Learning to Rank Collections

Jingfang Xu
Department of Electronic Engineering
Tsinghua University
Beijing, 100084, China
Xjf02@mails.tsinghua.edu.cn

Xing Li
Department of Electronic Engineering
Tsinghua University
Beijing, 100084, China
xing@cernet.edu.cn

ABSTRACT
Collection selection, ranking collections according to user query is crucial in distributed search. However, few features are used to rank collections in the current collection selection methods, while hundreds of features are exploited to rank web pages in web search. The lack of features affects the efficiency of collection selection in distributed search. In this paper, we exploit some new features and learn to rank collections with them through SVM and RankingSVM respectively. Experimental results show that our features are beneficial to collection selection, and the learned ranking functions outperform the classical CORI algorithm.

Categories and Subject Descriptors
H.3.3 [**Information Storage and Retrieval**]: Information search and Retrieval

General Terms: Algorithms

Keywords: Distributed Search, Collection Selection

1. INTRODUCTION
Distributed information retrieval, searching multiple collections in distributed environments, is an efficient way to integrate the information in deep web automatically. Given a query, a distributed search engine ranks the underlying collections and selects some of them to forward the query (*collection selection*) based on their content summaries (*resource representation*), and finally merges the results returned from the selected collections (*results merging*). Similar to page ranking in general web search, e.g., Google, collection selection plays a vital role in distributed search. Considerable efforts have been devoted to collection selection algorithms, e.g., CORI[2], ReDDE[6], and hierarchical database selection algorithm[4]. The features used by these algorithms are limited to terms' document frequencies and collection size, which is far different from web search scenario. The lack of features affects the performance of collection selection. On the other hand, many machine learning based ranking algorithms have been proposed in recent years, such as RankingSVM[5], RankNet[1], and RankBoost[3]. Instead of tuning ranking function empirically, these methods construct ranking functions through supervised learning. However, to the best of our knowledge, none of them has been used in distributed search.

In this paper, 20 features, including content, link, popularity, etc., are exploited for collection selection. Moreover, we learn to combine these features to rank collections through SVM and RankingSVM respectively. The learned ranking functions are evaluated on real web data. Experimental results show that our features are beneficial to collection selection, and the learned ranking functions outperform the classical CORI algorithm. The rest of the paper is organized as follows. First we propose some new features and learn to rank collections in section 2. Then section 3 describes the experiments and results. Finally we conclude in section 4.

Copyright is held by the author/owner(s).
SIGIR'07, July 23-27, 2007, Amsterdam, The Netherlands.
ACM 978-1-59593-597-7/07/0007.

2. LEARNING to RANK COLLECTIONS

2.1 Features
As only document frequencies and collection size are exploited in previous collection selection methods, more features are needed to represent collections exactly. This paper studies 20 features, including static features (query-independent) and dynamic features (query-dependent). While the static features represent the characteristic of a collection, the dynamic features describe the relationship between the collection and the query, shown in Table 1. In dynamic features, inter-anchors are the anchor texts from web pages in different collections, and intra-anchors are that from web pages in the same collection. Integrating the information of all the collections, we can acquire the information for inter-anchors. All the dynamic features have two values: absolute value, normalized by the total number, and relative value, normalized by the biggest value across collections. Both the static features and dynamic features can be exported initiatively by the collections in cooperative environments, or be estimated through query based sampling in uncooperative environments.

2.2 Ranking
To combine the features, we employ RankingSVM and SVM to learn a ranking function respectively, which ranks collections according to queries. For each query, every collection is represented as a vector of features, and it is classified into five categories: "very bad", "bad", "indifferent", "good", and "very good", according to the relevance between collection and query. With RankingSVM, ranking is viewed as ordinal regression, where the cost of misclassifying a good instance into "bad" is larger than that of misclassifying into "indifferent". With SVM, ranking is viewed as classification problem, which classifies collections into different categories for each query. Ignoring indifferent, SVM considers "good" and "very good" as positive, and considers "bad" and "very bad" as negative. For each query, the learned ranking function assigns a score to each collection, which is used to rank the collections.

3. EXPERIMENTAL RESULTS
We build a distributed search engine, *Net Compass Confederation* [1], which includes a broker and 50 site search

[1] Net Compass Confederation, http://www.compass.edu.cn

engines indexing 50 real Chinese web sites. 200 queries are randomly selected from the query log. For each query, volunteers are asked to label the collections as "very bad", "bad", "indifferent", "good", or "very good". The labeled data of 150 queries is used as training set, and the rest is used to test. For the 50 queries in testing set, collections are ranked according to the scores calculated with the learned ranking function.

Table 1: Features

Static Features
Size
the number of documents in the collection
PageRank
the pagerank score of the collection's homepage, reported by Google
Popularity
the network traffic of the collection, reported by Alexa
Homepage Length
the number of terms in the collection's homepage
Dynamic Features
Document Frequency
the number of documents containing the query
Title Document Frequency
the number of documents whose title contains the query
Inter-anchor Document Frequency
the number of inter-anchors containing the query
Intra-anchor Document Frequency
the number of intra-anchors containing the query
Term Frequency in Homepage
the occurring times of the query term in collection's homepage
Term Frequency in Homepage Title
the occurring times of the query term in the title of collection's homepage
Term Frequency in Inter-anchors
the occurring times of the query term in inter-anchors
Term Frequency in Intra-anchors
the occurring times of the query term in intra-anchors

Kendall's τ distance, measuring the similarity between two ordered lists, is adopted to evaluate the performance of ranking functions. Let r_a be the list of collections ordered by human, where "good" collections are in front of "bad" ones. Let r_b be the list of collections ordered by ranking functions. In r_a, there is a strict ordering between each pair of collections in different categories. Then, for each pair with strict ordering in r_a, it is concordant if their ordering in r_b agrees with that in r_a, and it is discordant otherwise. Let P be the number of concordant pairs and Q be the number of discordant pairs, and then the Kendall's τ distance of r_a and r_b is formulated as $(P-Q)/(P+Q)$.

The ranking functions, learned through SVM and RankingSVM, are evaluated with the testing set. In addition, the classical CORI collection selection algorithm[2] is used as a baseline. Comparing to the labeled data, the Kendall's τ distances of the three ranking functions are shown in Table 2. As we can see, both the functions learned by SVM and RankingSVM are better than the baseline, CORI. The ranking function learned through RankingSVM also outperforms the function learned through SVM, which is due to the information of the ordering between different categories.

In addition, in order to investigate the importance of features, we calculate the weights of features in the linear model learned by RankingSVM, as shown in Figure 1. Feature 1-4 are static features, feature 5-12 are absolute values of dynamic features, and features 13-20 are relative values of dynamic features. As we can see, feature 5, 13, absolute and relative values of document frequency are the most important features, followed by feature 14, the relative value of title document frequency. Generally, the relative values of dynamic features are more important than the absolute ones. Among static features, size and PageRank take a little advantage over popularity, while HomePage length is almost useless. To sum up, though the most important features have been used in previous collection selection methods, there are other beneficial features, such as relative document frequency and title document frequency, PageRank, and popularity, can be exploited to improve the performance of collection selection.

Table 2: Performance of ranking functions

Algorithms	Kendall's τ
CORI(baseline)	0.514
Learn by SVM	0.551
Learn by RankingSVM	0.586

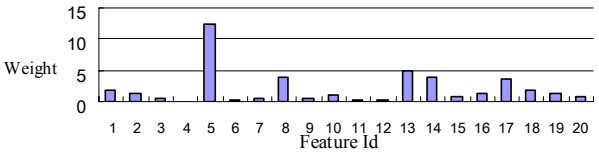

Figure 1: Weights of Features

4. CONCLUSIONS

In this paper, we propose some new features for collection selection in distributed search. In addition, SVM and RankingSVM are adopted to construct ranking functions combining all the features. Both of the ranking functions are evaluated on real web data. Experimental results show that both our functions with these new features outperform the classical CORI algorithm. It indicates that our features are beneficial to ranking collections, and the learning based ranking functions combine these features effectively.

5. REFERENCES

[1] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proceedings of ICML'05*, 2005, 89-96

[2] J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *Proceedings of SIGIR '95*, 1995, 21-28.

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

[4] P. G. Ipeirotis and L. Gravano. Distributed search over the hidden web: hierarchical database sampling and selection. In *Proceedings of VLDB '02*, 2002.

[5] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of SIGKDD '02*, 2002, 133-142.

[6] L. Si and J. Callan. Relevant document distribution estimation method for resource selection. In *Proceedings of SIGIR'03*, 2003, 289-305.