Context-Sensitive Query Auto-Completion*

Ziv Bar-Yossef*
Google Inc.
MATAM, Bldg 30
POB 15096, Haifa 31905, Israel
zivby@ee.technion.ac.il

Naama Kraus
Computer Science Department
Technion
Haifa 32000, Israel
nkraus@cs.technion.ac.il

ABSTRACT

Query auto completion is known to provide poor predictions of the user's query when her input prefix is very short (e.g., one or two characters). In this paper we show that context, such as the user's recent queries, can be used to improve the prediction quality considerably even for such short prefixes. We propose a context-sensitive query auto completion algorithm, NearestCompletion, which outputs the completions of the user's input that are most similar to the context queries. To measure similarity, we represent queries and contexts as high-dimensional term-weighted vectors and resort to cosine similarity. The mapping from queries to vectors is done through a new query expansion technique that we introduce, which expands a query by traversing the query recommendation tree rooted at the query.

In order to evaluate our approach, we performed extensive experimentation over the public AOL query log. We demonstrate that when the recent user's queries are relevant to the current query she is typing, then after typing a single character, NearestCompletion's MRR is 48% higher relative to the MRR of the standard MostPopularCompletion algorithm on average. When the context is irrelevant, however, NearestCompletion's MRR is essentially zero. To mitigate this problem, we propose HybridCompletion, which is a hybrid of NearestCompletion with MostPopularCompletion. HybridCompletion is shown to dominate both NearestCompletion and MostPopularCompletion, achieving a total improvement of 31.5% in MRR relative to MostPopularCompletion on average.

Categories and Subject Descriptors: H.3.3: Information Search and Retrieval.

General Terms: Algorithms.

Keywords: query auto-completion, query expansion, context-awareness.

*Despite the affiliation of the first author, this paper did not rely on privileged access to Google's resources, data, or technologies. The entire research was done at the Technion.
*Also an adjunct senior lecturer at the Technion, Department of Electrical Engineering.

WWW 2011, March 28–April 1, 2011, Hyderabad, India.
ACM 978-1-4503-0632-4/11/03.

1. INTRODUCTION

Query auto completion [4, 23, 14, 3] is one of the most visible features in Web Search today. It is offered by all major search engines and in almost all their search boxes. Query auto completion helps the user formulate her query, while she is typing it. Its main purpose is to predict the user's intended query and thereby save her keystrokes. With the advent of instant as-you-type search results (a la the recently released Google Instant[1]), the importance of correct query prediction is even more acute, because it determines the speed at which the user sees the suitable results for her intended search and the amount of irrelevant results that are displayed to her along the way.

The basic principle that underlies most query auto completion systems is the wisdom of the crowds. The search engine suggests to the user the completions that have been most popular among users in the past (we call this algorithm MostPopularCompletion). For example, for the prefix *am*, Bing suggests *amazon* and *american express* as the top completions, because these have been the most popular queries starting with *am*. As the user is typing more characters, the space of possible completions narrows down, and thus the prediction probability increases. For example, if the user is looking for *american presidents* in Bing, after typing the 14 characters *american presi* the desired query becomes the top completion.

Clearly, during the first few keystrokes the user is typing, the search engine has little information about her real intent, and thus the suggested completions are likely to mispredict her query. In our experiments, conducted over the public AOL query log [19], we found that after the first character, MostPopularCompletion's average MRR is only 0.187.

The objective of this study is to tackle the most challenging query auto completion scenario: after the user has entered only one character, try to predict the user's query reliably. Being able to predict the user's query on her first character rather than, say, on her 10-th character would not only save the user a few keystrokes, but would also make the whole search experience more interactive, as the feedback cycle of (query → results → query refinement) would be shortened significantly. In addition, cutting down the search time for all users implies lower load on the search engine, which translates to savings in machine resources and power.

But how can we overcome the inherent lack of information when the user has entered only a few characters of her

[1] http://www.google.com/instant/

intended query? Our main observation is that the user typically has some *context*, which can reveal more information about her intent. For example, if just before entering the characters *am* the user searched for *richard nickson*, it is more likely that the user is looking for *american presidents* than for *amazon* or *american airlines*. Similarly, if the user was browsing a page about President Lincoln or reading an article about american history. On the other hand, if the user has just tweeted about a planned trip, *american airlines* might be the more probable query. Recent queries, recently visited web pages, and recent tweets are examples of online activities that may indicate the user's intent and, if available, could be used by the search engine to better predict the query even after a few keystrokes. This is called *context-sensitive query auto completion*. While the idea is very intuitive and context has been used in other scenarios to disambiguate user intent (e.g., in search [16, 11] and in query recommendations [10, 13, 9, 6, 21]), there is almost no published work on its application to query completion.

Of the many different possible user contexts, our focus in this study is on the user's recent queries (within the same session), as they are readily available to search engines. Based on our empirical analysis of the AOL log, 49% of the searches are preceded by a different search in the same session, and are thus amenable to context-sensitive query completion.

One possible approach to use recent queries to improve query auto completion is to generalize MostPopularCompletion to rely on the popularity of query sequences rather than just the popularity of individual queries. Suppose the user's previous query in the same session is y and that the current user input (query prefix) is x. Then, the search engine will suggest the completions of x that were most frequently searched for after y. For example, if after the query *richard nixon* the most popular successive query starting with *am* is *american presidents*, the search engine will suggest *american presidents* as its top completion. This is in fact the main principle underlying most of the work on context-sensitive query recommendations [10, 13, 9, 6, 21]. The main caveat of this approach is that it heavily relies on the existence of reoccurring query sequences in search logs. Nevertheless, due to the long-tail distribution of query frequencies, many of the query sequences generated by users have never occurred before (by our estimate, 89% of the query pairs are new).

Some studies tried to mitigate the sparsity of query sequences by clustering similar query sequences together, based on features extracted from queries, like their topical categories or their terms [10, 9]. Machine learning techniques, like HMMs, are then used to predict the intended query, if the sequence of previous queries can be associated with a cluster in the model. This approach is still challenged by long-tail contexts, i.e., when the most recent query (or queries) have rarely occurred in the log (by our estimates, in 37% of the query pairs the former query has not occurred in the log before). In this case, the sequence of previous queries may not be easily associated with a cluster in the model. Moreover, none of these previous studies took the user input (prefix) into account in the prediction, so their applicability to query auto completion is still unknown.

We take a different approach to tackle this problem. Our algorithm relies on the following similarity assumption: when the context is relevant to the intended user query, the intended query is likely to be *similar* to the context queries. The similarity may be syntactic (e.g., *american airlines* → *american airlines flight status*) or only semantic (e.g., *american airlines* → *continental*). By our estimates, 56% of the refinements are non-syntactic.

Based on the similarity assumption, we propose the NearestCompletion algorithm, which works as follows: given a user input x and a context y, the algorithm suggests to the user the completions of x that are most similar to y. Choosing the suitable similarity measure is non-trivial, though, because we would like it to be both *correlated with reformulation likelihood* (that is, the more similar two queries A and B are the more likely they are to be reformulations of each other, and vice versa) and *universally applicable* (that is, the similarity is meaningful for any pair of queries A and B). The former requirement guarantees that the completions that are similar to the context are indeed more likely to be the user's intended query. The latter requirement makes sure that the algorithm can deal with any user input.

The above two requirements make many of the state-of-the-art query similarity measures less appealing for this problem. For example, syntactic measures, like edit distance, do not take all reformulation types into account. Similarity measures that are based on co-occurrence in search sessions [24, 12], on co-clicks [2, 10], or on user search behavioral models [6, 18, 9, 21], are not universally applicable to all query pairs due to their low coverage of queries, as long tail queries are rare in the query log. Similarity measures that are based on search result similarity [8] are not necessarily correlated with reformulation likelihood.

Given a context, query recommendation algorithms [2, 24, 21] output a list of recommendations that are likely reformulations of the previous query. So a possible similarity measure would be one that associates each query with its recommendations. The main caveat with this approach is that query recommendation algorithms are frequently designed to output only a few high quality recommendations and thus it is plausible that none of them are compatible with the user's input. Hence, this technique is not universally applicable.

We propose a new method of measuring query similarity, which expands on the latter recommendations based approach, but is universally applicable and is thus more suitable to query completion. Similarly to the result-based similarity of Broder *et al.* [8], we *expand* each query to a richer representation as a high-dimensional feature vector and then measure cosine similarity between the expanded representations. The main novelty in our approach is that the rich representation of a query is constructed not from its search results, but rather from its *recommendation tree*. That is, we expand the query by iteratively applying a black box query recommendation algorithm on the query, on its recommendations, on their recommendations, and so on. The nodes of the traversed tree of recommendations are tokenized, stemmed, and split into n-grams. These n-grams are the features of the expanded representation vector and the weight of each n-gram is computed based on its frequency and depth in the tree.

The above representation has two appealing properties. First, as the basic building block in the construction is a black-box query recommendation algorithm, we can leverage any state-of-the-art algorithm and inherit its power in predicting query reformulations. Second, the above scheme

provides a continuous spectrum of exceedingly rich representations, depending on the depth of the tree traversed. For example, a depth-0 traversal results in the n-grams of the root query itself, while a depth-2 traversal results in the n-grams of the query, its recommendations, and their recommendations. The main point is that the feature space remains the same, regardless of the traversal depth. So even if we cannot traverse the recommendation tree of a certain query (e.g., because it's a long-tail query for which there are no recommendations available), the similarity between its representation and the richer representation of other queries is meaningful. This property ensures that our query auto completion algorithm is applicable even for long-tail contexts that have never been observed in the log before.

Our empirical analysis shows that the average MRR of NearestCompletion (with depth-3 traversal) over queries whose context is relevant is 48% higher relative to the average MRR of MostPopularCompletion over the same queries. However, when the context is irrelevant to the intended query, NearestCompletion becomes destructive, so its average MRR is 19% lower relative to the average MRR of MostPopularCompletion over all queries. To mitigate this problem, our final algorithm, HybridCompletion, is a hybrid of MostPopularCompletion and NearestCompletion. Each of the two algorithms provides a list of top k matches. We aggregate the two lists by standardizing the contextual score and the popularity score of each candidate completion and then computing a final score which is a convex combination of the two scores. The completions are ranked by these final scores. We show that HybridCompletion dominates both NearestCompletion and MostPopularCompletion. Its average MRR is 31.5% higher relative to the average MRR of MostPopularCompletion.

As our new algorithm relies on the standard cosine similarity measure between vectors, it can be implemented efficiently over standard high-performance search architectures. This is a crucial property of the algorithm, because query auto completions need to be provided to the user in a split-second as she is typing her query. Note that the rich representation of the recent queries can be cached and retrieved quickly as the user is typing her current query. The current query requires no enrichment.

The rest of the paper is organized as follows. After reviewing some related work in Section 2, we provide a brief background about query auto completion algorithms in Section 3. We describe the NearestCompletion algorithm in Section 4 and the HybridCompletion algorithm in Section 5. In Section 6 we provide a detailed empirical study of the two algorithms and compare them to MostPopularCompletion. We end with some concluding remarks in Section 7.

2. RELATED WORK

Query auto-completion has received relatively little attention in the literature (see, e.g., [23, 4, 3, 14, 1]). All major search engines rely on query logs to generate query auto-completions. To the best of our knowledge, there is no published work on context-sensitive log-based query auto-completion. Arias *et al.* [1] suggest a context-sensitive query auto-completion algorithm for mobile search. Their completions, however, are thesaurus-based concepts whose relatedness to the user's context is determined by a rule-based mechanism. This approach does not seem to fit the scalability requirements of web search.

Similarly to query completions, query recommendations assist users in phrasing their intent. Numerous query recommendation algorithms have been introduced, relying on varied techniques, including topic clustering [2, 21, 5], query co-occurrence analysis [12], session analysis [24, 13, 9], and user search behavioral models [18, 21, 6, 22]. Query recommendation algorithms and query auto-completion algorithms differ in their input (a full query vs. a prefix of a query) and in their output (arbitrary reformulations of the query vs. (mostly) completions of the user's prefix). Note that our framework can leverage any state-of-the-art query recommendation algorithm to construct its rich query representations.

Several context-sensitive query recommendation algorithms have been proposed recently. Boldi *et al.* [6] compute query recommendations by running *Personalized PageRank* on their *Query Flow Graph*. As the mass of PageRank's teleportation vector is concentrated on the context queries, the recommendations generated are context-sensitive. The run time efficiency of this algorithm is questionable, as PageRank computation is heavy. Cao *et al.* [10, 9] and He *et al.* [13] train models for query sequences based on analysis of such sequences in search logs (using, among others, some machine learning models, like Variable Length HMM and Mixture Variable Memory MM). At run time, these models are used to predict the next user's query from her previous queries. It is not totally clear how these techniques deal with long-tail contexts that have never occurred in the log before. In comparison, our algorithms are adapted to query auto-completion, can deal with long tail contexts, have scalable runtime performance, and are robust to irrelevant contexts.

The framework we use in this paper for query auto completion is close to the one used by Broder *et al.* [8] to expand long-tail queries for the purpose of matching relevant ads. Broder *et al.* built an index of enriched representations of popular and semi-popular queries and in run time retrieved the queries that are most similar to the current user query. These similar queries are then used to retrieve relevant ads. Apart from the different applications and the different evaluation methodologies, the two frameworks differ in their core component: the rich query representation. Broder *et al.* mostly rely on their result-based query expansion approach [7], while we focus on traversal of the query recommendation tree.

Finally, query expansion is a well established field (see [17] for an overview). Classical methods expand the query using thesauri. This is limited and non-scalable. Roccio's method [20] expands the query from terms that occur in its search results. This is non-trivial to do without adding also a lot of noise that causes a "topic drift". More modern methods rely on query log analysis (e.g., [15]). Since query recommendations usually rely on such query expansion techniques as building blocks, our query expansion algorithm can be viewed as bootstrapping state-of-the-art expansion techniques to obtain richer vocabularies.

3. QUERY AUTO COMPLETION

A query auto completion (QAC) algorithm accepts a *user input* x, which is a sequence of characters typed by the user in the search engine's search box. The user input is typically a prefix of a complete query q that the user intends to enter. The algorithm returns a list of k *completions*, which are suggestions for queries, from which the user can select.

A completion c is said to be a *hit*, if it equals the query q that the user was about to enter. In this paper we will focus on hits as the main measure of success for QAC algorithms, as it is relatively easy to estimate hit rates when inspecting search logs. In reality, a QAC algorithm may be successful even if it returns a completion that is different from the query the user was about to type but that describes the same information need.

Most QAC algorithms share the following framework. In an offline phase a *query database* is built. The database consists of a large collection of queries that are of high quality and represent the intents of the search engine's users. Major search engines build this database from their query logs by extracting the most frequently searched queries. Smaller search engines, which do not have sufficient user traffic, construct the database from prominent phrases that occur in the corpus being searched.

Each QAC algorithm defines its own criteria for determining whether a query q is an eligible completion for an input x. Traditional QAC algorithms require that q is a proper string completion of x (i.e., that x is a prefix of q). For instance, *barack obama* is a proper completion of the input *bar*. Advanced QAC algorithms support also non-proper completions, like mid-string completions (e.g., *ob* $\rightarrow$ *barack obama*) and spell corrections (e.g., *barak* $\rightarrow$ *barack obama*). We will denote the set of queries that are eligible completions of an input x by completions(x).

At run-time, the QAC algorithm accepts an input x, and selects the top k eligible completions for x. Completions are ordered by a *quality score*, which represents how likely each completion is to be a hit. Since the algorithm needs to provide the user with the suggested completions as she is typing the query, it has to be ultra-efficient. To achieve this high performance, the algorithm needs a data structure, like a TRIE or a hash table, that supports fast lookups into the query database using prefix keys.

MostPopularCompletion is the standard and most popular QAC algorithm. The quality score it assigns to each query q is the frequency of this query in the log from which the query database was built. That is, for an input x, MostPopularCompletion returns the k completions of x that were searched for most frequently.

One way to look at this algorithm is as an approximate Maximum Likelihood Estimator. Let p be the probability distribution of incoming queries. That is, $p(q)$ is the probability that the next query the search engines receives is q. In an ideal world, the search engine would have known $p(q)$ for all q ahead of time. Given a user input x, the search engine could have then applied Maximum Likelihood Estimation to predict the user's query q as follows:

$$MLE(x) = \text{argmax}_{q \in \text{completions}(x)}\, p(q).$$

MostPopularCompletion essentially implements this algorithm, except that it approximates the next-query distribution p by the normalized frequencies of queries in the log.

4. NEAREST COMPLETION

As we will see in Section 6, MostPopularCompletion frequently fails to produce hits when the user input is still very short (say, 1-2 characters long). The NearestCompletion algorithm that we introduce next uses the user's recent queries as *context* of the user input x. When the context is relevant to the query the user is typing, the algorithm has better chances of producing a hit.

Search sessions. A *logical search session* is an interactive process in which the user (re-)formulates queries while searching for documents satisfying a particular information need. It consists of a sequence of queries $q_1, \ldots, q_t$ $(t \geq 1)$ issued by the user. The *context* of a user input x, where x is the prefix of some query q_i in the session, is the sequence of queries $q_1, \ldots, q_{i-1}$ preceding q_i. Note that if x is the prefix of the first query in the session, its context is empty.

Since all the queries in a logical session pertain to the same information need, the context of a user input is always relevant to this input by definition. In reality, however, detecting logical sessions is non-trivial as a user may switch her information need within a short time frame. Mis-detection of logical search sessions leads to mis-detection of contexts. In this section we ignore this problem and assume we have a perfect session detector, so contexts are always relevant. We will address this problem in the next section.

The reader may wonder at this point how a QAC algorithm that runs on the search engine's server can access the user's recent queries at run-time. The most straightforward solution is to keep the user's recent queries in a cookie, if the user agrees to it. Every time the user performs a search, the search engine returns the results and also updates a cookie (that the browser stores on the user's machine) with the latest search. When the user types characters in the search engine's search box, the browser sends the user's input along with the cookie to the search engine.

The basic algorithm. A context-sensitive extension of the Maximum Likelihood Estimator described above works as follows. Let $p(q|C)$ denote the conditional probability that the next query is q given that the current context is C. If the search engine would have known $p(q|C)$ for all q and C, it could have predicted the next query as follows:

$$MLE(x|C) = \text{argmax}_{q \in \text{completions}(x)}\, p(q|C).$$

Approximating $p(q|C)$ is trickier than approximating $p(q)$, because query logs are too sparse to provide meaningful estimates of $p(q|C)$ for most (q, C) pairs. We tackle this problem by casting it as an information retrieval (IR) problem: we treat the context C as a "query" and the queries in the query database as "documents". Indeed, our goal is to sift through the many possible completions of x and find the ones that are most related to the context C.

Looking at this problem through the IR prism, we can now resort to standard IR techniques. We chose to implement NearestCompletion using the traditional *Vector Space Model*, which is supported by the search library we used in our experiments. Each context C is mapped to a term-weighted vector v_C in some high dimensional space V. Similarly, each query q in the query database is mapped to a vector $v_Q \in V$. NearestCompletion then outputs the completion q of x whose vector v_q has the highest cosine similarity to v_C:

$$\text{NearestCompletion}(x, C) = \text{argmax}_{q \in \text{completions}(x)}\, \frac{\langle v_q, v_C \rangle}{||v_q|| \cdot ||v_C||}.$$

More generally, NearestCompletion outputs the k completions of x whose vectors are most similar to v_C.

Context representation. Contexts and queries are objects of different types, so it may not be clear how to rep-

resent both as vectors in the same space. However, since contexts are sequences of queries, then we can produce context representations from query representations. Formally, if $C = q_1, \ldots, q_t$ is a context and $v_{q_1}, \ldots, v_{q_t}$ are the corresponding vectors, we produce the context vector v_C as a linear combination of the query vectors:

$$v_C = \sum_{i=1}^{t} w_i v_{q_i}.$$

The weights $w_1, \ldots, w_t$ are non-negative. They specify the relative contribution of each context query to the context vector. As the more recent a context query is, the more likely it is to be relevant to the current query, the weights need to be monotonically non-decreasing. In our empirical analysis we experimented with different weight functions: recent-query-only ($w_t = 1$ and $w_i = 0$ for all $i < t$), linear decay ($w_i = 1/(t - i + 1)$), logarithmic decay ($w_i = 1/(1 + \ln(t - i + 1))$), and exponential decay ($w_i = 1/e^{t-i}$)).

Vector representation via query expansion. The most important ingredient of the NearestCompletion algorithm is the representation of queries as term-weighted vectors. This representation eventually determines how NearestCompletion ranks the completions of x. Ideally, we need this ranking to be consistent with the ranking of completions by the conditional probabilities $p(q|C)$. Note that $p(q|C)$ is the probability that the user reformulates the context queries C to the current query q. Hence, the vector representation of queries needs to yield a similarity measure so that similar queries are likely reformulations of each other and vice versa.

A naive approach would be to represent a query by its terms, as a bag of words. The resulting similarity measure can capture syntactic reformulations, such as *my baby is not eating well* → *baby eating disorder*, but not semantic refinements, like *baby eating disorder* → *infant nutrition*. The problem is that queries are short, and thus their vocabulary is too sparse to capture semantic relationships.

In order to overcome this sparsity problem, we expand each query into a *rich representation*. *Query expansion* [17] is used to augment the textual query with related terms, like synonyms. For example, the query *baby eating disorder* may be expanded to *baby infant eating food nutrition disorder illness* and the query *infant nutrition* may be expanded to *infant baby nutrition food*. The two expanded forms now have high cosine similarity.

Recommendation-based query expansion. We introduce a new query expansion technique that leverages a large body of work on *query recommendation*. A query recommendation algorithm suggests to the user high quality reformulations of her query. Query recommendation algorithms rely on existing query expansion techniques and in addition are tuned to provide likely reformulations of the query. Hence, a plausible expansion of a query could be the list of recommendations provided for it by such an algorithm. The advantage is that the added terms are high precision keywords that appear in likely reformulations of the query. It follows that two queries that have similar expanded forms of this sort share their reformulation vocabulary and are thus more likely to be reformulations of each other.

The above approach produces high precision query expansions, but may lack coverage. If the query recommendation algorithm produces a small number of recommendations for each query (as most such algorithms do), then the resulting expanded forms would be too sparse. To overcome this problem, we expand each query not just by its direct recommendations but rather by the whole recommendation tree rooted at this query.

Formally, let A be a query recommendation algorithm, and let us denote by $A(q)$ the top k recommendations that A provides for a query q.

Definition 4.1 (Query recommendation tree). *Let $d \geq 0$ be an integer. Let $T_{q,d}$ denote the depth-d* query recommendation tree *of query q. The root node of $T_{q,d}$ corresponds to the query q. The children of each node v in the tree correspond to the recommendations for v ($A(v)$).*

Note that the same query may occur multiple times in the tree and possibly at different levels of the tree.

The query recommendation tree is the main building block in the construction of the expanded form v_q of a query q. The coordinates of v_q correspond to n-grams that occur within the queries in the tree. n-grams are extracted as follows: each query in the tree is tokenized into terms, stop-words are eliminated, and the terms are stemmed. The resulting queries are split into overlapping n-grams, where $n = 1, \ldots, N$ and N is an upper bound on the size of n-grams we care about.

Let z be an n-gram. If z was not extracted by the above process, then $v_q[z] = 0$. If z was extracted, let $T(z)$ denote the nodes of the tree that contain z. We define the weight of z in v_q as follows:

$$v_q[z] = \left(\sum_{u \in T(z)} \text{weight}(\text{depth}(u)) \right) \cdot \ln(IDF(z)).$$

This is essentially a TF-IDF weight. The sum counts the number of occurrences of z in the tree, but it assigns different weights to different occurrences. The weight of an occurrence depends on the depth of the node in which it is found. Note that the deeper the node is, the weaker is its connection to the root query q, and hence the function weight$(\cdot)$ is monotonically non-increasing. We experimented with various weighting functions, including linear decay, logarithmic decay, and exponential decay. $IDF(z)$ is the inverse frequency of z in the entire query database.

System architecture. Since NearestCompletion relies on the Vector Space Model, it can leverage standard and optimized information retrieval architectures. In the offline phase, the algorithm computes the rich representation of each query in the query database. The resulting vectors are indexed in an inverted index. In addition, each query is indexed by its set of eligible prefixes, so one can retrieve all completions of a given prefix quickly (note that wildcard operator that is supported by standard search architectures can also be used for prefix-based retrieval).

In run time, the algorithm accepts the user input x and the context C. The rich representation of the queries that constitute C should be available in a cache, because these queries have been recently processed by the search engine. The algorithm can therefore compute the rich representation v_C of C. It then retrieves from the index the queries that are completions of x and whose rich representation is most similar to v_C.

5. HYBRID COMPLETION

NearestCompletion is designed to work well when the user input has a non-empty context and this context is relevant to the query that the user is typing. In practice, however, many queries have no context (51% by our experiments). In addition, due to incorrect segmentation of search sessions, recent queries that are deemed as context may not be relevant to the current query at all (by our experiments, 40% of the query pairs have different information needs). In all of these cases NearestCompletion relies either on no information or on false information and thus exhibits poor quality.

On the other hand, the standard MostPopularCompletion algorithm is not dependent on context, and thus can do well even if the context is empty or irrelevant. It would have been nice if one could identify these cases and use MostPopularCompletion instead of NearestCompletion in them. Recognizing that the context is empty is easy. However, how can one detect that the context is irrelevant, at run time? HybridCompletion circumvents this problem by using *both* algorithms when the context is non-empty.

Given a user input x and a context C, HybridCompletion produces two ranked lists of completions of x: L_{NC} consists of the top ℓ completions returned by NearestCompletion and L_{MPC} consists of the top ℓ completions returned by MostPopularCompletion. The final ranked list of completions L_{HC} is constructed by aggregating the two lists.

The results in each of the two lists are ranked by quality scores: L_{NC} is ranked by a similarity score, which we denote by simscore$(\cdot)$, and L_{MPC} is ranked by a popularity score, which we denote by popscore$(\cdot)$. The aggregated list L_{HC} is constructed by combining the two scoring functions into a single hybrid score, denote hybscore$(\cdot)$. As simscore and popscore use different units and scales, they need to be standardized before they can be combined. In order to standardize simscore, we estimate the mean similarity score and the standard deviation of similarity scores in the list L_{NC}. The standard similarity score is then calculated as

$$\text{Zsimscore}(q) = \frac{\text{simscore}(q) - \mu}{\sigma},$$

where μ and σ are the estimated mean and standard deviation. The standard popularity score is calculated similarly. The hybrid score is defined as a convex combination of the two scores:

$$\text{hybscore}(q) = \alpha \cdot \text{Zsimscore}(q) + (1 - \alpha) \cdot \text{Zpopscore}(q),$$

where $0 \leq \alpha \leq 1$ is a tunable parameter determining the weight of the similarity score relative to the weight of the popularity score. One can think of α as the prior probability that the next query has a relevant context and thus would require a context-sensitive completion. Note that when $\alpha = 0$ HybridCompletion is identical to MostPopularCompletion, and when $\alpha = 1$ HybridCompletion is identical to NearestCompletion (except for inputs that have empty context). In Section 6, we experiment with different values of α in order to tune it appropriately.

HybridCompletion, as described above, uses one global value for α, which is common to all inputs and contexts. There may be scenarios though where it is desired to alter the value of α adaptively. For example, if we have some indication that the context is not relevant to the current user input, we may want to reduce α, while if we have the opposite indication, we may want to increase α. Since the focus of this paper is not on session segmentation and context relevancy detection, we have not addressed this direction.

It is important to note that HybridCompletion may rerank the original lists of completions it receives. For example, among the most popular completions, it will promote the ones that are more similar to the context, and, conversely, among the most similar completions, it will promote the more popular ones. This implies that HybridCompletion can dominate both MostPopularCompletion and NearestCompletion not only on average, but also on individual inputs.

6. EMPIRICAL STUDY

Our empirical study has two goals: (a) compare the best configuration of our algorithms to the standard MostPopularCompletion algorithm; and (b) study the effect of the different parameters of our algorithms on the quality of the results. To this end, we came up with an automatic evaluation methodology for QAC algorithms, which estimates their MRR based on a given query log. We used the AOL query log [19] in our experiments, as it is publicly available and sufficiently large to guarantee statistical significance (other public query logs are either access-restricted or are small).

Experimental setup. For performing the empirical study we implemented the standard MostPopularCompletion algorithm and our two algorithms (NearestCompletion and HybridCompletion). The query database used by all algorithms was constructed from the queries that appear in the AOL log. We segmented the log into sessions, using a simple standard segmentation heuristic (every interval of 30 minute idle time denotes a session boundary). We eliminated from the data all click information and merged duplicate queries that belong to the same session. The final data sets consisted of 21,092,882 queries in 10,738,766 sessions. We found that 40% of the sessions were of length greater than 1 and 49% of the queries were preceded by one or more queries in the same session (hence being amenable to context-sensitive QAC).

We partitioned the AOL log into two parts: a training set, consisting of 80% of the log, and a test set, consisting of the remaining 20%. We computed a rich representation (see Section 4) for a subset of 55,422 of the training set queries. The query recommendation trees required for these rich representations were built using Google Suggest's query autocompletion service. We scraped the auto completions using the public external service (we did not rely on privileged access to internal Google services or data). Since Google poses strict rate limits on scrapers, we did not compute rich query representation for all the queries in the training set.

The query database used for training each of the 3 algorithms we considered consisted of these 55,422 queries. The frequency counts used by MostPopularCompletion were computed based on the entire training set, and not based only on the queries in the query database.

NearestCompletion and HybridCompletion were implemented by customizing Lucene[2] to our needs. The experiments were performed in October 2010 on a dual Intel Xeon 3.4GHz processor workstation with 4GB RAM and two 320GB disks.

Evaluation framework. We evaluate a QAC algorithm by how well it predicts the query the user is about to type. The prediction quality depends on whether the algorithm

[2]http://lucene.apache.org/java/docs/index.html

succeeds to generate a hit, and if it does, on the position of this hit.

Since all the algorithms we evaluate work the same when the current user input has no context, then our evaluation focused only on queries that have context. We generated sample queries with corresponding contexts as follows. We randomly sampled 40,000 sessions from the test set. For each selected session, we picked the first query among the queries in the session that have context (i.e., they are not the first one in the session) and that have a rich query representation (i.e., we scraped their recommendations/auto-completions from Google). If the session had no such queries, it was dropped from the sample. This resulted in 7,311 queries and contexts.

Our experiments focused on query auto-completion after having a single character from the current query. This setting is the most challenging and thus demonstrates the differences among the different algorithms most crisply.

Evaluation metric. Recall that for a particular query q and context C, an algorithm A is said to have a *hit* at position ℓ, if after receiving C and the first character of q, the algorithm returns q as the ℓ-th completion for this character. We write in this case that $\text{hitrank}(A, q, C) = \ell$ (if A has no hit at all, then $\text{hitrank}(A, q, C) = \infty$). Mean Reciprocal Rank (MRR) is a standard measure for evaluating retrieval that is aimed at a specific object. The reciprocal rank of an algorithm A on a particular (query, context) pair (q, C) is $1/\text{hitrank}(A, q, C)$ (note that the reciprocal rank is 0 when the algorithm has no hit). MRR is the expected reciprocal rank of the algorithm on a random (query, context) pair. To estimate MRR, we take a random sample S of (query, context) pairs, and compute the mean reciprocal rank of the algorithm over these pairs:

$$\text{MRR}(A) = \frac{1}{|S|} \sum_{(q,C)\in S} \frac{1}{\text{hitrank}(A, q, C)}.$$

MRR treats all (query, context) pairs equally. We observe, however, that some user inputs are easier to complete than others. For example, if the user input is the letter 'z', then since there are few words that start with z, the auto-completion task is easier and is likely to produce better predictions. On the other hand, if the user input is the letter 's', the numerous possible completions make the prediction task much harder. This motivates us to work with a *weighted* version of MRR (denoted wMRR). Rather than treating all (query, context) pairs uniformly, we weight them according to the number of completions available for the prefix of the query.

Comparison experiments. Our first set of experiments compare NearestCompletion and HybridCompletion to the standard MostPopularCompletion. The comparison is based on the 7,311 random (query, context) pairs collected from our training set. We used in these experiments the parameter values that we found to be the most cost-effective: (a) recommendation algorithm: Google's auto-completion; (b) recommendation tree depth: 3; (c) depth weighting function: exponential decay; (d) unigram model; (e) used only the most recent query; (f) α in HybridCompletion: 0.5.

We start with an anecdotal comparison (Table 1) of the completions provided by the three algorithms on some of the above pairs (note: these are real examples taken from the AOL log). The first two examples demonstrate the effect of context on query auto completion after the user has typed a single character of her intended query. In these examples, the user's intended query and the context are related (in the first example they are syntactically related and in the second one they are only semantically related). The NearestCompletion algorithm detects the similarity and thus provides the correct prediction at one of the top 2 positions. As the intended queries in these cases are not popular, MostPopularCompletion fails to hit the correct completion even in its top 10 suggestions. The utter failure of MostPopularCompletion has only a minor effect on HybridCompletion, which suggests the correct completion at one of its top 5 positions.

The third example exhibits the opposite scenario: the context is irrelevant to the user's intended query and the intended query is popular. Consequently, MostPopularCompletion hits the correct query at the top position, while NearestCompletion completely fails. This time HybridCompletion benefits from the success of MostPopularCompletion and hits the correct completion also at its top spot.

Figure 1 provides a comparison of weighted MRR of the three algorithms on the 7,311 (query, context) pairs. We validated that the results are all statistically significant. It is very clear that HybridCompletion dominates both NearestCompletion and MostPopularCompletion. For example, HybridCompletion's wMRR is 0.246 compared to only 0.187 of MostPopularCompletion (an improvement of 31.5%). It is also clear that the quality of NearestCompletion is inferior to MostPopularCompletion (by 19.8%), so it cannot be used as is for query auto completion. The graph also distinguishes between pairs in which the context has a rich representation (i.e., the recommendations for this context have been scraped from Google) and pairs in which the context has only a thin representation (based on the context query itself, without the recommendations). Note that the latter simulates the case the context is long-tail and the search engine has no recommendations for it. The results indicate that even such long-tail contexts are useful, and thus HybridCompletion is doing better than MostPopularCompletion.

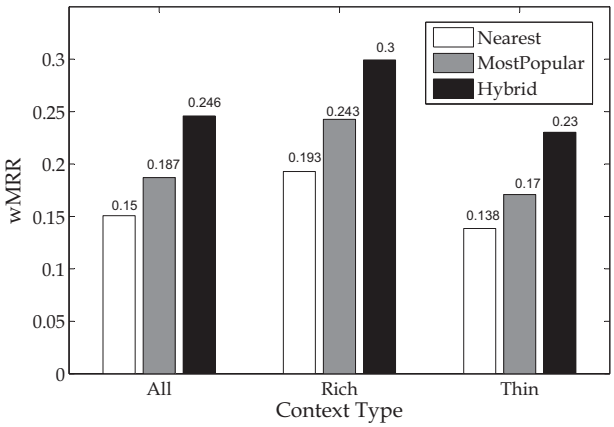

Figure 1: Weighted MRR of the 3 algorithms on 7,311 (query, context) pairs. Results are for all pairs, for pairs in which the context has a rich representation, and for pairs in which the context does not have a rich representation.

Figure 2 demonstrates that HybridCompletion dominates MostPopularCompletion not only on average but also with

Context	Query	MostPopularCompletion	NearestCompletion	HybridCompletion
french flag	italian flag	internet, im help, irs, ikea, internet explorer	**italian flag**, itunes and french, ireland, italy, irealand	internet, **italian flag**, itunes and french, im help, irs
neptune	uranus	ups, usps, united airlines, usbank, used cars	**uranus**, uranas, university, university of chicago, ultrasound	**uranus**, uranas, ups, united airlines, usps
improving acer laptop battery	bank of america	**bank of america** , bankofamerica, best buy, bed bath and beyond, billing	battery powered ride ons, battery plus charlotte nc, battery died while driving, best buy, battery replacement for palm tungsten c	**bank of america** , best buy, battery powered ride ons, bankofamerica, battery died while driving

Table 1: The top 5 completions provided by the 3 algorithms on (query, context) pairs taken from the AOL log. In the first two examples the context and the query are related, while in the last one they are not.

high probability. In this graph we compare the (weighted) fractions of (query,context) pairs on which the MRR values of one algorithm is higher than that of the other algorithm. Since small differences in MRR values are insignificant (e.g., if one algorithm ranks the intended query at position 9 and the other at position 10, the algorithms perform essentially the same on this query), we deem the two algorithms to do equally well on some input pair if the difference in their MRR values on this pair is at most ϵ. Clearly, the lower the value of ϵ, the tighter is the comparison.

After discarding input pairs on which the MRR of both algorithms was 0, we were left with 3,894 pairs. Figure 2 demonstrates that for a wide range of ϵ values, HybridCompletion is superior to MostPopularCompletion on a larger fraction of input pairs.

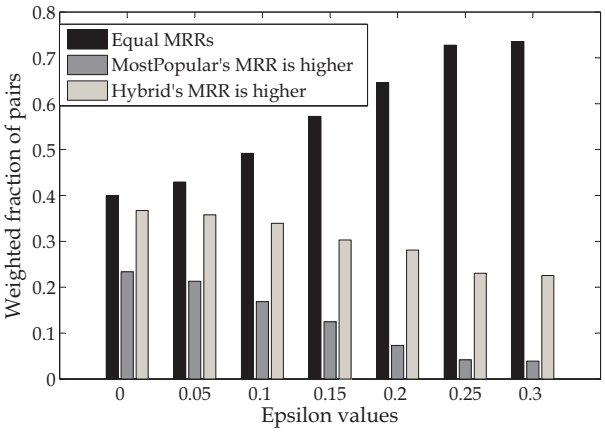

Figure 2: Weighted fractions of (query,context) pairs where MostPopularCompletion's MRR value is higher than that of HybridCompletion in at least an epsilon and vice versa.

We drilled down the results in order to understand the relative strengths and weaknesses of the three algorithms on different inputs. To this end, we selected a random sample of 198 (query, context) pairs from the set of 7,311 pairs, and manually tagged each of them as *related* (i.e., the query is related to the context; 60% of the pairs) and *unrelated* (40% of the pairs). Table 2 compares the quality of the three algorithms separately on the related pairs and on the unrelated pairs. The results indicate that when the intended query is related to the context, NearestCompletion is very successful, achieving wMRR that is 48% higher relative to MostPopularCompletion. HybridCompletion is even better because it takes both the context and the popularity into account. On the other hand, when the query and context are unrelated, NearestCompletion is essentially useless. HybridCompletion is even better than NearestCompletion for related pairs, while its quality for unrelated pairs is moderately lower (in 20.3%) than that of MostPopularCompletion.

	MostPopular	Nearest	Hybrid
Related context	0.163	0.242	0.280
Unrelated context	0.227	0	0.181

Table 2: Weighted MRR of the three algorithms, broken down by whether the intended query and the context are related or not.

Next, we broke down the 7,311 sample pairs into buckets based on the frequency of the intended query in the query log. The buckets correspond to exponentially increasing frequency ranges. Figure 3 plots the wMRR of each algorithm in each of the buckets. As expected, MostPopularCompletion is very successful at predicting popular queries. It supersedes NearestCompletion for such queries, because its success is independent of whether the context is related to the intended query or not. On the other hand, when the intended query is long-tail (low popularity), NearestCompletion manages to use the context to achieve a relatively high prediction quality, while MostPopularCompletion exhibits very poor quality. Note that HybridCompletion essentially takes the upper envelope of the two algorithms, and manages to achieve almost as high quality in all popularity ranges.

One peculiar artifact exhibited in this experiment is that the quality of NearestCompletion slightly deteriorates for popular queries (whose frequency in the log is above 10,000). We analyzed these queries and found that the fraction of unrelated contexts such queries have (58%) is much higher than the fraction of unrelated contexts for low popularity queries (only 36.5%). This explains the lower quality of NearestCompletion for such queries.

Parameter tuning experiments. We conducted a set of experiments in which we examined the effect of the different parameters of our algorithms on their quality. Figure 4 shows the influence of the depth of the query recommendation tree used in the construction of the rich query representation on the quality of NearestCompletion. We examined

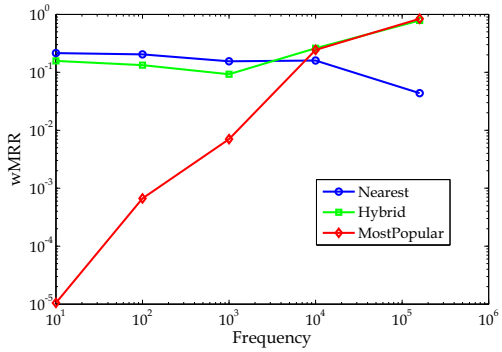

Figure 3: Weighted MRR of the 3 algorithms as a function of the frequency of the intended query in the query log (log-log scale).

depths 0 to 3. As expected, the quality of NearestCompletion increases, as the depth increases, since the vocabulary of the rich representation is richer. Note that the returns are starting to diminish at depth 3. While we could not run the experiment with larger depths, due to Google's scraping limitations, we expect this trend to continue as the depth increases. Thus, a recommendation tree of depth 2 or 3 seems to be the most cost-effective for this application.

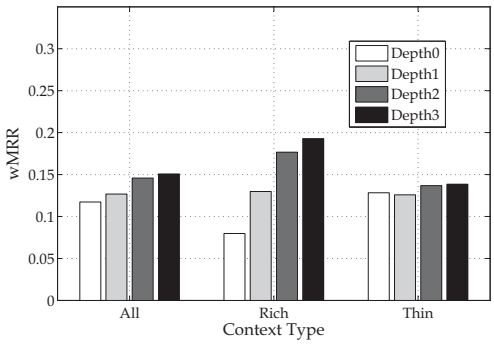

Figure 4: Weighted MRR of NearestCompletion as a function of the recommendation tree depth. Results are for all sample (query, context) pairs, for the ones in which the context has a rich representation, and for the ones in which the context does not have a rich representation.

Next, we measured the effect of the context length on the quality of NearestCompletion. Our experiments (see Table 3) demonstrate that increasing the number of recent queries being taken into account slightly improves the quality of the algorithm, as the vocabulary that describes the context is enriched. The effect is not as significant as the recommendation tree depth, though.

Context length	1	2	3
wMRR	0.139	0.154	0.164

Table 3: Weighted MRR of NearestCompletion as a function of the context length. Results are for 2,374 (query, context) pairs in which the context was of length at least 3.

Table 4 shows the dependence of NearestCompletion's quality on the query recommendation algorithm used to generate the recommendation trees. We compared two algorithms: Google's query auto-completions and Google's related search. The results demonstrate that the rich representations generated from Google's related searches are more effective (quality improves in 12.4%). Nevertheless, in most of our experiments we opted to use Google's query auto completions, because the difference is not huge and since they are much easier to scrape (the query latency is lower and the rate limits posed by Google are higher). We thus conclude that the auto completions are more cost-effective for this purpose.

	Auto-completions	Related searches
wMRR	0.177	0.199

Table 4: Weighted MRR of NearestCompletion using two different query recommendation algorithms (by Google) for generating recommendation trees of depth 2. Results are only for (query, context) pairs in which the context has a rich representation.

The parameter α in HybridCompletion controls the balance between NearestCompletion and MostPopularCompletion. Recall that for $\alpha = 1$ HybridCompletion is the same as NearestCompletion and for $\alpha = 0$ it is the same as MostPopularCompletion. Figure 5 analyzes the effect of α on the quality of the algorithm. As noted above, MostPopularCompletion is better than NearestCompletion when the intended query is popular and NearestCompletion is better when the context query is related to the intended query. The results show that $\alpha = 0.5$ is the best choice in aggregate.

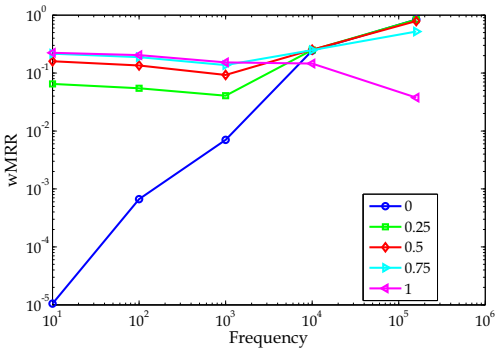

Figure 5: Weighted MRR of HybridCompletion as a function of α (log-log scale).

We measured the influence of other parameters of the algorithm, like the choice of the depth weighting function, the choice of the context weighting function and the N-grams maximum length. We have not found significant differences in quality among the different alternatives. We suppress the actual results from this draft, due to lack of space.

7. CONCLUSIONS

In this paper we proposed the first context-sensitive algorithm for query auto-completion. The algorithm, NearestCompletion, suggests to the user completions of her input prefix that are most similar to the recent queries the

user has just entered. We show that when the input prefix is short (1 character) and the context is relevant to the user's intended query, then the weighted MRR of NearestCompletion is 48% higher than that of the standard MostPopularCompletion algorithm. On the other hand, when the context is irrelevant, NearestCompletion is useless. We then propose HybridCompletion, which is a convex combination of NearestCompletion and MostPopularCompletion. HybridCompletion is shown to be at least as good as NearestCompletion when the context is relevant and almost as good as MostPopularCompletion when the context is irrelevant.

NearestCompletion computes the similarity between queries as the cosine similarity between their rich representations. To produce rich query representation we introduce a new query expansion technique, based on traversal of the query recommendation tree rooted at the query. This technique may be of independent interest for other applications of query expansion.

There are a number of possible interesting directions for further development of our techniques. (a) The choice of the optimal α value of HybridCompletion may be done adaptively. An algorithm which learns an optimal α as a function of the context features is likely to improve the quality of the combination. (b) Predicting the first query in a session still remains an open problem. Here one may need to rely on other contextual signals, like the user's recently visited page or the user's long-term search history. (c) The rich query representation may be further fine tuned by promoting the prominent keywords in the representation and demoting the less prominent ones. (d) We introduce a novel method for query expansion based on the query recommendation tree. It will be of interest to compare between the quality of our suggested technique and the quality of standard query expansion techniques. Such a comparison may be done in the scope of context-sensitive query auto-completion, as well as in other relevant IR tasks such as document search.

Acknowledgments. We thank Shraga Kraus for his careful review of the paper and for his guidance with MATLAB.

8. REFERENCES

[1] M. Arias, J. M. Cantera, J. Vegas, P. de la Fuente, J. C. Alonso, G. G. Bernardo, C. Llamas, and Á. Zubizarreta. Context-based personalization for mobile web search. In *PersDB*, pages 33–39, 2008.

[2] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Improving search engines by query clustering. *J. Am. Soc. Inf. Sci. Technol.*, 58(12):1793–1804, 2007.

[3] H. Bast, D. Majumdar, and I. Weber. Efficient interactive query expansion with complete search. In *CIKM*, pages 857–860, 2007.

[4] H. Bast and I. Weber. Type less, find more: fast autocompletion search with a succinct index. In *SIGIR*, pages 364–371, 2006.

[5] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 407–416, 2000.

[6] P. Boldi, F. Bonchi, C. Castillo, D. Donato, and S. Vigna. Query suggestions using query-flow graphs. In *WSCD*, pages 56–63, 2009.

[7] A. Z. Broder, P. Ciccolo, M. Fontoura, E. Gabrilovich, V. Josifovski, and L. Riedel. Search advertising using web relevance feedback. In *CIKM*, pages 1013–1022, 2008.

[8] A. Z. Broder, P. Ciccolo, E. Gabrilovich, V. Josifovski, D. Metzler, L. Riedel, and J. Yuan. Online expansion of rare queries for sponsored search. In *WWW*, pages 511–520, 2009.

[9] H. Cao, D. Jiang, J. Pei, E. Chen, and H. Li. Towards context-aware search by learning a very large variable length hidden markov model from search logs. In *WWW*, pages 191–200, 2009.

[10] 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.

[11] L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin. Placing search in context: the concept revisited. *ACM Trans. Inf. Syst.*, 20(1):116–131, 2002.

[12] B. M. Fonseca, P. B. Golgher, E. S. de Moura, and N. Ziviani. Using association rules to discover search engines related queries. In *LA-WEB*, page 66, 2003.

[13] Q. He, D. Jiang, Z. Liao, S. C. H. Hoi, K. Chang, E. Lim, and H. Li. Web query recommendation via sequential query prediction. In *ICDE*, pages 1443–1454, 2009.

[14] S. Ji, G. Li, C. Li, and J. Feng. Efficient interactive fuzzy keyword search. In *WWW*, pages 371–380, 2009.

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

[16] R. Kraft, C. C. Chang, F. Maghoul, and R. Kumar. Searching with context. In *WWW*, pages 477–486, 2006.

[17] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

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

[19] G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. In *1st InfoScale*, 2006.

[20] J. J. Rocchio. *Relevance Feedback in Information Retrieval*. Prentice Hall, 1971.

[21] E. Sadikov, J. Madhavan, L. Wang, and A. Halevy. Clustering query refinements by user intent. In *WWW*, pages 841–850, 2010.

[22] S. K. Tyler and J. Teevan. Large scale query log analysis of re-finding. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 191–200, 2010.

[23] R. W. White and G. Marchionini. Examining the effectiveness of real-time query expansion. *Inf. Process. Manage.*, 43(3):685–704, 2007.

[24] Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. In *WWW*, pages 1039–1040, 2006.