Juru at TREC 2004: Experiments with prediction of query difficulty

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

Abstract

Our experiments in the Robust track this year focused on predicting query difficulty and using this prediction for improving information retrieval. We developed two prediction algorithms and used the subsequent prediction in several ways in order to improve the performance of the search engine. These included modifying the search engine parameters, using selective query expansion, and switching between different topic parts. We also experimented with a new scoring model based on ideas from the field of machine learning.

Our results show that query prediction is indeed efficient in improving retrieval, although further work is needed in order to improve the performance of the prediction algorithms and their uses.

1 Introduction

Most search engines respond to user queries by generating a list of documents deemed relevant to the query. In this work we suggest to give the user additional information by predicting query difficulty; namely, how likely it is that the documents returned by the search engine are relevant to the query. This information is advantageous for several applications, among them are:

1. Simple evaluation of the query results.
2. Feedback to the user: The user can rephrase the query so as to improve query prediction.
3. Feedback to the search engine: The search engine can use the query predictor as a target function for optimizing the query, for example by adding terms to the query.
4. Use of different ranking functions for different queries based on their predicted difficulty: It is well-known in the Information Retrieval community that methods such as query expansion can help "easy" queries but are detrimental to "hard" queries [5]. The predictor can be used to mark easy queries on which the search engine should use query expansion. Our initial experiments suggest that such a method indeed improves the performance of the search engine.
5. For distributed information retrieval: As a method for deciding which search engine to use: Given a query and several search engines (e.g. Google, Yahoo, HotBot, etc), the predictor can estimate the results of which search engine are best for the given query. Additionally,

prediction can be used as a method for merging the results of queries performed on different datasets by weighing the results from each dataset.

The goal in our participation in the Robust Track this year was to experiment with using prediction of difficulty as a means for improving information retrieval. Our observations, which we describe in detail below, show that queries that are answered well by search engines are those whose keywords agree on most of the returned documents. Agreement is measured by the number of documents ranked best using the full query that were also ranked best by the sub-queries. Difficult queries (i.e. queries for which the search engine will return mostly irrelevant documents) are those where either all keywords agree on all results or cannot agree on them. The former is usually the case where the query contains one rare keyword that is not representative of the whole query and the rest of the query terms appear together in many irrelevant documents.

For example, consider the TREC query "What impact has the Chunnel had on the British economy and/or the life style of the British?". In this query many irrelevant documents, selected by the search engine, will contain the words British, life, style, economy, etc. But the gist of the query, the Chunnel, is lost. Another type of difficult query is one where the query terms do not agree on the target documents and each contributes very few documents to the final results. An example of such a case is the TREC query "Find accounts of selfless heroic acts by individuals or small groups for the benefit of others or a cause". In this query there are no keywords that appear together in the relevant documents and thus the final result set is poor.

Our observations have led us to suggest methods by which query difficulty can be predicted. The proposed methods are a wrapper to the search engine, i.e. they is not limited to a specific search engine or search method. Since they do not intervene in the workings of the search engine, they can be applied to any search engine. The main advantages of these algorithms, in addition to possible applications described above, are that they are simple, and can be applied by the search engine during query execution, since most of the data they use for their operation are generated by the search engine during its normal mode of operation.

As far as we know, the only attempt to measure query difficulty (albeit for a different reason) is [1]. In this work, prediction of query difficulty is attempted for a specific scoring model (i.e. A specific search engine). The purpose of this research is to decide on a query-by-query basis if query expansion should be used. In contrast, the proposed method is not limited to a specific search engine, and thus has many additional applications (As outlined above).

2 Predicting query difficulty

The basic idea behind the prediction is to measure the contribution of each query element to the final ranking of documents. The query elements are the keywords (i.e. the words of the query, after discarding stopwords) and *lexical affinities*, which are closely related keywords found in close proximity to each other [5]. We define a sub-query as the ranking of the documents returned by the search engine when a query element is run through it.

The features used for learning the estimator 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 intersection between the top N results of the two queries. Thus, the overlap is in the range of $[0, N]$.
2. The rounded logarithm of the document frequency, $\log(DF)$, of each of the sub-queries.

Learning to estimate 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 estimator varies from query to query. In contrast, most techniques for estimation are based on a fixed number of features. The second problem in learning to estimate is that the significance of the

sub-queries is not given, so any algorithm which performs a comparison or a weighted average on an ordered feature vector is not directly applicable.

In the following, we propose two solutions for these problems, based on finding a canonic representation for the data. The first solution bunches the features using a histogram, and the second one uses a modified tree-based estimator. We show that a histogram is useful when the number of sub-queries is large (i.e. there are many keywords and lexical affinities in the query). The tree-based classifier useful primarily for short queries.

2.1 Query predictor using a histogram

The histogram-based algorithm generates the prediction 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 $h(i), i = 0, 1, ..., N$. Entry $h(i)$ counts the number of sub-queries which agree with the full query on exactly i documents in the top N results. Unless otherwise stated, we used N=10.
3. Make the histogram into a binary histogram, that is:
   $h_b(i) = \begin{cases} 1 & h(i) > 0 \\ 0 & h(i) = 0 \end{cases}$
4. Sort the binary histogram, retaining the rank of the ordered vector. For example, if $h_b = [1\ 0\ 0\ 1\ 1]$, the result of the sorting is $r = [2\ 3\ 1\ 4\ 5]$
5. Predict query difficulty by multiplying the resulting rank r, by a linear weight vector c such that $Pred = c^T \cdot r$. The method by which this weight vector is computed is described below.

This algorithm can be improved significantly by using as features both the overlaps and $\log(DF)$ of the terms. For canonical representation we split $\log(DF)$ into 3 discrete values[1] $0-1, 2-3, 4+$. In this case, the algorithm is modified so that in stage (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 4 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:

$$h(DF, Overlap) = \begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \\ 2 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{array}$$

Before stage (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 estimator (after making the histogram into a binary histogram and concatenating rows) is $h(i) = [0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0]$;

The linear weight vector can be estimated in several ways, depending on the objective of the estimator. If the object of the estimator 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[6]:

$$c = \left(R \cdot R^T\right)^{-1} \cdot R \cdot t^T \tag{1}$$

where R is a matrix whose columns are the ranking vectors r 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

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

MAP (maximizing Kendall's-τ between estimated and actual order), in which case a more suitable optimization strategy is to modify the above equation as suggested in [9]. 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 $c^T \cdot r_i > c^T \cdot r_j$. This constraint can be rewritten in the form: $c^T(r_i - r_j) > 0$. In practice, it is better to modify the constraint such that we demand that $c^T(r_i - r_j) \geq 1$ in order to fix the scaling of c [10]. Therefore, instead of using the ranking vectors r directly, the matrix R is modified to a matrix whose columns are the differences between the ranking vectors; for each pair of training queries (i, j) the k-th column is:

$$R_k = r_i - r_j \quad \forall i, j = 1, 2, \ldots N_r \tag{2}$$

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

$$t_k = \begin{cases} +1 & if \;\; t_i > t_j \\ -1 & if \;\; t_j \leq t_i \end{cases} \forall i, j = 1, 2, \ldots N_r \tag{3}$$

2.2 Query predictor using a modified decision tree

The histogram-based estimator can be useful only when enough data is available to construct it. If the query is short, there are few sub-queries with which to build it, and the resulting canonical representation is too sparse to be of use for estimation. In contrast, as we will show below, a tree-based classifier can make use of much sparser data and thus it is useful for estimating the difficulty of shorter queries. The decision tree learning algorithm we present is similar to the CART algorithm [2], with modification described below.

The suggested tree is a binary decision tree. Each node consists of a weight vector and a score. Sub-queries are sorted according to increasing DF. Starting at the root, the estimation algorithm moves along the branches of the tree, using one sub-query for deciding the direction of movement at each node. Movement is terminated when no more sub-queries exist or when a terminal node is reached. The estimation of query difficulty is the score at the terminal node. The algorithm takes a left branch if the multiplication of the weights at the current node by the overlap and logarithm of DF of the current sub-query is smaller than a threshold, or a right branch if it is larger than the threshold.

During training of the decision tree, a linear classifier (such as the above-mentioned Moore-Penrose pseudo-inverse) is trained at each node to try and split the number of training queries equally among the branches of the tree according to their rank (i.e., their respective ranking according to P@10 or MAP). This is done by training the weight vector of the classifier so that it gives a negative value when it is multiplied by the patterns from those queries with a lower rank and a positive value when multiplied by the patterns from those queries that have a higher rank. Thus, low-ranked queries will have a negative result and be classified to the left child node and those with the higher rank will have a positive value for the multiplication and be classified to the right child node.

Scores are assigned to nodes so that the left part of the tree will have a lower score compared to its right part. This is achieved by the following method: The root node has a score of 1. Taking a left branch implies a division of the score by 1.5, while a right branch multiplies it by 1.2. These values were found through a heuristic search.

It is well-known in literature that better decision trees can be obtained by training a multitude of trees, each in a slightly different manner or using different data, and averaging the estimated results of the trees. This concept is known as a Random Forest [3]. Thus, we trained 50 different trees using a modified resampling of the training data, obtained via a modification of the AdaBoost algorithm [6]. The modified AdaBoost algorithm resamples the training data to improve the final performance of the estimator. Assume each query from the training set is characterized by the pair $\{x, r\}$, where x is the data from a single training query (i.e., pairs of $\log(DF)$ and overlaps for each sub-query, and r is the ranking (MAP or P10) of the query. The algorithm for training a single tree is shown below:

1. **Initialize** $D = \left\{(x,r)^1, \ldots, (x,r)^n\right\}$, k_{max}, δ (Minimum difference threshold), $W_1(i) = 1/n, i = 1, \ldots, n$, $k \leftarrow 0$.
2. Train a decision tree DT_k using D sampled according to $W_k(i)$.
3. Measure the absolute difference in the location of each training example classified by the decision tree and its correct rank. Denote these differences by $d(i)$.
4. Compute E_k, the training error of DT_k, measured on D, $E_k = \sum_i (d(i) > \delta)$.
5. Compute the importance of the hypothesis: $\alpha \leftarrow \frac{1}{2} \ln \left[\frac{(1-E_k)}{E_k}\right]$
6. Update the weights: $W_{k+1}(i) \leftarrow W_k(i) \times \begin{cases} e^{\alpha} & if \;\; d(i) < \delta \\ e^{-\alpha} & if \;\; d(i) \geq \delta \end{cases}$
7. Normalize weights: $W_{k+1}(i) = W_{k+1}(i) / \sum_i W_{k+1}(i)$
8. Repeat stages 2-7 for k_{max} iterations.

3 Evaluating the performance of the query predictor

The query prediction algorithm was tested using the Juru search engine [4] on two document collections from the TREC competition[8, 12]: The TREC collection and the WT10G collection. The TREC collection comprises of 528,155 documents. The WT10G collection encompasses 1,692,096 documents. The testing was performed using 200 TREC topics over the TREC collection and 100 topics from the Web track of TREC over the WT10G collection. Each topic contains two parts: A short (1-3 word) title and a longer description (Usually a complete sentence).We experimented with short queries based on the topic title and with long queries based on the topic description. Four-fold cross-validation was used for assessing the algorithm, i.e.: The topics were divided into four equal parts. A predictor was trained using three of the four parts, and was then applied to the remaining part for estimation of its performance. The process was repeated for each of the parts, and the quoted result is an average of the four results.

The set of queries used for evaluating our data, sorted by predicted MAP (or P10), can be thought of as a ranked list. The distance between the predicted ranked list and the real ranked list can be measured using Kendall's-tau (KT) distance.

The results of this test are shown in 1. Note that a random rank would result in a Kendall-tau distance of zero. Both predictors show a reasonable approximation of the correct ranking. Shorted queries (i.e. Title queries) are estimated better using the tree-based classifier, while long queries are addresses by the histogram-based classifier. The last rows of 1 showed the ability to use a single predictor for both databases. The results demonstrate that there was an improvement in performance through the use of both datasets. This is attributed to the fact that more data was available for building the predictors, and thus they performed better.

The weights calculated for the linear regression in the histogram-based predictor represent the relative significance of the entries in the binary histogram ($h_b(i) = 1$ if there is a least one sub-query that agrees with the full query on i top results, otherwise $h_b(i) = 0$). It is difficult to interpret the weights obtained by the linear regression. However, since there are a finite number of possible states that the binary histogram can have, some intuition regarding the weights (and the algorithm) can be gained by computing the estimated precision for each possible state of the histogram. If an overlap of 10 is considered between the full query and the sub-queries, there are a total of $2^{11} = 2048$ possible histogram states, corresponding to 2^{11} different binary histograms.

Thus we computed the predicted P10 for each such state using the linear prediction vector obtained using the description part of 200 topics from the TREC collection, ordered the states according to the prediction, and averaged each group of 128 consecutive states. The average histogram values for several slices (Each slice is an average of 128 individual histograms) are shown in figure 1. This figure

Database	Topic part	Tree		Histogram	
		KT_{MAP}	$KT_{P@10}$	KT_{MAP}	$KT_{P@10}$
TREC	Title	0.284	0.271	0.161	0.144
	Description	0.222	0.231	0.357	0.282
	Title + Description	0.200	0.194	0.357	0.265
Web	Title	0.118	0.175	0.110	0.155
	Description	0.202	0.098	0.093	0.140
	Title + Description	0.192	0.182	0.142	0.283
TREC + Web	Title	0.323	0.273	0.216	0.183
	Description	0.273	0.293	0.421	0.362
	Title + Description	0.273	0.265	0.345	0.312

Table 1. Kendall-tau scores for query prediction evaluated on the TREC and the Web datasets.

demonstrates that the easy queries have some overlaps between sub-queries and the full query at both high and medium ranges of overlap. Queries that perform poorly have their overlaps clustered in one area of the histogram, usually the lower one. This suggests that a good pattern of overlap would be one where the query is not dominated by a single keyword. Rather, all (or at least most) keywords contribute somewhat to the final results.

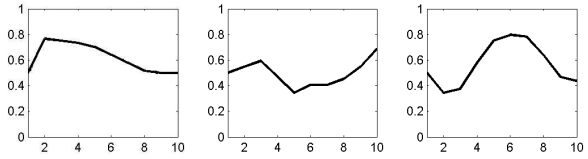

Fig. 1. Examples of the average weights as a function of the predicted P10. The leftmost figure shows a pattern which generated lower prediction for P10 (An average P10 of 3.1). This figure shows that a low prediction is generated when most overlaps are low, specifically here most sub-queries have an overlap of 2. The center figure are the average weights of representing an average predicted P10 of 7.3 and the rightmost figure the average weights representing a predicted P10 of 9.96. In all three figures the horizontal axis is the number of overlaps and the vertical axis the average weight.

4 Experiments in the Robust Track: Improving information retrieval using query prediction

As mentioned above, query prediction is useful not only as feedback to the user, but also as a method for improving information retrieval. In this section we describe several ways in which query prediction was used for this goal.

The results of these methods on the older 200 TREC queries (301-450, 601-650) are shown in table 2. These results were obtained using four-fold cross-validation. Figure 2 shows a comparison of these results and other TREC participants.

4.1 Description of the runs

Base runs

We performed five basic runs using Juru without any modification. These runs were:

1. Title-only run (Run name: JuruTit).
2. Description-only run (Run name: JuruDes).

3. Title and description run (Run name: JuruTitDes).
4. Title-only with Query Expansion run (Run name: JuruTitQE).
5. Description-only with Query Expansion run (Run name: JuruDesQE).

Selective query expansion

Query expansion (QE) is a method for improving retrieval by adding terms to the query based on frequently-appearing terms in the best documents retrieved by the original query. However, this technique works only if the search engine was able to give a high rank to relevant documents using the original query. If this is not the case, QE will add irrelevant terms, causing a decrease in performance [5].

Thus, it is not beneficial to use QE on every query. Instead, it is advantageous to have a switch that will predict when QE will improve retrieval, and when it would be detrimental to it.

Using the same features utilized for the histogram-based predictor we trained an SVM classifier [6, 11] to decide when to use QE and when not to. The SVM was trained with an RBF (Gaussian) kernel, with a width of 0.5.

There were two queries of this type. In the first the input was the description part of the topic. This run is labeled JuruDesSwQE . In the second the input was the title part of the topic. This run is labeled JuruTitSwQE .

Term selection

An alternative use for the predictor is to discover queries for which the search engine will return no or very few relevant results. Our working assumption was that if this is the case in a description query, the cause is likely to be noise arising from the length of the query. Therefore, for such queries, only a few terms were chosen in the query and the remaining terms were filtered out completely. Terms were chosen by two methods attempted to keep only terms which are similar in order to focus on the topic of the query. One used a hand-crafted rule and the other a tree-based classifier similar to the one described for the prediction:

1. In the hand-crafted method, for each query, the similarity between each pair of terms was calculated. The similarity measure used was a vector cosine between the terms (represented as sets of documents) divided by the log-distance between the occurrences of the terms in the query. The two pairs with the highest similarities were chosen, meaning either three or four terms.
2. In the automatic method, for each query, each term was given a list of features. This list included its occurrence's offset in the query, its vector cosine similarity to each term in the query and the distance between its occurrence and the occurrences of each term in the query. Several classified examples were used to train a tree-based classifier which was then used to choose the terms in each query.

Terms which were chosen by either of the methods were kept and those which were chosen by neither were removed from the query.

The input for this run was the description part of the topic. This run is labeled JuruDesTrSl .

Deciding which part of the topic should be used

TREC topics contain two relevant parts: The short title and the longer description. Our observations have shown than in general queries that are not answered well by the description and better answered by the title part of the topic.

The predictor was used to decide which part of the topic should be used. The title part of the topic was used for the queries ranked (by the predictor) in the lower 15% of the topics, while the description part was used for the remaining 85% of topics.

This run is labeled JuruTitSwDs .

Vary the parameters of the search engine according to query prediction

The Juru search engine uses both keywords and lexical affinities for ranking documents. Usually each lexical affinity is given a weight of 0.25 compared to 0.75 for regular keywords. However, this value is an average that can address both difficult and easy queries. In fact, difficult queries can be answered better by assigning greater weight to lexical affinity, while easy queries are improved by assigning lexical affinities a lower weight.

Unfortunately, the predictor is not sensitive enough to be used to modify these weights across the whole spectrum of difficulty. Instead we use a weight of 0.1 for the queries ranked (by the predictor) in the top 15% of the queries, while the regular weight of 0.25 was used for the remaining 85% of queries.

The input for this run was the description part of the topic. This run is labeled JuruDesLaMd .

Rank aggregation

Rank aggregation is a method designed to combine ranking results from various sources. The main applications include building meta-search engines, combining ranking functions, selecting documents based on multiple criteria, and improving search precision through word associations. Previous attempts [7, 9] to implement this method were somewhat "passive" in their design of the functions that provide the ranking results, and instead focus on techniques that combine the ranks provided by the available sources.

Our rank aggregation method adapted a more "active" approach: Similarly to our query prediction technique, we automatically constructed a set of sub-queries, based on the original query elements (keywords and lexical affinities), and used Juru to get ranking scores for every sub-query. We then used the resulting rank scores of the sub-queries as features in a training set, namely each document was represented as a vector of scores. Finally, we trained a non-linear SVM to learn how to re-rank documents (employing the strategy suggested in [9]), where the reference order for training was based on the original (full) query ranks. Thus, the learning system did not get the true order at any time, and in essence this scheme may be viewed as an "unsupervised learning" framework.

The input for this run was the description part of the topic. This run is labeled JuruDesAgg .

4.2 Results

Table 2 shows the results of running the methods described above on the older 200 TREC queries, using four-fold cross-validation. Note that for a fair comparison, runs should be compared according to the parts of the topics they use.

Selective QE is the most efficient method for improving queries based solely on the description. When both topic parts are used, it is evident that the switch between topic parts is better than using both at the same time or just one of the parts.

Figure 2 shows a histogram of the difference between the P10 obtained using the experimental methods and the median score of all the 2004 Robust Track runs. As this figure shows none of the methods performed as good as the best TREC submission. Of our runs, the best runs were the run which applied a switch between regular runs and those that used QE and the term-selection runs. However, the differences between all the runs were relatively minor, with an average difference of only 0.33.

Run name	MAP	P@10	% no
Title only	0.271	0.437	10.0
Description only	0.281	0.473	9.5
Title + Description	0.294	0.484	8.5
Description with QE	0.284	0.467	11.5
Description with selective QE	0.285	0.478	9.5
Description with selective QE and refinement	0.281	0.473	11.0
Description with modified parameters	0.282	0.467	9.5
Switch description-title	0.295	0.492	6.0

Table 2. Improvements in retrieval using different techniques. The measures used for assessing the performance are MAP, P@10, and % no. The last are the percentage of queries for which there was not a single relevant document in the top 10 documents.

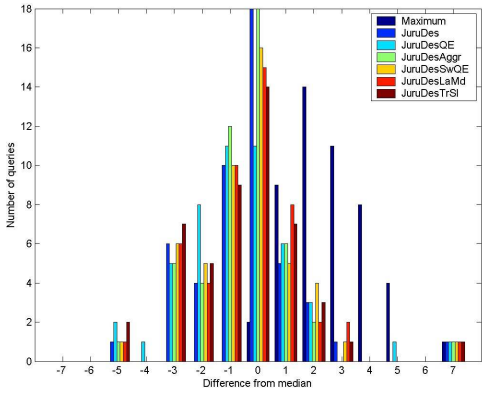

Fig. 2. Histogram of difference between P10 of each of the description-only methods proposed and the median P10 of all participating description runs in the Robust Track for the 49 new queries

The Kendall-tau score between the MAP obtained in our 10 runs and the predictions generated by the suggested algorithms is shown in table 3 (The displayed results are different from the original submitted runs due to a bug discovered after the runs were submitted). As can be seen the histogram-based predictor achieved good, consistent, predictions. Note the similarity between Kendall-tau for the older 200 queries and the 49 new queries, which demonstrate the robustness of the histogram predictor. The results of these predictions on all queries are much better than the median of all runs submitted to the Robust Track (At this time we do not yet know the median and best prediction as measured over the 49 new queries). The tree-based predictor obtained inferior results. This is likely not the results of over-fitting, since the prediction we submitted was the results of four-fold cross-validation. Instead, we suspect that this is due to an inherent difficulty in prediction of short queries. As shown above, this difficulty can be mitigated by using more training data (i.e. more queries).

5 Summary

Our experiments in the Robust Track of TREC centered on predicting query difficulty and using this prediction for improving information retrieval. We developed two algorithms for predicting query difficulty and used this prediction in several novel ways.

Run name	Type	All queries	200 queries	49 queries
Description only	Histogram	0.387	0.379	0.429
Description with QE	Histogram	0.365	0.351	0.432
Description with selective QE	Histogram	0.377	0.361	**0.435**
Description with selective QE and refinement	Histogram	0.370	0.364	0.407
Description with modified parameters	Histogram	0.392	0.383	**0.434**
Description with aggregation	Histogram	0.378	0.369	0.427
Title only	Tree	0.219	0.263	0.081
Title with selective QE	Tree	0.209	0.236	0.111
Title + Description	Histogram	0.379	0.373	0.412
Switch description-title	Histogram	0.327	0.307	0.432

Table 3. Kendall-tau scores of the 10 submitted runs

Our results demonstrate that query prediction is easier for longer queries compared to shorter ones. Our histogram-based predictor is able to predict query difficulty very well, but the use of these predictions for significant improvement of retrieval tasks is still an open issue for research.

References

1. G. Amati, C. Carpineto, and G. Romano. Query difficulty, robustness and selective application of query expansion. In *Proceedings of the 25th European Conference on Information Retrieval (ECIR 2004)*, pages 127–137, Sunderland, Great Britain, 2004.
2. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. Chapman and Hall, 1993.
3. Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
4. David Carmel, Einat Amitay, Miki Herscovici, Yoelle S. Maarek, Yael Petruschka, and Aya 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.
5. 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.
6. R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern classification*. John Wiley and Sons, Inc, New-York, USA, 2001.
7. C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation revisited. In *Proceedings of WWW10*, 2001.
8. D. Hawking and N. Craswell. Overview of the TREC-2001 Web Track. In E. M. Voorhees and D. K. Harman, editors, *Proceedings of the Tenth Text Retrieval Conference (TREC-10)*. National Institute of Standards and Technology. NIST, 2001.
9. 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.
10. B. Scholkopf and A.J. Smola. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT Press, Cambridge, MA, USA, 2002.
11. D. Stork and E. Yom-Tov. *Computer manual to accompany pattern classification*. John Wiley and Sons, Inc, New-York, USA, 2003.
12. E. M. Voorhees and D. K. Harman. Overview of the Tenth Text REtrieval Conference (TREC-10). In *Proceedings of the Tenth Text Retrieval Conference (TREC-10)*. National Institute of Standards and Technology (NIST), 2001.