Juru at TREC 2005: Query Prediction in the Terabyte and the Robust tracks

Elad Yom-Tov, David Carmel, Adam Darlow,
Dan Pelleg, Shai Errera-Yaakov, Shai Fine
IBM Haifa Research Lab
Haifa 31905, Israel
Email: {yomtov,carmel,darlow,dpelleg,shaie,fshai}@il.ibm.com

Abstract

Our experiments focus this year on the ad-hock tasks of the Terabyte and the Robust tracks. In both tracks we experimented with the query prediction technology we developed recently. In the Terabyte track, we investigated how query prediction can be used to improve federation of search results extracted from several indices. We show that federated search based on query prediction can achieve comparable results to single-index search. In the Robust track we trained a predictor over one collection (TREC-8) for predicting query difficulty over another collection (AQUAINT). The experimental results show that difficult topics on the TREC-8 collection were not found to be consistently difficult on the AQUAINT collection.

1 Introduction

Our experiments focused this year on the ad-hock tasks of the Terabyte and the Robust tracks. In both tracks we experimented with the query prediction technology we developed recently [7, 6]. In the Terabyte track, we investigated how query prediction can be used to improve federation of search results extracted from several indices. A predictor is learned for each index, based on last year's topics for training. When a (new) query is executed, a ranked list of documents is returned from each index and the prediction of query difficulty is computed for each result set. The predicted difficulty is used for weighting the scores of results and the final ranking is built by merging the lists using these weighted scores.

In addition, we experimented with two issues related to distributed search. The first is whether federated search over distributed index (a set of indices, each represents a sub-collection), can be comparable in terms of effectiveness to searching over one single index representing the whole collection. The second is whether the partitioning policy of the data to the distributed sub-collections affects the effectiveness of federated search. For the first question, we compared query-prediction based federated search over distributed index with search over one single equivalent index. For the second question, we experimented with several partition schemes, testing how they affect search results.

In the Robust track, one of the tasks is to predict the queries average precision. We trained our predictor on the TREC-8 collection using the old 249 TREC topics, and used the predictor over the AQUAINT collection for which no training data was given. Our hypothesis was that the TREC based predictor can predict the query difficulty on different collection than the collection it was trained for. In addition, following the good results obtained by several groups last year using web expansion, we upgraded our system to benefit web expansion based on the *answers.com* web search engine[1]. *answers.com* is a free search service, providing instant answers over many topics. Our expansion procedure worked by first submitting the topic title to *answers.com*, and then using the result page for query expansion.

The rest of the paper is organized as follows: Section 2 describes the distributed search over .gov2 collection, and compares federated search with single-index search. Section 3 describes our experiments in the Robust track. Finally, Section 4 concludes.

2 Terabyte track: Federated versus single-index search

2.1 Distributed search over the gov2 collection

Our experiments with federation required a baseline run. This baseline run should optimally have been a single search index representing the entire collection. However, our search engine Juru, at the time of experimentation, was not able to index the entire collection into one single index in reasonable time. Therefore, we used a distributed search framework in order to simulate a single search index. The distribution was done over a partition of the gov2 collection into ten equal disjoint parts.

The indexing process

Each document in the collection was indexed according to its textual content and according to its anchor data which includes all page's in-link anchors. A special anchor database was created that maps every page in the collection to its anchor data. We used the following process to collecting anchors:

1. Scan the .gov2 collection page by page and collect all anchor tuples (url, anchor, link-type) e.g. *(www.nasa.gov, "Nasa Home page", different-host-type)*.
2. Distribute the anchor tuples into 53 disjoint files according to the url's first characters. Filter out all urls linking outside the collection.
3. Sort each of the files according to the urls.
4. Collect all anchors related to a specific url and add them to the Anchor database.

The anchor database includes 19 million records, i.e. 76% of the pages have at least one anchor. The average number of anchors per page is 10, however there are few pages with more than 1,000 anchors, and even one page with 680,000 anchors. The database was used through the indexing process to extract the page's anchors to be concatenated with its textual content.

The page's tokens were marked according to their type (regular, headline, title, different-host-anchor, same-host-anchor). These types were associated with different

[1] http://www.answers.com

boosts during query execution time. A title token contributes three times more than a regular token to the document's score. In addition, every page was associated with a static score reflecting its authority. The page's static score was computed based on square root of the number of in-links pointing to the page. At query time, the textual score of the page is linearly combined with the page's static score to yield its final score.

The indexing process took 96 hours on a single machine (1 CPU, 2.4GH, 2GB physical memory). As already mentioned, following our system limitations, we partitioned the data into ten disjoint indices.

The search process

The distributed search algorithm has a two phase query process. In the first phase, the query retrieves all relevant term statistics, e.g. terms' document frequency, from each of the indices. These statistics are combined in order to create global statistics. These global statistics are sent back to the indices for a second phase which consists of the actual ranking of documents. The retrieved results from all of the indices can then be merged easily because all of their scores were calculated based on the same global statistics. In this approach, distributed search is equivalent to a single-index search.

Average query time was still relatively high with the distribution algorithm. Therefore we experimented with a pruning algorithm. Pruning was done using the WAND algorithm [1]. WAND is a document-at-a-time algorithm based on a two level approach: at the first level, it iterates in parallel over query term postings and identifies candidate documents using an *approximate evaluation* taking into account only partial information on term occurrences and no query independent factors; at the second level, promising candidates are *fully evaluated* and their exact scores are computed. The efficiency of the evaluation process is improved significantly using dynamic pruning techniques with very little cost in effectiveness. The amount of pruning can be controlled by the user as a function of time allocated for query evaluation.

In our experiment we tested the effect of the two stage evaluation process by comparing two runs. One run did pruning as described above and the other did no pruning at all. Interestingly, the results shown in Table 1 demonstrate that the run which did pruning achieved slightly better results than the run which did no pruning. This indicates that not only did the pruning reduce execution time by a factor of three, it was actually able to filter out some noisy documents which the normal ranking algorithm ranked too highly. A possible explanation stems from the fact that the much simpler scoring model used in the first phase does not take into account many document features such as occurrence counts, document length and link analysis. Therefore, it is not sensitive to documents that are noisy in relation to such features.

			2004		2005	
Run name	**Pruning**	**Time (sec)**	**MAP**	**P@10**	**MAP**	**P@10**
DF0	No	1.71	0.292	0.522	0.284	0.520
DF1	Yes	0.53	0.273	0.529	0.285	0.536

Table 1. Distributed search results. Average query execution time is measured over the 50,000 queries provided for the Terabyte track's efficiency task.

2.2 Federated search using query prediction

Using federated information retrieval on a large collection such as .gov2 requires two additional processing stages during indexing and retrieval. Before indexing, it is first required to partition the collection into smaller sub-collections, each of which are indexed separately. During retrieval, each sub-collection is queried separately and so it is necessary to federate the results into a coherent result list.

We experimented mainly with different partition schemes, which we detail below. Federation was based on query prediction methods we have previously developed [7, 6], although these were further developed as we describe in the remaining paragraphs.

Partitioning schemes

Our initial hypothesis was that the precise scheme used to partition the documents into indices might have an effect on retrieval performance. We identified several straightforward partitioning schemes, described in detail below. Due to the volume of the .gov2 collection, we decided to apply and evaluate the different schemes on the smaller .gov collection. The different schemes and identifiers follow.

random This scheme places each document in a random partition ("bin").

domain The document's URL is stripped to its two top-level domains. Then, a hash function maps each stripped URL to one of the bins. The hash functions guarantees that two identical strings are always mapped to the same bin. But it does not guarantee a unique bin to each string (and this constraint is impossible to meet, since the number of documents is several orders of magnitudes greater than the number of bins).

domain* One bin is dedicated to the `nasa.gov` documents (150K documents), and another to `noaa.gov` documents (102K documents). The other documents are placed in the remaining bins as in domain.

rand-ts Documents with no meaningful title string are placed into their own bin. Such documents have typically empty (or all-whitespace) titles, or titles such as: "untitled document", "image", "ppt", "you are now leaving the ...", etc. Additionally, another bin is dedicated to documents from large groups which all share the same title. Such are the entire domains `gcmd2.gsfc.nasa.gov` and `www.fleets.doe.gov`, as well as portions of `students.gov` and `clinicaltrials.gov`. All other documents are placed into the remaining $N - 2$ bins randomly.

titlehash A hash function is applied to the document title. If there is no title, the URL is used.

ttl-stop Two special bins are used for the documents with meaningless and repeating titles, as in rand-ts. Then, a hash function maps the title (or URL, if it is missing) to one of the remaining $N - 2$ bins.

sizelink Each bin is used for documents with a similar in-degree. To assure equally-sized bins, documents with in-degree one were split into four bins according to their size, and documents with in-degree of four or more were aggregated.

ttl-stem Document titles (or URLs) are tokenized, stemmed, and stripped of stop words. The resulting bags-of-words are then mapped into a bin by a hash function

cluster Clusters of documents are created with respect to the distance measure between bags of words. Each cluster corresponds to a bin.

sequential This partition simply mimics the directory structure of the distributed GOV2 files, generating partitions of roughly the same size. This is not a random partition, and seems to be based on a BFS order of the documents, starting from the site roots.

In our experiments, we normally used $N = 10$ bins. For evaluation, we trained the resulting federated search system on the 50 topic distillation queries of the TREC 2003 web track and tested it on the 50 topic distillation queries of the TREC 2002 web track. Comparison was based on both the mean average precision (MAP) and the precision-at-10 (P@10). In general, the partition methods that performed best were based on the document title, with random coming in at a close second. However, the overall differences in scores were minor. Consequently, it is our impression that the precise partitioning scheme is immaterial. Having said that, we note that some methods (such as cluster, and one based on cliques found in the hyperlink graph), result in consistently poor results.

Another issue which we briefly touched is the effect of partition size on retrieval results. This has implications when choosing the number of partitions and also when projecting the results to the much bigger .gov2 collection. To this end, we used partition sizes of $N = 15, 20, 30$ with the methods ttl-stem, random, ttl-stop, and sizelink. The results were slightly lower than for the $N = 10$ case. Due to the large amount of processing involved, we were not able to determine if this difference is significant or if the trend continues for larger N.

The TREC results were obtained using the sequential and random partitions, which needed the least amount of processing and therefore were ready first. As a sanity check, we later repeated the run with the ttl-stem partition. The results are summarized in Table 2.

Federation using query prediction

In a previous paper [6] we described the use of a query prediction algorithm for federated retrieval. The gist of the method we developed in that article is to train a query predictor for each sub-collection. When a query is executed, the ranked list of documents is returned from each sub-collection and the prediction of query difficulty is computed for each. The predicted difficulty is used for weighting the document scores (DSs) of the results and the final ranking is built by merging the lists using these weighted DSs. Thus the method we proposed sets a query-by-query weight for each search engine and document collection pair.

Training a predictor requires several training queries with known accuracies. In the context of the Terabyte track this year, only the 50 queries from TREC 2004 could be used.

The input features used for the query predictor are:

1. The overlap between the results set of the full query and those of the sub-queries (query terms containing keywords and lexical affinities).
2. The document frequency (DF) of the sub-queries.

In [6] these pairs of overlaps and document frequencies are then mapped into a histogram. Each cell in the histogram is considered a feature, and from these features a mapping is learned to the actual query difficulty. Unfortunately, due to the limited number of training queries and the size of the histogram (which is dependent on the span of the keyword document frequencies) the training set has more features than examples. This can cause over-fitting of the predictor to the training set [5].

Therefore, we developed a method for representing the histogram using an equivalent histogram with fewer independent cells. This feature reduction algorithm generates a new histogram starting with a histogram in which each cell is considered independent (and thus is later converted into a feature). Since a training query is represented by a list of overlap and DF pairs which map every sub-query into a histogram cell, as well as a difficulty (e.g., mean average precision (MAP)), the algorithm merges adjacent cells if one of the following conditions is met:

1. They contain no training data.
2. The points inside them are from queries with similar difficulty.
3. Each of the queries that have a point in the cells and a dissimilar difficulty have at least one sub-query which does not appear in other queries which appear in these cells. Thus, there is at least some evidence which can explain the dissimilar difficulty.

The algorithm randomly selects the cells to be merged and repeats merging until no merges have been performed for a given number of merge attempts. Our experiments showed that a histogram of 506 cells (11 overlaps and 46 DF values) can be efficiently represented using approximately 20 values. An example of such a mapping is shown in figure 1 in which each color represents cells which map into the same feature. This aggregation results in a lower dimensional feature vector, which greatly reduces the possibility of overfit, even when the number of training queries is small.

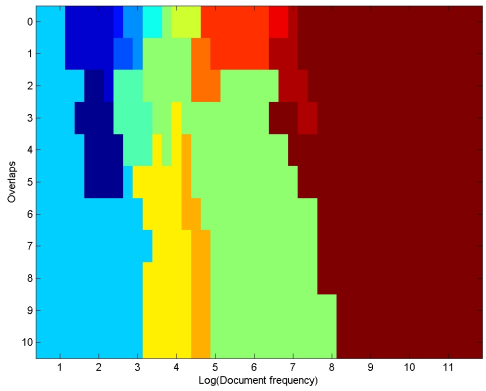

Fig. 1. A typical aggregated histogram. Areas of the same color represent cells which map into the same feature for the query predictor.

Data for the federation algorithm was obtained by running the WAND algorithm on each of the 10 partitions and collecting a full (10,000 document) results set from each. Finally, instead of the linear learning algorithm suggested in [6] we used a Support-Vector Machine (SVM) operating in regression mode as the learning algorithm. The SVM implementation we used was SVMlight [4], with a radial-basis kernel using the default parameters supplied by SVMlight. We used the SVM algorithm due to its improved accuracy, especially with limited training data.

Results of federation using query prediction

Table 2 shows the results of the different partition schemes compared to that of distributed search and the best and median TREC results. This table shows that the different partitioning schemes result in minor retrieval differences, which are also not significant. Additionally, the results of the federated search are very similar to those of the distributed search, which is equivalent to single-index search, thus exhibiting that prediction-based federation can be used as a viable alternative to single-index search. Such federated search has the additional benefits of lower computational cost and better scaling properties.

Type	**Partition**	**MAP**	**P@10**
Federated search	random	0.275	0.530
	sequential	0.269	0.538
	ttl-stem	0.264	0.522
Distributed search	DF1	0.285	0.536
TREC results	Best	0.51	0.9
	Median	0.28	0.57

Table 2. Performance of different partitions on the .gov2 collection, compared to the average best and median results of all participants.

3 Query Prediction in the Robust track

The Robust track addressed a new open question this year - does a query considered "difficult" by most systems on one collection will still be considered difficult when executed on a different collection? Another question to experiment with was whether the special techniques developed for the Robust tracks proved to be useful on one collection are still useful when invoked on another collection. For that, systems were tested on 50 old "difficult" TREC-8 topics. The systems' task was to run those topics against the AQUAINT collection, for which no Qrels exist.

Following this direction of research, we experimented with the question whether a query difficulty predictor, trained by the 249 old TREC-8 topics and their given TREC-8 Qrels, can be used to predict query difficulty on the AQUAINT collection. For that, we trained two predictors using TREC-8 collection and topics, one for description based queries and one for title based queries, and used those predictors to predict the queries difficulty as required by one of the track's tasks.

In addition, following the good results obtained by several groups last year using web expansion, we upgraded our system to benefit web expansion using *answers.com* web search engine. Answers.com is a free search service, providing instant answers over many topics. As opposed to standard search engines that serve up a list of links to follow, it displays quick, snapshot answers with concise, reliable information. If it fails to answer the query it returns the first result returned by *Google* for that query. Our expansion procedure worked by first submitting the topic title to *answer.com*, and then using the result page for query expansion. Each query term is expanded by its lexical affinities as found in the expanding Web page [2].

One of the issues related to query expansion is how to optimally weight the expanded terms. Rocchio formula and other expansion approaches set an ad-hock fixed weight for the expanded terms, however, it seems that optimal weights should

be set dynamically depending on the original query terms and the quality of the expanded terms.

We experimented with using the query predictor to set optimal weights for the expanded query terms. Given an expanded query, we searched over the weight vector space for an optimal weight vector. By exhaustive search we find an "optimal" weight vector that maximizes the predicted average precision for the given expanded query. For each weight vector we used the query predictor to predict the average precision of the expanded query, when the query terms are weighted according to the given weight vector. Since the weight space is infinite we discretized the weight values to 6 possible discrete values, reducing the search space size to $|q|^6$, where $|q|$ is the number of terms in the expanded query.

Note that our main hypothesis behind this run is that optimal weights for a query, computed using the TREC-8 collection, are also optimal weights for AQUAINT. Note also that the query's average precision was computed using TREC-8 Qrels, therefore this run actually tunes the system to optimally handles the queries given the Qrels, hence is considered as manual.

3.1 Robust track results

Table 3 shows the result of applying query expansion to the Robust topics of this year. The table shows that the Web expansion improves over simple retrieval. These differences are all significant (paired sign test, $p < 0.05$), except for the difference between P@10 of the run with no expansion and the manual expansion run.

Topic part	**Run description**	**GAP**	**MAP**	**P@10**
Title	Web expansion	0.157	0.239	0.496
	Manual expansion	0.059	0.173	0.340
	No expansion	0.099	0.189	0.376
	TREC best	0.233	0.332	0.592
	TREC median	0.129	0.224	0.434
Description	Web expansion	0.128	0.230	0.472
	No expansion	0.064	0.148	0.332
	TREC best	0.178	0.289	0.536
	TREC median	0.103	0.184	0.386
Title + Description	Web expansion	0.141	0.242	0.484
	No expansion	0.070	0.155	0.346
	TREC best	0.234	0.332	0.628
	TREC median	0.126	0.218	0.432

Table 3. Query expansion results

3.2 Results of query prediction

Table 4 shows the results of the query prediction algorithm tested on the TREC-8 and the AQUAINT collections. This table shows three types of comparisons: R^2, which is commonly used to estimate the quality of regression algorithms; Kendall's-τ, which is the measure used in [6]; and the area between the MAP of the queries when sorted by actual difficulty and when sorted by predicted difficulty, which is the measure used this year.

The table shows that even for the TREC-8, the query prediction algorithm did not do well compared to the results cited in [6], especially for description-part queries.

This is explained by the fact that the predictor learned to predict using 249 topics spanning a wide distribution of query difficulties, but tested on only 50 difficult topics. In such cases it may be prudent to use trasductive algorithms [3] rather than inductive algorithms. The former assume that the test topics are known, and thus focuses the learning on the area of the distribution where these queries are located. This, however, will results in a predictor which is not universally applicable.

A further conclusion from this table is that although there is some degradation caused by learning a predictor on one collection and using it on a different collection, the predictor is still useful, at least for the short title queries.

	TREC-8			AQUAINT		
Topic part	R^2	**Kendall's-τ**	**Area**	R^2	**Kendall's-τ**	**Area**
Title	0.295	0.362	3.638	0.0951	0.262	7.431
Description	0.095	0.053	5.676	0.0010	0.030	7.857

Table 4. Query prediction results

4 Summary

The results we obtained from this year's experiments demonstrate how prediction of query difficulty can successfully be used. In the Terabyte track the query predictor was used to effectively federate search results – federated search based on query prediction can achieve comparable results to single-index search. In the Robust track, a predictor learned over one collection could predict query difficulty over another collection, with reasonable results, at least for the title-based queries.

The experiments also provide some negative results. Different partition schemes of the data do not affect precision significantly; random partition achieved comparable results to sophisticated partition schemes. However, our experiments are limited in the sense that they split the data to only few bins - this conclusion should be explored more deeply when the data is partitioned into many more bins.

In the Robust track, most of the difficult TREC topics obtained high precision when executed against the AQUAINT collection. The difficulty characteristic of those topics was not preserved against AQUAINT. It is difficult to confidently judge the reasons for the improved precision, but it seems to lend support to our claim that query difficulty mostly depends on the collection features rather than on query independent features.

Finally, this is the first time that we were able to index and search over collection of (half) terabyte of data in a reasonable time, and to achieve decent results – an important milestone for our search system.

References

1. Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. Efficient query evaluation using a two-level retrieval process. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, pages 426–434, New York, NY, USA, 2003. ACM Press.
2. David Carmel, Eitan Farchi, Yael Petruschka, and Aya 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.

3. A. Gammerman, V. Vovk, and V. Vapnik. Learning by transduction. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 148–156. Morgan Kaufmann, 1998.
4. Thorsten Joachims. Making large-scale svm learning practical. In B. Scholkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT-Press, 1999.
5. Elad Yom-Tov. An introduction to pattern classification. In U. von Luxburg O. Bousquet and G. Ratsch, editors, *Advanced Lectures on Machine Learning. LNAI 3176*. Springer, 2004.
6. Elad Yom-Tov, Shai Fine, David Carmel, and Adam Darlow. Learning to estimate query difficulty including applications to missing content detection and distributed information retrieval. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 512–519, Salvador, Brazil, 2005.
7. Elad Yom-Tov, Shai Fine, David Carmel, Adam Darlow, and Einat Amitay. Improving Document Retrieval According to Prediction of Query Difficulty. In *Proceedings of the 13th Text REtrieval Conference (TREC2004)*, 2004.