learning a good ranking, I included only those features that were easy to implement. Furthermore, I did not do any feature selection or similar tuning, so that an appropriate design of features promises much room for improvement. The implemented features are the following:

1. Rank in other search engines (38 features total):

   rank_X: 100 minus rank in $X \in$ {Google, MSN-Search, Altavista, Hotbot, Excite} divided by 100 (minimum 0)

   top1_X: ranked #1 in $X \in$ {Google, MSNSearch, Altavista, Hotbot, Excite} (binary $\{0, 1\}$)

   top10_X: ranked in top 10 in $X \in$ {Google, MSNSearch, Altavista, Hotbot, Excite} (binary $\{0, 1\}$)

   top50_X: ranked in top 50 in $X \in$ {Google, MSNSearch, Altavista, Hotbot, Excite} (binary $\{0, 1\}$)

   top1count_X: ranked #1 in X of the 5 search engines

   top10count_X: ranked in top 10 in X of the 5 search engines

   top50count_X: ranked in top 50 in X of the 5 search engines

2. Query/Content Match (3 features total):

   query_url_cosine: cosine between URL-words and query (range $[0, 1]$)

   query_abstract_cosine: cosine between title-words and query (range $[0, 1]$)

   domain_name_in_query: query contains domain-name from URL (binary $\{0, 1\}$)

3. Popularity-Attributes ($\sim$ 20.000 features total):

   url_length: length of URL in characters divided by 30

   country_X: country code X of URL (binary attribute $\{0, 1\}$ for each country code)

Comparison	more clicks on learned	less clicks on learned	tie (with clicks)	no clicks	total
Learned vs. Google	29	13	27	19	88
Learned vs. MSNSearch	18	4	7	11	40
Learned vs. Toprank	21	9	11	11	52

Table 2: Pairwise comparison of the learned retrieval function with Google, MSNSearch, and the non-learning meta-search ranking. The counts indicate for how many queries a user clicked on more links from the top of the ranking returned by the respective retrieval function.

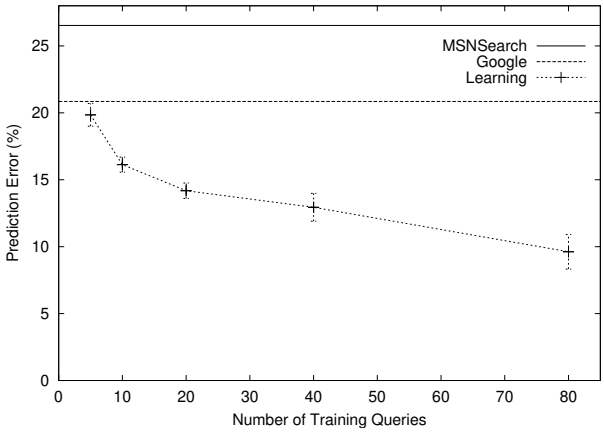

Figure 4: Generalization error of the Ranking SVM depending on the size of the training set. The error bars show one standard error.

domain_X: domain X of URL (binary attribute $\{0,1\}$ for each domain name)

abstract_contains_home: word "home" appears in URL or title (binary attribute $\{0,1\}$)

url_contains_tilde: URL contains "~" (binary attribute $\{0,1\}$)

url_X: URL X as an atom (binary attribute $\{0,1\}$)

From the 112 queries, pairwise preferences were extracted according to Algorithm 1 described in Section 2.2. In addition, 50 constraints were added for each clicked-on document indicating that it should be ranked higher than a random other document in the candidate set V. While the latter constraints are not based on user feedback, they should hold for the optimal ranking in most cases. These additional constraints help stabilize the learning result and keep the learned ranking function somewhat close to the original rankings.

Figure 4 shows the predictive performance of the Ranking SVM. To produce the graph, the full data set is split randomly into a training and a test set. The x-axis shows the number of training queries. The y-axis shows the percentage of pairwise preference constraints that are not fulfilled in the test set. Each point is an average over 10 (5-20 training queries) / 20 (40-80 training queries) different test/training splits. When training the Ranking SVM, no kernel was used and C, the trade-off between training error and margin, was selected from $C \in \{0.001, 0.003, 0.005, 0.01\}$ by minimizing leave-one-out error on the training set. The graph shows that the Ranking SVM can learn regularities in the preferences. The test error decreases to around 10%. The graph also shows the number of constraints violated by the rankings produced by Google and MSNSearch. Their error rates are substantially larger than for the learned retrieval function.

These results provide a first proof of concept and justify a larger-scale experiment with multiple users. In particular, while the offline experiment verifies that the Ranking SVM can learn to predict the preference constraints, it is not clear whether the learned retrieval function does improve the retrieval quality objectively. This question is addressed by the following experiment.

5.3 Interactive Online Experiment

To show that the learned retrieval function improves retrieval, the following online experiment was conducted. Starting on October 31^{st}, 2001, the Striver search engine was made available to a group of approximately 20 users. The group consisted of researcher and students of the AI unit at the University of Dortmund headed by Prof. K. Morik. They were asked to use Striver just like they would use any other WWW search engine. By November 20^{th}, the system had collected 260 training queries (with at least one click). On these queries, the Ranking SVM was trained using the same $\Phi(q, d)$ and the same general setup as described above. The learned function was then implemented in Striver and used for ranking the candidate set V. During the evaluation period lasting until December 2^{nd}, the learned retrieval function is compared against:

- Google
- MSNSearch
- Toprank: A baseline meta-search engine that ranks links retrieved at rank 1 by either Google, MSNSearch, Altavista, Excite, or Hotbot, before links ranked 2, before those ranked 3 etc.

The different strategies are compared using the method described in Section 5.1. The learned retrieval strategy is presented in combination with one of the three baseline rankings selected at random. Table 2 shows for how many queries users click on more/less links from the top of the learned retrieval function. The first line of the table compares the learned retrieval function with Google. On 29 queries, the users click on more links from the learned function, on 13 queries they click on more links from Google, and on 27+19 queries they click on an equal number (or none). Using a two-tailed binomial sign test, the difference is significant at a 95%-level, leading to the conclusion that the learned retrieval function is better than that of Google for this group of users. The same applies to the other two comparisons.

weight	feature
0.60	query_abstract_cosine
0.48	top10_google
0.24	query_url_cosine
0.24	top1count_1
0.24	top10_msnsearch
0.22	host_citeseer
0.21	domain_nec
0.19	top10count_3
0.17	top1_google
0.17	country_de
...	
0.16	abstract_contains_home
0.16	top1_hotbot
...	
0.14	domain_name_in_query
...	
-0.13	domain_tu-bs
-0.15	country_fi
-0.16	top50count_4
-0.17	url_length
-0.32	top10count_0
-0.38	top1count_0

Table 3: Features with largest and smallest weights as learned from the training data in the online experiment.

5.4 Analysis of the Learned Function

The previous result shows that the learned function improves retrieval. But what does the learned function look like? Is it reasonable and intuitive? Since the Ranking SVM learns a linear function, one can analyze the function by studying the learned weights. Table 3 displays the weights of some features, in particular, those with the highest absolute weights. Roughly speaking, a high positive (negative) weight indicates that documents with these features should be higher (lower) in the ranking.

The weights in Table 3 are reasonable for this group of users. Since many queries were for scientific material, it appears natural that URLs from the domain "citeseer" (and the alias "nec") received positive weight. The most influential weights are for the cosine match between query and abstract, whether the URL is in the top 10 from Google, and for the cosine match between query and the words in the URL. A document receives large negative weights, if it is not ranked top 1 by any search engine, if it not in the top 10 of any search engine (note that the second implies the first), and if the URL is long. All these weights are reasonable and make sense intuitively.

6. DISCUSSION AND RELATED WORK

The experimental results show that the Ranking SVM can successfully learn an improved retrieval function from clickthrough data. Without any explicit feedback or manual parameter tuning, it has automatically adapted to the particular preferences of a group of ≈ 20 users. This improvement is not only a verification that the Ranking SVM can learn using partial ranking feedback, but also an argument for personalizing retrieval functions. Unlike conventional search engines that have to "fit" their retrieval function to large and therefore heterogeneous groups of users due to the cost of manual tuning, machine learning techniques can improve retrieval substantially by tailoring the retrieval function to small and homogenous groups (or even individuals) without prohibitive costs.

While previous work on learning retrieval functions exists (e.g. [10]), most methods require explicit relevance judgments. Most closely related is the approach of Bartell et al. [2]. They present a mixture-of-experts algorithms for linearly combining ranking experts by maximizing a different rank correlation criterion. However, in their setup they rely on explicit relevance judgments. A similar algorithm for combining rankings was proposed by Cohen at al. [6]. They show empirically and theoretically that their algorithm finds a combination that performs close to the best of the basic experts. The boosting algorithm of Freund et al. [9] is an approach to combining many weak ranking rules into a strong ranking functions. While they also (approximately) minimize the number of inversions, they do not explicitly consider a distribution over queries and target rankings. However, their algorithm can probably be adapted to the setting considered in this paper. Algorithmically most closely related is the SVM approach to ordinal regression by Herbrich et al. [12]. But, again, they consider a different sampling model. In ordinal regression all objects interact and they are ranked on the same scale. For the ranking problem in information retrieval, rankings need to be consistent only within a query, but not between queries. This makes the ranking problem less constrained. For example, in the ranking problem two documents d_i and d_j can end up at very different ranks for two different queries q_k and q_l even if they have exactly the same feature vector (i.e. $\Phi(\mathrm{q}_k, \mathrm{d}_i) = \Phi(\mathrm{q}_l, \mathrm{d}_j)$). An elegant perceptron-like algorithm for ordinal regression was recently proposed by Crammer and Singer [8]. An interesting question is whether such an online algorithm can also be used to solve the optimization problem connected to the Ranking SVM.

Some attempts have been made to use implicit feedback by observing clicking behavior in retrieval systems [5] and browsing assistants [17][20]. However, the semantics of the learning process and its results are unclear as demonstrated in Section 2.2. The commercial search engine "Direct Hit" makes use of clickthrough data. The precise mechanism, however, is unpublished. While for a different problem, an interesting use of clickthrough data was proposed in [3]. They use clickthrough data for identifying related queries and URLs.

What are the computational demands of training the Ranking SVM on clickthrough data? Since SVM^{light}[15] solves the dual optimization problem, it depends only on inner products between feature vectors $\Phi(\mathrm{q}, \mathrm{d})$. If these feature vectors are sparse as above, SVM^{light}can handle millions of features efficiently. Most influential on the training time is the number of constraints in Optimization Problem 2. However, when using clickthrough data, the number of constraints scales only linearly with the number of queries, if the number of clicks per query is upper bounded. In other applications, SVM^{light}has already showed that it can solve problems with several millions of constraints using a regular desktop computer. However, scaling to the order of magnitude found in major search engines is an interesting open problem.

7. CONCLUSIONS AND FUTURE WORK

This paper presented an approach to mining logfiles of WWW search engines with the goal of improving their retrieval performance automatically. The key insight is that such clickthrough data can provide training data in the form of relative preferences. Based on a new formulation of the learning problem in information retrieval, this paper derives an algorithm for learning a ranking function. Taking a Support Vector approach, the resulting training problem is tractable even for large numbers of queries and large numbers of features. Experimental results show that the algorithm performs well in practice, successfully adapting the retrieval function of a meta-search engine to the preferences of a group of users.

This paper opens a series of question regarding the use machine learning in search engines. What is a good size of a user group and how can such groups be determined? Clearly, there is a trade-off between the amount of training data (ie. large group) and maximum homogeneity (ie. single user). Is it possible to use clustering algorithms to find homogenous groups of users? Furthermore, can clickthrough data also be used to adapt a search engine not to a group of users, but to the properties of a particular document collection? In particular, the factory-settings of any off-the-shelf retrieval system are necessarily suboptimal for any particular collection. Shipping off-the-shelf search engines with learning capabilities would enable them to optimize (and maintain) their performance automatically after being installed in a company intranet.

However, the algorithm is not limited to meta-search engines on the WWW. There are many situations where the goal of learning is a ranking based on some parameter (e.g. query). In particular, most recommender problems can be cast in this way. Particularly interesting in the recommender setting is the fact that the Ranking SVM can be trained with partial preference data. For example, consider a recommender system that aims to learn my TV watching preferences. By observing my "channel surfing" behavior, the system can infer which shows I prefer over other programs at a particular time. But again, this is a relative preference, not a preference on an absolute scale.

Open questions regarding the algorithm itself concern its theoretical characterization and its efficient implementation. Is it possible to prove generalization bounds based on the margin? Or, even more interesting from a practical point of view, is it possible to analyze the process of multiple learning/feedback steps in the sense of an incremental online algorithm? As elaborated before, there is a dependence between the links presented to the user, and those for which the system receives feedback. It would be interesting to explore active learning ideas to optimize this feedback. In this framework it might also be possible to explore mechanisms that make the algorithm robust against "spamming". It is currently not clear in how far a single user could maliciously influence the ranking function by repeatedly clicking on particular links. Regarding algorithms for solving the optimization problem, it seems likely that they can be further sped up, since the constraints have a special form. In particular, online algorithms would be most appropriate in many application settings.

8. ACKNOWLEDGEMENTS

Many thanks to Prof. Morik and the AI unit at the University of Dortmund for providing their help and the resources for the experiments. Thanks also to Rich Caruana, Alexandru Niculescu-Mizil, Phoebe Sengers, John Kleinberg, and Lillian Lee for helpful discussions, as well as Wim Verhaegh for helping to clarify the connection to recommender systems.

9. REFERENCES

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

[2] B. Bartell, G. Cottrell, and R. Belew. Automatic combination of multiple ranked retrieval systems. In *Annual ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR)*, 1994.

[3] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2000.

[4] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A traininig algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, 1992.

[5] J. Boyan, D. Freitag, and T. Joachims. A machine learning architecture for optimizing web search engines. In *AAAI Workshop on Internet Based Information Systems*, August 1996.

[6] W. Cohen, R. Shapire, and Y. Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 10, 1999.

[7] C. Cortes and V. N. Vapnik. Support–vector networks. *Machine Learning Journal*, 20:273–297, 1995.

[8] K. Crammer and Y. Singer. Pranking with ranking. In *Advances in Neural Information Processing Systems (NIPS)*, 2001.

[9] Y. Freund, R. Iyer, R. Shapire, and Y. Singer. An efficient boosting algorithm for combining preferences. In *International Conference on Machine Learning (ICML)*, 1998.

[10] N. Fuhr. Optimum polynomial retrieval functions based on the probability ranking principle. *ACM Transactions on Information Systems*, 7(3):183–204, 1989.

[11] N. Fuhr, S. Hartmann, G. Lustig, M. Schwantner, K. Tzeras, and G. Knorz. Air/x - a rule-based multistage indexing system for large subject fields. In *RIAO*, pages 606–623, 1991.

[12] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In *Advances in Large Margin Classifiers*, pages 115–132. MIT Press, Cambridge, MA, 2000.

[13] K. Höffgen, H. Simon, and K. van Horn. Robust trainability of single neurons. *Journal of Computer and System Sciences*, 50:114–125, 1995.

[14] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11. MIT Press, Cambridge, MA, 1999.

[15] T. Joachims. *Learning to Classify Text Using Support Vector Machines – Methods, Theory, and Algorithms*. Kluwer, 2002.

[16] T. Joachims. Unbiased evaluation of retrieval quality using clickthrough data. Technical report, Cornell University, Department of Computer Science, 2002. http://www.joachims.org.

[17] T. Joachims, D. Freitag, and T. Mitchell. WebWatcher: a tour guide for the world wide web. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, volume 1, pages 770 – 777. Morgan Kaufmann, 1997.

[18] J. Kemeny and L. Snell. *Mathematical Models in the Social Sciences*. Ginn & Co, 1962.

[19] M. Kendall. *Rank Correlation Methods*. Hafner, 1955.

[20] H. Lieberman. Letizia: An agent that assists Web browsing. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI '95)*, Montreal, Canada, 1995. Morgan Kaufmann.

[21] A. Mood, F. Graybill, and D. Boes. *Introduction to the Theory of Statistics*. McGraw-Hill, 3 edition, 1974.

[22] L. Page and S. Brin. Pagerank, an eigenvector based ranking approach for hypertext. In *21st Annual ACM/SIGIR International Conference on Research and Development in Information Retrieval*, 1998.

[23] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[24] C. Silverstein, M. Henzinger, H. Marais, and M. Moricz. Analysis of a very large altavista query log. Technical Report SRC 1998-014, Digital Systems Research Center, 1998.

[25] V. Vapnik. *Statistical Learning Theory*. Wiley, Chichester, GB, 1998.

[26] Y. Yao. Measuring retrieval effectiveness based on user preference of documents. *Journal of the American Society for Information Science*, 46(2):133–145, 1995.

APPENDIX

THEOREM 1. *Let r_{rel} be the ranking placing all relevant documents ahead of all non-relevant documents and let r_{sys} be the learned ranking. If Q is the number of discordant pairs between r_{rel} and r_{sys}, then the average precison is at least*

$$AvgPrec(r_{sys}, r_{rel}) \geq \frac{1}{R}\left[Q + \binom{R+1}{2}\right]^{-1}\left(\sum_{i=1}^{R}\sqrt{i}\right)^2$$

if there are R relevant documents.

PROOF. If $p_1, ..., p_R$ are the ranks of the relevant documents in r_{sys} sorted in increasing order, then Average Precision can be computed as

$$AvgPrec(\mathrm{r}_{sys}, \mathrm{r}_{rel}) = \frac{1}{R}\sum_{i=1}^{R}\frac{i}{p_i} \tag{24}$$

What is the minimum value of $AvgPrec(\mathrm{r}_{sys}, \mathrm{r}_{rel})$, given that the number of discordant pairs is fixed. It is easy to see that the sum of the ranks $p_1 + ... + p_R$ is related to the number of discordant Q as follows.

$$p_1 + ... + p_R = Q + \binom{R+1}{2} \tag{25}$$

It is now possible to write the lower bound as the following integer optimization problem. It computes the worst possible Average Precision for a fixed value of Q.

$$\text{minimize:} \quad P(p_1, ..., p_R) = \frac{1}{R}\sum_{i=1}^{R}\frac{i}{p_i} \tag{26}$$

$$\text{subject. to:} \quad p_1 + ... + p_R = Q + \binom{R+1}{2} \tag{27}$$

$$1 \leq p_1 < ... < p_R \tag{28}$$

$$p_1, ..., p_R \text{ integer} \tag{29}$$

Relaxing the problem by removing the last two sets of constraints can only decrease the minimum, so that the solution without the constraints is still a lower bound. The remaining problem is convex and can be solved using Lagrange multipliers. The Lagrangian is

$$L(p_1, ..., p_R, \beta) = \frac{1}{R}\sum_{i=1}^{R}\frac{i}{p_i} + \beta\left[\sum_{i=1}^{R} p_i - Q - \binom{R+1}{2}\right]. \tag{30}$$

At the minimum of the optimization problem, the Lagrangian is known to have partial derivatives equal to zero. Starting with the partial derivatives for the p_i

$$\frac{\delta\, L(p_1, ..., p_R, \beta)}{\delta\, p_i} = -i\, R^{-1}\, p_i^{-2} + \beta \doteq 0, \tag{31}$$

solving for p_i, and substituting back into the Lagrangian leads to

$$L(p_1, ..., p_R, \beta) = 2\, R^{-\frac{1}{2}}\beta^{\frac{1}{2}}\sum_{i=1}^{R} i^{\frac{1}{2}} - \beta\left[Q - \binom{R+1}{2}\right]. \tag{32}$$

Now taking the derivative with respect to β

$$\frac{\delta L(p_1, ..., p_R, \beta)}{\delta\, \beta} = R^{-\frac{1}{2}}\beta^{-\frac{1}{2}}\sum_{i=1}^{R} i^{\frac{1}{2}} - \left[Q - \binom{R+1}{2}\right] \doteq 0, \tag{33}$$

solving for β, and again substituting into the Lagrangian leads to the desired solution. □