# Learning to Estimate Query Difficulty

## Including Applications to Missing Content Detection and Distributed Information Retrieval

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

## ABSTRACT

In this article we present novel learning methods for estimating the quality of results returned by a search engine in response to a query. Estimation is based on the agreement between the top results of the full query and the top results of its sub-queries. We demonstrate the usefulness of quality estimation for several applications, among them improvement of retrieval, detecting queries for which no relevant content exists in the document collection, and distributed information retrieval. Experiments on TREC data demonstrate the robustness and the effectiveness of our learning algorithms.

## Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

## General Terms

Algorithms

## Keywords

Query Difficulty Estimation

## 1. INTRODUCTION

Most search engines respond to user queries by generating a list of documents deemed relevant to the query. However, many information retrieval (IR) systems suffer from a radical variance in performance; Even for systems that succeed very well on average, the quality of results is poor for some of the queries [22, 11]. Thus, it is desirable that IR systems be able to identify "difficult" queries in order to handle them properly.

Estimating query difficulty is an attempt to quantify the quality of results returned by a given system for the query over a given collection. An example for such a measure of quality is the precision-at-10 (P@10) or the mean average precision (MAP) [22] of the query. Estimation of query difficulty is advantageous for several reasons:

1. Feedback to the user: The user can rephrase a "difficult" query to improve system effectiveness.
2. Feedback to the search engine: The search engine can invoke alternative retrieval strategies for different queries according to their estimated difficulty.
3. Feedback to the system administrator: The administrator can identify queries related to a specific subject that are "difficult" for the search engine, and expand the collection of documents to better answer poorly covered subjects.
4. For distributed information retrieval: Estimation can be used to decide which search engine to use by estimating the results of which search engine are best. Additionally, estimation can be used as a method for merging the results of queries performed on different datasets by weighing the results from each dataset.

Recently, the concept of a *clarity score* [8] has been proposed as an attempt to quantify query difficulty. A clarity score is defined as the difference between the language model used in the document collection, and the language model used for the query. If the models are similar, then the query has a low clarity score – the query describes the entire collection. If the models are disparate, then the query identifies a subset of the collection which is likely to be the answer set for the query. The clarity score has been shown to correlate positively with the query's average precision, hence it could serve as a predictor of query performance.

In this work, we suggest an alternative approach, based on machine learning techniques, for estimating query difficulty. Our observations, which we describe in detail below, 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.

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, 'Chunnel', is lost.

Our method learns to estimate query difficulty based on the overlaps between the results of the full query and its sub-queries. The estimator is induced from training queries and their associated relevance sets. In the following, we describe two learning approaches for inducing an estimator from training data.

The main advantage of our estimator is its simplicity, and the search engine's ability to efficiently apply it during query execution, since all data it uses 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. Furthermore, the learned estimator can be used as a wrapper to the search engine since it can be applied without intervention in the workings of the search engine. This can be done by submitting each sub-query independently to the search engine. Therefore, the learned estimator is not limited to a specific search engine or a search method.

The rest of the paper is organized as follows: Section 2 describes related work to ours. Section 3 describes our methods for learning an estimator for query difficulty in full details. Section 4 describes the experiments we conducted on TREC data. Section 5 describes three applications for these estimations, namely: Improving information retrieval (IR), identifying queries for which no relevant document exist in the document collection, and merging query results in the setting of distributed IR. Section 6 concludes and addresses some directions for future work.

## 2. RELATED WORK

Estimation of query difficulty has been recently recognized by the IR community as an important capability for IR systems. In the Robust track of TREC 2004 [22], systems were asked to rank the topics by predicted difficulty, with the goal of eventually being able to use such predictions to do topic-specific processing. System performance was measured by comparing the ranking of topics based on their actual precision to the ranking based on their predicted precision. Prediction methods suggested by the participants varied from measuring clarity based on the system's score of the top results [19, 15], through analyzing the ambiguity of the query terms [18], to learning a predictor using old TREC topics as training data [14, 24]. The track results clearly demonstrated that measuring query difficulty is still intrinsically difficult.

Amati et. al [1] showed positive correlation with the DFR (Divergence From Randomness) scoring model to query precision. By predicting query precision, they were able to expand only easy queries and to improve their system performance. He and Ounis [12] studied the usefulness of several alternative predictors, including one based on the standard deviation of the inverse document frequency (idf) of the composing query terms. This predictor is based on the assumption that the terms of a poorly-performing query tend to have similar idf values. Plachouras et al. [21] experimented with the idf-based predictor and with a similar variant based on the average inverse collection term frequency (avICTF) of the query terms. Both predictors showed positive correlation with query precision over the Robust track's topics.

Diaz and Jones [9] use meta-data, attached to documents in the form of time stamps, to measure the distribution of documents retrieved over the time domain. They showed that using these temporal distribution together with the content of the documents retrieved can improve the prediction of average precision for a query.

Kwok et. al [14] learn an estimator from training queries, employing epsilon support vector machine for regression. The queries are represented by a feature vector containing document frequencies and query term frequencies of the most significant keywords of the query. The authors conclude that although there is some correlation between observed and prediction topic ranking, better feature settings may lead to improved results.

The Reliable Information Access (RIA) workshop [11] investigated the reasons for system variance in performance across queries. By performing failure analysis on TREC topics, 10 failure categories were identified [4], including four categories for which systems fail due to emphasizing only partial aspects of the query. One of the conclusions of that workshop was that "...comparing a full topic ranking against ranking based on only one aspect of the topic will give a measure of the importance of that aspect to the retrieved set" [11]. Our work follows this direction by estimating query difficulty based on the overlap between the full query and its sub-queries.

## 3. ESTIMATING QUERY DIFFICULTY

The basic idea behind the estimation of query difficulty is to measure the contribution of each query term to the final result set. In our system, query terms are defined as the keywords (i.e., the words of the query, after discarding stopwords) and the *lexical affinities*, which are closely related query terms found in close proximity to each other [6]. 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 query difficulty is that the sub-queries are not ordered 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 lexi-

cal affinities in the query). The tree-based classifier is useful primarily for short queries.

The overlap between a sub-query and the full query is a measure of agreement between these two queries. The $\kappa$-statistics [7] is a standard measure for the strength of agreement between two experts. The Appendix shows that the $\kappa$-statistics is a linear function of the overlap between the two queries. Thus, our query estimator is based on meta-statistics over the $\kappa$-statistics.

## 3.1 Query estimator using a histogram

The histogram-based algorithm generates the estimation 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. 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 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:

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

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 is $h(i) = [0\ 0\ 1\ 2\ 0\ 0\ 0\ 1\ 0]$;

Two additional features which were found to be advantageous to the success of the estimator are the score of the highest-ranked document (as suggested by [19]) and the number of words in the query. We concatenated these features to the histogram data, and the reported results use these features.

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[10]:

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

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

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 (maximizing Kendall's-$\tau$ between estimated and actual order), in which case a more suitable optimization strategy is to modify the above equation as suggested in [13]. The idea is that a linear weight vector will maximize Kendall's-$\tau$, 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 $c$ [16]. 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:

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

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

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

## 3.2 Query estimator 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 [3], with modification described below.

The suggested tree is a binary decision tree. Each node consists of a weight vector, a threshold, 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. The algorithm takes a left branch if the multiplication of the weights at the current node by the overlap and the logarithm of DF of the current sub-query is smaller than the threshold, or a right branch if it is larger than the threshold. 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 terminated node.

An example of a decision tree is shown in Figure 1. Consider the data of a sample query shown at the top of the figure. Starting with the first sub-query, its data (pairs of overlap and $\log(DF)$) are multiplied by the weights of the corresponding node. The multiplication of the first sub-query (Overlap=3, DF=0) by the root node weights (-1.0,-1.0) is -3, so the left branch is taken, as shown by the thicker line. (For simplicity, the threshold for all nodes is assumed to be zero.) Movement is continued until no more sub-queries exist or, as in this case, a terminal node is reached. The resulting estimation in this example is 0.96.

During training of the decision tree, the weight vector (learned by 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 [2]. Thus, we trained 50 different trees using a modified resampling of the training data, obtained via a modification of the AdaBoost algorithm [10]. 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, as in the sample query in Figure 1), and $r$ is the ranking (according to MAP or P10) of the query. The algorithm for training a single tree proceeds as follows:

1. **Initialize** $D = \left\{(x,r)^1, \ldots, (x,r)^n\right\}$, $k_{max}$, $\delta$ (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.

## 4. EVALUATION

The query estimation algorithms were tested using the Juru search engine [24] on two document collections from TREC: The TREC-8 collection (528,155 documents, 200 topics) and the WT10G collection (1,692,096 documents, 100 topics). 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

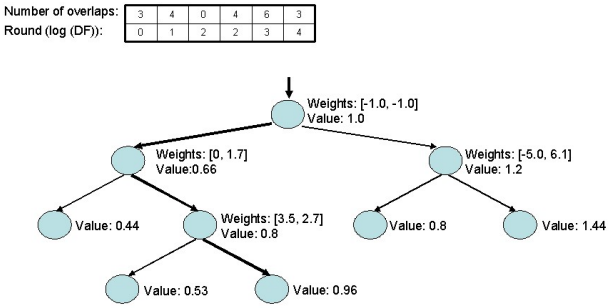


**Figure 1: Example of a decision tree. See text for an explanation of this example.**

the algorithm: The topics were divided into four equal parts. An estimator 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 topics used for evaluating our data was ranked according to the estimated MAP/P@10 and according to the actual MAP/P@10. The distance between the estimated ranked list and the actual ranked list was measured using Kendall's-$\tau$ (KT) distance.

The results of this test are shown in Table 1. Both estimators show a reasonable approximation of the correct ranking. Short queries (i.e., title queries) are estimated better using the tree-based classifier, while long queries are addressed better by the histogram-based classifier. However, overall the results for queries based on the title are inferior to those based on the description. We attribute this to two causes: First, title queries are shorter, and thus less information exists to estimate the query difficulty. The second is that in some queries, the title contradicts the description and narrative, but since documents are judged by the latter, queries might be estimated wrongly.

The last two rows of Table 1 describes the use of a single estimator for a collection, which is the union of both datasets. The results point to an improvement in the performance of the predictors. This is because more data (300 topics) was available for building the estimators, and thus they performed better.

| Database | Top part | Tree | | Histogram | |
|---|---|---|---|---|---|
| | | $KT_{MAP}$ | $KT_{P@10}$ | $KT_{MAP}$ | $KT_{P@10}$ |
| TREC-8 | T | 0.305 | 0.268 | 0.254 | 0.253 |
| | D | 0.218 | 0.249 | 0.439 | 0.360 |
| WT10G | T | 0.118 | 0.175 | 0.143 | 0.187 |
| | D | 0.202 | 0.098 | 0.140 | 0.172 |
| Both | T | 0.287 | 0.280 | 0.312 | 0.291 |
| | D | 0.252 | 0.280 | 0.464 | 0.414 |

**Table 1: Kendall's-$\tau$ scores for query estimation evaluated on TREC-8, WT10G, and a combination of both datasets. Queries were either short and based on the title (T) part of the topic, or longer and based on the description (D) part of the topic.**

Note that a random ranking would result in a Kendall's-$\tau$ score of zero, and a complete agreement would result in a score of one. For more than about 40 topics, the distribution

of Kendall's-$\tau$ is approximately normal, with a mean of zero and a variance of $2(2n+5)/(9n(n-1))$ (where $n$ is the number of topics) [20]. Thus, in our case, significance ($P < 0.001$) is reached when Kendall's-$\tau$ is greater than 0.0046 (for 100 topics) or 0.0015 (for all 300 topics). Thus our results are extremely significant statistically.

We compared Kendall's-$\tau$ scores of the algorithms described in this article to some methods of estimation, previously suggested by some participants in the Robust track, namely:

1. Estimation based on the score of the top result [19].
2. Estimation based on the average score of the top ten results [15].
3. Estimation based on the standard deviation of the query term inverse document frequency (IDF) [21].
4. Estimation based on learning a Support Vector Machine for regression ([14]).

Table 2 shows the Kendall's-$\tau$ scores of those methods for both the 200 older queries and 49 new queries first provided in TREC 2004. For the 200 older queries we estimated the learning algorithms' performance using four-fold cross-validation, while for the 49 new queries, the 200 older queries were used for learning. As this table shows, the algorithms suggested in this article are much more precise compared to the other four algorithms. The improvement in Kendall's-$\tau$ scores shows that the suggested algorithms were not overfit to the training data.

The weights computed by linear regression in the histogram-based estimator represent the relative significance of the entries in the histogram. It is difficult to interpret the weights obtained by the linear regression. However, if we assume that each bin in the histogram can have a finite number of values then there are a finite number of possible states that the histogram can have. If an overlap of 10 is considered between the full query and the sub-queries, and assuming a binary histogram, there are a total of $2^{11} = 2048$ possible histogram states, corresponding to $2^{11}$ different histograms. By computing the estimated precision for each possible histogram some intuition regarding the weights (and the algorithm) is gained.

Thus we computed the estimated P@10 for each such histogram (using the linear estimation vector obtained using the title part of 249 topics from the TREC collection), ordered the histograms according to the estimation, and averaged each group of 128 consecutive histograms. The average histogram values for the first, the middle, and the last slice (each slice is an average of 128 individual histograms) are shown in Figure 2. This figure demonstrates that 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 the lower end of the histogram. This suggests that a good pattern of overlap would be one where the query is not dominated by a single sub-query. Rather, all (or at least most) sub-queries contribute somewhat to the final results.

| Estimation method | Title | | Description | |
|---|---|---|---|---|
| | 200 queries | 49 queries | 200 queries | 49 queries |
| Top score | 0.207 | 0.260 | 0.278 | 0.379 |
| Avg. top 10 | 0.129 | 0.211 | 0.255 | 0.294 |
| Std IDF | 0.222 | 0.110 | 0.186 | 0.243 |
| Ref [14] | | 0.223 | | 0.330 |
| Overlap+Tree | **0.305** | 0.201 | 0.218 | 0.294 |
| Overlap+Hist | 0.254 | **0.371** | **0.439** | **0.571** |

**Table 2: Comparison of Kendall's-$\tau$ scores (according to MAP) for several estimation methods, for the 200 older TREC queries and the 49 new TREC queries. Row 4 cites the highest Kendall's-$\tau$ scores of the method described in [14] (results for the 200 older queries are not given there).**

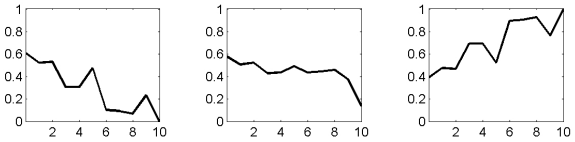

**Figure 2: Examples, from left to right, of the averaged histogram patterns, which generate the lowest, medium, and the highest estimation for P@10, respectively.**

# 5. APPLICATIONS

## 5.1 Application 1: Improving IR using query estimation

As mentioned above, query estimation 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 estimation can be implemented to attain this goal.

### 5.1.1 *Selective automatic query expansion*

Automatic query expansion (AQE) is a method for improving retrieval by adding terms to the query, based on frequently appearing terms in the top documents retrieved by the original query. However, this technique works only for easy queries, i.e., when the search engine is able to rank high the relevant documents. If this is not the case, AQE will add irrelevant terms, causing a decrease in performance [6].

Thus, it is not beneficial to use AQE for every query. Instead, it is advantageous to have a switch that will estimate when AQE will improve retrieval, and when it would be detrimental to it. Using the same features utilized for the histogram-based estimator, we trained an SVM classifier [10, 17] to decide when to use AQE and when not to. The SVM was trained with an RBF (Gaussian) kernel, with a width of 0.5.

### 5.1.2 *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 that in our system, some topics that are not answered well by the description part are better answered by the title part.

The estimator was used to decide which part of the topic should be used. The title part was used for difficult topics, i.e., those ranked (by the estimator) at the bottom 15% of the topics, while the description part was used for the remaining 85%.

### 5.1.3 Varying the parameters of the search engine according to query estimation

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.

We use a weight of 0.1 for lexical affinites of easy queries, i.e., queries ranked (by the estimator) at the top 15% of the queries, while the regular weight of 0.25 was used for the remaining 85% of the queries.

### 5.1.4 Results

The results of these methods on the 200 TREC queries are shown in Table 3. These results were obtained using four-fold cross-validation. Note that for a fair comparison, runs should be compared according to the parts of the queries they use.

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

## 5.2 Application 2: Detecting missing content

There are some queries for which all the results returned by the search engine are irrelevant. We define missing content queries (MCQs) as queries for which there is no relevant document in the document collection. A useful application of the query estimator is to identify MCQs.

As far as we know, there were no previous attempts to identify such queries. However, knowing if a query is an MCQ is useful for both the user and the search engine manager. The former will know if the document collection contains any answers to her query. The latter can note information that is of interest to his customers but is not answered by his sources of information.

We tested the possibility of using the query estimation for identifying MCQs by using a novel experimental protocol. The relevant documents for 166 queries from a set containing 200 description-part queries and 200 title-part queries were deleted from the TREC collection. Thus we artificially created 166 MCQs. A tree-based estimator was then trained

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

**Table 3: Improvements in retrieval based on query estimation. %no measures the percentage of queries for which no relevant document ranked in the top 10 documents.**

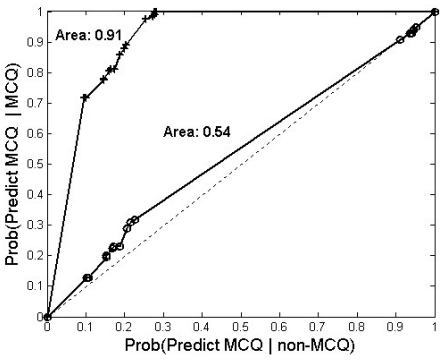


**Figure 3: Receiver operating characteristic (ROC) curve for distinguishing MCQs from non-MCQ queries. The line denoted by circles is of the MCQ classifier without prefilter, while the line denoted by pluses represents the ROC with prefilter.**

to classify MCQs from non-MCQs.

Our experiment consisted of two parts: In the first part, the estimator was trained using the complete set of 400 queries. In the second part, the query estimator trained in the previous sections was used as a prefilter of easy queries before the MCQ classifier. Ten-fold cross-validation was used throughout the experiment.

The results of our experiment are shown as a Receiver Operating Characteristic (ROC) curve in Figure 3. Different points on the graph represent different thresholds for deciding if a query is an MCQ or not. This figure shows that the MCQ classifier coupled with an estimator is extremely efficient at identifying MCQs. The fact that such a prefilter is needed (as demonstrated by the poor performance of the classifier without the prefilter) indicates that the MCQ classifier, while invoking by itself, groups together easy queries with MCQ queries. This is alleviated by prefiltering easy queries using the difficulty estimator.

## 5.3 Application 3: Merging the results in a distributed IR system according to difficulty estimation

Our third application for query estimation involves the setting of a distributed IR system. In this application, the search engine queries different datasets, and the results from the different datasets are merged to form a single ranking of documents.

The score for a document retrieved from a collection might or might not be provided. In the later case, only apriori knowledge about the datasets can be used. In the former, which we describe in this application, the score can be used as additional information for merging. However, it is difficult to rerank the documents since scores are local for each specific dataset. This can be avoided by computing global statistics [23], e.g., the idf of the query terms as though all datasets were merged to a single collection. Other approaches to the problem are through collection ranking, whereby each collection is given a score based on its statistics. This ranking is then used for merging the different rankings by weighting the scores. One of the state-of-the-art known algorithms that uses this approach is CORI [5]. In this article, we compare our results to the results achieved

using CORI.

The approach we describe below is based on the assumption that only minimal information is supplied by the search engine operating on a specific dataset, namely, the score of documents and the idf of all query terms. Thus the method we describe uses less information than CORI does. Given this data, our approach is to train a query estimator for each dataset. When a query is executed, the ranked list of documents is returned from each dataset and the estimation of query difficulty is computed for each dataset. The estimated difficulty is used for weighting the scores and the final ranking is built by merging the lists using these weighted scores.

We tested this approach using the TREC-8 collection. This collection is comprised of four different sub-collections: FBIS, FR94, FT, and LA-TIMES. We indexed each of these separately, and trained a tree-based estimator for each of these collections. Given a query, the Juru score returned for each document from its corresponding dataset was multiplied by the estimation. The final document ranking was generated by sorting the pool of retrieved weighted documents. Ten-fold cross-validation was used throughout the experiment.

Our initial results (see the first three lines of Table 4) showed that merging using the query estimator reached significant improvement in results compared to those obtained by simple merging, and comparable to CORI (one-tailed paired t-test, $P = 0.10$ for both methods).

We then clustered the queries based on their estimations for each of the datasets in order to see if there is a common pattern to those queries where weighting using the estimations improved results compared to queries where simple merging sufficed. Clustering was performed using the well-known k-means algorithm, implemented in [17]. The clustering revealed two distinct clusters. In one cluster, the variance of the estimations was small. This cluster corresponded to queries where unweighted scores were better for merging results. The second cluster contained queries where the variance of estimations was large (by an order of magnitude compared to the first cluster). This cluster corresponded to cases where the weighted merging was superior to unweighted merging.

Therefore a better merging scheme is to first test the variance of the estimations. If the variance is large, it is useful to use the estimations as a weight. Otherwise, simple merging suffices. The results of this scheme are shown in the last row of Table 4. These results are significantly better than CORI and simple merge alone (one-tailed paired t-test, $P = 0.07$ and $P = 0.05$, respectively).

We hypothesize that the necessity for a switch between weighted and unweighted merging is due to the fact that in cases where there is little variance in the estimations of difficulty, the actual difference comprises of noise rather than of information. In this case, it is better to ignore the estimator. However, when the estimator identifies one or more of the databases as better than the others, weighting is indeed useful.

| Merge method | P@10 | MAP | % no |
|---|---|---|---|
| Unweighted scores | 0.414 | 0.305 | 12.50 |
| CORI | 0.428 | 0.315 | 12.00 |
| Weighted scores | 0.430 | 0.315 | 10.75 |
| Selective weighted scores | 0.433 | 0.318 | 10.50 |

**Table 4: Results of different merging strategies.**

## 6. SUMMARY

In this work we describe two methods for learning an estimator of query difficulty. The learned estimator predicts the expected precision of the query by analyzing the overlap between the results of the full query and the results of its sub-queries. Experiments with TREC data demonstrated the robustness and the effectiveness of the learning methods. The estimators, trained over 200 TREC topics, were able to predict the precision of new unseen 49 topics, while the quality of prediction was equivalent to the quality of prediction for the trained topics. Thus our methods are robust and do not overfit with the training data.

We also showed that such an estimator can be used to improve the effectiveness of a search engine, by performing selective automatic query expansion for "easy" queries only, or by tuning the system parameters according to the expected query difficulty. We also described two other applications based on the estimator; identifying missing content queries (queries with no relevant results in the collection), and merging search results retrieved form distributed datasets.

There are still many open issues related to this research. Our results definitely show that the quality of query prediction strongly depends on the query length. Therefore, a question arises how the predictor can be improved for short queries, which are the type of queries that a search system is expected to serve. One of the direction we would like to consider is looking for additional features that indicate the query difficulty but on the other hand do not depend on the query length.

Another difficulty in the learning approach is the restricted amount of training data. There is much evidence in our experiments that the quality of prediction is increased with the number of training examples. Whether more training data for learning a predictor can be accumulated in automatic, or at least semi-automatic manner, is left for future research.

Following TREC's Robust track, we evaluated our predictors by measuring the similarity between the ranking of the training topics based on their actual precision, to their ranking based on the predicted precision. Therefore, we trained our predictors to maximize that measure. However, for real applications, an optimal predictor should estimate query precision independently of other queries. Hence, we think that in such setups alternative evaluation methodologies for query prediction should be considered.

## APPENDIX

The $\kappa$-statistic [7] is used for estimating the strength of agreement between two experts. It is estimated as follows:

$$\kappa = \frac{AG_{Observed} - AG_{Chance}}{1 - AG_{Chance}} \tag{4}$$

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 5, where $N_D$ is the number of documents in the index, and $Ov$ is the overlap (the number of documents that both queries agree should be in the top 10 results).

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

**Table 5: Agreement matrix for two queries.**

Based on this table, the $\kappa$-statistic is [7]:

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

In this case the $\kappa$-statistic is a linear function of the overlap. Thus, the query estimator learns a meta-statistic over the $\kappa$-statistic.

# A. 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. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. Chapman and Hall, 1993.

[4] C. Buckley. Why current IR engines fail. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 584–585. ACM Press, 2004.

[5] 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.

[6] 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.

[7] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, pages 37–46, 1960.

[8] S. Cronen-Townsend, Y. Zhou, and W. B. Croft. Predicting query performance. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 299–306. ACM Press, 2002.

[9] F. Diaz and R. Jones. Using temporal profiles of queries for precision prediction. In *SIGIR '04: Proceedings of the 27th annual international conference on Research and development in information retrieval*, pages 18–24. ACM Press, 2004.

[10] R. Duda, P. Hart, and D. Stork. *Pattern classification*. John Wiley and Sons, Inc, New-York, USA, 2001.

[11] D. Harman and C. Buckley. The NRRC reliable information access (RIA) workshop. In *SIGIR '04: Proceedings of the 27th annual international conference on Research and development in information retrieval*, pages 528–529. ACM Press, 2004.

[12] B. He and I. Ounis. Inferring query performance using pre-retrieval predictors. In A. Apostolico and M. Melucci, editors, *String Processing and Information Retrieval, 11th International Conference, SPIRE 2004*, volume 3246 of *Lecture Notes in Computer Science*, 2004.

[13] 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.

[14] K. Kwok, L. Grunfeld, H. Sun, P. Deng, and N. Dinstl. TREC 2004 Robust Track Experiments using PIRCS. In *Proceedings of the 13th Text REtrieval Conference (TREC2004)*, 2004.

[15] C. Piatko, J. Mayfield, P. McNamee, and S. Cost. JHU/APL at TREC 2004: Robust and Terabyte Tracks. In *Proceedings of the 13th Text REtrieval Conference (TREC2004)*, 2004.

[16] B. Scholkopf and A. Smola. *Leaning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT Press, Cambridge, MA, USA, 2002.

[17] D. Stork and E. Yom-Tov. *Computer manual to accompany pattern classification*. John Wiley and Sons, Inc, New-York, USA, 2004.

[18] B. Swen, X.-Q. Lu, H.-Y. Zan, Q. Su, Z.-G. Lai, K. Xiang, and J.-H. Hu. Part-of-Speech Sense Matrix Model Experiments in the TREC 2004 Robust Track at ICL, PKU. In *Proceedings of the 13th Text REtrieval Conference (TREC2004)*, 2004.

[19] S. Tomlinson. Robust, Web and Terabyte Retrieval with Hummingbird SearchServer at TREC 2004. In *Proceedings of the 13th Text REtrieval Conference (TREC2004)*, 2004.

[20] G. Upton and I. Cook. *Oxford dictionary of statistics*. Oxford university press, Oxford, UK, 2002.

[21] B. H. Vassilis Plachouras and I. Ounis. University of Glasgow at TREC 2004: Experiments in Web, Robust and Terabyte tracks with Terrier. In *Proceedings of the 13th Text REtrieval Conference (TREC2004)*, 2004.

[22] E. M. Voorhees. Overview of the TREC 2004 Robust Retrieval Track. In *Proceedings of the 13th Text Retrieval Conference (TREC-13)*. National Institute of Standards and Technology (NIST), 2004.

[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. Improving Document Retrieval According to Prediction of Query Difficulty. In *Proceedings of the 13th Text REtrieval Conference (TREC2004)*, 2004.