CONQUER: A System for Efficient Context-aware Query Suggestions

Christian Sengstock
Institute of Computer Science
University of Heidelberg, Germany
sengstock@informatik.uni-heidelberg.de

Michael Gertz
Institute of Computer Science
University of Heidelberg, Germany
gertz@informatik.uni-heidelberg.de

ABSTRACT

Many of today's search engines provide autocompletion while the user is typing a query string. This type of dynamic query suggestion can help users to formulate queries that better represent their search intent during Web search interactions. In this paper, we demonstrate our query suggestion system called CONQUER, which allows to efficiently suggest queries for a given partial query and a number of available query context observations. The context-awareness allows for suggesting queries tailored to a given context, e.g., the user location or the time of day. CONQUER uses a suggestion model that is based on the combined probabilities of sequential query patterns and context observations. For this, the weight of a context in a query suggestion can be adjusted online, for example, based on the learned user behavior or user profiles. We demonstrate the functionality of CONQUER based on 6 million queries from an AOL query log using the time of day and the country domain of the clicked URLs in the search result as context observations.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Query formulation; Search process

General Terms

Algorithms

Keywords

Dynamic query suggestion, Query context

1. INTRODUCTION

Query autocompletion provides an important functionality of many of today's search engines. Autocompletion aims at helping users to formulate queries that better represent their search intent. A well-known example is the Google search autocomplete feature, which tries to predict queries while the user is entering a query string. We call this functionality *dynamic prefix-based query suggestion*, with prefix meaning the partial query string typed by the user. The circumstances when, where, and how a query is entered can have a significant influence on the relevance of suggestions. For a user in Berlin typing *'pizza'*, a local suggestion like *'... berlin mitte'* will have more relevance than *'... hut'*. Considering the temporal context, for a user typing *'berlin'* the suggestion *'... shopping'* might have more relevance at 2pm than at 9 pm, whereas it is the opposite case for the suggestion *'... concert'*. In addition to geographic and temporal contexts also other contexts like length of a query session or the clicked country domains of the URLs in a query hit list might have a valuable influence on suggestions.

While there exists a variety of approaches to query suggestion (also called *query recommendation*, see, e.g., [1, 3, 4, 7, 9, 11]), the underlying models are not that flexible with respect to the influence of query contexts. Furthermore, existing query recommendation approaches do not cover the case of prefix-based suggestions. This means that the input is a partial query string and should be extended by the intended query. Although the latter property is not exclusive for dynamic query suggestion, it is a valuable feature for a text-field autocompletion feature. As mentioned above, autocompletion features of today's search engines already take context information into account. However, there is no related work on a detailed discussion of the models and techniques underlying these approaches.

In this paper and the demonstration of our system called CONQUER, we show how prefix-based query suggestions can be efficiently realized by considering available context observations. To achieve this functionality, our system computes dynamic query suggestion based on combined probabilities of occurring sequential query patterns and context observations in real-time.

The remainder of this paper is organized as follows. In Section 2, we review related work. In Section 3, the context-aware query suggestion model is introduced. After an outline of the architecture of our system in Section 4, we give an overview of our demonstration scenarios in Section 5.

2. RELATED WORK

Query suggestion, also known as query recommendation or query intention prediction, is a growing research area. Several approaches have been suggested to compute and recommend queries similar to a given query string. Query session-based approaches suggest similar queries if they happen in the same query session. Fonesca et al. [4] present an approach to extract query transactions based on query sessions and to employ association rules to suggest queries. Zhang and Nasaroui [11] and Anagnostopoulos et al. [1] also consider the sequential order of queries in their models. A different approach to measure the semantic similarity of queries is by looking at the clicks in the query result (hit list). Mei et al. [7] model queries and URL-clicks in a graph and use random walks to measure query distances. Sahami

WWW 2011, March 28–April 1, 2011, Hyderabad, India.
ACM 978-1-4503-0637-9/11/03.

and Heilman [9] propose a URL-click based kernel function to measure query similarity. Content-based approaches measure query similarity using the content of the search result. Baeza-Yates et al. [3] introduce an approach based on document similarity of resulting URLs. Janses et al. [6] present a technique to propose a categorization of the user intents of Web queries using supervised and unsupervised clustering and identified four major intents that might be incorporated into suggestion models.

In contrast to these approaches, our base model is simple in that it uses the frequency of sequential query patterns. This textual similarity is an important property of dynamic prefix-based query suggestion, which, to the best of our knowledge, has not been studied in other work so far. Other than in session-, click-, or content-based approaches, not the semantic similarity but the frequency of textual patterns is used to measure the relevance of suggestions. For this, the relevance is tailored to the context within which the query is entered by the user. Context-aware suggestion models are also not covered in the related work on query suggestion so far. An approach addressing this aspect is by Backstrom et al. [2] where an evaluation of a large-scale query log regarding the geographic distribution of queries was studied and the importance of query contexts was shown.

3. QUERY SUGGESTION MODEL

In this section, we first present the basic query suggestion model and then discuss an extension of the model to include query context observations for computing query completions.

3.1 Query Pattern Probability

We treat each query as a sequence of words. We assume that a word subsequence in a query can be a valuable hint for suggesting a completion of the query string a user is starting to type. For example, assume that a lot of queries are *'italian restaurant'*. A valuable suggestion for *'it...'* is *'...alian restaurant'* and for *'rest...'* it is *'...aurant'*. However, we want to be aware of query word boundaries. For example, for the query *'li...'*, *'...an restaurant'* is not a suggestion, because *'lian restaurant'* is not a valid word subsequence.

We assume a query log-file, which we transform into a query-transaction database. For this, the queries are tokenized into word sequences. A query $Q = \langle w_1, w_2, ..., w_m \rangle$ thus consists of a sequence of m words $w_i \in W$, with W being the set of all words that occur in the log. Now assume the input query A is given as a sequence of words (and not yet an arbitrary query string). A is a subsequence of Q, denoted $A \sqsubseteq Q$, if the sequence A is contained in Q. Given a database of query transactions $\mathcal{T} = \{T_1, ..., T_n\}$ with $T_i = (id, Q)$ being a query transaction consisting of an id and a sequence of words Q, the support of an input query A is the number of query transactions in $\mathcal{T}$ where A is a subsequence, denoted $support(A)$.

The probability that a word sequence Q exists in $\mathcal{T}$ is simply the relative frequency $P(Q) = \frac{support(Q)}{|\mathcal{T}|}$. The conditional probability that a word sequence B exists, with B being an extension of A (denoted $A \oplus B$) is

$$P(B|A) = \frac{P(A \oplus B)}{P(A)}. \quad (1)$$

Given a query A as input, we can return the top-k query extensions with the highest conditional probabilities $P(B|A)$. The suggestions based on the conditional probability given A can be efficiently determined by first mining the set of frequent sequential query patterns

$$\mathcal{F}_\delta = \{Q \mid Q \sqsubseteq T.Q \in \mathcal{T} \wedge support(Q) > \delta\} \quad (2)$$

with δ being a user-defined support threshold and $F \in \mathcal{F}$ being a frequent sequential query pattern. By looking up all super-sequences $\mathcal{S}_A = \{F \mid F \in \mathcal{F}_\delta \wedge A \sqsubseteq F\}$ calculating the conditional probability for every extension B such that $S = A \oplus B$ becomes

$$P(B|A) = \frac{P(S)}{P(A)}. \quad (3)$$

If we know the frequent sequential query patterns and their support values, we can concatenate the queries $F \in \mathcal{F}_\delta$ and index them in a character prefix-tree. By adding the support information to the nodes where a query ends, we can look up all frequent sequential query patterns that are a superset of an arbitrary query string A and return the k patterns with the highest conditional probabilities.

3.2 Query Context Probability

Assume that for each query transaction $T \in \mathcal{T}$, a context observation $T.x$ exists. A query context observation can, for example, be the time of the day when the query was submitted, a geographic location or the age of the user. An observation $T.x$ is a valid event for the complete word sequence $T.Q$. Consequently, if the sequence $Q = T.Q$ occurs n times in the transaction database, we have a set of n observations $X_Q = (x_1, \ldots, x_n)$ for Q, which we treat as a random variable describing the observations for Q. Since a transaction T supports not only the word sequence $Q = T.Q$ but also every subsequence of Q, the observation $T.x$ is a valid event for every subsequence $A \sqsubseteq Q$. This means that we have a random variable X_F for every sequential query pattern $F \in \mathcal{F}_\delta$. For every word sequence Q, we can now use X_Q to calculate the probability $P(X_Q = x^*)$ that a given context event x^* happens.

For the simple case of a discrete random variable X describing the hour of the day when the query was submitted, we can use the relative frequency to calculate the probability that query Q was submitted on a given hour of the day x^*:

$$P(X_Q = x^*) = \frac{\text{number of events in } X_Q \text{ where hour is } x^*}{\text{number of all events in } X_Q} \quad (4)$$

3.3 Combined Model

The conditional probability of a query extension B and the probability that $S = A \oplus B$ happens given a context parameter x^* and the observations in X_S can be combined by parameterization of $P(B|A)$:

$$P(B|A; x^*) = P(B|A)\ P(X_{A \oplus B} = x^*). \quad (5)$$

If a number of context observations exists for each sequential query pattern, with $X_Q^i, 1 \leq i \leq l$ denoting the ith of the l context random variables for pattern Q, the general context-aware probability can be formulated as

$$P(B|A; C, \omega) = P(B|A) \prod_{i=1}^{l} P(X_{A \oplus B}^i = C_i)^{\omega_i} \quad (6)$$

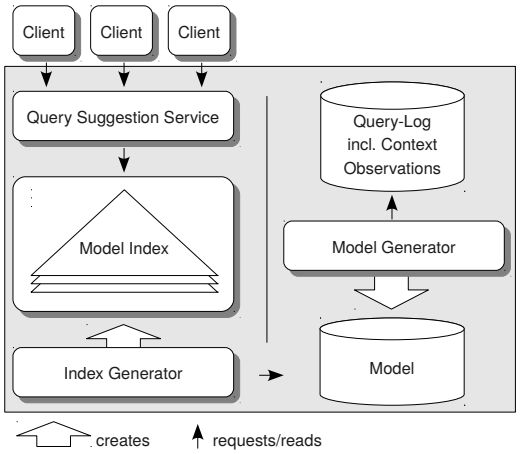

Figure 1: Basic components and data flow of CONQUER.

with $C_i \in C$ being the ith context parameter and $\omega_i \in \omega$ in the interval $[0, 1]$ being a weight to control the influence of the ith context parameter.

4. SYSTEM ARCHITECTURE

The architecture of the CONQUER system is composed of (1) a model generation component, (2) a model index, and (3) a suggestion service. The model generator performs the offline model generation by mining frequent sequential query patterns and their associated context aggregations (synopsis data structures). The patterns and their synopses are then used by the index generator to create the model index. The suggestion service queries the model index and provides a Web service interface to receive requests for query suggestions. The basic architecture is shown in Figure 1.

4.1 Model Generator

The model generator mines a query transaction database $\mathcal{T}$ for frequent sequential query patterns and aggregates the context observations of each pattern in synopsis data structures. The output of the model generator is a set $\mathcal{M} = \{M_1, M_2, ..., M_p\}$ of p tuples $M_i = (Q, count, syn_1, ..., syn_l)$ with Q being a frequent sequential query pattern, *count* the number of times the pattern occurs in the transaction database $\mathcal{T}$, and syn_i the synopsis data structure summarizing the ith of the l contexts of the pattern.

A synopsis can be any data structure that can summarize observations iteratively and that can be merged with another synopsis. A simple synopsis is a sparse vector keeping track of the number of added discrete observations and the total number of observations. Thus, one can easily retrieve the relative frequency of a given context value. Different synopses can be used to summarize continuous observations. However, we will not discuss them in this paper and refer to Zhang et al. [10] for a detailed discussion of an approach using the Clustering-Feature-tree.

To mine frequent sequential query patterns, we use an extended FP-Growth implementation. FP-Growth, introduced by Han et al. [5], is a divide-and-conquer algorithm for mining frequent itemsets. We use a preprocessing phase to add item-prefixes and a post-processing phase to extract

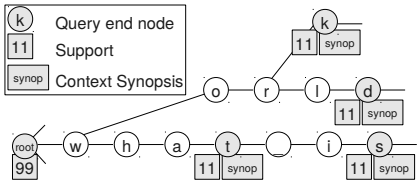

Figure 2: Query pattern prefix-tree containing count and context synopsis information.

only valid sequential patterns to allow for sequential pattern mining. An extension to FP-Growth for sequential pattern mining was proposed by Pei et al. [8]. However, for our needs a standard FP-Growth extension is sufficient. During the mining process the synopsis information is updated and merged in the same way as the pattern counts during conditional FP-tree construction. The additional space and runtime-complexity of our FP-Growth extension depend solely on the chosen synopsis data structure to aggregate the context observations. For example, aggregating the time of day observation using a histogram with bins representing fixed-size time intervals will cause an additional space-complexity of $O(1)$ per node in the FP-tree and $O(1)$ runtime-complexity overhead for each node update operation. Thus, it will not change the overall FP-Growth space- and runtime-complexity.

4.2 Model Index and Scoring

The model $\mathcal{M}$ generated by the model generator is indexed by a prefix-tree to allow for efficient query extension lookups, as shown in Figure 2. For this, the word sequences are concatenated to strings and each end node of a query pattern is marked in the tree with its count and synopsis data structure. This allows to efficiently determine the frequent sequential query patterns $\mathcal{S}_A$, given an arbitrary partial query string A. For each query extension B with $S = A \oplus B$, a score is calculated using the generic context-aware probability $score(B) = P(B|A; C, \omega)$. Given an arbitrary query string A, a list of context parameters C, and the context weights ω, the index can then be queried to return the top-k scored extensions.

4.3 Query Suggestion Service

The suggestion service provides an interface to receive requests for query suggestions from clients (JSON/REST-based). The layer input is a query prefix A, the context parameters C and their weights ω. The layer forwards the input to the model index layer to look up the top-k scored query extensions. The service layer is able to receive the index results as asynchronous notifications in parallel environments. When all parallel running tree traversals returned their query suggestions, the top-k results can be merged and are returned to the service client.

5. DEMONSTRATION

Our demonstration is based on the AOL-query log-files published in 2006. The complete dataset contains 60M query transactions. For each query, the following parameters are available: *userID*, *query*, *timestamp*, *clicked URL*, and *position of clicked URL in result list*. We use a sample of 6M query transactions and extract the hour of the day from the

Figure 3: AOL-query suggestion showing completions for query *work* at 6 am, 3 pm and 9 pm.

timestamp as a statistical observation. We also extract the country domain of the clicked URL as a discrete observation. This allows to suggest queries based on a given query prefix A and the context parameters *hour of day* and *clicked country domain in the query result*. Mining for frequent sequential patterns with a minimum support threshold of 3 results in a total number of $841,255$ query patterns indexed in the model index. The demo includes a Web GUI that allows to type an arbitrary query string for which a maximum of 10 query suggestions are displayed while entering the query. The suggestions are ordered by their score. Sliders and check-boxes permit to change the context parameters and to set the weight of a context.

5.1 Explore Context Dependency

The GUI allows to change the values of the context parameters on the fly so that the context dependency of the suggestion can be explored dynamically. For example, when typing the query prefix A = *'work'*, 10 suggestions are displayed that are dominated by the very frequent pattern *'works'*. When moving the hour slider, the suggestions change as shown in Figure 3. At 6 am one of the top suggestions is *'workwear'* while at 3 pm it is *workerforce* and at 9 pm it is *'workout'*. Sliding the hours gives intuitive results for a large number of queries and demonstrates the great dependency of query suggestions by time of day.

Exploring the dependency for the *clicked country domain in the query result* context is possible by selecting a corresponding domain check-box. This ranks those query suggestions higher where users have more likely clicked a URL with the checked country domain. The AOL query logs are dominated by *.com* domains and consequently the dependency of the queries by the context value is less significant than the temporal context. However, some valuable examples exist for country specific queries like *berlin*, with the according country domain selected. For example, the suggestion *berlin germany* gets a lower score due to its higher relevance for English speaking users. Also, when preferring queries leading to *.gov* pages by selecting the corresponding check-box, a significant change to suggestions having a governmental content can be recognized.

5.2 Dynamic Context Weight

As indicated in Section 3, the context dependency can be weighted. A weight of 0 means that the context has no influence on the suggestion, a value of 1 means that the query pattern probability for a given partial query A is multiplied by the probability of the context value. Thus, if a suggestion has a low probability for a given context value, it will less likely be considered a suggestion. The weight of the context influence can be set on the fly by moving the corresponding slider. Given a query prefix A, the suggestions change dynamically while the slider is moved. Learning the context weight dynamically can be a valuable feature, for example, by evaluating the context probabilities of selected suggestions. A user choosing suggestions that have a high context probability can be supported by increasing the weight of the context. In our demonstration we show how the suggestions will change based on the chosen weight value.

6. REFERENCES

[1] A. Anagnostopoulos, L. Becchetti, C. Castillo, and A. Gionis. An Optimization Framework for Query Recommendation. In *Proc. WSDM '10*, 161–170, 2010.

[2] L. Backstrom, J. Kleinberg, R. Kumar, and J. Novak. Spatial variation in search engine queries. In *Proc. WWW '08*, 357–366, 2008.

[3] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Query Recommendation using Query Logs in Search Engines. In *Int. Workshop on Clustering Information over the Web (with EDBT '04)*, 588–596, 2004.

[4] B. Fonseca, P. Golgher, E. de Moura, and N. Ziviani. Using association rules to discover search engines related queries. In *Proc. LA-WEB '03*, 66–71, 2003.

[5] J. Han, J. Pei, Y. Yin, and R. Mao. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, Jan. 2004.

[6] B. J. Jansen, D. L. Booth, and A. Spink. Determining the User Intent of Web Search Engine Queries. In *Proc. WWW '07*, 1149–1150, 2007.

[7] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *Proc. CIKM '08*, 469–478, 2008.

[8] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Mining sequential patterns by pattern-growth: the PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering*, 40(11):31–1440, Nov. 2004.

[9] M. Sahami and T. D. Heilman. A Web-based Kernel Function for Measuring the Similarity of Short Text Snippets. In *Proc. WWW '06*, 377–387, 2006.

[10] T. Zhang, R. Ramakrishnan, and M. Livny. Fast Density Estimation Using CF-kernel for Very Large Databases. In *Proc. KDD '99*, 312–316, 1999.

[11] Z. Zhang and O. Nasraoui. Mining Search Engine Query Logs for Query Recommendation. In *Proc. WWW '06*, 1039–1040, 2006.