Context-Aware Query Suggestion

Research Thesis

In Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

Naama Kraus

Submitted to the Senate of the Technion - Israel Institute of Technology

Iyar 5772 Haifa April 2012

The research thesis was done under the supervision of Dr. Ziv Bar-Yossef and
Prof. Shaul Markovitch in the Faculty of Computer Science

The generous financial help of the Technion is gratefully acknowledged

Acknowledgment

I would like to sincerely thank the many people who made this research possible.

- I wish to express my deep gratitude to my supervisor, Dr. Ziv Bar-Yossef, for the guidance, partnership, support and encouragement. Thanks for the many things I've learned while working on this research. I highly appreciate the ability to balance between guiding me, and giving me the freedom to explore.
- I am grateful to Prof. Shaul Markovitch for co-advising, and his contribution to this research.
- I wish to thank Dr. Oren Kurland for the interest in the research and the useful review.
- I would like to thank my friends at the Technion, for warm words and beneficial discussions.
- My deep thanks to my parents, Sara and Melech Westreich, for their continuous support and encouragement. Thanks for teaching me to strive for knowledge and wisdom.
- I wish to thank my parents in law, Leah and Jehoshua Kraus, for the kind support. I am grateful for the assistance, which enabled me to invest time and resources in my research.
- To my beloved children, Mattan, Noa and Yuval, who gave me the light and happiness in a way that only children can.
- I am deeply grateful to my husband, Shraga, for accompanying me along the way, and for believing me even more than I believed in myself. Thanks also for the technical assistance in various aspects of this research.

List of Publications

1. Ziv Bar-Yossef and Naama Kraus, "Context-sensitive query auto-completion", *Proceedings of the 20th International Conference on World Wide Web, WWW 2011*, pp. 107-116, March 2011.

Table of Contents

List of Figures

List of Tables

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 study 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 Mean Reciprocal Rank (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.

List of Acronyms

ESA	Explicit Semantic Analysis
HMM	Hidden Markov Model
HC	Hybrid Completion
IDF	Inverse Document Frequency
IR	Information Retrieval
MLE	Maximum Likelihood Estimator
MM	Markov Model
MPC	Most Popular Completion
MRR	Mean Reciprocal Rank
NC	Nearest Completion
PPR	Personalized Page Rank
PPV	Personalized Page Rank Vector
QAC	Query Auto-Completion
QFG	Query Flow Graph
TF	Term Frequency
URL	Uniform Resource Locator
VSM	Vector Space Model
wMRR	Weighted Mean Reciprocal Rank

List of Symbols

∞	Infinity
μ	Mean
σ	Standard deviation
Z	Standard score

Chapter 1

Introduction

Query auto completion (QAC) [6, 58, 27, 5] 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

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

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 [42], 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 $\rightarrow$ results $\rightarrow$ 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 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 [33, 18] and in query recommendations [13, 24, 12, 9, 50]), 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 [13, 24, 12, 9, 50]. 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 [13, 12]. 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* $\rightarrow$ *american airlines flight status*) or only semantic (e.g., *american airlines* $\rightarrow$ *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 [61, 19], on co-clicks [2, 13], or on user search behavioral models [9, 41, 12, 50], 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 [11] are not necessarily correlated with reformulation likelihood.

Given a context, query recommendation algorithms [2, 61, 50] 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.* [11], 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 nov-

elty 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.

Chapter 2

Related Work

2.1 Overview

The following section surveys art that relates to different aspects of this study. We walk through query assistance techniques, namely query auto-completion and query recommendations. In particular, we review context-sensitive query assistance algorithms. As context has been studied in other Information Retrieval (IR) domains, we expand our review to contextual IR in general. Last, we review related art on query similarity and query expansion.

2.2 Query Assistance

Search engine users form natural language queries for the purpose of expressing their information need. Query formulation is thus a crucial step in the search process, as it directly effects the returned search results and therefore the user's search experience. Modern search engines assist users at the query formulation stage, by suggesting to the user a list of queries to choose from. Two major query assistance methods exist in the literature: *query auto-completion* and *query recommendations*. Despite the similar output, the two methods defer in their input and objective. The main objective of query auto-completion is to predict the user's query while she types it. The input is an incomplete

query, thus the user's intent is highly vague or even unknown. In contrast, in query recommendations, the input is a full user query that has just been typed. The main objective of query recommendations is to suggest alternative queries, also named *query re-formulations*. These recommendations assist the user to improve expressing her intent, or explore additional information related to her query.

2.3 Query Auto-Completion

Query auto-completion has received relatively little attention in the literature. Several methods have been proposed for completing user queries, differing in various aspects such as the target application, the type of the data being searched, the data-source used for extracting completions, the output format and the ranking method used.

2.3.1 Corpus Based Query and Document Suggestion

Bast and Weber [6] suggest an algorithm and an efficient indexing infrastructure for supporting query auto-completion. Given the user's input, which is composed of complete terms followed by a prefix of the next term, their algorithm suggests to the user term completions. For example, if the user has typed 'car d', the algorithm will suggest terms starting with 'd', such as 'dealer' or 'driver'. These will imply the queries 'car dealer' and 'car driver'. The suggested term completions are extracted from documents containing the user's typed terms, 'car' in our example. In addition to suggesting completions of the user's prefix, Bast and Weber's algorithm outputs top document hits of the suggested completions. Thus for example, if 'car dealer' was suggested as a completion, top hits of the query 'car dealer' will be presented to the user. Bast and Weber propose to rank the suggested completions, based on the scores of their corresponding hit documents. Their intuition is that queries resulting in highly scored documents should be ranked higher, since such queries are expected to be more useful to the user. Bast and Weber propose an indexing data-structure implementation that is time and space efficient. The proposed data-structure efficiently supports auto-completion while the user types her query, while requiring no additional space over a state-of-the-art indexing infrastructure.

2.3.2 Fuzzy Query Auto-Completion Over Structured Data

Ji *et al.* [27] explored an interactive and fuzzy search paradigm, focusing on searching over structured records containing textual information. While the user types her query, the proposed algorithm auto-completes query terms and displays matching data records. The search is fuzzy in the sense that returned records may contain text that is similar enough to query terms, not necessarily an exact match. For example, if the user types 'professor smyt', records containing 'professor smyth' are returned, as well as ones containing 'professor smith'. In this work, the user is not given completion suggestions to choose from, rather, gets the final results with query prefixes highlighted. In terms of ranking, the authors propose a ranking mechanism that is based on the following factors: (1) similarity between the predicted keyword and the input prefix (2) query independent keywords weight such as Inverse Document Frequency (IDF) (3) query independent record weight.

Ji *et al.* observe the crucial aspect of performance in auto-complete, as queries need to be processed for every keystroke. Thus they propose a highly efficient algorithm.

2.3.3 Probabilistic Corpus Based Query Auto-Completion

Bhatia *et al.* [8] propose a probabilistic method for generating query auto-completions from a given corpus. At a pre-processing stage, they create a repository of phrases by extracting N-grams from the corpus. At run-time, the algorithm's gets as an input the user's typed incomplete query. The algorithm's task is to suggest phrase completions that have a high probability to be eventually typed by the user. The proposed model estimates the desired probability by estimating two probabilities: (1) the probability that some phrase will be typed by the user, given the prefix that user is still typing, i.e. the last query term (2) the correlation between a phrase and the user's query, i.e. all preceding query terms.

2.3.4 Thesaurus Based Query Auto-Completion for Mobile Search

Arias *et al.* [1] propose a query auto-completion algorithm for mobile search. Their method is semantic, rather than syntactic and is additionally context-sensitive. Their completions 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. In comparison, we suggest a scalable algorithm that is based on query log data. Our algorithm is able to efficiently suggest completions given any user context, including rare contexts that have never occurred in the log before.

2.3.5 Context Sensitive Query Auto-Completion Based on Query Logs

Christian and Gertz [51] propose a context-sensitive query auto-completion algorithm, where completion candidates are extracted from the query log. The authors propose to model the probability that a completion is relevant to the user, based on the user's prefix as well as her contextual information. They demonstrate context-sensitive query auto-completion for time and location contextual information. Our algorithm addresses different contextual information, namely, recent queries based context. Applying recent queries based context in auto-complete, poses non trivial challenges that are not tackled by Christian and Gertz work. Examples are identifying completions relevant to the recent queries, ranking completions, coping with rare context and with irrelevant context. In comparison, our work tries to address those challenges.

2.4 Query Recommendations

Similarly to query auto-completion, query recommendations assist users in phrasing their intent. Numerous query recommendation algorithms have been introduced, relying on varied techniques, including topic clustering [2, 7], query co-occurrence analysis [19], session analysis [61, 24, 12], and user search behavioral models [41, 50, 9, 56]. As our context-sensitive query auto-completion algorithm is based on similarity between queries, we will elaborate on two query recommendation algorithms, which are based

on recommending similar queries.

2.4.1 Cluster Based Query Recommendations

Baeza-Yates *et al.* [2] query recommendation algorithm is based on suggesting queries that semantically relate to the user's query. Query log based queries form the repository of candidate query recommendations. A query is represented as a weighted vector of terms, where the terms are extracted from the query's clicked documents. Term weights are based on a variant of TF-IDF, which takes into account document click frequency, in addition to the standard term frequencies information. At a pre-processing stage, queries are clustered into groups of similar queries, based on the proposed similarity measure. Queries within a cluster are ranked by the *support of the query*, which measures how relevant a query is within the cluster, based on click-through information. Upon runtime, given a user's input query q, the algorithm locates the cluster C that q belongs to. It then suggests queries from C ranked by their support, as well as by their similarity to q.

Similarly to Baeza-Yates *et al.* [2], Beeferman and Berger [7] cluster query log queries into groups of similar queries, later used for query recommendations. Their clustering algorithm is based on the query log bipartite graph of queries and their associated clicked documents. The principal difference from Baeza-Yates *et al.* method is that Beeferman and Berger's algorithm is (as they phrase) 'content-ignorant'. I.e., the algorithm makes no use of the content of the documents or queries, rather is based on query-click relations only. The similarity between two queries is measured by the overlap between their associated sets of clicked documents. The intuition is that queries which share a large fraction of common clicked documents are ones that express a similar information need. Thus clusters of queries represent different ways or reformulating a similar intent.

2.4.2 Context-Sensitive Query Recommendations

Several context-sensitive query recommendation algorithms have been proposed in the literature. Boldi *et al.* [9] compute query recommendations by running *Personalized PageR-*

ank (PPR) on their *Query Flow Graph (QFG)*. 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.* [13, 12] and He *et al.* [24] train models for query sequences based on analysis of such sequences in search logs. They use, among others, some machine learning models, like Variable Length Hidden Markov Model (HMM) and Mixture Variable Memory Markov Model. 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.

2.5 Contextual IR

Contextual IR has been identified as one of the important and central challenges in the area [16]. It has been observed that considering the user's current query solely is not sufficient, as user queries are short and ambiguous [16, 25, 34]. Additionally, different users or even the same user, phrase different information needs using the same query or syntactically similar queries [14, 4, 36]. Therefore, IR applications should leverage user's context in order to exploit additional information about the user's information need and thus improve effectiveness.

Previous work tackles the context sensitivity challenge by exploiting different contextual resources, and proposing a variety of context models. The different approaches differ in the methods they use for leveraging the context, and in the IR applications they apply their methods to.

2.5.1 Context Type

Two categories of user context are addressed in the literature: *long-term context* and *short-term context* [52]. Long-term context is based on global information about the user such

as demographic, long-term search activity and user's interests information. Short-term context is based on the user's recent search activity, e.g., the user's recent queries and clicks in the last 30 minutes.

2.5.2 Contextual Resources

A large variety of contextual resources are used in the literature for modeling the user context. Examples are search queries [12, 9, 52], click-through information [52, 46], the amount of time a user spent on a specific result page [59], browsing history [55, 40], and the user's current location and time [51].

2.5.3 Context-Sensitive Retrieval

The most explored application is contextual retrieval, having numerous papers, e.g. [55, 52, 46, 33, 18, 43, 40, 44]. The objective of contextual retrieval is to exploit the user's context in order to personalize search results, and thus improve retrieval accuracy. A variety of context modeling and personalization methods where proposed in the literature.

Interests Based User Model

One approach for personalizing retrieval is to create a user profile based on her general interests. Interests are represented by categories extracted from a pre-defined taxonomy [26]. Users need to explicitly describe their categories of interests, or alternatively, interests may be deduced implicitly, e.g. from the user's query history [37]. Documents are also mapped to categories, and thus search results are re-ranked according to the user's interests. For example, Personalized PageRank [26] computes a personalized Page Rank Vector (*PPV*) for each set of documents belonging to some category. *PPV*'s and user model of interests are used at query time for re-ranking search results.

User Model as a Weighted Vector of Terms

A second approach is to extract terms rather than high level categories from the user's context, and use those terms for re-ranking search results. Teevan *et al.* [55] model the

user by a rich term index, which is based on the user's personal content such as web pages viewed, desktop documents and more. The authors propose to re-rank search results by assigning new weights to terms in the search formula, based on term occurrences in the user's content. Kraft *et al.* [33] represent the user context as a weighted vector of terms, which may be obtained from textual resources such as the current page viewed. They propose three different techniques for personalizing search results: (i) *query rewriting* - augmenting context terms to the original query; (ii) *rank-biasing* - giving boost to context terms in retrieved documents; (iii) *iterative filtering meta search* - generating a set of sub-queries from the query and context vector. Sub queries' result lists are then combined into the final retrieved documents list.

Machine Learning Based Personalization

A third approach proposes to exploit the user's search context by applying *learning to rank* methods. Radlinski and Joachims [46] coined the term *query chain* which they define as a sequence of queries that relates to a similar information need. In contrast to learning methods that consider each query solely, they take advantage of query chains in order to generate new preference judgment types. For example, a clicked document for a query q is preferred over any result skipped in queries preceding q in the same query chain. Thus, considering query chains implies deducing preference judgments of many more documents. The authors demonstrate that their method outperforms static ranking functions, as well as learning to rank methods that do not consider query chains.

2.5.4 User Search Behavior

User search behavior is a research area that relates to contextual IR. In order to effectively exploit user's context, it is beneficial to understand how users conduct search. An extensive research exists on different aspects of users search behavior. Examples are studies that explore the way users re-find information [54, 57]. Others try to predict the user's future search actions [35, 15], interests [31], and state, e.g., whether the user is satisfied by search results [17, 22].

2.5.5 Detecting Logical Sessions

When considering search history based context, an essential building block is the ability to segment user's past sessions into logical sessions. Previous research tackles this problem in different ways. He and Goker [21] proposed a method for detecting session boundaries using a time interval threshold. Later, He *et al.* [23] extended their time-based method to additionally consider features of similarity between queries, which improves the quality of session segmentation. Subsequent papers point out that users tend to be multi-tasking within a particular session [38]. In addition, users tend to span their search goals across multiple sessions [46, 10, 30, 32]. Different methods were proposed for identifying logical sessions given the user's search history. Several studies [46, 30, 32] propose machine learning classifiers for detecting queries belonging to the same search task. Lucchese *et al.* [38] propose a similarity measure between queries and clustering methods for partitioning a given search session into multiple task-based sessions. Boldi *et al.* [10] suggest the *Query Flow Graph* which is an aggregating model of all users' search sequences as reflected by the query log. The Query Flow Graph is used for computing the most probable partitioning of a particular search history into logical sessions.

2.6 Query Similarity

A variety of query similarity measures exists in the literature. This includes trivial syntactic similarity measures such as Jaccard distance, expansion techniques that overcome query sparsity problem such as Roccio's method [49], methods that exploit query log information [2, 7, 41], and semantic relatedness techniques [48, 28, 53, 20, 45, 60, 47].

2.6.1 Query Expansion

Query expansion is a well established field (see [39] for an overview). Query expansion techniques add terms such as synonyms, to the user's original query, in order to improve the way the query represents an information need. A classical application of query expansion is improving search recall. Query expansion methods alter the user's

query under the hoods, and thus the user is not aware of the query modification.

Classical methods expand the query using thesauri. This is limited and non-scalable. Roccio's relevance feedback method [49] expands the query from terms that occur in its search results. The algorithm (which was originally formulated in the vector space model) alters the original query by forming a new vector that maximizes similarity with relevant documents, while minimizing similarity with irrelevant documents. I.e., a vector that optimally separates between relevant and irrelevant documents. In the vector space model, the optimal query is formulated as the difference between the centroid of the relevant documents, and the centroid of the irrelevant documents. In many cases, only positive feedback is considered, and thus the centroid of relevant documents serves as the expanded query. The set of documents used for expansion are ones retrieved as a result of the query. These documents are manually marked as relevant or irrelevant, and the expansion is done presumably in several iterations. In order to avoid the costly manual intervention, *pseudo relevance feedback* [39] was proposed. The method automates the manual tagging of documents relevancy by considering the top k results of the query as the relevant set of documents.

2.6.2 Query Similarity Based on Query Log Analysis

Various methods rely on query log analysis for computing query similarity, e.g., [29, 2, 7, 3, 9]. Baeza-Yates [3] proposes several ways to measure similarity between queries, based on various data sources: (1) common terms (2) session co-occurrence (3) common clicked URLs (4) links between clicked URLs (5) common terms in clicked document.

Many of existing approaches use variants of these data sources for computing query similarity.

2.6.3 Semantic Relatedness

Semantic relatedness techniques aim to measure the level of similarity between two natural language texts. Various semantic relatedness techniques exist in the literature, e.g. [48, 28, 53, 20, 45, 60, 47]. Gabrilovich and Markovitch [20] propose *Explicit Semantic Anal-*

ysis (ESA), which leverages human knowledge that is maintained in Wikipedia. Their core idea is to map text into a high-dimensional space, where features are Wikipedia based concepts. Text is thus modeled as a weighted vector of concepts that represent the meaning behind the text. Similarity is measured using standard metrics such as the cosine similarity. Semantic relatedness techniques that accept multi-term texts as an input (such as ESA), may be used for measuring the semantic similarity between search queries.

Chapter 3

Most Popular Completion

3.1 Query Auto-Completion Definition

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.

3.2 Hit Definition

A completion c is said to be a *hit*, if it equals the query q that the user was about to enter. In this work 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.

3.3 Query auto-Completion Framework

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 qual-

ity 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.

3.4 Completion Criteria

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).

3.5 Completion Ordering

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.

3.6 MostPopularCompletion Definition

MostPopularCompletion (MPC) is the standard and most popular QAC algorithm. The quality score it assigns to each query q is the relative 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.

More formally, let $p(q)$ denote the probability distribution of incoming queries; we assume that $p(q)$ is identical for all users. Let $p(q|x)$ denote the probability that the user's intended query is q, given her typed input x. MostPopularCompletion ranks completion queries by $p(q|x)$, such that the most likely completion is returned at the top rank.

In order to rank completions by $p(q|x)$, we first apply *Bayes Rule* and express $p(q|x)$ as:

$$p(q|x) = \frac{p(x|q) \cdot p(q)}{p(x)}$$

$p(x|q)$ denotes the probability that the user will type x, given that she has query q in mind. For simplicity, we pose the following assumptions:

1. Users type queries by starting from the beginning of the query. Thus, x is a prefix of q, rather than any sub-string.

2. Users type the exact characters of q. In particular, we don't consider errors while typing.

3. Auto-complete does not affect users. This implies that a user will fully type q, even if q was suggested to her while typing.

Under the above assumptions we get:

$$p(x|q) = \begin{cases} 1 & \text{iff } x \text{ is an exact prefix of } q \\ 0 & \text{otherwise} \end{cases}$$

As x is given, $p(x)$ is constant over all queries q. Consequently, $p(x)$ does not affect the ranking and may be ignored. Thus, we end up with:

$$p(q|x) \stackrel{rank}{=} \begin{cases} p(q) & \text{iff } x \text{ is an exact prefix of } q \\ 0 & \text{otherwise} \end{cases}$$

Note however, that in the more general case, $p(q|x) \stackrel{rank}{=} p(x|q) \cdot p(q)$. This requires estimating $p(x|q)$, which we don't cover in the current research.

MPC applies *Maximum Likelihood Estimation (MLE)* for estimating $p(q)$. Recall that we assumed $p(q)$ is identical for all users, and thus may be estimated by the aggregation of all observed queries. Thus, $p(q)$ is estimated by the relative frequency of q in the query log:

$$\hat{p}(q) = p_{MLE}(q) = \frac{\text{freq}(q)}{\sum_{q' \in log} \text{freq}(q')}$$

Finally, we get:

$$p(q|x) \overset{rank}{=} \begin{cases} \frac{\text{freq}(q)}{\sum_{q' \in log} \text{freq}(q')} & \text{iff } x \text{ is an exact prefix of } q \\ 0 & \text{otherwise} \end{cases}$$

Chapter 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 (NC) 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.

4.1 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 ses-

sion 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.

4.2 The Basic Algorithm

A context-sensitive QAC algorithm takes into account the user's context, when estimating the probability of her intended query. More formally, let $p(q|x, C)$ denote the conditional probability that the next query is q, given the prefix x and the current context C. Similarly to MostPopularCompletion, we apply *Bayes Rule* and get:

$$p(q|x, C) = \frac{p(x|q, C) \cdot p(q|C)}{p(x|C)}$$

We assume that the input x is independent of the context C, given the query q. That is, $p(x|q, C) = p(x|q)$. Thus:

$$p(q|x, C) = \frac{p(x|q) \cdot p(q|C)}{p(x|C)}$$

As x and C are given, $p(x|C)$ is constant over all queries q. Consequently, it may be ignored for the purpose of ranking.

We pose the same assumptions regarding $p(x|q)$, as we did in MostPopularCompletion and thus have:

$$p(x|q) = \begin{cases} 1 & \text{iff } x \text{ is an exact prefix of } q \\ 0 & \text{otherwise} \end{cases}$$

Finally, we end up with:

$$p(q|x,C) \overset{rank}{=} \begin{cases} p(q|C) & \text{iff } x \text{ is an exact prefix of } q \\ 0 & \text{otherwise} \end{cases}$$

Estimating $p(q|C)$ using MLE is problematic, because query logs are too sparse to provide meaningful estimates 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 similar 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 (VSM)*, 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 ranks the completions q of x, by the cosine similarity between v_q and v_C:

$$\text{Similarity}(q, C) = \frac{\langle v_q, v_C \rangle}{||v_q|| \cdot ||v_C||}.$$

In other words, NearestCompletion outputs the k completions of x whose vectors are most similar to v_C.

4.3 Context Representation

Contexts and queries are objects of different types, so it may not be clear how to represent 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 ($w_i = 1/e^{t-i}$)).

4.4 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* $\rightarrow$ *baby eating disorder*, but not semantic refinements, like *baby eating disorder* $\rightarrow$ *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* [39] 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.

4.5 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 4.5.1 (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.

4.6 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.

Note that a similar architecture was proposed by Broder *et al.* [11] to expand long-tail queries for the purpose of matching relevant ads.

Chapter 5

Hybrid Completion

5.1 Motivation

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.

5.2 HybridCompletion Definition

Given a user input x and a context C, HybridCompletion (HC) 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.

Returning to the probabilistic formalization of NearestCompletion and MostPopularCompletion, HybridCompletion may be formalized as an estimation of $p(q|x, C)$. HybridCompletion is thus defined as the following convex combination of probabilities:

$$\hat{p}(q|x, C) = p_{HC}(q|x, C) \stackrel{def}{=} \alpha \cdot p_{NC}(q|x, C) + (1 - \alpha) \cdot p_{MPC}(q|x)$$

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 $\alpha = 0$ Hybrid-

Completion 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.

5.3 An Adaptive Combination Approach

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 work is not on session segmentation and context relevancy detection, we have not addressed this direction.

5.4 Re-ranking of Original Lists

It is important to note that HybridCompletion may re-rank 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.

Chapter 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 [42] 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).

6.1 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). Figures 6.1 and 6.2 demonstrate the session length distribution in the AOL log.

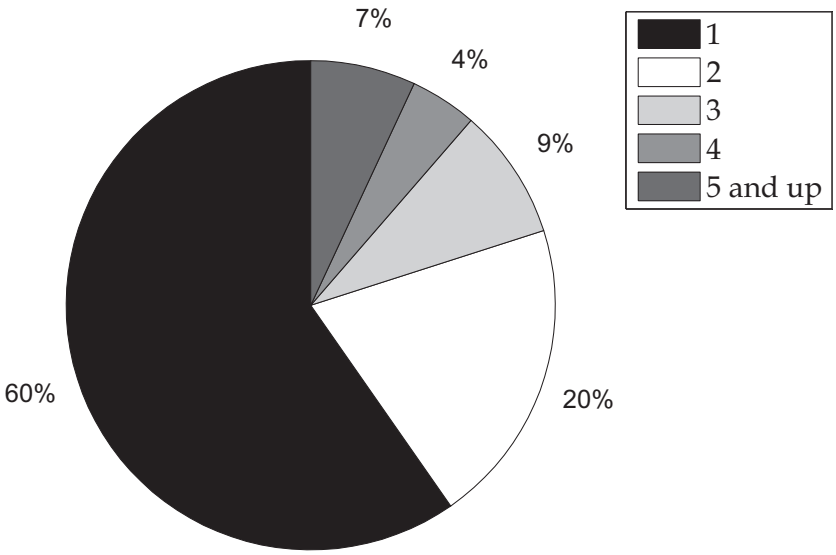

Figure 6.1: Session length distribution in the AOL log

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 auto-completion 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[1] 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.

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

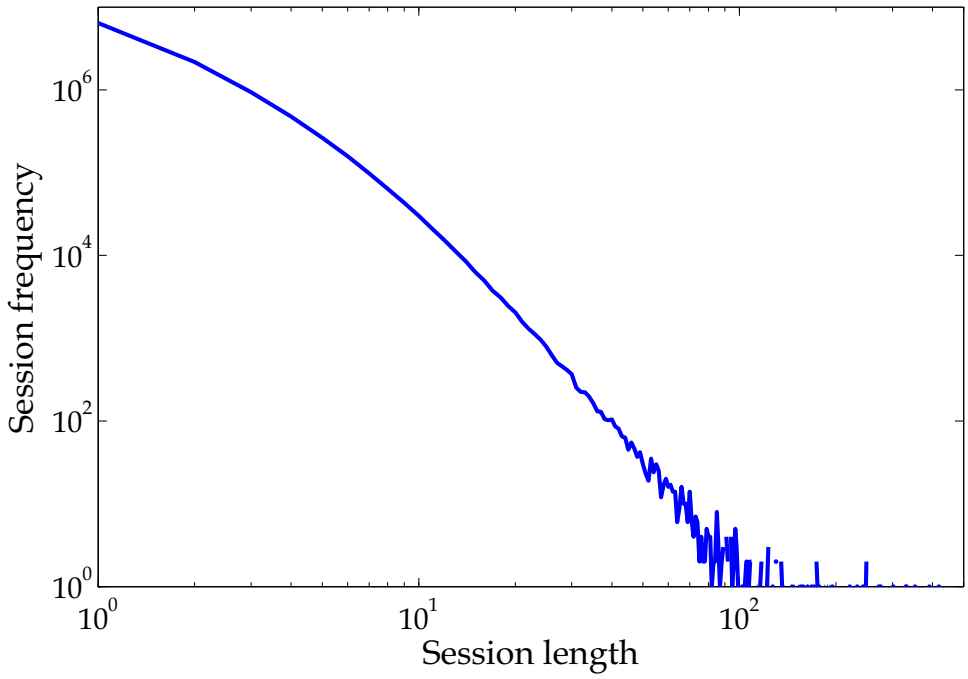

Figure 6.2: Session length distribution in the AOL log; log log scale

6.2 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 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.

6.3 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.

6.4 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. Table 6.1 summarizes the most cost-effective parameters.

Recommendation algorithm	Google's auto-completions
Recommendation tree depth	3
Depth weighting function	exponential decay
n-gram length	1
Context length	1
α used by HybridCompletion	0.5

Table 6.1: Most cost parameters-effective parameters of NearestCompletion and HybridCompletion as found in the comparison experiments.

We start with an anecdotal comparison (Table 6.2) 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 6.3 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.

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 6.2: 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.

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.

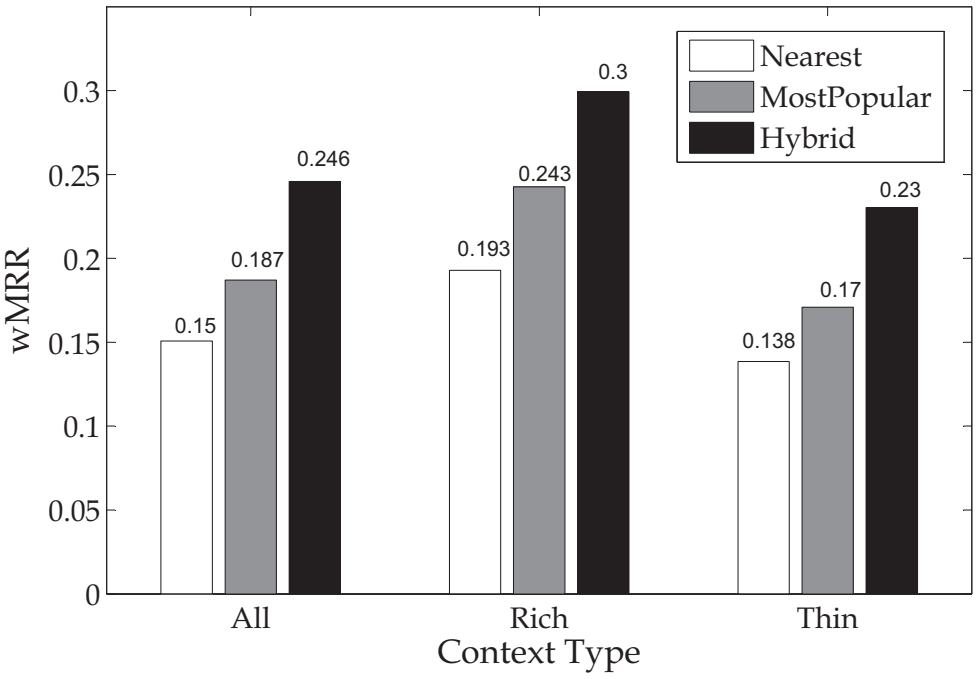

Figure 6.3: Weighted MRR of the 3 algorithms on 7,311 (query, context) pairs. Results are for all pairs, for pairs in which the context has a rich representation, and for pairs in which the context does not have a rich representation.

Figure 6.4 demonstrates that HybridCompletion dominates MostPopularCompletion not only on average but also with 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 6.4 demonstrates that for a wide range of ϵ values, HybridCompletion is superior to MostPopularCompletion on a larger fraction of input pairs.

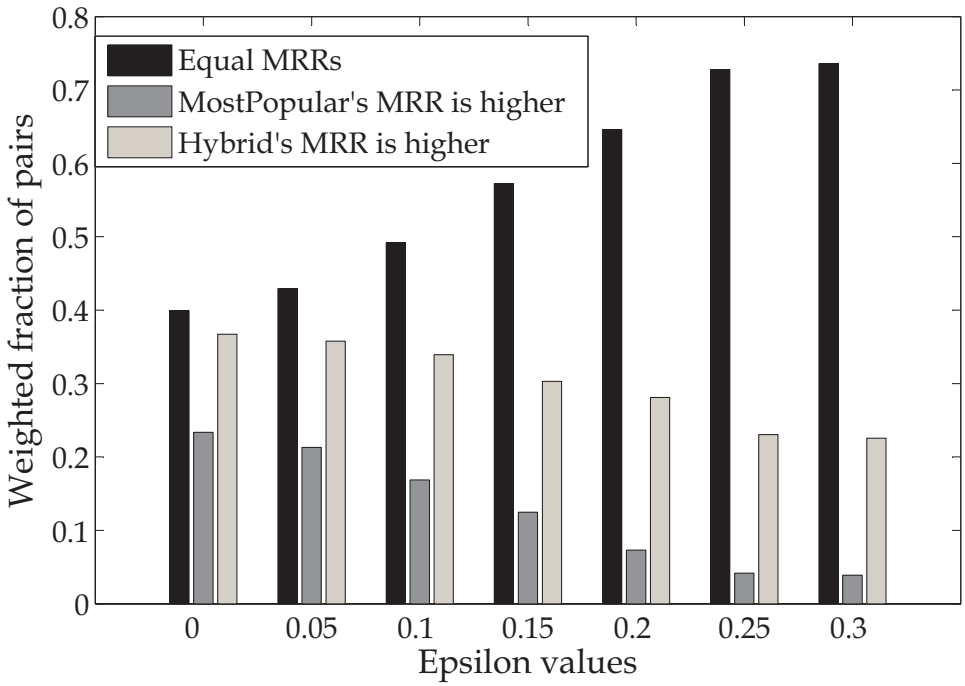

Figure 6.4: Weighted fraction of (query,context) pairs where MostPopularCompletion's MRR value is higher than that of HybridCompletion in at least an epsilon and vice versa.

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 6.3 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 6.3: 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 6.5 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.

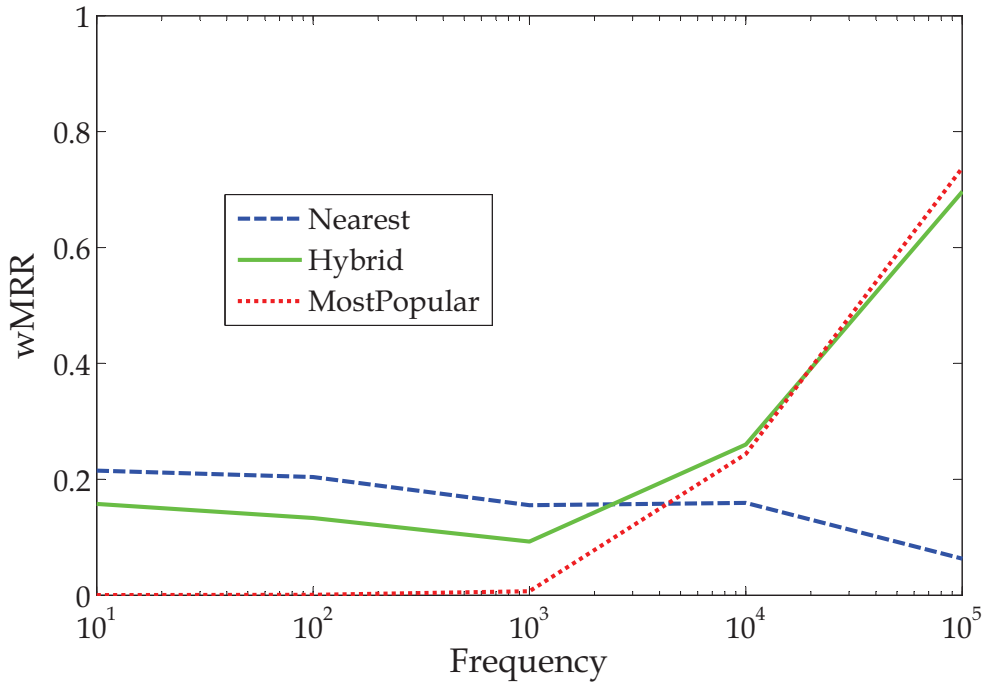

Figure 6.5: Weighted MRR of the 3 algorithms as a function of the frequency of the intended query in the query log (log-log scale).

6.5 Scalability Experiments

Our baseline query database for comparing NearestCompletion, MostPopularCompletion and HybridCompletion consisted of 55,422 rich queries. In practice, the query database is expected to be much larger. When the query database size increases, there are more query candidates to choose a completion from, which makes the prediction task even more challenging. Therefore, we were interested to measure the scalability of our algorithm as a function of the query database size.

We created a larger rich query database by scraping additional Google Auto-Completions, and came up with 273,127 rich queries. We then compared the wMRR of the three algorithms over the larger queries database as demonstrated in Figure 6.6. Note that we used the same comparison method that we described in the previous section. We observed that the relative improvement of HybridCompletion over MostPopularCompletion increases when the query database size increases, and is 45.6 % compared to 31.5 % relative improvement in the 55,422 query database. Figure 6.7 is a side by side comparison of the

two database size experiments for all query pairs.

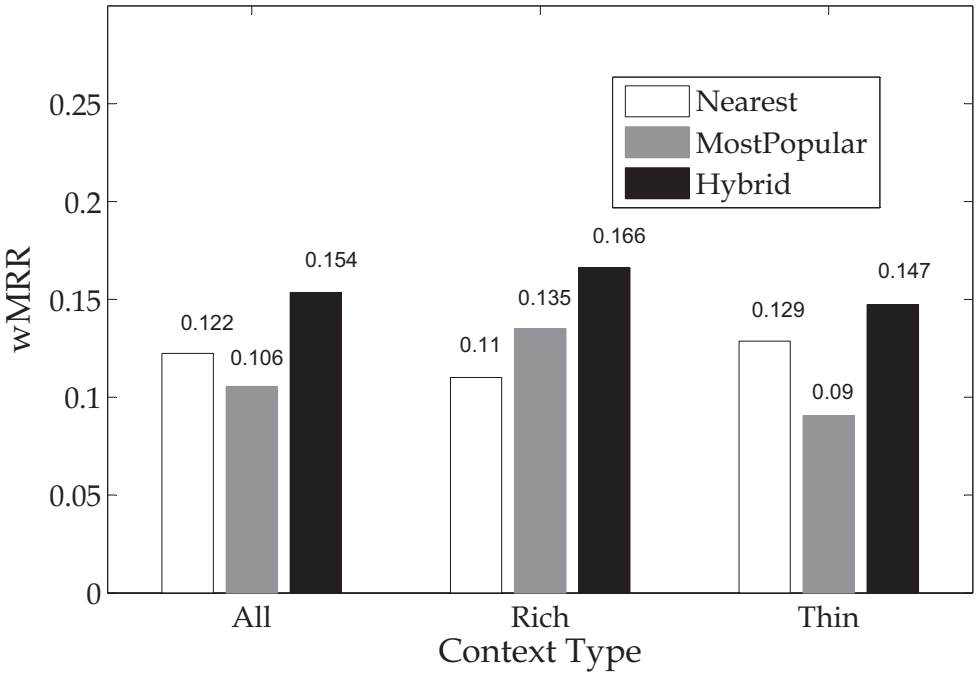

Figure 6.6: Weighted MRR of the 3 algorithms on 273,127 (query, context) pairs. Results are for all pairs, for pairs in which the context has a rich representation, and for pairs in which the context does not have a rich representation.

It appears that the quality of both MostPopularCompletion and NearestCompletion (and thus of HybridCompletion) degrades when the rich query database size increases. We measured the relative quality degradation of the two algorithms and observed that MostPopularCompletion's quality degrades more significantly than that of NearestCompletion: NearestCompletion's wMRR decreases to 0.12, i.e. it relatively degrades in 20 %, while MostPopularCompletion's wMRR decreases to 0.1 and thus degrades in 46.5 %.

Another interesting outcome we observed is that in the 273,127 query database experiment, NearestCompletion's wMRR is higher than that of MostPopularCompletion. Recall that in the 55,422 query database experiment, MostPopularCompletion dominated NearestCompletion.

All the above hints that in terms of quality, NearestCompletion scales better than MostPopularCompletion does, when the size of the query database increases.

As in the smaller query database experiment, we created a graph of the wMRR as

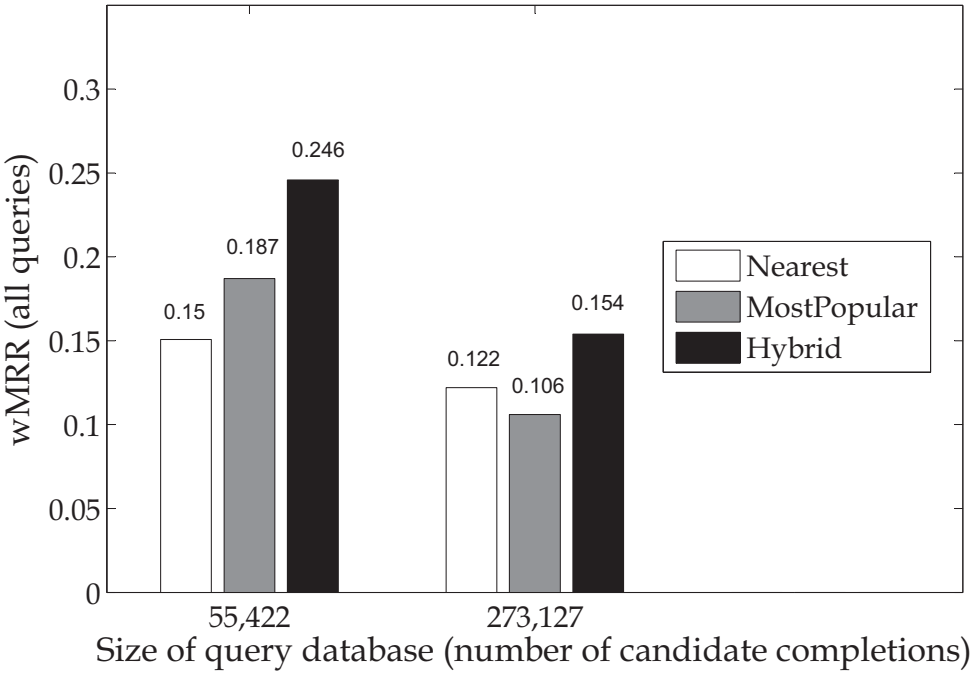

Figure 6.7: Weighted MRR of the 3 algorithms on a database consisting of 7,311 (query, context) pairs vs. a database consisting of 273,127 (query, context) pairs.

a function of the intended query frequency. As shown in Figure 6.8, the overall behavior pattern of the three algorithms is similar to what we observed in the smaller query database.

6.6 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 6.9 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 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.

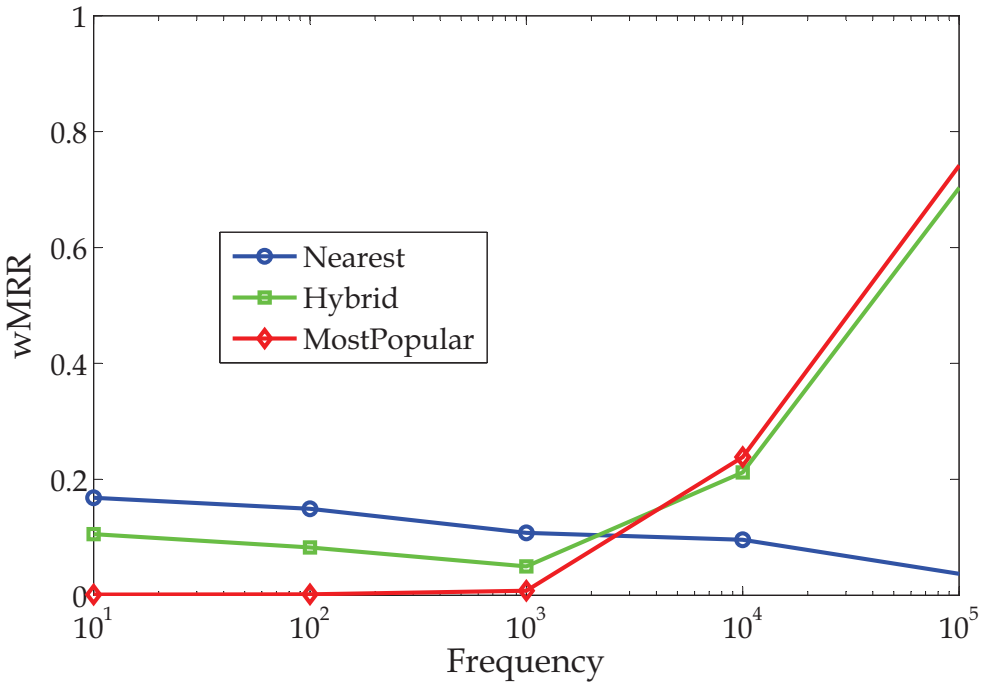

Figure 6.8: Larger (273,127) database experiment. Weighted MRR of the 3 algorithms as a function of the frequency of the intended query in the query log (log-log scale)

Next, we measured the effect of the context length on the quality of NearestCompletion. Our experiments (see Table 6.4) 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 6.4: 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 6.5 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

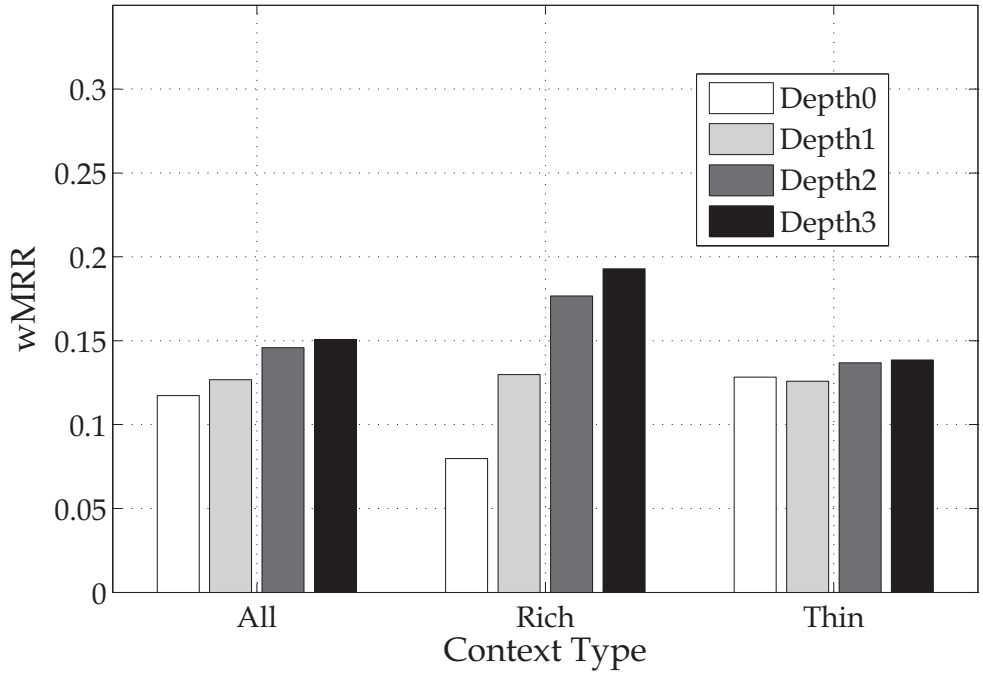

Figure 6.9: Weighted MRR of NearestCompletion as a function of the recommendation tree depth. Results are for all sample (query, context) pairs, for the ones in which the context has a rich representation, and for the ones in which the context does not have a rich representation.

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 6.5: 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 6.10 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.

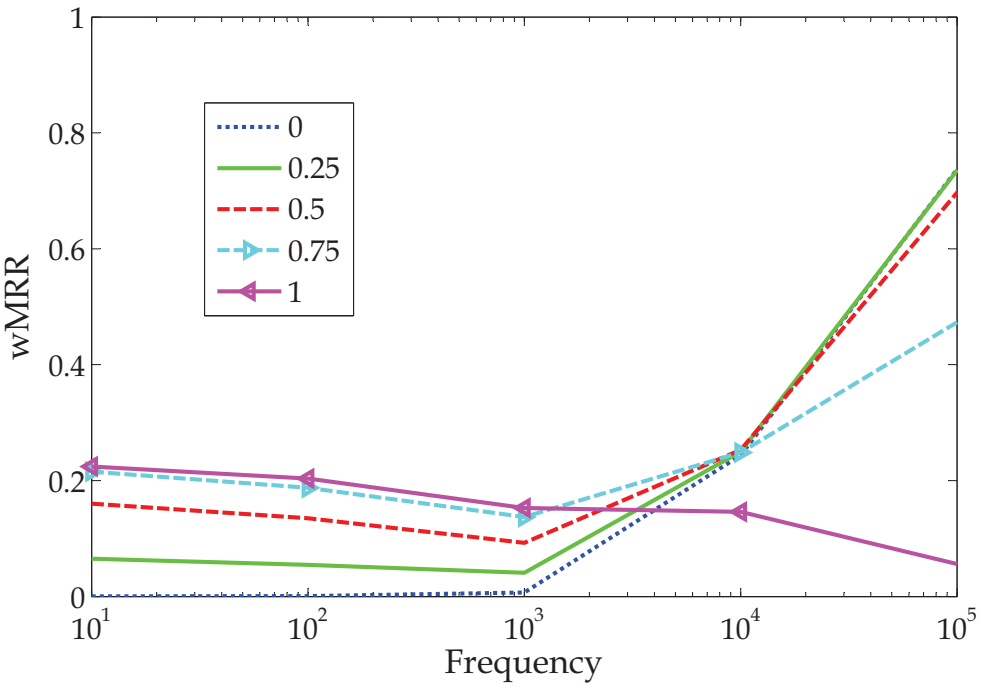

Figure 6.10: Weighted MRR of HybridCompletion as a function of α (log-log scale).

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.

Chapter 7

Discussion and Conclusions

In this work 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) 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.

Bibliography

[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] Ricardo A. Baeza-Yates. Graphs from search engine queries. In *SOFSEM (1)*, pages 1–8, 2007.

[4] Ricardo A. Baeza-Yates, Carlos A. Hurtado, and Marcelo Mendoza. Improving search engines by query clustering. *JASIST*, 58(12):1793–1804, 2007.

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

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

[7] 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.

[8] Sumit Bhatia, Debapriyo Majumdar, and Prasenjit Mitra. Query suggestions in the absence of query logs. In *SIGIR*, pages 795–804, 2011.

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

[10] Paolo Boldi, Francesco Bonchi, Carlos Castillo, Debora Donato, Aristides Gionis, and Sebastiano Vigna. The query-flow graph: model and applications. In *CIKM*, pages 609–618, 2008.

[11] 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.

[12] 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.

[13] 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.

[14] Zhicheng Dou, Ruihua Song, and Ji-Rong Wen. A large-scale evaluation and analysis of personalized search strategies. In *WWW*, pages 581–590, 2007.

[15] Doug Downey, Susan T. Dumais, and Eric Horvitz. Models of searching and browsing: Languages, studies, and application. In *IJCAI*, pages 2740–2747, 2007.

[16] James Allan et al. Challenges in information retrieval and language modeling: report of a workshop held at the center for intelligent information retrieval, university of massachusetts amherst, september 2002. *SIGIR Forum*, 37(1):31–47, 2003.

[17] Henry Allen Feild, James Allan, and Rosie Jones. Predicting searcher frustration. In *SIGIR*, pages 34–41, 2010.

[18] 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.

[19] 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.

[20] Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *IJCAI*, pages 1606–1611, 2007.

[21] Ayse Göker and Daqing He. Analysing web search logs to determine session boundaries for user-oriented learning. In *AH*, pages 319–322, 2000.

[22] Ahmed Hassan, Rosie Jones, and Kristina Lisa Klinkner. Beyond dcg: user behavior as a predictor of a successful search. In *WSDM*, pages 221–230, 2010.

[23] Daqing He, Ayse Göker, and David J. Harper. Combining evidence for automatic web session identification. *Inf. Process. Manage.*, 38(5):727–742, 2002.

[24] 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.

[25] B. J. Jansen, A. Spink, J. Bateman, and T. Saracevic. Real life information retrieval: a study of user queries on the web. *SIGIR Forum*, 32(1):5–17, 1998.

[26] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *WWW*, pages 271–279, 2003.

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

[28] Jay J. Jiang and David W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. *CoRR*, cmp-lg/9709008, 1997.

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

[30] Rosie Jones and Kristina Lisa Klinkner. Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. In *CIKM*, pages 699–708, 2008.

[31] Alexander Kotov, Paul N. Bennett, Ryen W. White, Susan T. Dumais, and Jaime Teevan. Modeling and analysis of cross-session search tasks. In *SIGIR*, pages 5–14, 2011.

[32] Alexander Kotov, Paul N. Bennett, Ryen W. White, Susan T. Dumais, and Jaime Teevan. Modeling and analysis of cross-session search tasks. In *SIGIR*, pages 5–14, 2011.

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

[34] Robert Krovetz and W. Bruce Croft. Lexical ambiguity and information retrieval. *ACM Trans. Inf. Syst.*, 10(2):115–141, 1992.

[35] Tessa Lau and Eric Horvitz. Patterns of search: analyzing and modeling web query refinement. In *Proceedings of the seventh international conference on User modeling*, pages 119–128, 1999.

[36] Uichin Lee, Zhenyu Liu, and Junghoo Cho. Automatic identification of user goals in web search. In *WWW*, pages 391–400, 2005.

[37] Fang Liu, Clement T. Yu, and Weiyi Meng. Personalized web search by mapping user queries to categories. In *CIKM*, pages 558–565, 2002.

[38] Claudio Lucchese, Salvatore Orlando, Raffaele Perego, Fabrizio Silvestri, and Gabriele Tolomei. Identifying task-based sessions in search engine query logs. In *WSDM*, pages 277–286, 2011.

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

[40] Nicolaas Matthijs and Filip Radlinski. Personalizing web search using long term browsing history. In *WSDM*, pages 25–34, 2011.

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

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

[43] James E. Pitkow, Hinrich Schütze, Todd A. Cass, Robert Cooley, Don Turnbull, Andy Edmonds, Eytan Adar, and Thomas M. Breuel. Personalized search. *Commun. ACM*, 45(9):50–55, 2002.

[44] Benjamin Piwowarski, Georges Dupret, and Rosie Jones. Mining user web search activity with layered bayesian networks or how to capture a click in its context. In *WSDM*, pages 162–171, 2009.

[45] Kira Radinsky, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. A word at a time: computing word relatedness using temporal semantic analysis. In *WWW*, pages 337–346, 2011.

[46] Filip Radlinski and Thorsten Joachims. Query chains: Learning to rank from implicit feedback. *CoRR*, abs/cs/0605035, 2006.

[47] Joseph Reisinger and Raymond J. Mooney. Multi-prototype vector-space models of word meaning. In *HLT-NAACL*, pages 109–117, 2010.

[48] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI*, pages 448–453, 1995.

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

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

[51] Christian Sengstock and Michael Gertz. Conquer: a system for efficient context-aware query suggestions. In *WWW (Companion Volume)*, pages 265–268, 2011.

[52] Xuehua Shen, Bin Tan, and ChengXiang Zhai. Context-sensitive information retrieval using implicit feedback. In *SIGIR*, pages 43–50, 2005.

[53] Michael Strube and Simone Paolo Ponzetto. Wikirelate! computing semantic relatedness using wikipedia. In *AAAI*. AAAI Press, 2006.

[54] Jaime Teevan, Eytan Adar, Rosie Jones, and Michael A. S. Potts. Information re-retrieval: repeat queries in yahoo's logs. In *SIGIR*, pages 151–158, 2007.

[55] Jaime Teevan, Susan T. Dumais, and Eric Horvitz. Personalizing search via automated analysis of interests and activities. In *SIGIR*, pages 449–456, 2005.

[56] 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.

[57] Sarah K. Tyler and Jaime Teevan. Large scale query log analysis of re-finding. In *WSDM*, pages 191–200, 2010.

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

[59] Ryen W. White, Paul N. Bennett, and Susan T. Dumais. Predicting short-term interests using activity-based search context. In *CIKM*, pages 1009–1018, 2010.

[60] Eric Yeh, Daniel Ramage, Christopher D. Manning, Eneko Agirre, and Aitor Soroa. Wikiwalk: Random walks on wikipedia for semantic relatedness. In *Graph-based Methods for Natural Language Processing*, pages 41–49, 2009.

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

Hebrew Section

הנוכחי. ההקשר שאנו מתרכזים בו הינו השאילתות הקודמות שביצע המשתמש לאחרונה. אנו מראים שכאשר תחילית הקלט הינה קצרה מאוד (באורך תו בודד) וההקשר רלבנטי לחיפוש הנוכחי, אזי ה-MRR הממושקל של NearestCompletion גבוה ב-48% מזה של MostPopularCompletion. מאידך, כאשר ההקשר אינו רלבנטי, NearestCompletion הינו חסר תועלת. אנו מציעים אלגוריתם בשם HybridCompletion שהינו שילוב של שני האלגוריתמים האחרים. אנו מראים כי HybridCompletion הינו טוב לפחות כמו NearestCompletion כשההקשר רלבנטי, וטוב כמעט כמו MostPopularCompletion כשההקשר אינו רלבנטי.

כמו כן, אנו מציעים בעבודה זו טכניקה חדשה של הרחבת שאילתות, למדידת דמיון בין שאילתות. ההרחבה המוצעת מתבססת על עץ המלצות ששורשו מכיל את השאילתא המבוקשת. הטכניקה המתוארת עשויה להיות רלבנטית לאפליקציות אחרות של הרחבת שאילתות.

ישנם מספר כיוונים אפשריים להרחבה ולהמשך.

1. חיזוי השאילתא הראשונה של המשתמש כאשר לא קיימות שאילתות קודמות שנשאלו לאחרונה, נותרה בעיה פתוחה. במקרה זה, יש להסתמך על מקורות הקשר אחרים כגון היסטוריית החיפוש ארוכת הטווח של המשתמש, או הדפים אותם ביקר לאחרונה.
2. שימוש בהרחבת שאילתות מבוססת המלצות באפליקציות חיפוש אחרות כגון איחזור מסמכים.
3. לימוד הפרמטרים האופטימליים של HybridCompletion, המשמשים לשילוב בין התוצאות של NearestCompletion ו-MostPopularCompletion . בניגוד לשיטה המוצעת בה אנו לומדים פרמטרים גלובליים לכל הקלטים, היה מעניין ללמוד פרמטרים אופטימליים המותאמים לתכונות ההקשר הנתון. פרמטרים כאלה עשויים לשפר את איכות השילוב.

ניסויים ותוצאות

במסגרת המחקר ערכנו קבוצת ניסויים שלהם שתי מטרות: המטרה הראשונה הינה ביצוע השוואה בין אלגוריתמי המלצת השאילתות שהצענו, NearestCompletion ו-HybridCompletion, לאלגוריתם הבסיס MostPopularCompletion. המטרה השניה הינה בדיקת ההשפעה של הפרמטרים השונים על איכות התוצאות.

אנו מציעים במחקרנו מתודולוגית הערכה אוטומטית לאלגוריתמי המלצת שאילתות, המבוססת על יומן שאילתות נתון של משתמשים. המתודולוגיה המוצעת הינה כדלקמן:

חילקנו את יומן השאילתות לשני חלקים. גודלו של החלק הראשון הינו 80% מהיומן ותפקידו לשמש כקבוצת האימון. גודלו של החלק השני הינו 20% מהיומן ותפקידו לשמש כקבוצת הבדיקה. קבוצת האימון שימשה לנו ליצירת מאגר של שאילתות מועמדות להמלצה. לכל שאילתא במאגר יצרנו ייצוג עשיר של וקטור במימד גבוה, המבוסס על שיטת הרחבת שאילתא שהצענו בעבודה זו. את הייצוג יצרנו על בסיס שרות המלצת השאילתות (נטולת הקשר) של גוגל – Google Suggest.

כדי להעריך אלגוריתם המלצת שאילתות נדרשנו לשערך את איכות החיזוי של שאילתת המשתמש בהנתן תחילית השאילתא. איכות החיזוי תלויה ביכולת האלגוריתם לחזות את השאילתא, וכן בדרוג הניתן ע"י האלגוריתם. ככל ששאילתת המשתמש מדורגת גבוה יותר ברשימת ההמלצות, כך עולה איכות החיזוי. לצורך הניסויים דגמנו קבוצת זוגות של שאילתות עוקבות מהיומן, שנשאלו ע"י אותו משתמש בהפרש זמן קצר יחסית (הפרש זמן של 30 דקות המקובל במחקר בתחום). לכל זוג שאילתות התייחסנו כיחידת בדיקה, כאשר השאילתא הראשונה משמשת כשאילתת הקשר. על האלגוריתם הנבדק לחזות את השאילתא השניה בהנתן התו הראשון שלה כתחילית הקלט. אנו מציעים במחקר להשתמש בואריאציה של מדד ה- MRR. מדד ה-MRR הינו מדד מקובל להערכת יכולת אלגוריתם לאחזר עצם מבוקש בדרוג גבוה. פרטי הניסויים ותוצאותיהם מופיעים בגוף החיבור.

מסקנות

בעבודה זו הצענו את האלגוריתם הראשון להמלצת שאילתות שהינו תלוי הקשר. האלגוריתם, NearestCompletion, מקבל כקלט את תחילית השאילתא שהמשתמש זה עתה מקליד, ומציע למשתמש שאילתות הדומות ביותר להקשר שבו מתבצע החיפוש

של שאילתות, ע״י הפעלה איטרטיבית של אלגוריתם להמלצת שאילתות (נטול הקשר) המשמש כקופסה שחורה. העץ נבנה באופן הבא: שורש העץ מכיל את השאילתא הנתונה; הבנים הישירים של השורש מכילים המלצות ישירות של השאילתא; הדרגה הבאה מכילה המלצות של המלצות; וכן הלאה. תהליך הבניה נמשך עד לגובה רצוי, שהינו פרמטר נתון לאלגוריתם. ניתן לראות כי העץ שנוצר מכיל מילים נוספות על מילות השאילתא המקורית, המהוות בסיס להרחבת השאילתא. ההיפותזה שלנו הינה שהמילים שנוספו תהיינה קשורות סמנטית לשאילתא המקורית, כיון שמקורן בשאילתות שהינן בסבירות גבוהה, ניסוחים אפשריים לשאילתא זו. מהעץ שנוצר אנו גוזרים ייצוג וקטורי של מילים ממושקלות, המייצג את השאילתא המורחבת. אנו מציעים שיטה למישקול המילים, הלוקחת בחשבון את כמות מופעי המילה בעץ, וכן את הרמה בעץ בה מופיעה מילה. ככל שמילה מופיעה פעמים רבות יותר, הסבירות שהיא קשורה סמנטית לשאילתא עולה. בנוסף, רמה גבוהה יותר בעץ מעידה על קשר ישיר יותר לשאילתא הנתונה, ולכן משקלה גדול יותר.

מניתוח אמפירי שערכנו מתברר כי כאשר ההקשר רלבנטי לשאילתא הנוכחית, החיזוי של NearestCompletion טוב יותר ב-48% באופן יחסי מזה של MostPopularCompletion עבור אותן שאילתות. אולם, כאשר ההקשר אינו רלבנטי לשאילתא הנוכחית, NearsetCompletion נכשל לגמרי בחיזוי. וכך, עבור כלל השאילתות (רלבנטית להקשר ושאינן רלבנטיות להקשר), יכולת החיזוי של NearestCompltion נמוכה ב-19% מזו של MostPopularCompletion. כדי להתגבר על התופעה המתוארת, אנו מציעים אלגוריתם בשם HybridCompletion שהינו שילוב של שני האלגוריתמים האחרים. כל אחד משני האלגוריתמים נותן כפלט רשימה מדורגת של k ההשלמות הטובות ביותר. אנו מאחדים את שתי הרשימות על ידי חישוב של ציון סופי שהינו קומבינציה קונווקסית של הציון הסטנדרטי על פי פופולריות והציון הסטנדרטי על פי דמיון להקשר. ההמלצות הסופיות מדורגות לפי הציון הסופי. במדידות שערכנו אנו מראים כי HybridCompletion טוב יותר מ-NearsestCompletion וכן מ-MostPopularCompletion. בפרט, הוא משפר את יכולת החיזוי של MostPopularCompletion ב-31.5%.

מכיון שהאלגוריתם שלנו מתבסס על דמיון קוסינוס הזוית המקובל כפונקציית דמיון בין מסמכים, ניתן לממש אותו ביעילות על ידי שימוש בארכיטקטורת חיפוש סטנדרטית. תכונה זו היא חיונית לאלגוריתם, מכיון שהשלמת שאילתות נעשית בזמן אמת, תוך כדי שהמשתמש מקליד את השאילתא.

מטרת העבודה הזו הינה להתמודד עם התרחיש המאתגר ביותר: בהינתן קלט של תחילית קצרה, יש לחזות את השאילתא של המשתמש בהסתברות גבוהה.

מכיוון שעבור קלט קצר אין לנו מידע על כוונת המשתמש, אנו מציעים לקחת בחשבון את ה"הקשר" של המשתמש בו נעשה החיפוש, העשוי לרמוז על צרכי המידע שלו. אנו קוראים לגישה זו "המלצת שאילתות תלויית הקשר". הקשר יכול להיות היסטוריית החיפוש של המשתמש, היסטוריית הגלישה שלו, תחומי עיניין, מידע המאוחסן במחשבו האישי, דואר אלקטרוני ששלח ועוד. בעבודה זו אנו מתמקדים בשאילתות האחרונות שביצע המשתמש כמקור להקשר. לדוגמה, אם השאילתא הקודמת של המשתמש היתה "מזון מהיר" וכעת הוא מקליד את התו "פ", נציע לו את השאילתא "פיצריה" מכיון שהיא קשורה לשאילתא הקודמת. הסיבות לכך שאנו מתמקדים בשאילתות קודמות הן: (1) הן נגישות למנוע החיפוש (2) בבדיקות שערכנו, מצאנו כי ל-49% מהחיפושים קדמה שאילתא אחרת בזמן קצר (פחות מ-30 דקות). כלומר, לכמות נכבדה של חיפושים ישנו הקשר של שאילתות קודמות המתאים לגישתנו.

אנו מציעים שיטה חדשה לשימוש בשאילתות קודמות לצורך חיזוי השאילתא הבאה של המשתמש. שיטתנו מתבססת על ההנחה כי כאשר ההקשר של המשתמש רלבנטי לחיפוש הנוכחי, ישנה סבירות גבוהה שהשאילתא שהוא עומד להקליד דומה לשאילתות ההקשר. הדמיון יכול להיות סינטקטי (לדוגמה, מזון מהיר -> מזון מהיר וזול), או סמנטי בלבד (מזון מהיר -> פיצריה). לפי בדיקות שערכנו, ב-56% מהמקרים בהם שאילתא עוקבת קשורה לשאילתא שקדמה לה, ינסח המשתמש שאילתא חדשה שאינה דומה סינטקטית לשאילתא הקודמת.

בהתבסס על הנחתנו לגבי דמיון שאילתא נוכחית לשאילתת ההקשר, אנו מציעים אלגוריתם בשם NearestCompletion העובד באופן הבא. בהנתן קלט x והקשר y, האלגוריתם יציע למשתמש השלמות של x הדומות ביותר ל-y. בחירת פונקציית הדמיון בין שאילתות אינה טריויאלית, זאת מכיון שהיא צריכה לקיים שתי דרישות. דרישה ראשונה היא שככל ששתי שאילתות דומות יותר על פי הפונקציה, כך הסבירות שלהן להיות ניסוחים שונים של אותו צורך במידע גבוה יותר. דרישה שניה היא שפונקצית הדמיון תקבל כקלט כל זוג שאילתות שהוא.

בעבודה זו אנו מציעים פונקציה חדשה למדידת דמיון בין שאילתות ע"י הרחבת שאילתא לייצוג עשיר של וקטור במימד גבוה. בהינתן שני וקטורים, הדמיון ביניהם יחושב ע"י מדד סטנדרטי כגון מדד קוסינוס הזוית. החידוש בשיטה שלנו הוא שהרחבת השאילתא מבוססת על שאילתות אחרות קרובות סמנטית. אנו מייצרים עץ

תקציר

המלצת שאילתות הינה אחד מהשירותים הפופולריים ביותר של מנועי חיפוש מודרניים. מטרתה של המלצת שאילתות היא לעזור למשתמש לנסח טוב יותר את הצורך שלו במידע, ובכך לשפר את חווית החיפוש. בעבודה זו אנו מתמקדים בהשלמת שאילתות באופן אוטומטי, שהיא אחת הדרכים הקיימות להמלצת שאילתות. השלמת שאילתות מתבצעת בזמן שהמשתמש מקליד את מילות השאילתא. הקלט הינו תחילית השאילתא שהוקלדה עד כה, כלומר שאילתא חלקית. הפלט הינו רשימה קצרה יחסית של השלמות אפשריות לרצף התווים שהוקלד. אחת המטרות של השלמת שאילתא, בה נתמקד בעבודה זו, הינו חיזוי השאילתא של המשתמש כדי לחסוך לו את ההקלדה המלאה של השאילתא.

העיקרון הראשי העומד מאחורי אלגוריתמים של המלצת שאילתות הינו "חוכמת ההמונים". מנוע החיפוש מציע למשתמש את השאילתות הפופולריות ביותר שנשאלו על ידי משתמשים אחרים בעבר. מידע זה ניתן להפיק מיומנים (לוגים) של שאילתות המנוהלים על ידי מנועי החיפוש. לדוגמה, עבור התחילית "יש" יציע מנוע החיפוש "גוגל" את השאילתות הפופולריות "ישראכרט" ו-"ישראל היום", בראש רשימת השאילתות המומלצות. נקרא לאלגוריתם מבוסס פופולריות בשם MostPopularCompletion.

כפי שציינו, השלמת שאילתות מתבצעת בזמן שהמשתמש מקליד את מילות השאילתא. בשלבים הראשונים, הקלט הינו מספר תווים קטן, וייתכן אף תו אחד בלבד. עבור קלט קצר, מרחב ההמלצות האפשריות גדול מאוד. בנוסף, תחילית קצרה של שאילתא הינה עמומה, ואינה מספקת מידע על כוונת המשתמש. מסיבות אלה, חיזוי שאילתת המשתמש בהנתן תחילית קצרה הינה משימה לא טריויאלית. ואכן, בניסויים שביצענו מצאנו כי אחרי הקלדת תו בודד, יכולת החיזוי של המלצת שאילתות על בסיס פופולריות הינה נמוכה.

- תודה להוריי, שרה ואלימלך וסטרייך, על שנתנו לי את הדחיפה ואת הכח להתחיל; על שתמכו, חיזקו, והאמינו במסירות ובהתמדה; ובעיקר, על שהפנימו בי את השאיפה להרחיב ולהעמיק את ידיעותיי, ללמוד ולהשכיל.
- תודה לחמי ולחמותי, יהושע ולאה קראוס, על תמיכתם בנדיבות וברוחב לב, ועל העזרה בכל עת, שאפשרה לי להפנות זמן ומשאבים ללימודים.
- תודה לילדיי האהובים, מתן, נועה ויובל, שנתנו לי את האור והשמחה שרק ילדים יכולים לתת.
- לבסוף, תודה לשרגא בעלי, על שהיה לצידי בכל זמן ובכל מצב, ותמך בדבקות לאורך כל הדרך, ועל האמונה בי יותר משהאמנתי בעצמי. בנוסף, תודה על העזרה הטכנית שתרמה למחקר עצמו.

ברצוני להקדיש מחקר זה לסביי ולסבתותיי זכרם לברכה, יפה ושמואל גולדנר, בלומה וחיים וסטרייך, שאיבדו את בני משפחותיהם בשואה, והעמידו דורות חדשים בארץ ישראל למרות הכל.

המחקר נעשה בהנחיית ד״ר זיו בר יוסף ופרופ׳ שאול מרקוביץ׳

תודתי נתונה לטכניון – מכון טכנולוגי לישראל
על התמיכה הכספית הנדיבה בהשתלמותי

תודות

״אמר רבי יצחק: אם יאמר לך אדם ׳יגעתי ולא מצאתי׳ – אל תאמין;
׳לא יגעתי ומצאתי׳ – אל תאמין;
׳יגעתי ומצאתי׳ – תאמין״.
(בבלי מגילה ו)

ברצוני להודות מקרב לב לכל האנשים שתמכו בי לאורך היגיעה שבדרך, ושבלעדיהם לא הייתי משלימה את עבודת המחקר.

- בראש ובראשונה, תודה לד״ר זיו בר-יוסף על ההנחיה המסורה; על ההשקעה הרבה, הרצינות, התמיכה והעידוד; על כל הדברים הרבים שלמדתי במהלך העבודה המשותפת. תודה על היכולת לשמור על האיזון העדין בין נתינת חופש מחשבתי לבין הכוונה.
- תודה לפרופ׳ שאול מרקוביץ׳ על ההנחיה המשותפת ועל תרומתו למחקר.
- תודה לד״ר אורן קורלנד על ההתעניינות במחקר ועל המשוב המועיל.
- תודה לכל ידידיי בטכניון על מילים טובות ועל שיחות מועילות.

המלצת שאילתות תלוית הקשר

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים במדעי המחשב

נעמה קראוס

הוגש לסנט הטכניון – מכון טכנולוגי לישראל

אייר תשע״ב חיפה אפריל 2012