Collaborative Ranking Function Training for Web Search Personalization

Giorgos Giannopoulos
School of EC Engineering
NTU Athens
Greece
giann@dblab.ece.ntua.gr

Theodore Dalamagas
IMIS Institute
"Athena" Research Center
Greece
dalamag@imis.athena-innovation.gr

Timos Sellis
IMIS Institute
"Athena" Research Center
Greece
timos@imis.athena-innovation.gr

ABSTRACT

In this paper, we present a framework for improving the ranking function training and the re-ranking process for web search personalization. Our method is based on utilizing clickthrough data from several users in order to create multiple ranking functions that correspond to different topic areas. Those ranking functions are combined each time a user poses a new query in order to produce a new ranking, taking into account the similarity of the query with each of the topic areas mentioned before. We compare our method with the traditional approaches of training one ranking function per user, or per group of users and we show preliminary experimental results.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information Search and Retrieval—*Relevance feedback, Search process, Clustering*.

General Terms

Algorithms, Experimentation, Measurement

Keywords

Search engine, ranking, training, clickthrough data, relevance judgement, clustering

1. INTRODUCTION

Lately, a lot of effort has been put in finding ways to personalize web search results. Several models for ranking function training have been proposed [7, 3, 6] and a great number of studies on user search behavior and feedback have been performed [20, 1, 7, 12].

To the best of our knowledge, most approaches of learning ranking functions for search personalization, utilize machine learning techniques that train a ranking function for a user or a group of users with similar search behaviour. So, no matter how diverse the search topics of a user, or a group of users sharing a common ranking function training, are, the re-ranking/personalization of the search results is based on the same model.

However, it is often the case that a user searches in more than one different areas of interest that could lead to completely different ranking function training models. Consider for example the following search scenario. A phd student working on information retrieval (IR) issues would like to search for papers related to clickstream data, machine learning techniques and ranking. Her search is usually based on keywords related to IR. Since ACM is a well-known digital library, she would probably prefer clicking on results from that data source. So, a proper model for this search behavior would train a ranking function that favors results whose titles have high textual similarity with the query and results containing the word "acm" in their title or url.

However, the user would also like to search for information about a new cellphone she would like to buy. She would use the brandname of the cellphone as well as words like "review" or "hands on" as keywords. Also, she would probably click on results coming from forums where users discuss their experience/opinion about the cellphone and/or on video results which would present the phone's functionality. So a model suitable for this kind of search would favor results containing the word "forum/blog" in their url and video results.

Thus, it is evident that training a single ranking function per user or per group of users with similar search behavior does not capture the diversity in topics areas that are of user interest.

Our approach. In this work, we address the aforementioned problem. We present a framework that creates multiple ranking functions, each one corresponding to a different topic area, which are finally combined to produce a final score for each search result. The main idea is that a ranking function is not trained for a single user, neither a group of users with similar search behavior. On the contrary, one ranking function is trained per *topic area*. That is:

1. We gather clickthrough data from several users and extract triplets of the form: (*query*, *result list*, *clicked results list*).
2. Then, we cluster the clicked results based on their tex-

WOODSTOCK '97 El Paso, Texas USA

tual similarity and for each result cluster C_i, we train a different ranking function F_i using Ranking SVM. So, when a user poses a new query, each ranking function F_i will give a different rank r_{ij} for each query result j.

3. We compute the similarity between the query and each cluster C_i, and we represent it with a weight w_i.

4. We then exploit w_is to produce a final ranking list of results to present to the user.

Outline. In Section 2 we discuss the background issues related to ranking function learning. In Section 3, we present our method for adjusting the ranking function training and result re-ranking according to the content of each new query. In Section 4 we present a preliminary experimental evalution. Section 5 presents the related work and, finally, Section 6 conludes and discusses further work.

2. RANKING FUNCTION TRAINING

In this section, we present background information on ranking function training. We outline the steps that comprise this process, including: a) extraction of relevance relations, b) extraction of features, and c) training of ranking function.

Relevance relation extraction. Based on the results of a query, relevance relations can be extracted from either *absolute* or *relative* preferences. An absolute preference suggests that a search result is either relevant or irrelevant to a query. A relative preference suggests that a search result is more relevant to a query than another result.

The extraction of those relations is mainly based on the results the user viewed or clicked. There are many approaches for extracting relevance relations. For instance, in [7, 11], where an approach that uses relative preferences is adopted, relevance relations are extracted by taking into consideration the relative position of a pair of results in the initial ranking. For example, a clicked result c is considered more relevant than all non-clicked results which are placed higher than c in the ranking list. An example of absolute preference is presented in [14]. In this work, if a result is clicked, then it is considered relavant to the query.

Usually, every query-result pair is assigned a score or a label, named its *relevance judgement*, that denotes the degree of relevance between the result and the query. The set of query-result pairs, together with the corresponding relevance judgements, comprise the first component that is input to the training process. The second component involves feature extraction from the query-result pairs.

Feature extraction. Every query-result pair is represented by a feature vector which quantifies the matching quality between the query and the result. There exists a great variety of features that can be used in this process. Content-based features, which can be extracted from the title, the body, the anchor and the url of a result, can be used to estimate content similarity between the query and the result [9]. Some other features are based on hyperlink information (i.e., pagerank values) [2], or on specific information of the results such as the domain of the url or the rank of the result in several search engines [7]. Also, such features may incorporate statistical information over user behaviour, e.g., deviation from average time spent for viewing pages [1].

Ranking function training. Ranking function training aims at assigning a proper weight for each feature used in feature vectors. Those weights indicate which features are more important for ranking the search results in the context of the particular training process. Several training methods have been proposed, such as Ranking SVM [7], RankNET [3], RankBoost [6] as well as tuning methods for the aforementioned techniques [4].

Figure 1 shows the overall training process. Once the ranking function is trained, the results for each new query are re-ranked according to scores produced by the ranking function. Those scores are calculated utilizing the weights found in the previous step and the feature vector of each new search result. In the example, three labels are used for relevance judgement: *irrelevant*, *partially relevant* and *strongly relevant* results respectively. For each query-result pair viewed or clicked by the user, these relevance judgements can be extracted implicitly as described previously in this section.

We should note that, regardless of the approach adopted, there has to be a specific representation of the relevance relations, so that they can be used as input to a ranking function training system. In this paper, we adopt the representation format of SVM$^{\text{light}}$[1], which is a popular tool that implements Ranking SVM [7]. Also, we should point out that our method sticks to the process described above, proposing a different perspective in the training and re-ranking phases.

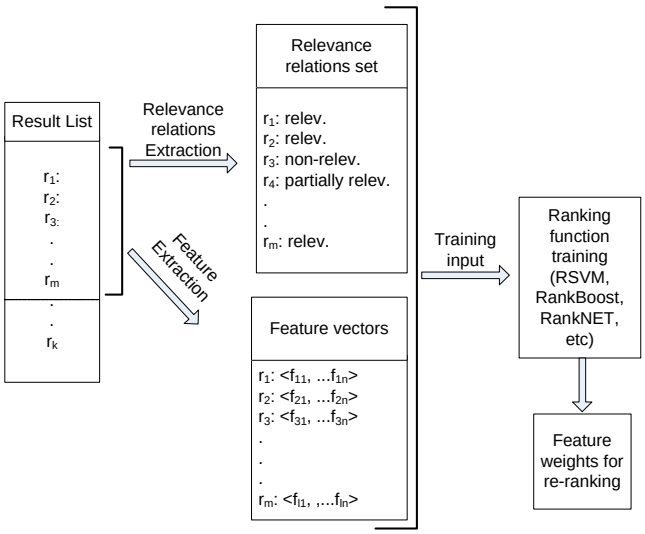

Figure 1: Ranking function training.

3. COLLABORATIVE TRAINING

In our method, we consider clickthrough data of the form: (*query*, *result list*, *clicked results list*). We are based on Joachims model [7] according to which a clicked result is more relevant to the query than all non-clicked results in higher rank. These relevance relations produce relevance judgments to train a ranking function and adapt ranking to user needs.

Our aim is to have multiple ranking functions, each one trained per *search topic*. We identify search topics by clus-

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

tering user clickthrough data, and more specifically results clicked by users. Our approach involves the following steps:

- First, we perform clustering on the clicked results of all queries posed by users. We choose to cluster the clicked results instead of the queries to capture user behaviour because we consider them more informative about the user information need. According to the well known example, if a user searches for "jaguar", some results will be related to the animal and some others to the car. If we use the query text and cluster the queries, the information about which of these two concepts the user is actually interested in will be lost. On the contrary, if we cluster the clicked results using, i.e., title and abstract text, we will be able to capture her actual interest.

  After the clustering is performed, groups of results with similar content are formed. We call these groups *topic clusters*.

- We need a mechanism to calculate the similarity of every new query with each of the extracted clusters. For this reason, we use titles and abstracts of all (clicked) results belonging to a cluster as the textual representation of this cluster. We build an inverted file index to be able to calculate the similarity of the query with the textual representation of each cluster.

- We need to create one ranking function model per topic cluster. To achieve that, we use as training input only clickthrough data related to the corresponding topic cluster.

- This final step deals with the re-ranking of the results, utilizing the clusters created previously. When a new query is posed, we re-rank its results using all available ranking function models producing N different rankings, where N the number of topic clusters. In order to compose the final ranking, we combine those rankings taking into account the similarity of the query with each topic cluster.

Next, we describe in detail the aforementioned steps.

3.1 Clustering of Search Results

Consider a feature space ϕ of n terms, $\phi = \{t_1, t_2, \ldots, t_n\}$, where n is the total number of distinct terms in all clicked results for all queries of all users. We represent each result by a feature vector $v_i = \{wtd_{i1}, wtd_{i2}, \ldots, wtd_{in}\}$, where $wtd_{ik} = tf_{ik} * log(N/df_k)$, tf_{ik} is the frequency of occurrence of term t_k in document i, N is the number of results, and df_k is the number of results that contain the term t_k. We should note that, in our approach, we take into consideration the title and the abstract of each result in order to extract those features.

After extracting the features, we cluster the results having similar content. We use a partitional clustering method that utilizes repeated bisections [16, 17]. This method has been shown to have excellent performance when dealing with document collections [16]. In the following, we give an overview of the clustering method.

All documents (i.e., search results) are initially partitioned into two clusters (i.e., bisected). Then, one of these clusters is selected and is further bisected. This process is repeated until we get the desired number of clusters. In each step, the selection of the cluster to be bisected and the bisection itself, is done in such a way that the bisection optimizes the value of a clustering criterion function.

The criterion function used to form the clusters aims at maximizing the following quantity:

$$\sum_{i=1}^{k} \sqrt{\sum_{v,u \in S_i} sim(v,u)}$$

where k is the number of clusters, S_i is the set of documents of cluster i, and $sim(v,u)$ is the similarity value between documents v and u in S_i.

We employ the cosine similarity as the metric that compares two documents. We apply the cosine similarity function on the documents' feature vectors as shown in the following Equation:

$$sim(v,u) = \frac{\sum_{i=1}^{n}(wtd_{vi} \times wtd_{ui})}{\sqrt{\sum_{i=1}^{n} wtd_{vi}^2} \times \sqrt{\sum_{i=1}^{n} wtd_{ui}^2}}$$

Following this process, we obtain N *topic clusters*, with each one containing similar clicked results.

3.2 Cluster Indexing

Given a topic cluster C_i, we extract the text from the titles and abstracts of all the results it contains. We regard this text as the cluster's textual representation. In this way, we obtain a collection of "documents" representing the topic clusters. Then, we build an inverted file index on these documents using the Lucene[2] IR engine. The index is then used to calculate the similarity of any new query (see Section 3.4) with each cluster:

1. We pose the query to the Lucene search engine.
2. The search engine uses its own scoring function to calculate the simirarity of each indexed document with the query.
3. A list of documents is returned, along with a score that indicates how similar to the query they are.
4. The scores are normalized so that the sum of all scores equals to 1.
5. We assign each normalized document score to the corresponding topic cluster.

3.3 Ranking function training

For every extracted topic cluster C_i we train a different ranking function F_i as described in Section 2. The training is based on Joachims' model [7, 11] using ranking SVMs. Relevance judgments and feature vectors are used as input to the SVMs. The relevance judgements are produced by the users' clicks. To construct feature vectors, we have implemented the following features:

1. Textual similarity between query and (title, abstract, URL) of the result. The similarity is computed with 3 different types: tfidf, BM25 and Lucene scoring function. This results to 9 different similarity features.
2. Domain of the result (.com, .edu, etc): 73 boolean features (i.e., exist or not).

[2]http://lucene.apache.org/

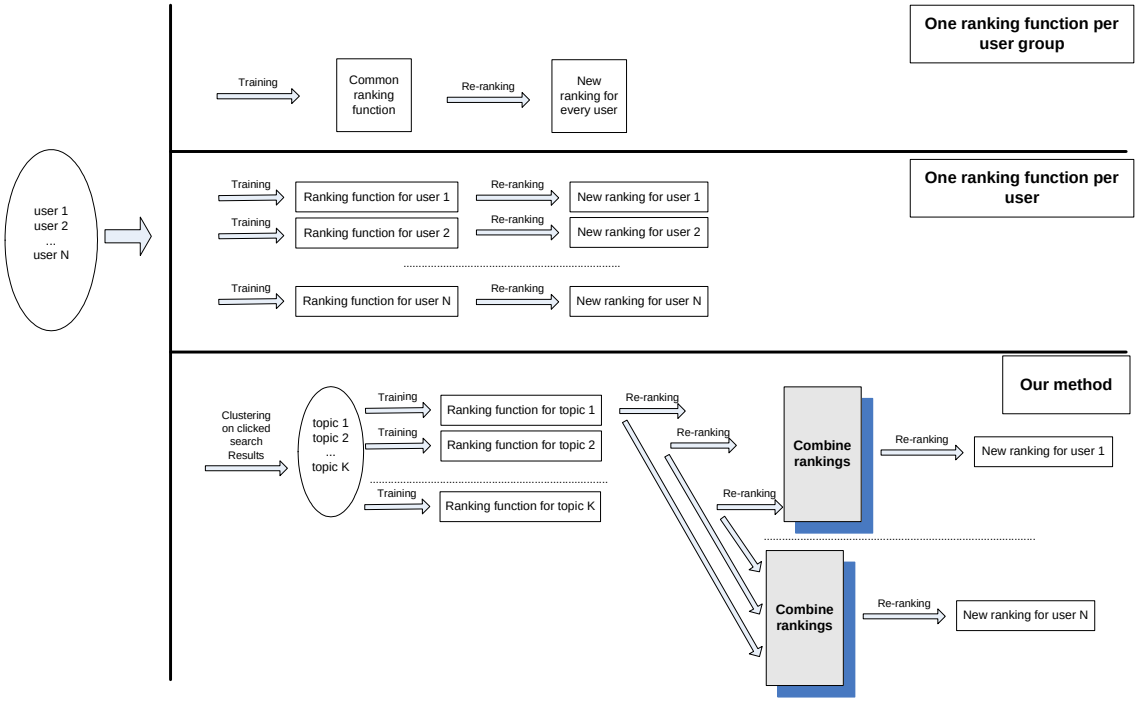

Figure 2: Training and re-ranking approaches

3. Rank of the result in Google search engine.

4. Special words found such as "blog", "forum", "wiki", "portal", etc, found in the result's title, abstract or url. Each word corresponds to 3 feature values. These values depend on the similarity scores of this word on the result's title, abstract and url.

5. URL suffix (.html, .pdf, .ppt): boolean features.

6. 100 most frequent words found in all result documents of all previous searches. Each word corresponds to 3 feature values. These values depend on the similarity scores of this word on the result's title, abstract and url.

As training input for the Ranking SVM, we use only clickthrough data related to the corresponding topic cluster. That is, we regard only query-result pairs, where results belong to the same cluster. This process results to the creation of N ranking function models, where N is the number of topic clusters.

3.4 Re-Ranking

All steps described in the previous sections regard the training phase. This section deals with re-ranking which is performed after training is over. Re-ranking involves the following issues:

- When a user poses a new query q, its similarity score w_{qi} with every topic cluster C_i is calculated as described in Section 3.2.

- Then, using each ranking model F_i, we produce N different rankings R_{qi} for query q (r_{qij} the rank of result j) according to model F_i, corresponding to cluster C_i.

- The final rank for each result j for query q is given by the type:

$$rank(q, j) = \sum_{i=1}^{N} w_{qi} r_{qij}$$

4. EXPERIMENTAL EVALUATION

In this section, we present a preliminary experimental evaluation of our method. First, we present our experimental dataset.

4.1 Dataset

In order to obtain clickthrough data, we set up a logging mechanism over Google Search Engine. The application is able to keep track of the queries posed by the users, the result list returned from Google (i.e., title, abstract and URL), and the result URLs that were clicked by the users. Also, it records auxiliary information, such as (a) the IPs, so that we are able to distinguish the users, and (b) the exact date and time of queries and result clicks, in order to be able to divide the clickthrough dataset into training and test dataset.

We asked from 10 users, phd students and researchers from our lab, to search on Google for information relevant to a given set of topics for a two-months period. The search topics were the following: *Gadgets, Cinema, Auto & Moto, Life & health*, and *Science*. The users were asked to select 1 to 3 out of these topics and focus their search mainly on those, without, however, to restrict them from searching for information relevant to the other topics. During this period of time, we got clickthrough data for 671 queries. We used 75% of clickthrough data as the training dataset (501 queries), and the 25% as the test dataset (170 queries).

4.2 Preliminary Results

We evaluate the effectiveness of the following approaches for training ranking functions:

T1: Training one ranking function per user.

T2: Training one ranking function per group of users who are expected to have similar search behaviors.

T3: Collaborative ranking function training (our approach).

Our dataset is based on implicit user feedback, and not on data explicity judged. Thus, there are only a few relevance judgements for the results of each query. So, it was not possible to use evaluation metrics such as Precision and Mean Average Precision, since those metrics require a great number of query results in the test dataset to have relevance judgements. In our case, for most queries we have judgements (i.e., clicks by the users) for 1 or 2 results.

However, what we were able to do, was to compare the ranks coming from the different ranking function training approaches. Specifically, we executed 3 rounds of ranking function training and re-ranking (one for each approach T1, T2 and T3), and compared the ranks of clicked results in the new re-ranked result lists.

The results are presented in Table 1. Since our method depends on clustering quality, we also give results obtained varying the number of clusters created (see Section 3.1). The values presented in column "T3-T1" ("T3-T2") are the average differences in the ranks of the clicked results in the new re-ranked result lists produced by approaches T1 and T3 (T2 and T3). For example, the value 3 shows that our approach T3 moves the clicked results 3 positions higher, on average, compared to T1.

num of clusters	T3-T1	T3-T2
5	3	27
10	-14	11
15	-11	10
20	-9	15
25	-12	12

Table 1: Average differences of clicked results ranks in the new re-ranked result lists

Determining the proper clustering arrangement, we can improve the re-ranking process for personalization of search results. We can see that for the first clustering arrangement (5 clusters) our method outperforms the other two.

Some interesting results are presented in Table 2 where each cell shows the percentage of clicked results belonging to each cluster for each user. For example, 31% of results clicked by user 2 belong to the topic area represented by cluster 1.

We observe that there are users who searched only in one topic area (users 1 and 10), users who searched mainly in two areas (users 2, 5 and 7) and users who searched in many areas (users 3, 4, 6, 8 and 9). This observation supports our intuition that we should train and use more than one ranking models even for the same user.

We also observe that users 1, 3, 4 and 7 dedicated a respectable percent of their searches in the topic area represented by cluster 4, and, similarly, users 5, 7, 8 and 10 for cluster 5, and users 5, 6, 8 and 9 for cluster 3. This supports our intuition that we should train and use ranking models by combining clickthrough data from more than one users.

Finally, we should note here that we do not aim at comparing the efficiency of different training and ranking models, but at examining how our method improves the effectiveness of a given model (in our case Ranking SVM), by creating multiple ranking functions and combining them according to the user search needs.

5. RELATED WORK

In what follows, previous approaches regarding the problem of increasing the quality of ranking function training and re-ranking are presented. In [18] the authors utilize concept hierarchies like ODP[3] to categorize queries and build user profiles. They, then, use collaborative filtering techniques to re-rank query results based on those profiles.

The approaches described in [12] and [10] are based on rank promotion of results having low rank. The initial ranking presented to users is slightly altered using several strategies, so that more results are judged by the users. As a result, a better ranking function training is achieved.

Finally, clustering is exploited in SVM Ranking methods, but to achieve different goals. In [8], clustering of results is utilized in order to refine a large training data set, by rejecting those data that are not necessary for the SVM training phase. Also, in [5], clustering is performed on the ranking functions learned from training data taken from different users. In [19] we used clustering to increase the training input, inferring relevance judgements for unjudged results.

6. CONCLUSION

In this paper, we presented a methodology for improving the quality of ranking function training. Performing clustering on clickthrough data involving search results clicked by the users, we find groups of similar results that represent topic areas. Based on those groups, we train multiple ranking functions each one corresponding to a different topic area, which are finally combined to produce a final score for each search result.

The intuition behind our method is that a user may exhibit more than one search behaviors, which cannot be represented by a single ranking model. However, if we exploit multiple instances of a ranking model, that are collaboratively trained for all users, better results can be achieved. Thus, we suggest an approach where one ranking function is trained per topic area, contrary to approaches where a ranking function is trained per user, or for a group of users with similar search behavior. The experiments show that our approach gives better results compared to those approaches.

Our work is in progress and this is a first-cut approach to this problem with preliminary experimental results. Evidently, there is room for improvements and expansions. First, we plan to perform more extended experiments with larger datasets in order to obtain more reliable results and study the effects of clustering in the topic area detection. We also plan to study whether classification techniques using pre-defined concept hierarchies (ODP) can help in detecting more appropriate topic areas. Finally, we will study more sophisticated mechanisms for inferring topic areas. Now, we cluster clicked results based on their textual content similarity. We believe that we could achieve better results by

[3]http://www.dmoz.org/

user	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
1	0	0	0	100	0
2	31	67	0	2	0
3	6	11	16	67	0
4	11	48	0	41	0
5	0	0	32	4	64
6	10	4	72	7	7
7	2	0	4	24	70
8	9	7	24	5	55
9	27	4.5	55	4.5	9
10	2	2	2	0	94

Table 2: Percentage of clicked results belonging to each cluster (topic area) for each user.

performing clustering exploiting the similarity values of feature vectors for each query-result pair.

7. REFERENCES

[1] E. Agichtein, E. Brill, and S. Dumais. Improving web search ranking by incorporating user behavior information. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 19–26, 2006.

[2] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30:107–117, 1998.

[3] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96, 2005.

[4] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon. Adapting ranking svm to document retrieval. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 186–193, 2006.

[5] J. Diez, J. J. del Coz, O. Luaces, and A. Bahamonde. Clustering people according to their preference criteria. *Expert Systems with Applications: An International Journal*, 34:1274–1284, 2008.

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

[7] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, 2002.

[8] X. Li, N. Wang, and S.-Y. Li. A fast training algorithm for svm via clustering technique and gabriel graph. In *Proceedings of the International Conference on Intelligent Computing*, 2007.

[9] 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 on Learning to Rank for Information Retrieval*, 2007.

[10] S. Pandey, S. Roy, C. O. J. Cho, and S. Chakrabarti. Shuffling a stacked deck: the case for partially randomized ranking of search engine results. In *Proceedings of the 31st international conference on Very large data bases*, pages 781–792, 2005.

[11] F. Radlinski and T. Joachims. Query chains: Learning to rank from implicit feedback. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 239–248, 2005.

[12] F. Radlinski and T. Joachims. Active exploration for learning rankings from clickthrough data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 570–579, 2007.

[13] S. E. Robertson. Overview of the okapi projects. *Journal of Documentation*, 53(1):3–7, 1997.

[14] G.-R. Xue, H.-J. Zeng, Z. Chen, Y. Yu, W.-Y. Ma, W. Xi, and W. Fan. Optimizing web search using web click-through data. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 118–126, 2004.

[15] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 334–342, 2001.

[16] Y. Zhao and G. Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning*, 55(3):311–331, 2004.

[17] Y. Zhao, G. Karypis, and U. Fayyad. Hierarchical clustering algorithms for document datasets. *Data Mining and Knowledge Discovery*, 10(2):141–168, 2005.

[18] U. Rohini and V. Ambati. Improving Re-ranking of Search Results Using Collaborative Filtering. *Information Retrieval Technology, Third Asia Information Retrieval Symposium, AIRS 2006*, pages 205–216, 2006.

[19] G. Giannopoulos, T. Dalamagas, M. Eirinaki and T. Sellis. Boosting the ranking function learning process using clustering. *10th ACM International Workshop on Web Information and Data Management (WIDM 2008)*, pages 125–132, 2008.

[20] S. Fox, K. Karnawat, M. Mydland, S. Dumais and T. White. Evaluating implicit measures to improve web search. *ACM Transactions on Information Systems (TOIS)*, 23(2):147–168, 2005.