

Metasearch and Federation using Query Difficulty Prediction

Elad Yom-Tov, Shai Fine, David Carmel, Adam Darlow
IBM Haifa Research Labs
Haifa 31905, Israel
{yomtov,fshai,carmel,darlow}@il.ibm.com

ABSTRACT

Fusing the results of a distributed Information Retrieval system has often been viewed as two separate problems: metasearch, where several search engines are retrieving from one document collection, and federation, where one search engine is retrieving from several document collections. In this work we present a unified framework for fusion based on predicting query difficulty. We validate our approach using a rigorous, quantitative, experimental procedure, which uses Desktop Search Engines and a standard document collection. We show that the proposed algorithm performs significantly better than previously-proposed algorithms for both metasearch and federation.

1. INTRODUCTION

The most commonly analyzed scenario for information retrieval is that of a single search engine retrieving documents from a single document collection. This scenario can be extended to handle more complex systems that utilize multiple search engines retrieving documents from a single document collection, or a single search engine retrieving documents from a plethora of document collections. These scenarios, known as **fusion**, require that the final result of the retrieval effort be a single, unified ranking of documents, based on several ranked lists. The first scenario with multiple engines searching a single collection is known as Metasearch, while the latter, with a single engine searching multiple collections is known as Federation.

In the information retrieval realm, the process of fusion is usually divided into three phases: collection (or engine) selection, number of document selection, and merging [16]. The idea in collection/engine selection is to narrow down the queried datasets to the most relevant collections, thus reducing the amount of noise present. Number of document selection is the process of deciding how many documents should be retrieved from each collection, the simplest being an identical number of documents. Finally, merging is the process of generating a unified list from the retrieved lists of

documents.

Metasearch is of major interest when retrieving information from the Internet. There are many successful, general purpose, search engines over the Web such as Google¹, MSN², Yahoo!³. It is desirable to have a single interface submit a query to many of these and return a unified document list based on the ranked lists returned by these engines. This is preferred over requiring the user to search each engine separately and form a unified list manually. The metasearch engine is not required to maintain a search index of the documents. Instead it submits queries to the underlying search engines and recombines the returned results. The goal of a metasearch engine is to first select which search engines to query, and then to form a combined list from the retrieved lists such that the documents it contains are more relevant than any single list from the underlying engines. See Dreilinger & Howe [7] for an overview of metasearch engines on the Web.

Federation of document collections is addressed mainly in the context of retrieving information from a multitude of specialized datasets, using a single search engine. Maintaining the document sets in several datasets, and handling a separate search index for each one of them, is usually required for scalability and security issues. Other reasons prohibiting centralization of the data are legal constraints, geographical limitations (data transfer is forbidden), etc. In such a model, searching each dataset would result in a ranked list of documents. The task of the federation engine in this case is, as in metasearch, to select the datasets to query, and to output a unified ranked list of results.

1.1 Related Work

Aside from the document ranking, most search engines provide very little information with which to perform the merging. The document score (DS) assigned by a search engine to a document retrieved from a collection may or may not be provided. When scores are not provided, only the document ranking and some a priori knowledge about the datasets can be used for merging the different result sets. Metacrawler [18] and Borda-Fuse [1] utilize document rank and its appearance in the results list of several engines to perform merging. This is done by summing the ranks of the document in the different ranked lists.

When document scores are given they can be used as additional information for merging. However, it is difficult to

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.

SIGIR'05, August 15–19, 2005, Salvador, Brazil.

Copyright 2005 ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

¹<http://www.google.com>

²<http://www.msn.com>

³<http://www.yahoo.com>

re-rank the documents since DSs are local for each specific dataset and engine combination. This can be avoided by computing global scores, based on global statistics of the query terms, as though all datasets were merged to a single collection [23]. Another approach is to map the scores returned by each of the search engines to relevance probabilities [15].

Fox and Shaw [9] proposed several combination techniques, including setting the score of each document to the sum of the scores obtained by the individual search engines (COMBSUM) or by multiplying this sum by the number of engines which have non-zero scores (COMBNZ). Lee [14] extended those techniques by normalizing each engine on a per query basis. He observed that the best combination obtained when systems retrieved similar sets of relevant documents and dissimilar sets of non-relevant documents.

Other approaches to merging were suggested by assigning a weight to every ranking, such that each collection is given a score based on its statistics. This score is then used for merging the different rankings by weighting the DSs. Vogt et al. [21] used a linear combination of the DSs, where the linear weights are constant for the search engines, and are learned during a training session. Two known state-of-the-art algorithms that use this approach are CORI [2], which is applicable to the framework of federation, and ProFusion [11], created for metasearch. CORI computes a query dependent score for each dataset. It requires, in addition to DSs, some statistics for each query term from each dataset. ProFusion creates an engine-specific weight by measuring the precision of each search engine over a known set of predefined queries.

An alternative model, Bayes-Fuse [1], learns the evidence of relevance as a distribution of relevance given the search engines' rankings. The final ranked list is obtained using Bayes optimal decision rule. A more sophisticated model for combining rankings using conditional probabilities was introduced in [13], where models of permutation were used in order to combine rankings. Cohen et al. [6] show how metasearch can be formulated as an ordering problem, and present an on-line algorithm for learning a weighted combination of ranking systems which is based on an adaptation of Freund and Schapire's Hedge algorithm [10].

Recently, Joachims [12] demonstrated a user-driven approach to metasearch. His system learns user preferences based on past activity and assigns weight to individual search engines. Thus, this system is similar to ProFusion, the main difference being that weights are assigned based on the preference of an individual user (or group of users) rather than search engine precision.

1.2 Our Approach

In this paper we present a novel algorithm for performing both metasearch and federation using query difficulty prediction, a recently developed technique for predicting the quality of retrieved results for a given combination of query and search engine [24]. Our observations show that queries that are answered well by search engines are those whose query terms agree on most of the returned documents. Agreement is measured by the overlap between the top results for the full query and the top results for each of the query terms. Difficult queries are those where either the query terms cannot agree on top results or most of the terms do agree beside a few outliers. This is usually the case where

the query contains one rare term that is not representative of the whole query and the rest of the query terms appear together in many irrelevant documents.

Our method learns to predict query difficulty based on the overlaps between the results of the full query and its sub-queries. The predictor is induced from training queries and their associated relevance sets. The approach we describe is based on the assumption that only minimal information is supplied by the search engine operating on a specific dataset, namely, the ranking of documents, the inverse document frequency (idf) of all query terms, and possibly their scores (i.e., the DSs). Given this data, our approach is to train a query predictor for each dataset and search engine combination. When a query is executed, the ranked list of documents is returned from each dataset and search engine combination and the prediction of query difficulty is computed for each. The predicted difficulty is used for weighting the DS of the results and the final ranking is built by merging the lists using these weighted DSs. Thus the method we propose sets a query-by-query weight for each search engine and document collection pair.

In the following we present a novel experimental procedure which enables rigorous, quantitative, testing of distributed information retrieval algorithms. We use the same document collection for both federation and metasearch. Federation is performed by splitting the collection to four sub-collections which are accessed separately by a single search engine. Metasearch is performed using publicly available desktop search engines. These engines, which are designed to help personal computer users find information on their desktop computers, search over the full collection, and the proposed algorithm is used to merge their outputs. The experimental results show that in the federation scenario our algorithm significantly and consistently improves system's precision compared to simple merging and CORI. In the metasearch experiment, we show improvement of 7% over the best single search engine, and 12% over other metasearch strategies.

The paper is organized as follows. Section 2 describes the method used for query prediction. Section 3 presents our metasearch and federation experiments, and Section 4 discusses our results.

2. PREDICTION OF QUERY DIFFICULTY

Recently, a learning algorithm for predicting query difficulty was proposed [24]. The algorithm was tested for a single search engine. In the course of our experiments, detailed below, we were able to show that the algorithm can be trained to predict query difficulty for the fusion case as well. As detailed below, several simplifications have been made to the original algorithm. We found that these modifications do not change the quality of the prediction given sufficient training data.

2.1 The prediction algorithm

The idea employed by the method demonstrated in [24] is to learn a meta-statistic from the contribution of each query term to the final result set⁴. This information is then used

⁴In essence, this method is reminiscent of committee-based approaches commonly used to address many machine learning problems, where each query term is identified as a committee member.

to predict the query difficulty as measured by the precision at 10 (P@10), by the mean average precision (MAP), and by the number of queries with no relevant results in the top 10 results (%no) [22]. In the context of query difficulty prediction, we define query terms as the keywords (i.e., the words of the query, after discarding stop-words) and the *lexical affinities*, which are closely related query terms found in close proximity to each other [4]. The features used for the predictor learning are:

1. The overlap between each sub-query (a query based on one query term) and the full query. The overlap between two queries is defined as the size of the intersection between the top N results of the two queries. Thus, the overlap is in the range of $[0, N]$. Section 2.2 explains the rationale for using this feature.
2. The rounded logarithm of the document frequency (the number of documents containing the term), $\log(DF)$, of each of the sub-queries.

Learning to predict query difficulty using the aforementioned data presents two challenges. The first is that the number of sub-queries is not constant, and thus the number of features for the predictor varies from query to query. In contrast, most techniques for prediction are based on a fixed number of features. The second problem in learning to predict is that the significance of the sub-queries is not given, so any algorithm that performs a comparison or a weighted average on an ordered feature vector is not directly applicable.

As noted above, each query may generate a variable number of sub-queries. Thus, there is a need to represent the data in a canonical representation. A histogram is used for this in the following manner:

1. Find the top N results for the full query and for each of the sub-queries.
2. Build a histogram of the overlaps. Denote this histogram by $h(i)$, $i = 0, 1, \dots, N$. Entry $h(i)$ counts the number of sub-queries that agree with the full query on exactly i documents in the top N results. Unless otherwise stated, we used $N=10$.
3. Predict query difficulty by multiplying the histogram h , by a linear weight vector c such that $Pred = c^T \cdot h$. The method by which this weight vector is computed is described below.

This algorithm can be improved significantly by using both the overlaps and $\log(DF)$ of the terms as features. For canonical representation we split $\log(DF)$ into three discrete values⁵ $0 - 1, 2 - 3, 4+$.

In this case, the algorithm is modified so that in Step (1), a two-dimensional histogram is generated, where entry $h(i, j)$ counts the number of sub-queries with $\log(DF) = i$ and j overlaps with the full query. For example, suppose a query has four sub-queries, an overlap vector $ov(n) = [2 \ 0 \ 0 \ 1]$ and a corresponding $\log(DF)$ vector $\log(DF(n)) = [0 \ 1 \ 1 \ 2]$. The two-dimensional histogram for this example would be:

$$h(i, j) = \begin{matrix} & \begin{matrix} 0 & 1 & 2 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \end{matrix} & \begin{pmatrix} 0 & 0 & 1 \\ 2 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{matrix}$$

⁵In cases where the DF of a term is zero, we take $\log(DF) = 0$.

Before Step (3), the histogram is made into a vector by concatenating the lines of the histogram, corresponding to the overlap histogram at each $\log(DF)$ of the term, one after the other. In the example above, the corresponding vector for the linear predictor is $h(i) = [0 \ 0 \ 1 \ 2 \ 0 \ 0 \ 0 \ 1 \ 0]$;

An additional feature that was found to be advantageous to the success of the predictor is the number of words in the query. We concatenated this feature to the histogram data and the reported results use this feature.

The linear weight vector can be predicted in several ways, depending on the objective of the predictor. If the object of the predictor is to predict the P@10 or MAP of a query, a logical error function would be the Minimum Mean Square Error (MMSE), in which case the weight vector is computed using the Moore-Penrose pseudo-inverse[8]:

$$c = (H \cdot H^T)^{-1} \cdot H \cdot t^T \quad (1)$$

where H is a matrix whose columns are the histogram vectors h for the training queries, computed as described above, and t is a vector of the target measure (P@10 or MAP) for those queries. However, the objective might also be to rank the queries according to their expected P@10 or expected MAP by maximizing Kendall's- τ [20] between predicted and actual order. In this case, a more suitable optimization strategy is to modify the above equation as suggested in [12]. The idea is that a linear weight vector will maximize Kendall's- τ , if for each query i which is ranked higher than query j , we enforce the constraint $c^T \cdot h_i > c^T \cdot h_j$. This constraint can be rewritten in the form: $c^T(h_i - h_j) > 0$. In practice, it is better to modify the constraint such that we demand that $c^T(h_i - h_j) \geq 1$ in order to fix the scaling of c [17]. Therefore, instead of using the histogram vectors h directly, the matrix H is modified to a matrix whose columns are the differences between the histogram vectors. For each pair of training queries (i, j) the k -th column is:

$$H_k = h_i - h_j \quad \forall i, j = 1, 2, \dots, N_r \quad (2)$$

where N_r is the number of training queries. In this case, the target vector is modified as follows:

$$t_k = \begin{cases} +1 & \text{if } t_i > t_j \\ -1 & \text{if } t_j \leq t_i \end{cases} \quad \forall i, j = 1, 2, \dots, N_r \quad (3)$$

2.2 Overlaps as κ -statistics

The overlap between a sub-query and the full query is a measure of agreement between these two queries. The κ -statistics [5], a standard measure for the strength of agreement between two experts, is predicted as follows:

$$\kappa = \frac{AG_{Observed} - AG_{Chance}}{1 - AG_{Chance}} \quad (4)$$

where $AG_{Observed}$ is the fraction of cases where both experts agreed on the outcome, while AG_{Chance} is the fraction of cases they would agree upon if both were making random decisions.

In the case of two queries, the agreement matrix is shown in Table 1, where N_D is the number of documents in the collection, and Ov is the overlap (the number of documents that both queries agree should be in the top 10 results).

		Query 1		Total
		Top 10	Not top 10	
Query 2	Top 10	Ov	$10 - Ov$	10
	Not top 10	$10 - Ov$	$N_D - (20 - Ov)$	$N_D - 10$
Total		10	$N_D - 10$	N_D

Table 1: Agreement matrix for two queries.

Based on this table, the κ -statistic is [5]:

$$\kappa = \frac{\frac{Ov + (N_D - (20 - Ov))}{N_D} - \left[\left(\frac{10}{N_D} \right)^2 + \left(\frac{N_D - 10}{N_D} \right)^2 \right]}{1 - \left[\left(\frac{10}{N_D} \right)^2 + \left(\frac{N_D - 10}{N_D} \right)^2 \right]} \quad (5)$$

In this case the κ -statistic is a linear function of the overlap. Thus, the query predictor learns a meta-statistic over the κ -statistic.

2.3 Use of query difficulty prediction for metasearch and collection federation

Using the methods described in the previous sections, once a query is submitted to a search engine, its difficulty can be predicted. We note that such prediction should be performed separately for each search engine or document collection, on a query-by-query basis. In the federation case, the overlap between the results of the full query and the results of its sub-queries can be measured efficiently during query execution, since all data needed can be generated by the search engine during its normal mode of operation. For example, the top results for each of the sub-queries can be accumulated simultaneously during evaluation of the full query. However, in the metasearch case, several more queries are required to be submitted to each search engine. A separate query for each query term is needed for calculation of its overlap with the full query results. In both cases, the computational load required for difficulty prediction is negligible.

The query difficulty prediction, computed for each dataset / engine, is then used to form a unified ranked list of results. Document ranks or, if available, the document scores are weighted in each ranked document list, by the query difficulty score they have been assigned. The top N documents are then taken as the unified ranked list.

3. EXPERIMENTS

We conducted two sets of experiments. The first tested the use of difficulty prediction in the framework of metasearch, with several search engines retrieving from the same document collection. The second experiment was in the framework of federation, where one search engine retrieves data from several indices, and each index represents a different document collection.

The data for the experiment comprised of the TREC-8 collection (528,155 documents, 249 topics) [22]. This is a natural benchmark because it is a well studied collection for which there are a large number of topics (249) and corresponding relevant documents. Each topic comprises of a short query based on the topic title and a longer query based on the topic description. An additional benefit of this collection is that it is built from four distinct sub-collections, which enabled us to use it for both parts of the experiment. Three-fold cross-validation was used to assess the algorithm.

3.1 Metasearch

The use of the TREC-8 collection and the queries based on its topics allowed us to make a quantitative comparison of different metasearch algorithms. In order to test our algorithm for metasearch, several search engines are required to be run on the TREC-8 document collection. We used a novel method for performing this by using several publicly available desktop search engines.

Desktop search engines are a relatively new class of applications that attempt to retrieve documents saved on a user's computer. Just a year ago, desktop search was only available from a few companies, such as Copernic⁶. This year, all major Web search companies have entered the desktop search world, providing beta releases of desktop search for free. We downloaded eight of these and attempted to index the document collection. Most of the search engines could not index the whole collection, and thus we reduced the requirement to use only the LA-TIMES dataset, which is comprised of approximately 140,000 documents. This was still prohibitively high for four of the search engines, and we thus remained with four working search engines: Google, MSN, Yahoo!, and HotBot. We note that except for Yahoo!, all search engines provide sorting results by relevance. Yahoo! could only sort them by date.

After indexing was completed, we submitted the 249 short (title) queries to each search engine and collected the top 10 results from each. We then repeated the process for each word in the query. This was done so as to enable the computation of the overlap. In this experiment, only document ranking (not document scores) and term idf's were available for merging the results.

Since three-fold cross-validation was used, two thirds of the data was used to train a query predictor used for metasearch over the remaining third. The predictor attempted to predict the P@10. Recall that MAP could not be computed since only the top 10 documents from each search engine were collected. We discovered that better results were obtained by optimizing the predictors to minimum mean square error rather than to Kendall's- τ , probably due to the discrete nature of the P@10.

If all engines used very similar search algorithms, the documents returned by them would be very similar and the metasearch algorithm would not be able to improve significantly upon the individual search engines. Therefore, we tested how the desktop search engines agreed on the results set by measuring the overlap (As defined above) between them on their top 10 results. A dendrogram was drawn based on the average overlap, as shown in Figure 1. Recalling that nodes in a dendrogram are linked higher if they are less similar to each other, this figure shows that even the most similar of the two engines disagreed on approximately 50% of the results, and the highest dissimilarity was approximately 85%. Thus, it appears that there is ample dissimilarity between the desktop engines for the metasearch algorithm to work on.

After running each query through each of the desktop search engines, we weighted the ranking (1 through 10) of the documents by the prediction computed for this query. The documents were then pooled and sorted according to their combined score. The top 10 were taken as the result of the prediction-based metasearch algorithm. If a document

⁶<http://www.copernic.com>

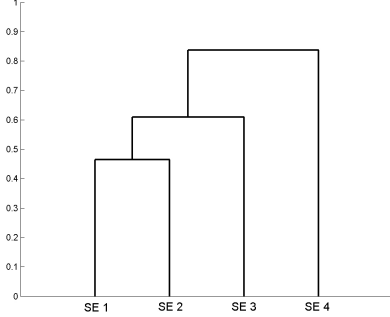


Figure 1: Similarity in the results of the four desktop search engines.

Table 2: Results of the different metasearch strategies on LA-TIMES collection, using the topic titles as queries.

		P@10	% NO
SINGLE SEARCH ENGINE	SEARCH ENGINE 1	0.139	47.8
	SEARCH ENGINE 2	0.153	43.4
	SEARCH ENGINE 3	0.094	55.2
	SEARCH ENGINE 4	0.171	37.4
MERGE METHOD	ROUND-ROBIN	0.164	45.0
	METACRAWLER	0.163	34.9
	Prediction based	0.183	31.7

was retrieved by more than one search engine, its highest combined score was used.

The results of the prediction-based metasearch algorithm are compared to two other algorithms: Round-Robin and MetaCrawler [18]. The first generates the combined list of documents by taking the first ranked documents from each search engine, then the second ranked documents, etc. The first 10 unique documents are returned as the combined list. In contrast, MetaCrawler assigns new weights to documents by summing the reciprocal rank (i.e., 10 minus the rank) of the top documents in each search engine. Thus, a document ranked high by two or more engines is weighted higher.

The results of the experiment are depicted in Table 2. They show an improvement (in P@10) of 7% over the best single search engine and of 12% over the other metasearch strategies. The improvement of the proposed method over both every single engine and the two metasearch strategies is statistically significant (two-tailed paired t-test, $p < 0.02$ for all methods and single collections). By contrast, the two alternative metasearch algorithms performed lower on average than (and not significantly different from) the best single search engine.

Note that all the desktop engines that participated in the experiment are still in their beta stage. This might explain the poor effectiveness they all demonstrate on the TREC data. This is also the reason for our decision not to expose the particular results of each of the participants. However, in the context of metasearch, this experiment demonstrates that by clever merging, we are able to significantly improve the results of several non-effective search engines.

3.2 Federation

The TREC-8 collection is comprised of four different sub-collections: FBIS, FR94, FT, and LA-TIMES. Thus, this document collection could be used for federation by indexing each of the sub-collections separately using the Juru search engine [3]. A query predictor was then trained for each of these collections.

During the federation experiment, either one or both parts of the TREC topic (the short titles and the longer descriptions) were used, thus generating a total of 498 queries. Given a query, the Juru document score (DS) returned for each document from its corresponding collection was weighted by the prediction of that collection. The final document ranking was generated by sorting the pool of retrieved, weighted, documents.

Our results, depicted in Table 3, demonstrate that merging methods all provide a large gain in performance over the single document collections. This difference is statistically significant (two-tailed paired t-test) at $p < 10^{-3}$. However, using the query predictor achieved significant improvement in results compared to those obtained by both simple merging and CORI (two-tailed paired t-test, $p < 10^{-3}$ for both methods).

Some previously proposed algorithms attempted to solve the federation problem by attempting to approximate it to searching the collections as though they were merged into a single collection [19]. Thus, it is interesting to compare the results of the prediction-based federation algorithm to those of the search engine working on the entire TREC-8 collection.

The results of this comparison are shown in Table 4. This table and a two-tailed paired t-test show that the prediction-based federation algorithm achieved significantly better MAP, but lower P@10, compared to unfederated retrieval ($p < 10^{-5}$ for both comparisons). Thus, the proposed federation algorithm might be preferred not only in a distributed scenario, but also when the document collection can be split naturally into separate sub-collections. The lower P@10 obtained by the prediction-based federation algorithm is due to the fact that the predictor was, in our case, trained to predict the MAP and not the P@10.

Table 4: Comparison of searching the whole TREC-8 collection (Unfederated) and prediction-based federation, using different topic parts as queries.

TOPIC PART	METHOD	P@10	MAP	% NO
TITLE (249 QUERIES)	UNFED.	0.437	0.257	8.4
	FED.	0.363	0.266	15.5
DESCRIPTION (249 QUERIES)	UNFED.	0.435	0.253	12.8
	FED.	0.377	0.319	13.2
TITLE AND DESCRIPTION (498 QUERIES)	UNFED.	0.435	0.255	10.6
	FED.	0.373	0.295	14.9

4. SUMMARY

Traditionally, distributed Information Retrieval has been divided into two separate problems: Federation (one search engine searches many collections), and metasearch (many engines search one collection). This work suggests a novel, unified, approach to fusion of search results, which deals with both scenarios. Our fusion method is based on pre-

Table 3: Results of the different federation strategies on TREC-8 collection, using different topic parts as queries.

TOPIC PART			P@10	MAP	% NO
TITLE (249 QUERIES)	DOCUMENT COLLECTION	FBIS	0.176	0.158	51.8
		FR94	0.070	0.103	75.1
		FT	0.262	0.255	32.1
		LA-TIMES	0.256	0.242	28.9
	MERGE METHOD	UNWEIGHTED DS	0.338	0.237	20.9
		CORI	0.347	0.243	17.7
		Prediction based	0.363	0.266	15.5
DESCRIPTION (249 QUERIES)	DOCUMENT COLLECTION	FBIS	0.158	0.153	53.8
		FR94	0.071	0.103	75.5
		FT	0.275	0.251	24.9
		LA-TIMES	0.239	0.225	29.3
	MERGE METHOD	UNWEIGHTED DS	0.359	0.287	15.3
		CORI	0.375	0.294	14.1
		Prediction based	0.377	0.319	13.2
TITLE AND DESCRIPTION (498 QUERIES)	DOCUMENT COLLECTION	FBIS	0.167	0.156	52.8
		FR94	0.071	0.103	75.3
		FT	0.269	0.253	28.5
		LA-TIMES	0.247	0.233	29.1
	MERGE METHOD	UNWEIGHTED DS	0.348	0.262	18.1
		CORI	0.361	0.268	15.9
		Prediction based	0.373	0.295	14.9

dicting the difficulty of a given query for a specific collection/engine, and weighting the retrieved results accordingly. Thus, expected poor results contribute less to the final merged list. Experiments with several desktop search engines searching over TREC data, and a single search engine searching over sub-collections of TREC, showed significant improvement over state-of-the-art fusion techniques both for metasearch and federation.

Our approach can be easily extended to the many-to-many case, in which many engines search over many collections of documents. In that case, a difficulty predictor should be trained for each engine-collection pair. Given a query submitted to many engines searching many collections, any result set retrieved by any pair will be weighted according to the associated prediction. Actually this is the case in metasearch on the Web in which each search engine indexes a different fraction of the Web (with some overlap). Experiments with prediction-based fusion approach for the many-to-many case is left for future work.

Training a difficulty predictor for a given engine-collection pair requires training data based on a large set of queries, each associated with its own relevance set of results. In previous work it was shown that the quality of prediction strongly depends on the amount and the quality of the training data. Accumulating training data is not an easy task. However, in previous work [24] it was demonstrated that a predictor learned over one collection-engine pair can be useful, even though it is less accurate, for other collection-engine pairs. Thus, a predictor trained for a specific pair, can be used for other pairs. The effect on performance in such a scenario is also left for future work.

5. REFERENCES

- [1] J. A. Aslam and M. Montague. Models for metasearch. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 276–284. ACM Press, 2001.
- [2] J. Callan, Z. Lu, and W. Croft. Searching distributed collections with inference networks. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 21 – 28, Seattle, Washington, 1995.
- [3] D. Carmel, E. Amitay, M. Herscovici, Y. S. Maarek, Y. Petruschka, and A. Soffer. Juru at TREC 10 - Experiments with Index Pruning. In *Proceeding of Tenth Text REtrieval Conference (TREC-10)*. National Institute of Standards and Technology. NIST, 2001.
- [4] D. Carmel, E. Farchi, Y. Petruschka, and A. Soffer. Automatic query refinement using lexical affinities with maximal information gain. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 283–290. ACM Press, 2002.
- [5] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, pages 37–46, 1960.
- [6] W. W. Cohen, R. E. Schapire, and Y. Singer. Learning to order things. In *NIPS '97: Proceedings of the 1997 conference on Advances in neural information processing systems 10*, pages 451–457. MIT Press, 1998.
- [7] D. Dreilinger and A. Howe. Experiences with selecting search engines using meta-search. *ACM Transactions on Information Systems*, 15(3):195222, 1997.
- [8] R. Duda, P. Hart, and D. Stork. *Pattern classification*. John Wiley and Sons, Inc, New-York, USA, 2001.
- [9] E. A. Fox and J. A. Shaw. Combination of multiple searches. In E. M. Voorhees and D. Harman, editors, *Overview of the Second Text REtrieval Conference (TREC-2)*, pages 243–249, 1994.
- [10] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT '95: Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37. Springer-Verlag, 1995.
- [11] S. Gauch, G. Wang, and M. Gomez. Profusion:

- Intelligent fusion from multiple, distributed search engines. *Journal of Universal Computing*, 2(9):637–649, 1996.
- [12] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*. Association of Computer Machinery, 2002.
- [13] G. Lebanon and J. D. Lafferty. Cranking: Combining rankings using conditional probability models on permutations. In *Proceedings of the 19th International Conference on Machine Learning*. Morgan Kaufmann Publishers, 2002.
- [14] J. H. Lee. Analyses of multiple evidence combination. In *Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 267–276. ACM Press, 1997.
- [15] R. Manmatha, T. Rath, and F. Feng. Modeling score distributions for combining the outputs of search engines. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 267–275. ACM Press, 2001.
- [16] W. Meng, C. Yu, and K.-L. Liu. Building efficient and effective metasearch engines. *ACM Computing Surveys*, 34(1):48–89, 2002.
- [17] B. Scholkopf and A. Smola. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT Press, Cambridge, MA, USA, 2002.
- [18] E. Selberg and O. Etzioni. Multi-service search and comparison using the MetaCrawler. In *Proceedings of the 4th International World-Wide Web Conference*, Darmstadt, Germany, December 1995.
- [19] L. Si and J. Callan. Using sampled data and regression to merge search engine results. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 19–26. ACM Press, 2002.
- [20] G. Upton and I. Cook. *Oxford dictionary of statistics*. Oxford university press, Oxford, UK, 2002.
- [21] C. Vogt, G. W. Cottrell, R. Belew, and B. Bartell. Using relevance to train a linear mixture of experts. In E. M. Voorhees and D. Harman, editors, *Overview of the Fifth Text REtrieval Conference (TREC-5)*, pages 503–515, 1997.
- [22] E. M. Voorhees. Overview of the trec 2003 robust retrieval track. In *Proceedings of the Twelfth Text Retrieval Conference (TREC-12)*. National Institute of Standards and Technology (NIST), 2003.
- [23] J. Xu and D. Lee. Cluster-based language models for distributed retrieval. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 254–261, Berkeley, California, 1999.
- [24] E. Yom-Tov, S. Fine, D. Carmel, A. Darlow, and E. Amitay. Juru at TREC 2004: Experiments with Prediction of Query Difficulty. In *Proceeding of the 13th Text REtrieval Conference (TREC-2004)*. National Institute of Standards and Technology. NIST, 2004.