Adaptive Query Suggestion for Difficult Queries[*]

Yang Liu[⋆], Ruihua Song[†], Yu Chen[†], Jian-Yun Nie[‡] and Ji-Rong Wen[†]
[⋆] Beijing Institute of Technology,[†] Microsoft Research Asia,[‡]University of Montreal
yan9liu@gmail.com, {rsong,yu.chen,jrwen}@microsoft.com, nie@iro.umontreal.ca

ABSTRACT

Query suggestion is a useful tool to help users formulate better queries. Although this has been found highly useful globally, its effect on different queries may vary. In this paper, we examine the impact of query suggestion on queries of different degrees of difficulty. It turns out that query suggestion is much more useful for difficult queries than easy queries. In addition, the suggestions for difficult queries should rely less on their similarity to the original query. In this paper, we use a learning-to-rank approach to select query suggestions, based on several types of features including a query performance prediction. As query suggestion has different impacts on different queries, we propose an adaptive suggestion approach that makes suggestions only for difficult queries. We carry out experiments on real data from a search engine. Our results clearly indicate that an approach targeting difficult queries can bring higher gain than a uniform suggestion approach.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: query formulation

General Terms

Algorithms, Experimentation, Measurement

Keywords

Difficult queries, adaptive query suggestion, query suggestion evaluation

1. INTRODUCTION

It is often difficult for users to compose appropriate queries in Web search. Even if a query expresses well the right information need from the point of view of human beings, the query can still fail to retrieve the desirable documents. This is the case for the query "what's in fashion", which clearly describes the information need on learning the current fashion trends. However, most top search results are irrelevant, as shown in the top results from Google (`http://www.google.com`) in Table 1: the second result is irrelevant while the third one is not authoritative enough. The query is difficult because the current search methods cannot find the desired documents effectively. To be general, we consider all the queries with a low NDCG score *difficult queries* as in [17]. The low NDCG scores could be due to several reasons: the query's key terms mismatch those in the desired documents; the query is too general, the query is too narrow, and so on. No matter what the underlying reason is, one can alleviate the problem by suggesting appropriate alternative queries to the user, which can perform better in search.

Query suggestion is a technique which can assist users to interactively refine queries. However, most previous work on query suggestion, such as [29, 23, 7, 8], tries to identify alternative queries that bear a strong similarity or relevance to the original query. For easy queries, the suggested queries can often result in good search results, as is also the case for the original query. In such cases, query suggestion is less crucial, and can even be annoying sometimes (imagine, for example, the case when a user submits a very good query, but becomes uncertain when a number of alternatives are suggested). However, for difficult queries, it is critical but much harder to suggest queries that perform well. For example, for the same example query "what's in fashion", the suggestions "what's in fashion 2010" and "what's in fashion for men" are both relevant , but they are not more effective than the original query, as will be shown in Table 4. To be useful, a suggested query should not only be relevant to the original query, but also allow to better retrieve the desired documents.

Query reformulation is another technique aiming at improving the relevance of search results. Different from query suggestion, query reformulation is performed automatically on the original query without explicit interactions with the user. In such a setting, one should make sure that the automatic reformulation has a high precision. In practice, it is often limited to replacing some original query terms by correcting misspelled words, or suggesting some more frequently used terms that are slightly different from the original terms in morphology. However, such a replacement does not apply to the difficult queries we target in this paper such

[*]The work was done when the first author was visiting Microsoft Research Asia.

Table 1: One example of a difficult query, a better suggestion, and their top three search results returned by Google in November 4, 2011.

Query: "what's in fashion"	Suggestion: "latest fashion trends"
Fashion - Women's **Fashion** - **Fashion** Website Get a look at 2011 women's fashion. ELLE is the leading ... *www.elle.com/Fashion*	**Fashion** - Women's **Fashion** - **Fashion** Website Get a look at 2011 women's fashion. ... for seasonal ... *www.elle.com/Fashion*
Quiz: **What's** Your **Fashion** Style? What's Your Fashion Style? Find out what your likes ... *www.lhj.com/lhj/quiz.jsp?quizId=/.../WhatsYourStyleQuiz...*	**Fashion Trends** and News on Style.com NEW ON STYLE.COM. fashion shows. Spring 2012 ... *www.style.com/trendsshopping/*
Celebrity **Fashion**|**What's** hot|High Street Style Tips... OUR Fashion Ed's top fashion and beauty picks ... *www.thesun.co.uk/.../Celebrity-Fashion-Whats-hot-High-S...*	Trends - **Fashion Trends** (Vogue.com UK) Get a look at 2011 women's fashion. ... for seasonal ... *www.vogue.co.uk/fashion/trends*

as "what's in fashion", in which all the words are spelled correctly and are quite frequent.

In this paper, we do not intend to design a system that solves the problem of difficult queries alone; rather we propose an interactive suggestion method that adaptively suggests alternative queries when this is necessary or useful, namely for difficult queries. The goal is to suggest more effective queries for them. As an example, a good suggestion for the original query "what's in fashion" is "latest fashion trends", which both corresponds well to the original search intent and leads to better search results, as shown in Table 1.

As we hinted earlier, query suggestion can be useful in some cases (especially difficult queries), but annoying in some others (especially easy queries). It is then natural to determine when it is useful to perform query suggestion and to use it only for the useful cases. To do this, we use a regression model to predict the retrieval performance of queries and then return suggestions according to the query difficulty. Experimental results show that our adaptive query suggestion approach significantly improves the uniform query suggestion.

In addition to proposing an effective query suggestion method that targets difficult queries, we also propose some new measures to evaluate the quality of query suggestion, which we call Max@n (the maximal NDCG using the first n suggestions) and SDCG@n (the discounted cumulative gain using the first n suggestions). Compared to the measures used in previous studies, these measures can better reflect the use of query suggestions by end users.

The contributions of this work are twofold. First, to our best knowledge, it is the first time that query suggestion is studied specifically to help difficult queries. Second, we successfully demonstrate that our adaptive approach significantly improves the retrieval effectiveness of original queries and outperforms the current state of the art.

The remainder of this paper is organized as follows. We briefly review related work in Section 2. We propose two new evaluation measures in Section 3. Then, we describe our proposed approach for difficult queries in Section 4 and evaluate the approach in Section 5. The adaptive query suggestion approach is described and tested in Section 6. Finally, we present concluding remarks and future work in Section 7.

2. RELATED WORK

Our work is related to query suggestion, query reformulation and query performance prediction. We will review some work in these three areas.

2.1 Query Suggestion

In the past decade, many approaches have been proposed to perform query suggestion. Much work takes advantage of click-through information from query logs and leverages co-clicked URLs to identify related queries. For example, [6] constructed a bipartite graph based on click-through and clusters similar queries by assuming that queries for which the same documents are clicked on are similar. Here, clicked documents are considered as representing the user's search intent. [32] further extended the approach by also taking into account the content words of the queries. Therefore, queries in the same cluster are both similar to each other and share the same search intent. Baeza-Yates et al. [2] proposed to cluster similar queries by considering queries along with the text of their clicked URLs. Given an initial query, the queries from its cluster can be considered as possible suggestions.

In recent years, other types of useful information have been included into the bipartite graph. Mei et al. [24] proposed to compute hitting time on a large-scale bipartite graph mined from click-through data. Then candidate queries are ranked by the hitting time. Cao et al. [8] clustered queries into concepts and then by mapping user query sessions to concept sessions, they enhanced query suggestion by considering enriched contextual information. In [23], the authors proposed a query suggestion framework including two parts: The offline part builds a query similarity graph by user-query and query-click bipartite graphs; The online part applies a ranking algorithm to the query similarity graph and then suggests latent semantically relevant queries to users. To tackle the problem of rare query suggestion, Song et al. [28] built two bipartite graphs by leveraging both click and skip information from query logs and used an optimal random walk and combination model to determine query correlations.

Most of these approaches focus on enhancing user search experiences by providing related queries to expand searches [29]. The evaluation measure commonly used reflects whether the suggested queries are relevant to the original query. As we argued, a useful query suggestion is not only relevant, but also more effective. This is particularly important for difficult queries. To cope with this requirement, we will propose a new evaluation methodology in this paper.

2.2 Query Reformulation

Query reformulation techniques are widely used to modify user queries in order to improve retrieval effectiveness. Traditional methods include pseudo relevance feedback, which adds some terms extracted from the top ranked documents into the query [22, 30, 33]. Recent work exploits query logs to collect reformulation candidates. In [20], the authors identify query-level and phrase-level candidate substitutions for user queries by mining session data. Candidates are ranked according to two relevance measurements, i.e., "precise rewriting" and "broad rewriting". To address the problem of term mis-specification and under-specification within a query, Wang and Zhai [31] define two novel term association patterns, i.e., context-sensitive term substitution and term additions, and propose a method to discover these patterns by analyzing term co-occurrences in query logs. Gao et al. [15] propose a ranker-based search query speller that gathers correction candidates from query logs and then use a ranker trained from manually annotated data to rank the candidates. Guo et al. [16] propose a unified CRF model for query refinement by incorporating four independent techniques for query correction.

Anchor text is an alternative data source for query reformulation. It has been observed that there is a similarity between search queries and anchor texts [13]. Therefore, Kraft and Zien [21] propose a method to generate refinements for queries by mining anchor texts. They also employ a ranking algorithm for combining multiple factors to select query refinements. By using anchor texts to simulate click-through data of query log, Dang and Croft [12] employ the approaches described in [25, 31] to reformulate queries. Their results show that anchor texts are at least as effective as a real query log for this purpose.

The refinement work usually outputs alternative queries that will most likely change the user query's search results. To ensure high precision, the previous work often performs changes that are safe and hesitates to do risky refinements. The same technique can be hardly applied to difficult queries, for which more different suggestions are needed.

2.3 Query Performance Prediction

Query performance prediction is the task of estimating the quality of the search results for a query. Recently, a number of predictors have been proposed for this task [11, 1, 34, 10, 35, 36]. Carmel et al. [9] conducted a comprehensive comparison among these predictors over several TREC benchmarks and further discussed several methods for combing different predictors to obtain performance enhancement. Different from previous predictors evaluated on TREC-like collections, Balasubramanian et al. [5] propose an effective and efficient query performance prediction technique for real Web search that uses aggregates of retrieval scores and retrieval features. Our work is in the same setting. Therefore, we will use a similar approach to determine difficult queries in Section 6.

3. QUERY SUGGESTION AND QUALITY MEASURES

In this section, we first formulate the problem of query suggestion for difficult queries. Then we define measures to evaluate how good a suggestion list is for a difficult query.

3.1 Query suggestion problem

Query suggestion can be formulated as a two-step process: First, given a query q, a set of candidate queries $C = \{c_1, c_2, \ldots, c_m\}$ for suggestion are identified; then the candidates are ranked according to some quality criterion. Similar to the Probability Ranking Principle (PRP) [27], the best suggestion list can be seen as the one in which the suggested queries are ranked in decreasing order of their relevance probability $P(rel = 1|q, c)$, in which $rel = 1$ means relevance, q is the original query and c refers to a candidate. We then choose at most n ($n < m$) top candidate as suggestions to q, denoted by $C_s =< c_{s_1}, c_{s_2}, \ldots, c_{s_n} >$. The key problem is to find an optimal function $r(q, c_i)$ to estimate $P(rel = 1|q, c_i)$.

This is the standpoint taken in many previous studies. The measurements proposed are based on direct human judgments [29, 23, 7, 28, 8] or indirect human judged resources like Open Directory Project (ODP) data [23, 3]. For example, for each pair of the original query and a suggestion, assessors are given the queries and their corresponding search result pages side-by-side, and asked to make a binary decision of whether two queries are related [29].

As we discussed previously, a useful suggestion should be the one that improves the search effectiveness. Therefore, we change the previous relevance probability to the usefulness probability $P(useful = 1|q, c)$. For example, although both "what's in fashion for men" and "latest fashion trends" are relevant to the query "what's in fashion", the latter is more effective, thus its usefulness is higher. We will define two measures to reflect the usefulness in the next subsection.

Some measures in a similar vein have been used in query reformulation. For example, [12] used the metric precision@n, which is the fraction of top n retrieved documents that are relevant, to measure the quality of a refined query. In this paper, we will assume that our goal is to improve NDCG@k. Accordingly, we propose two measurements Max@n and SDCG@n to measure the effectiveness of a suggestion list.

3.2 Quality measures

3.2.1 *Evaluating an individual suggestion*

We use the judged document set of an original query q, which is pooled from several popular search engines, to evaluate a query suggestion c_{s_i}. Instead of precision@n, we choose NDCG to evaluate an individual suggestion, which is commonly used for Web search. Given an original query q, we have a set of documents, denoted as D, judged by human assessors. Then, NDCG@k is defined as $\frac{DCG@k}{IDCG@k}$, where DCG@k is calculated by as follows:

$$DCG@k = \sum_{i=1}^{k} \frac{2^{rating(i)} - 1}{log(1 + i)} \quad (1)$$

Here $rating(i)$ is the relevance rating of the document at position i. In our experiments, we have five grades of relevance, i.e., perfect, excellent, good, fair, and bad, corresponding to the ratings 4, 3, 2, 1, and 0, respectively. IDCG@k is the ideal discounted cumulated gain which is produced by the perfect ordering of D.

We evaluate the retrieval performance of a suggestion c_{s_i} by calculating NDCG@k measure for the list of retrieved documents $S(c_{s_i})$ based on the judgments of D. Note that although on average there are more than 100 documents judged for each original query, there are still some un-judged documents retrieved by suggested queries. We regard these un-judged documents as irrelevant, or bad, as is commonly done in IR evaluation.

3.2.2 *Evaluating a suggestion list*

Based on the NDCG measure of a single suggestion, we can measure the quality of a suggestion list. Usually search engines show at most ten suggestions for a query in rows at the bottom of search results or a list on the left side bar. The suggestion list is usually short so that the users can easily go through it and choose a suggestion that looks promising. Given a list of n suggestions, it is impossible to tell which of them will be selected by the user. Therefore, we use the maximum NDCG@k achievable by these n suggestions, denoted by Max@n, as a quality measure of the list.

For example, for an original query, the NDCG@3 values of its top five suggestions are:

$$< 0.4, 0.6, 0.5, 0.7, 0.2 >$$

Then Max@1 is 0.4; Max@2 is 0.6; Max@3 is 0.6; Max@4 is 0.7; and Max@5 is 0.7. By connecting these Max@n points, we can use a monotonically increasing curve to describe these values.

We also use the metric SDCG@n to measure the overall quality of a suggestion list, which assumes that the user scans the suggestion list from top to bottom. This is similar to the assumption used in the general DCG measure. SDCG@n is defined as follows:

$$SDCG@n = \sum_{i=1}^{n} \frac{NDCG@k(i)}{log(1+i)} \quad (2)$$

Here n is the total number of suggestions in a suggestion list and NDCG@k(i) is the quality of the suggestion at position i.

As search engines often return less than ten suggestions for a query, it is unfair to compare them with our methods if they return less suggestions. Thus, we choose Max@n and SDCG@n with $n \leq 5$ as our main metrics and use the queries with at least five suggestions in our experiments.

4. OUR APPROACH

In this section, we describe our approach to address the problem of query suggestion for difficult queries. Figure 1 shows the work flow of our approach.

Given a query q, we first retrieve candidates based on clicks and query terms. Then we use a search system to obtain search result pages for the original query q and candidates $\{c_1, c_2, \ldots, c_m\}$ [1]. For each pair of q and c_i, we extract features to predict how well the candidate will perform. Our proposed features are extracted from queries, candidates, and their search result pages. Next, we apply a ranking model $f(\mathbf{x}_{c_i})$ to estimate $P(useful = 1|q, c_i)$. The model is learned on a training set composed of difficult queries. Finally, we sort the candidates by their estimated scores and return top n candidates as suggestions.

We will describe the above processes in detail in the following subsections.

4.1 Retrieving Candidates

Candidates can be collected from many data sources. In our work, we mine query clusters from click-through data and retrieve candidates based on query clusters.

First, we build a click-through bipartite graph from the search logs collected on Bing (`http://www.bing.com`) from January 1st to March 25th, 2010. An edge e_{ij} is created between a query node q_i and a URL node u_j if u_j has been clicked when users issued q_i. The weight w_{ij} of edge e_{ij} is the aggregated click number. The query q_i is then represented as an L_2-normalized vector, in which each dimension corresponds to a URL. If edge e_{ij} exists, the value of the dimension is normalized w_{ij}; otherwise, it is zero.

[1]Note that the search results of the candidates can be prefetched offline.

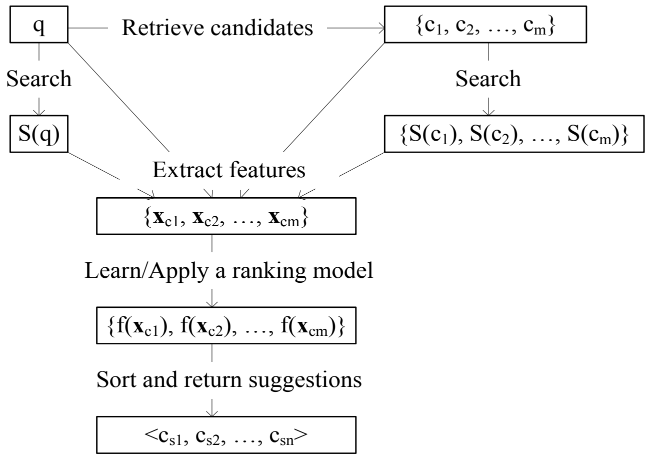


Figure 1: Flowchart of generating suggestions for a query

Second, we apply the clustering method proposed in [8]. The algorithm creates a set of clusters as it scans through the queries. For each query q, the algorithm first finds the closest cluster C, and then tests the diameter of $C \cup \{q\}$. If the diameter is not larger than D_{max}, q is merged into C. Otherwise, a new cluster is created with q as its first member.

Third, we divide queries in each query cluster into intent groups. When examining query clusters, it is observed that the search intents of some queries in a query cluster are almost the same. Despite the slight difference in their forms, they can be safely treated as duplicates to one another. This phenomenon is mainly caused by misspellings, with or without stop words, different tenses, equivalent syntax, and so on. For example, “jet bleu”, “jetbue” and “jetb” share the same intent with “jetblue”. It is annoying if we show these misspelled or duplicate queries as suggestions. To solve this problem, we apply a sequence of transformation operations, such as spelling correction, stop words removal, stemming, and term sorting, to all queries in each cluster and then group two queries together if their edit distance after the transformation operations is less than a threshold. Then, we select the most frequent query in each group to be the group leader, which will be returned as a candidate on behalf of the whole group.

Given an original query q, we identify the query-cluster map to find the cluster containing q. All the group leaders in the cluster are returned as candidates for q.

4.2 Extracting Features

We extract some features to measure how well candidate c_i performs. These features will be used within a learning-to-rank framework (see the next section). The features are extracted from $\langle q, S(q), c_i, S(c_i)\rangle$, where $S(q)$ and $S(c_i)$ are search results returned by Bing for q and c_i respectively. In our experiments, we use the first page containing top ten search results for each query.

Our proposed features can be divided into four categories: match features, cross match features, similarity features and an estimated NDCG feature.

4.2.1 Match Features

The match features aim to measure how well a candidate matches its own search result. Our intuition is that only well-matched candidates are qualified to be suggestions.

Match features are extracted from c_i and $S(c_i)$. As each result in $S(c_i)$ is usually composed of three parts: title, snippet, and URL, we calculate match features for each of them. Take title as an example. Given the candidate c_i and a title $T_{i,j}$, which is the title of the j-th result in $S(c_i)$, we first remove stop words by Fox's list [14] and stem terms by Porter Stemmer [26]. Then we count how many times a term t_k of c_i occurs in the title and normalize the term frequency (TF) by the length of title in words:

$$MatchScore(c_i, T_{i,j}) = \sum_{t_k \in c_i} \frac{TF(t_k, T_{i,j})}{length(T_{i,j})} \tag{3}$$

Finally, we calculate the feature of TitleMatch by aggregating all titles from the top N results and discounting MatchScore by the position of the title as follows:

$$TitleMatch = \sum_{j=1}^{N} \frac{MatchScore(c_i, T_{i,j})}{log(j+1)} \tag{4}$$

Similar to TitleMatch, we also calculate the features of SnippetMatch and URLMatch. The three features compose our match features.

4.2.2 Cross Match Features

The cross match features aim to measure how well a candidate's search result $S(c_i)$ matches the original query q. The intuition is that a search result $S(c_i)$ that matches better the original query is more likely to correspond the original query's information need.

Cross match features are extracted from q and $S(c_i)$. Similar to the match features, we calculate three features based on title, snippet, and URL. For example, $MatchScore(q, T_{i,j})$ represents the cross match feature on the title of the j-th result in $S(c_i)$. TitleCrossMatch can be computed as:

$$TitleCrossMatch = \sum_{j=1}^{N} \frac{MatchScore(q, T_{i,j})}{log(j+1)} \tag{5}$$

SnippetCrossMatch and URLCrossMatch can be computed similarly.

4.2.3 Similarity Features

The similarity features are other features trying to measure the similarity between a candidate and the original query. This similarity is measured on the candidate's search result $S(c_i)$ and the original query's search result $S(q)$. On the one hand, as we want $S(c_i)$ to satisfy the original query's information need, the topic of $S(c_i)$ should not drift too much away from $S(q)$. On the other hand, $S(c_i)$ would fail to provide new and better results if it is too similar to $S(q)$.

To cover the two aspects, we extract three similarity features on the result pages, URLs and domains. PageSimilarity tries to prevent from a possible topic drift. It is calculated by the cosine similarity between the vectors of $S(q)$ and $S(c_i)$. The vectors are formed by terms in the search results after stopword removal and stemming. TF-IDF formula is used to weigh the terms. URLSimilarity is estimated by the number of common URLs between $S(q)$ and $S(c_i)$ and DomainSimilarity is based on the number of common domains, which are derived from URLs, between the two search results.

4.2.4 Estimated NDCG Feature

As we discussed earlier, difficult queries are more in need of suggestions. The estimated NDCG feature tries to capture, to some extent, how relevant a set of search results is to the original query, by which we want to reflect the difficulty of the query. This estimate is made using the sets of search results from different query formulations. The intuition is that, if a document appears in the top search results of many candidates, a.k.a. voted by many candidates, it is very likely to be relevant. Therefore, an estimate of relevance can be obtained for each search result.

Given q, the top search results of q's candidates form a search result collection $\bigcup_{i=1}^{m} S(c_i)$. For each unique document in the collection, we count the number of times that the candidates return it among top ten as the document's estimated relevance ratings. For a candidate c_i, every document in $S(c_i)$ will have an estimated rating. Consequently, we can calculate the estimated NDCG for the candidate using Equation 1, in which $rating(i)$ is the estimated relevance rating instead.

4.3 Learning to Rank Suggestions

We use a pairwise learning-to-rank method, RankSVM [19] to rank the candidates. RankSVM focuses on the relative order between two items in a ranking list and its objective of learning is to directly minimize the number of item pairs with reverse order. Given a list of candidates, RankSVM outputs the prediction score for each candidate, which can be used to rank candidates by sorting them on the prediction score in descending order.

Formally, suppose a candidate set $\mathcal{C} = \{(\boldsymbol{x_i}, y_i) | \boldsymbol{x_i} \in \mathbb{R}^d, y_i \in R^1\}$ and let y_i be the retrieval performance, e.g., NDCG@3, of a candidate $\boldsymbol{x_i}$. The ranking function has the form:

$$f(\boldsymbol{x}) = \sum_{i=1}^{l} (-\alpha_i) y_i K(\boldsymbol{x_i}, \boldsymbol{x}) + b^* \tag{6}$$

where $K(\cdot)$ is the kernel function. In our experiments, we choose the RBF (radial basis function) kernel. Notice that RankSVM tries to rank queries according to its search effectiveness (y_i) rather than their relevance or relatedness to the original query. A higher-ranked candidate is the one that is more effective in search. This allows us to capture the desired usefulness.

We define the error function for incorrect pairwise ordering as follows [4]:

$$\Theta_f(\boldsymbol{x_i}, \boldsymbol{x_j}) = \begin{cases} 1, & \text{if } \mathrm{sign}((f(\boldsymbol{x_i}) - f(\boldsymbol{x_j})) \neq \mathrm{sign}(y_i - y_j) \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

Then, the optimal ranking function f^* can be learned by minimizing the overall ranking errors:

$$f^* = \underset{f}{\mathrm{argmin}} \sum_{\boldsymbol{x_i} \in \mathcal{C}} \sum_{\boldsymbol{x_j} \in \mathcal{C}} \Theta_f(\boldsymbol{x_i}, \boldsymbol{x_j}) \tag{8}$$

Note that we only use the difficult queries, which perform worse than a threshold (e.g., 0.4 in this paper) in terms of NDCG@3, in a training set to learn ranking models.

5. EXPERIMENTS ON SUGGESTION APPROACHES

In this section, we conduct experiments on a set of real Web search queries to evaluate the effectiveness of our proposed approach and compare it with several baseline approaches.

5.1 Data Collection

The dataset collected contains about 10,000 real Web queries from the search logs of Bing. On average, each query has more than 100 documents judged in five relevance grades. As our evaluation will use the top 5 suggestions, to be fair, we discard the queries for which the baseline approaches or our approach returns less than five suggestions. Finally we have 4,068 queries for experimentation.

We fetch top three search results from Bing for each of the original 4,068 queries, and calculate NDCG@3 to measure how difficult each original query is. We divide the original queries into 10 bins according to their NDCG@3 values:$[0, 0.1), [0.1, 0.2), \ldots, [0.9, 1]$. The number of queries in each bin is shown in Figure 2. As we can see, there are a quite large number of queries (1,019) in the four lowest bins below 0.4. These queries account for around 25% of the total queries. Their low NDCG values indicate that it is difficult to retrieve relevant documents for them with the original queries, and they need better suggestions to improve search.

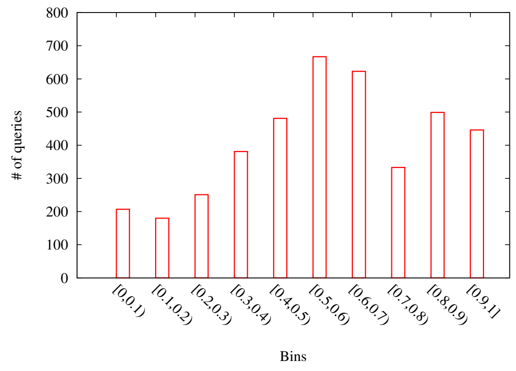


Figure 2: Distribution of original queries in different bins of retrieval performance

For all the 4,068 original queries, our approach identifies 638,391 suggestion candidates from the three-month search log. On average, each original query has about 157 candidates. Again, we fetch the top three search results from the search engine for each candidate and compute their NDCG@3 values based on the relevance judgments of the original query. If a candidate has a higher NDCG@3 value than its original query, it is an *improved candidate*. Figure 3 shows the average number of candidates and improved candidates per original query in each NDCG bin. As we can see, there are a larger number of improved queries in the lower bins than in the higher bins. For the $[0, 0.1)$ bin, each query has 33 improved candidates on average. The number of improved candidates decreases progressively as the original queries become easier. For the original queries in the bins of $[0.8, 0.9)$ and $[0.9, 1]$, almost none of their candidates is better than the original ones. This is not a surprise because search engines already perform very well in returning relevant documents for the easy queries, and hence there is less space left for further improvement. The above observation supports our conjecture that query suggestion is more necessary for difficult queries. It is on difficult queries that we can obtain large improvements on search effectiveness.

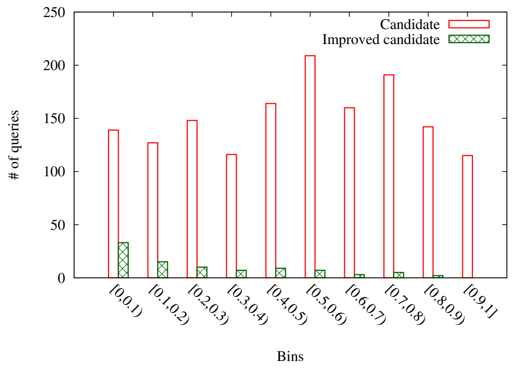


Figure 3: Number of candidates per query and number of improved candidates per query in different bins

As additional baselines, we also evaluate the suggestions from two commercial search engines, SE1 and SE2, for the original queries. For a suggestion from SE1 or SE2, we fetch the top three search results from them to evaluate its effectiveness in NDCG@3 .

5.2 Evaluating Ranking Models

We first conduct experiments to investigate the ranking models that are learnt from different types of features. The 4,068 original queries are randomly divided into ten subsets, and we conduct ten-fold cross validation for all learning experiments in this paper. In each trial, the queries in nine subsets are used as the training set, and the remaining subset as the testing set. In order to train a RankSVM model specifically for difficult queries, we only preserve the queries whose NDCG values are below 0.4 in the training set. For testing, however, we use all the original queries in order to reflect the fact that we cannot know their NDCG values in advance. The reported performance measures are averaged on the ten trials.

Figure 4 shows the results measured by Max@1 for the first six bins. Due to limited space, we do not show the results for original queries in other bins and other Max@n, but the trend is similar and has been clearly presented in Figure 4. First, we find that cross match features perform consistently better than match features. This observation is intuitive because the cross match features capture how well a candidate matches the original query. Second, we observe that the estimated NDCG feature is the most significant type in the bin of $[0, 0.1)$, more effective than the similarity measures. This is interesting and consistent with our assumption that for difficult queries we need to suggest queries that are more different from the original queries. In contrast, when queries become easier, the similarity measures stand out as the most effective criteria. This is also intuitive — easy queries do not need suggestions that are very different from the original ones; otherwise, there is a high risk of topic drift. However, we also observe that the combined approach that uses all types of feature (**All**) does not always outperform the one using the similarity measures only, namely in the bins $[0.3, 1]$. This can be explained by the fact that the models are trained on the difficult queries only (from bins below 0.4). During the training process, some of the features may be over-used and their possible negative impact on the queries in other bins may not be observed during the training. It would be more appropriate to use all the queries in the training in order to separate a model for difficult queries from the one for easy queries. We will leave this to future work.

The very good performance of similarity measures for easy queries does provide a good indication that these measures can be used as a good model for easy queries. It is then intuitive to combine the model trained on all the features (which performs well on difficult queries) and the one with similarity features only.

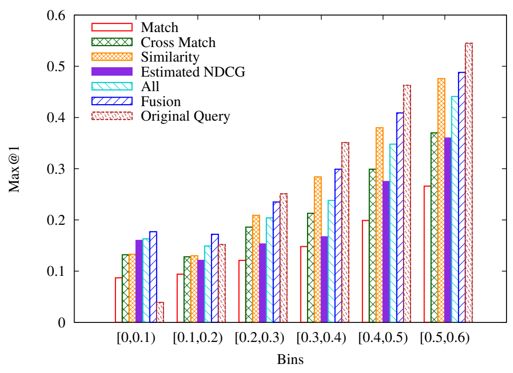


Figure 4: Evaluating ranking models based on different type of features over original queries in different bins

We propose using a variant of Borda's ranking fusion method [18] to combine the two models as follows:

$$Score(c) = \lambda \frac{1}{\sqrt{r_a(c)+1}} + (1-\lambda)\frac{1}{\sqrt{r_s(c)+1}} \quad (9)$$

where r_a is the position of candidate c in the list of suggestions returned by the model based on all features, and r_s is the position of c in the list returned by the model based on similarity features. λ is a trade-off parameter to balance the two models. The experimental results show that for a wide range of λ, from 0.3 to 0.7, the fusion outperforms both models. In our remaining experiments, we empirically set λ to 0.5.

The fusion method is consistently the best among our proposed methods as shown in Figure 4. This indicates that the simple fusion method is capable of taking advantage of both models for difficult and easy queries. We also calculate the difference between the fusion method and the two separate models in terms of Max@1-5 for all queries and SDCG@5 for queries in different bins. The results are shown in Table 2 and Table 3 respectively. In all the cases except one, the fusing approach improves the two separate models, and the improvements are statistically significant. Therefore, we use the fusion method as our query suggestion approach in remaining experiments.

5.3 Comparing Our Approach with Baselines

We compare our fusion approach with several baseline approaches. We collect suggestions from **SE1** and **SE2** as two additional baselines representing the current state of the art for query suggestion in commercial search engines. Also we randomly choose five queries from candidates to form suggestions. We denote this baseline as **Random**. The comparison results are evaluated by Max@1-5 and SDCG@5, shown in Table 2 and Table 3. As our method contains a candidate identification step followed by a candidate selection step, in order to see the impact of each of them, we also report the case when 5 suggestions are randomly chosen from the identified candidates (Random), i.e. we only use the first step.

As shown in the two tables, our fusion approach is dramatically better than any baseline in terms of all Max@1-5 and SDCG@5. The differences are statistically significant. This indicates that the existing query suggestion technologies used in commercial search engines have not well solved the problem of difficult queries. A possible reason may be that too much emphasis is put on the similarity between the suggestions and the original query. As we observed, the similarity criterion does not work well for difficult queries. To illustrate this, we show two examples in Table 4. In each example, we show an original query and top five suggestions returned by three approaches along with their NDCG@3 values. Given the query "what's in fashion", SE1 and SE2 provide some related queries that are subtopics of the query by adding a few key terms. Unfortunately, they fail to find better results on fashion because of the ineffective original query terms. Our approach suggests queries that are more different, such as "latest fashion trends" and "fashion now", which remove some ineffective key terms, like "what's in", and add a few better ones, like "trends", "latest", and "now". In terms of NDCG@3, our suggestions can improve the retrieval performance from 0.1564 to 0.5307. For the query "gallery furniture", SE1 and SE2 provide specific furniture brands that are related to the original query. However, there is a clear topic drift. Our fusion approach is more effective and the suggestions are more specific and within the scope of the original query.

In Table 2, we observe that although our approach is the best one among all suggestion approaches, it performs worse than original queries in terms of Max@1 and Max@2, meaning that we have to use at least 3 suggestions in order to catch up and improve the original queries. In order to better understand the situation, we have a closer look into the six bins in Figure 5. It turns out that our fusion approach works better for queries that are more difficult. For the most difficult queries (in the bins of $[0, 0.2)$) there is already improvements on Max@1, i.e. the first suggestion is already better than the original queries. When we extend to larger bins in $[0, 0.3)$, it still achieves improvement starting from Max@2. For easier queries, it becomes more difficult to improve the result by one or two suggestions only. However, it consistently improves the original queries from Max@3. This means that the approach is capable of proposing better queries for these bins when we allow at least 3 suggestions.

Nevertheless, we observe a clear trend: the easier the original query, the more difficult to propose a better suggestion. It is then intuitive to suggest queries according to the difficulty of the query. We investigate such an approach in the next section.

Table 2: Max@1-5 of suggestion approaches over all 4,068 queries. We conduct paired t-test for the difference between our fusion method with any other method and bold face indicates statistically significant with $p < 0.01$.

Max@n	1	2	3	4	5
Fusion	0.497	0.553	0.58	0.594	0.604
All	**0.442(-11%)**	**0.504(-9%)**	**0.536(-8%)**	**0.558(-6%)**	**0.573(-5%)**
Similarity	**0.485(-2%)**	**0.541(-2%)**	**0.568(-2%)**	**0.585(-2%)**	**0.596(-1%)**
Random	**0.205(-59%)**	**0.307(-44%)**	**0.372(-36%)**	**0.413(-30%)**	**0.443(-27%)**
SE1	**0.214(-57%)**	**0.296(-46%)**	**0.347(-40%)**	**0.386(-35%)**	**0.415(-31%)**
SE2	**0.172(-65%)**	**0.252(-54%)**	**0.299(-48%)**	**0.331(-44%)**	**0.357(-41%)**
Original Queries	0.572(15%)	0.572(3%)	**0.572(-1%)**	**0.572(-4%)**	**0.572(-5%)**

Table 3: SDCG@5 of suggestion approaches over queries in different bins. We conduct paired t-test for the difference between our fusion method with any other method and bold face indicates statistically significant with $p < 0.01$.

SDCG@5	$[0, 1]$	$[0, 0.1)$	$[0.1, 0.2)$	$[0.2, 0.3)$	$[0.3, 0.4)$	$[0.4, 0.5)$	$[0.5, 0.6)$
Fusion	1.358	0.456	0.451	0.63	0.805	1.104	1.36
All	**1.255(-8%)**	0.481(5%)	**0.427(-5%)**	**0.585(-7%)**	**0.694(-14%)**	**1.008(-9%)**	**1.271(-7%)**
Similarity	**1.329(-2%)**	**0.343(-25%)**	**0.406(-10%)**	**0.575(-9%)**	**0.766(-5%)**	**1.062(-4%)**	**1.338(-2%)**
Random	**0.618(-54%)**	**0.245(-46%)**	**0.22(-51%)**	**0.315(-50%)**	**0.357(-56%)**	**0.482(-56%)**	**0.634(-53%)**
SE1	**0.556(-59%)**	**0.265(-42%)**	**0.242(-46%)**	**0.305(-52%)**	**0.386(-52%)**	**0.486(-56%)**	**0.574(-58%)**
SE2	**0.429(-68%)**	**0.250(-45%)**	**0.265(-41%)**	**0.321(-49%)**	**0.329(-59%)**	**0.39(-65%)**	**0.372(-73%)**

Table 4: Two queries and their suggestions returned by our fusion approach, SE1, and SE2 together with their NDCG@3. All suggestions are changed to lowercase.

Our Approach		SE1		SE2	
Query: what's in fashion (NDCG@3=0.1564)					
latest fashion trends	0.5307	what in fashion 2011	0	fashion trends	0
fashion now	0.4693	fashion what in 2010	0	what's in fashion 2010	0.2961
hottest casual fashion trends women	0.2346	what's in fashion now 2010	0	what's in fashion for men	0
fashion trends women	0.2961	what's in fashion fall 2010	0.1564	what's in fashion for teens	0
styles cool fashion	0	what in fashion this summer	0	what's in fashion 2009	0.1564
Query: gallery furniture (NDCG@3=0.2939)					
gallery furniture address	0.3728	carson pirie scott furniture gallery	0	fingers furniture	0
gallery furniture bedroom	0.2939	ashley furniture gallery	0	ashley furniture	0
gallery furniture sales you money card	0.2346	the dump furniture store	0	star furniture	0
gallery furnisher	0.2387	macy's furniture galleries	0	furniture store	0
gallery furniture post oak	0.2387	fingers furniture	0	hilton furniture	0

6. ADAPTIVE QUERY SUGGESTION

To make good suggestions for all the queries, it is important to distinguish difficult queries from easy ones. A key problem is to predict how difficult a query is. Although we have used a simple feature (Estimated NDCG) to reflect the query difficulty in the learning-to-rank model, there is still the need to determine query difficulty with a more sophisticated method and explicitly use it to vary the strategy of query suggestion. We will carry out experiments using such a prediction of query difficulty.

6.1 Learning to Predict Difficult Queries

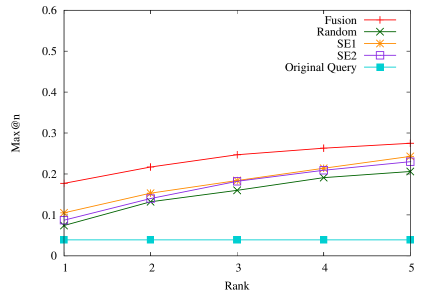
Various methods have been proposed to predict query difficulty. Any effective predictor can be used in our experiments. We choose to use the RAPP method proposed in [5]. The key idea behind this approach is to use the ranking scores, as well as the features that are used for ranking documents (e.g. BM25, click and PageRank), to predict the quality of the results. We re-implement the approach of [5].

We conduct three-fold cross validation to train a regressor using all queries, including easy queries and difficult queries. The predicted difficulty value for a query q is denoted by $g(q)$. If the query q is judged as difficult according to $g(q)$, we return suggestions that are generated by our fusion approach to users. This leads to an adaptive approach for query suggestion, i.e. more query suggestions for difficult queries than for easy queries.

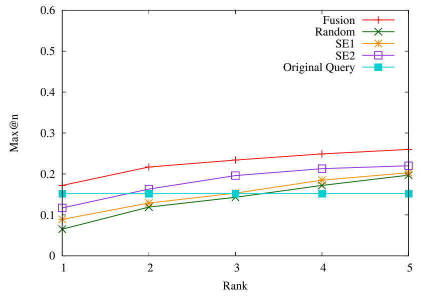
6.2 Experiments on Adaptive Query Suggestion

To quantitatively test the idea, we assume that a certain suggestion budget (the number of suggestions) is allowed for the entire query set, and our goal is to distribute the budget on different queries according to their difficulty. The budget is defined by m suggestion slots per query on average. The total suggestion slots available for all our test queries is then defined by $4,068 \times m$. To make our test easier, we will assume that whenever a query is considered to be worthy for suggestions, we propose 5 suggestions. This is in line with our evaluation method that uses the top 5 suggestions. Therefore, we will select to perform query suggestion on $n = 4,068 \times m \div 5$ most difficult queries. For the other queries, no suggestion will be provided. This evaluation schema is of course a simplification of the real situation; but it does reflect the latter to some extent.

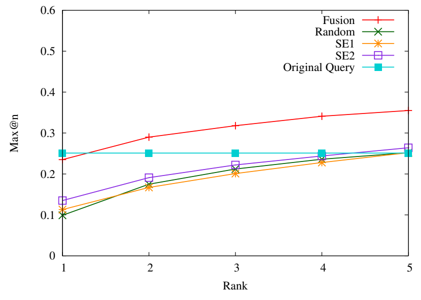
For a budget of m slots per query, we evaluate our adaptive query suggestion approach and the fusion approach on the Max@ns metric. Here ns is the total number of the



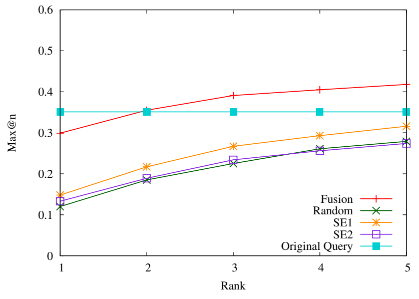
(a) $[0, 0.1)$



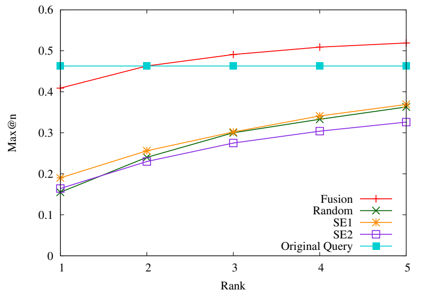
(b) $[0.1, 0.2)$



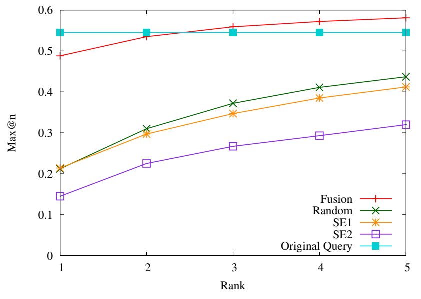
(c) $[0.2, 0.3)$



(d) $[0.3, 0.4)$



(e) $[0.4, 0.5)$



(f) $[0.5, 0.6)$

Figure 5: Effectiveness comparison among different approaches in different bins

suggestions assigned to a query. For the queries without suggestion, we use NDCG@3 of the original query to calculate Max@ns. Figure 6 shows the average Max@ns scores of the adaptive approach (**Adaptive**) and the fusion approach (**Fusion**), along with different number of suggestions: $m = 1 \ldots 5$. We also plot the average NDCG@3 values of all original queries (**Original Query**) in the figure.

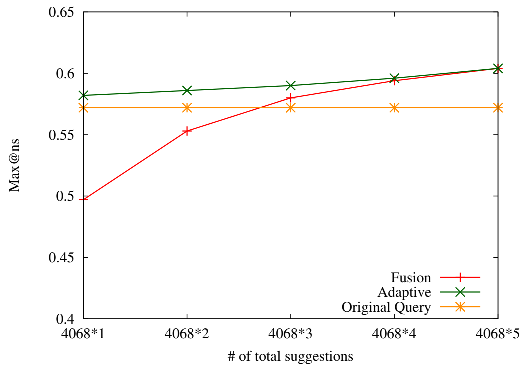


Figure 6: Comparing adaptive query suggestion with the fusion approach and the original queries with budget control

As shown in Figure 6, our adaptive approach consistently outperforms the fusion approach and the original queries. Specifically, when m is small (one or two), the fusion approach cannot suggest queries better than the original ones. One can only find better queries when there are 3 slots or more. This is because the slots are uniformly distributed over all the queries, including on easy queries which do not need suggestions. The adaptive approach successfully avoids this problem. It can target more difficult queries for which query suggestion is more useful. Its improvements over the original queries are statistically significant for all budget settings.

The above simulation provides a good indication that one can gain more in targeting difficult queries for query suggestion. However, we assumed an equal number (5) of suggestions for all the queries whose $g(q)$ is lower than a threshold. In practice, this strategy can be improved by proposing different numbers of suggestions according to $g(q)$. We will leave this to future work.

7. CONCLUSION

Query suggestion is widely used by search engines. Although it is found useful, the method is used uniformly on all the queries. In this paper, we show that query suggestion is more beneficial for difficult queries than for easy queries. To the best of our knowledge, this is the first investigation of query suggestion according to query difficulty.

In this paper, we used a learning-to-rank method to learn to rank suggestion candidates by using different features. Then an adaptive approach is proposed to provide suggestions according to the estimation on query difficulty. The experiments reported in this paper show that:

1. There is much more to gain in proposing query suggestions for difficult queries;
2. To be useful, the suggestions for more difficult queries should be less similar to the original query;
3. An adaptive suggestion according to query difficulty is the most useful approach;

In addition to the above conclusions, we also proposed two new evaluation measures that take into account the possible improvements in NDCG by the suggested queries, as well as their ranking in the suggestion list. These measures can better reflect the use of suggestions by end users than the previous measures.

This work is a first step towards adaptive query suggestion. There is much room for future improvements. First, in order to gain more insights on query suggestion for the whole spectrum of queries, we may need to train different, difficulty-dependent, query suggestion models. That is, for each category of queries, a specific model is trained to predict whether it is useful to suggest queries and what type of query is the most useful. Second, the true usefulness of different query suggestion methods should be tested with true users. Third, the proposed method can be used in different tasks. For example, as we have shown, some suggested queries lead to better search results. It is then possible to utilize our suggestions to automatically re-rank the search results of difficult queries.

8. REFERENCES

[1] G. Amati, C. Carpineto, and G. Romano. Query difficulty, robustness, and selective application of query expansion. In *ECIR*, pages 127–137, 2004.

[2] R. A. Baeza-Yates, C. A. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *EDBT Workshops*, pages 588–596, 2004.

[3] R. A. Baeza-Yates and A. Tiberi. Extracting semantic relations from query logs. In *KDD*, pages 76–85, 2007.

[4] N. Balasubramanian, G. Kumaran, and V. R. Carvalho. Exploring reductions for long web queries. In *SIGIR*, pages 571–578, 2010.

[5] N. Balasubramanian, G. Kumaran, and V. R. Carvalho. Predicting query performance on the web. In *SIGIR*, pages 785–786, 2010.

[6] D. Beeferman and A. L. Berger. Agglomerative clustering of a search engine query log. In *KDD*, pages 407–416, 2000.

[7] S. Bhatia, D. Majumdar, and P. Mitra. Query suggestions in the absence of query logs. In *SIGIR*, pages 795–804, 2011.

[8] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *KDD*, pages 875–883, 2008.

[9] D. Carmel and E. Yom-Tov. Estimating the query difficulty for information retrieval. In *SIGIR*, page 911, 2010.

[10] D. Carmel, E. Yom-Tov, A. Darlow, and D. Pelleg. What makes a query difficult? In *SIGIR*, pages 390–397, 2006.

[11] S. Cronen-Townsend, Y. Zhou, and W. B. Croft. Predicting query performance. In *SIGIR*, pages 299–306, 2002.

[12] V. Dang and W. B. Croft. Query reformulation using anchor text. In *WSDM*, pages 41–50, 2010.

[13] N. Eiron and K. S. McCurley. Analysis of anchor text for web search. In *SIGIR*, pages 459–460, 2003.

[14] C. Fox. Lexical analysis and stoplists. *Information Retrieval - Data Structures & Algorithms*, pages 102–130, 1992.

[15] J. Gao, X. Li, D. Micol, C. Quirk, and X. Sun. A large scale ranker-based system for search query spelling correction. In *COLING*, pages 358–366, 2010.

[16] J. Guo, G. Xu, H. Li, and X. Cheng. A unified and discriminative model for query refinement. In *SIGIR*, pages 379–386, 2008.

[17] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[18] J.C.Borda. Mémoire sur les élections au scrution. *Histoire de l'Acadêmie Royal des Sciences*, 1781.

[19] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, pages 133–142, 2002.

[20] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *WWW*, pages 387–396, 2006.

[21] R. Kraft and J. Y. Zien. Mining anchor text for query refinement. In *WWW*, pages 666–674, 2004.

[22] V. Lavrenko and W. B. Croft. Relevance-based language models. In *SIGIR*, pages 120–127, 2001.

[23] H. Ma, H. Yang, I. King, and M. R. Lyu. Learning latent semantic relations from clickthrough data for query suggestion. In *CIKM*, pages 709–718, 2008.

[24] Q. Mei, D. Zhou, and K. W. Church. Query suggestion using hitting time. In *CIKM*, pages 469–478, 2008.

[25] F. Peng, N. Ahmed, X. Li, and Y. Lu. Context sensitive stemming for web search. In *SIGIR*, pages 639–646, 2007.

[26] M. Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137, 1980.

[27] S. E. Robertson. The probability ranking principle in ir. *Journal of Documentation*, 33:130–137, 1977.

[28] Y. Song and L. wei He. Optimal rare query suggestion with implicit user feedback. In *WWW*, pages 901–910, 2010.

[29] Y. Song, D. Zhou, and L. wei He. Post-ranking query suggestion by diversifying search results. In *SIGIR*, pages 815–824, 2011.

[30] T. Tao and C. Zhai. Regularized estimation of mixture models for robust pseudo-relevance feedback. In *SIGIR*, pages 162–169, 2006.

[31] X. Wang and C. Zhai. Mining term association patterns from search logs for effective query reformulation. In *CIKM*, pages 479–488, 2008.

[32] J.-R. Wen, J.-Y. Nie, and H. Zhang. Clustering user queries of a search engine. In *WWW*, pages 162–168, 2001.

[33] J. Xu and W. B. Croft. Improving the effectiveness of information retrieval with local context analysis. *ACM Trans. Inf. Syst.*, 18(1):79–112, 2000.

[34] E. Yom-Tov, S. Fine, D. Carmel, and A. Darlow. Learning to estimate query difficulty: including applications to missing content detection and distributed information retrieval. In *SIGIR*, pages 512–519, 2005.

[35] Y. Zhou and W. B. Croft. Ranking robustness: a novel framework to predict query performance. In *CIKM*, pages 567–574, 2006.

[36] Y. Zhou and W. B. Croft. Query performance prediction in web search environments. In *SIGIR*, pages 543–550, 2007.