Synthesizing High Utility Suggestions for Rare Web Search Queries

Alpa Jain
Yahoo! Labs
alpa@yahoo-inc.com

Umut Ozertem
Yahoo! Labs
umut@yahoo-inc.com

Emre Velipasaoglu
Yahoo! Labs
emrev@yahoo-inc.com

ABSTRACT

Search engines are continuously looking into methods to alleviate users' effort in finding desired information. For this, all major search engines employ query suggestions methods to facilitate effective query formulation and reformulation. Providing high quality query suggestions is a critical task for search engines and so far most research efforts have focused on tapping various information available in search query logs to identify potential suggestions. By relying on this single source of information, suggestion providing systems often restrict themselves to only previously observed query sessions. Therefore, a critical challenge faced by query suggestions provision mechanism is that of coverage, i.e., the number of unique queries for which users are provided with suggestions, while keeping the suggestion quality high. To address this problem, we propose a novel way of generating suggestions for user search queries by moving beyond the dependency on search query logs and providing synthetic suggestions for web search queries. The key challenges in providing synthetic suggestions include identifying important concepts in a query and systematically exploring related concepts while ensuring that the resulting suggestions are relevant to the user query and of high utility. We present an end-to-end system to generate synthetic suggestions that builds upon novel query-level operations and combines information available from various textual sources. We evaluate our suggestion system over a large-scale real-world dataset of query logs and show that our methods increase the coverage of query-suggestion pairs by up to 39% without compromising suggestion quality or utility.

Categories and Subject Descriptors

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

General Terms

Algorithms, Design, Experimentation, Measurement

Keywords

Query recommendations, search query logs, rare queries

SIGIR'11, July 24–28, 2011, Beijing, China.

1. INTRODUCTION

Search engines provide a variety of tools to assist users in better formulating their information needs. Examples include, suggestions for query completion, spell corrections and query reformulation. Among these, *post-submit query reformulation* suggestions, typically found in the left-column of a search engine's results page, focus on providing follow-up queries related to a user's original query. These suggestions can be categorized as: (1) *specializations* that expand the original query by adding terms, (2) *generalizations* that drop terms from the original query, and (3) *laterals* that suggest alternatives related to the original query and do not lexically overlap with the original query (e.g., suggesting 'myspace' to 'facebook').

Generating query suggestions has been the focus of several past research efforts [8, 13, 25, 29, 35] (see Section 7). Virtually all query suggestion techniques strictly depend on search query logs to discover query suggestions with sufficient statistics. Particularly, to mine lateral suggestions, the primary source of information is past *user sessions*, i.e., sequence of queries issued by a single user within a specific time window [4]. Unfortunately, relying on query session logs can restrict the coverage of queries for which post-submit suggestions are available: It is well known that the query distributions in web search are heavy tailed. To be more precise, in the query logs of a major search engine, we found that about 30% of unique queries that are observed in a given month are queries that were not seen within the past year before that month. User session logs are even further infrequent: in our experiments, we mined user session logs to derive lateral suggestions and over a random sample observed that 26.6% of unique queries have more than 5 suggestions, and 7.4% have between 1 and 4 suggestions, and the biggest portion is the queries with no suggestions that constitute 66% of the unique query traffic.

In this paper, we look into *building synthetic query reformulation suggestions*, i.e., suggestions that are not solely based on past user sessions. Our work is motivated by the following observation. Consider a simple query such as 'supercuts new jersey' for which no suggestions were provided by all major search engines. By appropriately tokenizing and relaxing this query, we may ascertain that 'supercuts' can be substituted by 'great clips' or 'fantastic sams' or 'cost cutters.' Further, by appending the context 'new jersey' from the original query, we can generate query suggestions such as 'great clips new jersey' or 'fantastic sams new jersey.' In line with this example, we are adding a forth category of suggestions which can be regarded as *partially lateral*, where we break the query into segments and make a lateral move of the parts, such as suggesting 'buca di beppo nutrition information' for 'macaroni grill nutrition information', or 'tampa bay fisheries' for 'tampa bay agriculture.'

The above example underscores several interesting challenges that need to be addressed. First, generating post-submit query suggestions differs from the task of query rewriting [3, 9, 16, 22, 26, 33, 34] which has mostly focused on rewriting queries to improve search results. Generating query suggestions differs from query rewriting in that we are now also interested in reformulations that are semantically related but not identical to the original query. For instance, for the user query 'tampa bay fisheries' we may provide 'tampa bay agriculture' as an interesting (lateral) query suggestion; however, this reformulation would not be considered for the task of improving search results since the latter query brings about different search results. Second, candidates for applying transformations need to be systematically identified. For instance, given the query 'supercuts new jersey' not all terms (e.g., 'new') can be substituted by related terms. Third, exploring synthetic queries can lead to suggestions that are not 'well-formed' or suggestions that are not semantically related to the user query. For instance, for the query 'tampa bay fisheries' we may generate 'tampa tribune fisheries' after identifying 'tampa bay' and 'tampa tribune' as related concepts. The resulting suggestion is however garbled and should be pruned. Finally, most query re-writing techniques consider query reformulation on a single query basis and do not account for the *utility* of the suggestion set as a whole. For instance, the query 'hawaii sunset pictures' could be reformulated as 'hawaii sunset photos' or 'hawaii sunset images'; while these suggestions are well-formed and relevant to the original query, they are less likely to differ in their result sets and thus in turn yield a low utility suggestion set. Note that this is not an issue when rewriting queries to improve search results.

To address these challenges, we develop an end-to-end system (see Figure 1) for synthesizing query suggestions. Our approach relies on a variety of query transformations over the original query in order to systematically explore candidates for query suggestions. As a key contribution, these generation methods do not solely rely on past user sessions but instead using query logs as a guide considers in concert several sources such as individual queries (i.e. queries not restricted by a session) in query logs, semantic information from web pages, and search results page. Given a set of candidate suggestions, we propose a holistic ranking mechanism which takes into account suggestion's (a) *well-formedness*, (b) *relevance* to the original query, and (c) *utility* to the user given other suggestions. In summary, our main contributions are as follows:

- A general framework for *synthesizing* suggestions for query reformulations that goes beyond query session logs.
- A variety of query transformations to systematically generate suggestion candidates (Sections 2 and 3).
- A holistic suggestion ranking technique that focuses on suggestion quality and utility (Section 4).
- An extensive experimental evaluation that includes user studies and real-world datasets (Section 5 and 6).

2. RELAXING USER QUERIES

Query suggestions is an assistance technique provided by search engines where given a user query q the search engine returns an ordered set S of suggestions for follow-up queries. For a large fraction of queries (e.g., tail queries), providing query suggestions is challenging since user session logs may not contain sufficient information. Our goal is to build for such queries an ordered set of S of useful suggestions for follow-up queries that are relevant to the user's original mission.

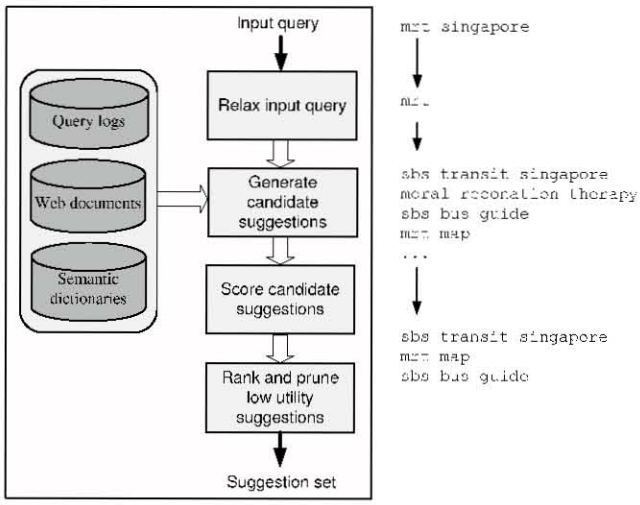

Figure 1: Stages of synthesizing query suggestions.

Given a query q, we begin by relaxing q to query q' by identifying and eliminating phrases in q that are not deemed important to the original information sought by the user. The intuition for this step is that rare queries often contain extraneous terms that are not critical to the underlying user intent. Such long queries thus suffer from the problem of insufficient query log information [22], and by eliminating unimportant terms we can derive the root query for which we may have richer information in the query logs or web pages. For example, the term "store" in the query "big lots furniture store" does not bring any significant information, hence we could relax this query to "big lots furniture" and further to "big lots."

We formulate the problem of identifying non-critical terms in a user query as a sequence labeling problem. More specifically, a user query q is treated as a sequence of tokens $\mathbf{x} = u_1 u_2 \cdots u_n$ which needs to be assigned a sequence of labels $\mathbf{y} = y_1 y_2 \cdots y_n$ such that $y_i \in \{C, D\}$. A label of C indicates a critical term whereas a label of D indicates a term that can be dropped from the original query. A promising approach for solving sequence labeling problems is Conditional Random Fields (CRFs) [23].

Training data: A practical challenge here is generating annotated data consisting of queries where each term in the query is labeled as C or D. Building large-scale annotated data that comprehensively covers a wide classes of queries is challenging and further such data is not readily available. To address this, we propose a method to automatically construct annotated data based on information available in search query logs. Our method is based on the observation that users often reformulate their queries when their original query did not lead to any interesting search results and furthermore, these reformulations contain important information regarding which terms may be replaced (i.e., dropped terms) and which terms may not be replaced (i.e., critical terms). Specifically, starting with user search queries, we split them into search sessions by considering all queries that occurred within a time frame of 15 minutes. Within a query session, we look at subsequent queries such that (a) the queries differ by one term, (b) the first query did not lead to clicks on any of the results, indicating an unsuccessful query or uninteresting search results page, (c) the second query led to at least one click on the results, indicating a potentially interesting search results page. Given a query pair q_1, q_2, we label all terms that are common to both queries as C and the terms that were removed from q_1 when generating q_2 as D.

Features: We used a combination of features including lexical-, query-logs-frequency-, and dictionary-based features derived from

description	source
frequency of t_i	query logs
standalone frequency of t_i	query logs
pairwise mutual information for (t_i, t_{i-1})	query logs
is first name	dictionary
is last name	dictionary
is location	dictionary
is stop word	dictionary
is wikipedia entry	wikipedia
has digit	lexical
has punctuations	lexical
position in query	lexical
length	lexical

Table 1: CRF features for each term t_i in a query $q = t_1, t_2, \cdots, t_n$.

various sources such as three months of query logs of a commercial search engine, Wikipedia, etc. Table 1 describes our feature set along with the sources used to derive them. In particular, given a query $q = t_1, t_2, \cdots t_n$ along with labels $l_1, l_2 \cdots l_n$ for each term, respectively, we compute the listed feature for each term t_i. Of the features shown in Table 1, the query-logs-based features, namely, standalone frequency and pairwise mutual information need further discussion.

The *standalone score* tries to capture whether a given term is an entity or a real-world concept or not: intuitively, a concept (e.g., california, ipod, madonna) should often occur in a standalone form among the search query logs. Therefore, among the query logs we must find queries of the form $Q == t_i$, capturing the fact that users are looking to learn more about the given concept. More formally, we compute the query-log-based standalone score as:

$$s(t_i) = \frac{|Q == t_i|}{|queries\ that\ contain\ t_i|} \quad (1)$$

The *pairwise mutual information (pmi)* score is computed for a pair of consecutive terms occurring in the query: intuitively, this score measures the "cohesiveness" of pairs of terms (e.g., 'san francisco' would have a high pmi score compared to 'drinking water'). More formally, we compute the pmi score for a pair (t_i, t_{i+1}) as:

$$pmi(t_i, t_{i-1}) = log \frac{C(t_i, t_{i+1})}{C(t_i) \cdot C(t_{i-1})} \quad (2)$$

where $C(x)$ is the number of queries that contain term x and $C(x, y)$ is the number of queries that contain ordered pair (x, y).

Upon eliminating terms that are labelled as non-critical, we use the resulting relaxed queries to generate candidates for query suggestions for the original query. Note that the set of relaxed queries themselves serves as suggestion candidates.

3. GENERATING SUGGESTION CANDIDATES

We now discuss a suite of algorithms to generate candidates for query suggestions for a query. Our algorithms *extend* information available in session logs utilizing information from query logs and web pages to simulate user sessions and derive suggestions.

3.1 Co-occurrence in query sessions

While searching for information, users usually issue related queries within a given session [4]. Sometimes they add or drop a term to their queries, or sometimes make a *lateral move* such as issuing "nikon d40" and "canon 50d". One rich source for generating suggestion candidates for a given query is to fetch the past session logs, and find queries that it is being manually formulated into [20, 29]. In practice, this rich reformulation data is typically available only for frequent queries and thus, we work with the root query obtained after query relaxation.

A simple approach to identify candidates would be to provide queries with high reformulation probability $p(q_n|q_c)$ as suggestions. This can simply be measured over the frequency counts in the session logs:

$$p(q_n|q_c) = \frac{f(q_n, q_c)}{f(q_c)} \quad (3)$$

where $f(q_n, q_c)$ is the frequency that these two queries are issued by the same user within a short time frame (will be referred as co-occurrence hereafter). The problem with using reformulation probability as the merit for suggestion candidate generation is that a query that is not dependent on q_c might have a high reformulation probability just because of its high marginal probability. To account for this, one can use pointwise mutual information (PMI) instead,

$$pmi(q_n, q_c) = log\left(\frac{f(q_n, q_c)}{f(q_c)f(q_n)}\right) \quad (4)$$

where $f(q_n)$ and $f(q_c)$ are individual marginal counts. The difference is that PMI normalizes the reformulation probability by a factor of $f(q_n)$; hence, it measures the dependency of these two queries. One weakness of PMI is that it might become very unstable for pairs of rare queries. As $f(q_c)$ and $f(q_n)$ get low, even a single coincidental co-occurrence might lead to a high PMI value. To account for this, one can use reformulation log-likelihood ratio (LLR) instead [20],

$$\begin{aligned} LLR(q_n, q_c) = {} & p(q_n, q_c)\ pmi(q_n, q_c) + p(q_n, \overline{q}_c)\ pmi(q_n, \overline{q}_c) \\ & + p(\overline{q}_n, q_c)\ pmi(\overline{q}_n, q_c) + p(\overline{q}_n, \overline{q}_c)\ pmi(\overline{q}_n, \overline{q}_c) \end{aligned}$$

where $\overline{q}_{next}$ denotes the set of all queries except q_n and similarly for q_c. LLR fixes the stability problem by taking the size of the all session data into account, and when the marginal query frequencies $f(q_n)$ and $f(q_c)$ get lower, other terms will start to dominate. We generate suggestion candidates using query reformulations with LLR above a certain threshold optimized empirically. (We will provide details about the session data later in Section 5). Additionally, we also apply various filters to guard against robots and spam details of which are out of scope of this paper.

As an example of applying this co-occurrence-based operator, for the query 'acoustic guitar strings', we obtain 'classical guitar' as a suggestion candidate after eliminating the term 'strings' from the original query. Or for 'cheap vegas hotels resorts', it gives 'las vegas specials' after dropping the term 'cheap'.

3.2 Semantic relations from web corpus

Our next method, explores candidates by replacing query terms by terms that are *distributionally similar* (i.e., synonyms, siblings, hypernyms, etc.) to them. We apply this transformation to both critical as well as the dropped query terms. This transformation relies on distributional similarity methods [24] that model the *Distributional Hypothesis* [14]: the distributional hypothesis links the meaning of words to their co-occurrences in text and states that *words that occur in similar contexts tend to have similar meanings*.

In practice, distributional similarity methods that capture this hypothesis are built by recording the surrounding contexts for each term in a large corpus and storing them in a *term-context matrix* [24]. Term-context matrix consist of weights for contexts with terms as rows and context as columns, and each cell x_{ij} is assigned a score to reflect the co-occurrence strength between the term i and context j. Methods differ in their definition of a context (e.g., text window or syntactic relations), or in their means to weigh contexts (e.g., frequency, tf-idf, pointwise mutual information), or ultimately in measuring the similarity between two context vectors (e.g., using Euclidean distance, Cosine, Dice). We build a term-context matrix as follows: we process a large corpus of text (e.g.,

web pages in our case) using a text chunker [24]. Terms are all noun phrase chunks with some modifiers removed; their contexts are defined as their rightmost and leftmost stemmed chunks. We weigh each context f using pointwise mutual information [7]. Specifically, we construct a pointwise mutual information vector $PMI(w)$ for each term w as: $PMI(w) = (pmi_{w1}, pmi_{w2}, \cdots, pmi_{wm})$, where pmi_{wf} is the pointwise mutual information between term w and feature f and is derived as:

$$pmi_{wf} = log\left(\frac{c_{wf} \cdot N}{\sum_{i-1}^{n} c_{if} \cdot \sum_{j-1}^{m} c_{wj}}\right) \quad (5)$$

where c_{wf} is the frequency of feature f occurring for term w, n is the number of unique terms, m is the number of contexts, and N is the total number of features for all terms. Finally, similarity scores between two terms are computed by computing a cosine similarity between their pmi context vectors [31]. As an example of applying the distributional similarity-based operator, for the query 'football hall of fame 2010 inductees' we generate 'football hall of fame 2010 award winners' since 'inductees' and 'award winners' are distributionally similar phrases.

3.3 Substitutions from co-clicked URL queries

Aside from co-occurrences, another dimensionality in the session logs that can bring additional valuable information is to consider the queries that lead to clicks to common URLs as in [29]. Query pairs with co-clicked URLs can be used to build a *substitutions dictionary*. This dictionary can be used to provide alternatives for the root query as well as for providing substitutions for the dropped terms *in a context-aware manner*, such that the term "turkey" can be substituted with "thanksgiving" in the context of "recipes" and with "turkish" in the context of "embassy" but not the opposites.

To build a substitutions dictionary, we build a query-URL bipartite graph with queries and urls as nodes and an edge $q \rightarrow u$ between a query node q and an url node u if a user clicked on url u after issuing query q. To eliminate outliers or noisy data, we only retain edges with clickthrough rate (i.e., clicks vs. views) above 0.01. Using this graph, we identify all pairs of queries that are connected to at least 2 and at most 10 common URLs[1]. There are two things that we have considered while constructing the query-URL bipartite graph:

- We removed URLs that are connected to more than 200 queries (most of which turn out to be very popular destination pages like `youtube.com`, `amazon.com` etc.). This prevents us from connecting vaguely related queries and bringing in irrelevant substitutables.
- For the URLs from tail domains, we used the domain-level information instead of the particular URL. If for a given domain, we have less than 30 unique queries leading to a click to a URL in that particular domain, we consider this a as a tail domain, and define the co-clicks over the domain, instead of the particular URLs. We found that this helps for enriching the suggestion sets for tail intents without losing the context.

When generating suggestion candidates, we simply look into the substitution dictionary for all possible substitutions. For example, for the query 'turkey recipes', if the relaxed query is 'turkey', we look at the substitutables of the dropped term 'recipes' to find 'roasting times', 'stuffing recipe' , 'how to roast' and many others, and generate the candidates 'turkey roasting times', 'turkey stuffing recipe' and 'how to roast turkey'[2].

3.4 Context from original query

After exploring suggestion candidates of the relaxed query, in many cases it is helpful to push the dropped term back into the query to generate a *partially lateral* move. Naturally, this context is only pushed to suggestions without the dropped terms or their substitutions. As an example, for the query 'acoustic guitar strings', we generated 'classical guitar' using the co-occurrence-based operator (Section 3.1) to which we push the term 'strings' to generate the suggestion 'classical guitar strings.' This operator re-inserts the original query context into the suggestions.

To summarize, we consider an expanded set of suggestions derived by relaxing the original query as well as using different combinations of operators discussed above.

4. RANKING SUGGESTIONS

While combining various information sources and text-based operations discussed in the previous section allow us to systematically construct candidates for query suggestions, they may also lead to erroneous query suggestions. Specifically, we may have at hand suggestions that are garbled; for example, in our experiments, for the query 'tampa bay fisheries' one of the suggestions generated using co-occurrence data and pushing the dropped term from the query was 'tampa tribune fisheries' (the original query was relaxed to generate 'tampa bay' for which session co-occurrence data produces 'tampa tribune' as a suggestion). Similarly, the candidate suggestions by applying various operators may not be relevant to the original query. To address these issues, we build a supervised ranking model to decide which candidate suggestions are valid.

The desiderata for ranking candidate suggestion for a given query include: (a) *well-formedness* of the suggestion (Section 4.1), (b) *relevance* to the query (Section 4.2, and (c) *utility* to users (Section 4.4). Additionally, we use a variety of lexical and syntactical features and train a gradient boosted decision tree (GBDT) [12] over this feature space for ranking candidate suggestions.

4.1 Well-formedness of a suggestion

In this step, our goal is to demote suggestions that are garbled, i.e., that do not conform to a real-world concept or language formulation. For instance, the suggestion 'tampa tribune fisheries' is not a a real-world concept and similarly 'marilyn monroe compendium' is a garbled suggestion. This is a critical in our setting since the suggestions are 'user-facing'; in contrast traditional query rewriting methods send rewritten queries to the search engine (i.e., backend) where the engine simply returns few or no results in case of poorly formed queries. To capture the well-formedness of a suggestion, we use *statistical language models* which is a probability distribution $P(s)$ over a sequence $w_1, w_2, \cdots w_m$ of words expressed as: $P(w_1, w_2, \cdots w_m) = \prod_{i-1}^{m} P(w_i|w_1, w_2, \cdots, w_{i-1})$. We use an n-gram model which computes the above probability based on "memory" of past (n-1) words given as:

$$P(w_1, w_2, \cdots w_m) \approx \prod_{i=1}^{m} P(w_i|w_{i-(n-1)} \cdots, w_{i-1})$$

In our experiments, we use a tri-gram model where

$$P(w_1, \cdots w_m) \approx \prod_{i=1}^{m} P(w_i|w_{i-2}), w_{i\ \ 1})$$

which is estimated using a *maximum likelihood estimator* as:

[1] We decided to use an upper bound here as well, because with too many common URLs, the query pairs are starting to become synonymous with almost identical result sets, which provides almost no utility to the users. This will be detailed in Section 4.4.

[2] While generating the candidates we keep the structure of the co-clicked URL queries and do not put the placed term into the position of the dropped term (as it would end up with many garbled suggestions like 'turkey how to roast'

$$P(w_i|w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}w_{i-1}w_i)}{w_{i-2}w_{i-1}} \quad (6)$$

where $C(w_{i-2}w_{i-1}w_i)$ is the relative frequency of observing the word w_i given that it is preceded by the sequence $w_{i-2}w_{i-1}$.

A common problem in building language models is that of word sequences that do not occur in the training set used to compute quantities in the above equation. In such cases, $C(w_{i-2}w_{i-1}w_i)$ and therefore $P(w_i|w_{i-2}, w_{i-1})$ equals 0. To address this problem, several smoothing techniques have been proposed that appropriately discount the MLE estimates in (6) and use the left-over probability mass for sequences not observed in the training data. Several smoothing techniques have been proposed in the past and we use a commonly used method, namely, Kneser-Ney smoothing [21]. Kneser-Ney smoothing interpolates higher-order models with lower-order models based on the number of distinct *contexts* in which a term occurs instead of number of occurrences of a word. Formally,

$$P(w_3|w_1w_2) = \frac{max\{C(w_1, w_2) - D, 0\}}{C(w_1, w_2)} + \frac{D}{C(w_1, w_2)} N(w_2) P(w3|w_2) \quad (7)$$

where D is a discount factor [21] and $N(x)$ is the number of unique contexts following term x.

An important observation in our setting is that we are dealing with potentially valid suggestions that may not occur in the query logs. Specifically, since our suggestions are synthetically derived we want to incorporate sources other than query logs. With this in mind, we build and combine language models from query logs as well as web pages. We combine these models as:

$$P_{\mathcal{C}}(w_3|w_1, w_2) = \lambda \cdot P_{\mathcal{Q}}(w_3|w_1, w_2) + (1 - \lambda)P_{\mathcal{W}}(w_3|w_1, w_2)$$

where λ is the interpolation weight optimized on a heldout training set. In the ranking step, we use $P_{\mathcal{C}}$, $P_{\mathcal{Q}}$ and $P_{\mathcal{W}}$ as features that capture well-formedness of a candidate.

4.2 Relevance to original query

To assess the relevance of each suggestion candidate to the original query, we designed three types of features for each pair of query and its suggestion candidate. In this section, we discuss how we derive these features using query logs as well as search results.

4.2.1 Click-vector similarity

An obvious relevance feature along the lines with the co-clicked URL candidate generation method is to look at the overlap between the clicked URLs for each query pair. For a given query, consider the following document click-vector over the set of all documents $\mathbf{cl}(q) = [cl_1(q), cl_2(q), \ldots, cl_K(q)]$, where K is the number of clicked documents. We calculate the cosine similarity between the overlapping URLs in the click-vectors of each query and suggestion candidate pair as our first relevance feature,

$$Sim_{click} = \frac{\mathbf{cl}(q_1) \cdot \mathbf{cl}(q_2)}{\|\mathbf{cl}(q_1)\|\|\mathbf{cl}(q_2)\|} \quad (8)$$

This feature by definition is non-zero for the candidates that are coming from the co-clicked URL candidate generation method. On the other hand, the coverage over other candidate generation methods is quite low since we are working with synthetic suggestions not observed in the past. Note that although this feature relies on query logs, it does not require the query and suggestion to occur in the same session; later in this section, we derive features that are not derived from query logs.

4.2.2 Context-vector similarity

A relevance feature along the lines with the distributional hypothesis is to look at the similarity of the distribution of other terms that each query is searched along with. For example, consider two queries q_1 and q_2, and assume that in the session logs, the most frequent queries that include these two queries include "$\langle q_1 \rangle$ download", "$\langle q_2 \rangle$ download", "install $\langle q_1 \rangle$" and "install $\langle q_2 \rangle$". From the context with which the query is searched together with, it is clear that both queries are software related.

For a given query, consider the context vector as the frequencies of the terms that it is searched along with $\mathbf{co}(q) = [f_1, f_2, \ldots, f_L]$, where L is the number of co-queried terms in the session logs. Similar to above, we evaluate the cosine similarity for a pair of context-vectors of the overlapping context terms of each query - suggestion candidate pair as:

$$Sim_{context} = \frac{\mathbf{co}(q_1) \cdot \mathbf{co}(q_2)}{\|\mathbf{co}(q_1)\|\|\mathbf{co}(q_2)\|} \quad (9)$$

Although this is a useful feature for frequent queries, over a uniformly sampled query set the context-vector similarity has a very low coverage. In our experiments, many queries in our data set are rare (and long) queries that have a very low probability of being searched within other queries, and do not have a context vector. Furthermore, the suggestion candidates that are coming from substitutions from web corpus are not even guaranteed to be observed in the query logs, hence context is not defined.

Require: Concept dictionary D, query q
1: Retrieve set R of top-k results for q
2: T = Terms from D contained in R
3: Eliminate from T terms in q
4: **for** term $t \in D$ **do**
5: $d(t)$ = number of results that t appears
6: $r(t)$ = total rank that t appears
7: $R(t) = \frac{((k+1)\ d(t)) - r(t)}{d(t)\ k}$
8: $S(t) = \frac{d(t)\ R(t)}{k}$
9: **end for**
10: Get the 20 terms with highest score $S(t)$

$$\mathbf{ab}(q) = [S(t_1), S(t_2), \ldots, S(t_{20})]$$

Algorithm 1: Algorithm to compute the aboutness vector.

4.2.3 Web-based aboutness similarity

So far, we discussed two features based on user query logs (i.e., observed data); however, of course, these features are sparse and will fail to assess the relevance of rare queries or queries that are *not* observed in the past. In the absence of session logs, one can use the search engine itself and look at the results to determine how related two queries are. Specifically, an earlier work by Raghavan and Sever [28] compares the ordered result sets returned for each query to measure query similarity. Their method requires ranking of all documents and $O(N^2)$ complexity, where N the number of documents, which is intractable for the web search scenario.

A subsequent approach is given by Fitzpatrick and Dent where they use the set overlap of the top k documents [10]. This is a tractable solution, however the result set overlap is only good at finding almost synonymous queries, and the overlap drops sharply when the queries are related but not (almost) identical. In the context of query suggestions, we are not interested in identifying nearly identical queries; instead, we need to assign reliable relevance scores to related query pairs as well. For example, although they are related and would make useful suggestions for each other, 'python' and 'ruby' have zero results in common in top 50. Thus, a measure simply based on the search results is insufficient and our next feature is captures what the results are *about*.

To assess a suggestion's relevance to the query, we build an *aboutness vector* [6] of the query and the suggestion and compute the similarity between these vectors. An aboutness vector suc-

cinctly describes a document and is represented as a set of salient *concepts* in the document along with their scores. Methods to build aboutness vectors differ in their definition of concepts (e.g., terms, named-entities, unigrams). In this work, we rely on a pre-built concept dictionary built using the method proposed in [2]. Our dictionary consist of 27 million phrasal concepts and named-entities obtained from a large web crawl as well as well as query logs and is intended to cover most "interesting" phrases on the web. (For steps to build the concept dictionary please refer to [2].)

Algorithm 1 describes our algorithm to compute the aboutness vector for a given a query q using the concept dictionary D. The final score $S(t)$ is a function of $d(t)$, the number of documents that the term t appears and, $R(t)$ the rank score. $R(t)$ gives higher weights to the terms that are contained in higher ranked documents; since documents that are ranked higher are more important than ones that rank lower, terms that are contained in higher ranked documents are more important. Given the aboutness vectors of the query q_1 and the suggestion candidate q_2, we calculate the cosine similarity between these two vectors to find the web-based aboutness similarity

$$Sim_{about} = \frac{\mathbf{ab}(q_1) \cdot \mathbf{ab}(q_2)}{\|\mathbf{ab}(q_1)\|\|\mathbf{ab}(q_2)\|} \tag{10}$$

We found that Sim_{about} is quite useful since it can assign reliable relevance scores to related queries. For example, for the above example, 'python' and 'ruby' have a Sim_{about} score of 0.29, with the aboutness terms 'download' 'programming language' and 'implementation' in common. This first may sound lower than expected, but note that this is a quite difficult comparison due to other meanings of the queries (snake and gemstone) that occupy some portion of the aboutness vector. Hence, 'python' vs 'boa snake' or 'ruby' vs 'saphire' also have non-zero Sim_{about} as well.

Another good thing about Sim_{about} is that it has full coverage. It can be computed as long as the query returns some results. Although it cannot be computed for zero-result queries, we still consider it as full coverage, since query suggestions with zero results cannot be relevant to the user by definition.

4.3 Suggestions ranker

In addition to the above, we also employ features that capture the lexical characteristics of a suggestion such as binary features which include, whether the suggestion contains a digit, punctuation, alphanumeric characters, as well as the length of the suggestion. Other family of features include the source that generated the suggestion. For instance, the feature `SRC.CO` is set to 1 when a suggestion is generated using the co-occurrence statistics. Finally, we also employ dictionary-based features that check if query terms that were dropped, inserted, or left intact belonged to any specific category. In particular, we use dictionaries to determine if these terms are locations (e.g., city, state, or country), wikipedia entities (e.g., surgery, alzheimer), or stop words (e.g., the, of, an).

We use Gradient Boosting Decision Tree (GBDT) as the learner [12], and pose the problem as a classification of good and bad queries. In this classification setting, for each test sample GBDT outputs the probability of good. We prune the test samples with probability of good less than $p(good) < 0.5$ and rank the remaining by this probability.

4.4 Suggestion utility

The final step is to filter out the low utility suggestions. Of all the pieces in the pipeline, there is nothing that ensures that utility of the suggestions, and thus the system may generate irrelevant suggestions, e.g., suggestions with no search results. Also, in all likelihood, the system can generate query suggestions that are synonymous to the original query, with an almost identical result set. We assert that a query suggestion should be presented only if it leads to a sufficiently different result set as compared to those of the original query and other presented suggestions. For this purpose, we define a measure of utility of the suggestion q_s conditioned on q_p, a query that is already presented to the user, $U(q_s|q_p)$.

Given q_s and q_p, let $URL_s = [u_{s1}, \ldots, u_{sN}]$ and $URL_p = [u_{p1}, \ldots, u_{pN}]$ be the result sets of these two queries. For the top 10 URLs in the result page of q_s, we define the examination probability of the URLs using the rank discounts in the commonly used DCG formula [18].

$$p(e(u_{si}) = 1|q_s) = d(u_{si}, q_s) = \frac{1}{\log_2(r_i + 1)} \tag{11}$$

where r is the rank of the URL and e is a binary random variable that shows whether the URL is examined or not. Again for each $u_{si} \in URL_s$, we also define the examination probability that the user will examine the URL in the result page of q_p as follows,

$$p(e(u_{si}) = 1|q_p) = \begin{cases} 0 & : u_{si} \notin URL_p \\ 1 & : \begin{array}{c} u_{si} \in URL_p, \\ E\{d(q_p, u_{si})\} \geq E\{d(q_s, u_{si})\} \end{array} \\ \frac{E\{d(q_p, u_{si})\}}{E\{d(q_s, u_{si})\}} & : \begin{array}{c} u_{si} \subset URL_p, \\ E\{d(q_p, u_{si})\} < E\{d(q_s, u_{si})\} \end{array} \end{cases} \tag{12}$$

In words,

- u_{si} cannot be observed via q_p, if it is not in the result set of q_p, hence the examination probability is zero.
- If q_p returns u_{si} at least as high as q_s does, the examination probability is 1
- If q_p returns u_{si} lower than q_s does, the examination probability of this URL is the ratio of the rank discounts of the two corresponding ranks.

We define the pairwise conditional utility $U(q_s|q_p)$ by combining (11) and (12) as

$$U(q_s|q_p) = 1 - \sum_{u \in URL_s} p(e(u) = 1|q_s)\, p(e(u) = 1|q_p) \tag{13}$$

Intuitively, the first term in the summation gives how important this particular URL is for the query q_s, and the second term gives how likely it is that the same user would examine this URL in q_p, with the assumption that the user would go as deep into the result set in q_p. Hence, $U(q_s|q_p)$ is, by definition is 0 if the results of the two queries are exactly the same or q_s has zero results. Also, $U(q_s|q_p)$ would be close to 0 for queries that share many URLs and rank them similarly. After defining $U(q_s|q_p)$, we use the following greedy approach to ensure that all suggestions are sufficiently different from the original query as well as each other.

- Get the ranked suggestion list with decreasing scores.
- For the first one in the ranked suggestion list, and put it into the final suggestion set if it satisfies $U(q_s|q_p) \geq \gamma$ where q_p is the original query.
- For all remaining queries, get the one with the highest score and put it into the final suggestion set if it satisfies $U(q_s|q_p) \geq \gamma$ for the original query and all queries in the final set.

User study to select the optimal γ value for the utility model is presented in Section 6.5.

Putting it all together, our end-to-end framework systematically explores candidate for query suggestions while allowing suggestions that may or may not lexically overlap with the original query. To this end, we proposed a supervised ranking algorithm to eliminate irrelevant or low utility suggestions. It is noteworthy that our pipeline is amenable to be incorporated in a web search engine since it can be run in an offline manner to generate a dictionary of queries and their suggestions.

5. EXPERIMENTAL SETUP

Query Set: We collected a random sample of 100,000 fully anonymized queries sent to Yahoo! search engine in the month of August, 2010 along with their frequency. Of these, we identified those for which the search engine does not present any suggestion and drew a random sample of 10,000 queries biased by their frequency (see Introduction for query suggestion distribution.) . Our goal was to capture queries from different quantiles of this distribution. Note that by nature the queries that the search engine cannot provide any suggestions are very likely to be tail queries of the overall query distribution. We perform 10-fold cross-validation over this dataset.

Distributional similarity filters: For this, we use a collection of 500 million web pages crawled by a commercial search engine crawl. We construct our distributional similarity database by adopting the methodology proposed in [27]. We POS-tagged our web corpus using Brill's tagger [5] and chunked it using a variant of the Abney chunker [1]. We built a distributional database from this chunked corpus using the method outlined in Section 3.2.

Clickthrough data: To generate the co-click graph and compute the co-occurrence log-likelihood ratio, we used 6 month of anonymized query clickthrough logs of Yahoo! search engine. The co-click graph consists of 2.7M unique queries and 55.6M unique URLs in total, with 167.3M edges in between them. The co-occurrence data has 4.3M unique (q_c, q_n) pairs for 913K unique q_c.

Compared methods: We are unaware of any existing system for synthetically generating suggestions from several candidate generation methods and blends them. However, several methods for generating suggestions have been proposed, which we use as sources for candidate suggestion generation [20, 29]. Therefore, we build baseline techniques for comparison using each candidate generation source:

- `CO+PS`:co-occurrence with pushing the dropped terms
- `CO` :co-occurrence
- `DS` :distributional similarity of the dropped term
- `Q+CK` :coclicks of the original query
- `R+CK` :coclicks of the relaxed query
- `DS+CK`:coclicks of `DS`
- `CO+CK`:coclicks of co-occurrence of relaxed query
- `HYB` :hybrid method that combines all

Note that we allow all the above methods to share the same query relaxation techniques and the ranking model.

User studies: All user studies and manual annotation tasks described were performed by a group of eight professional search engine quality evaluators experienced with assessing the quality of query suggestions and search results.

Evaluation method: Professional annotators provided binary judgments for the 10,000 queries we sampled, good, for the relevant and useful suggestions, and bad for the irrelevant, garbled, zero-result, or synonymous (hence useless) suggestions. Annotators were asked to input their judgments after looking at the results page and comparing those for the query and suggestion; this is important to capture the utility of a suggestion.

Evaluation metrics: We evaluate the performance of each system using information retrieval measures, namely, precision and recall defined as:

- **Precision:** Given a list L of suggestions for a query, we compute precision as $\frac{Number\ of\ correct\ suggestions\ in\ L}{|L|}$. We study the precision values at varying ranks averaged over queries with at least one suggestion.

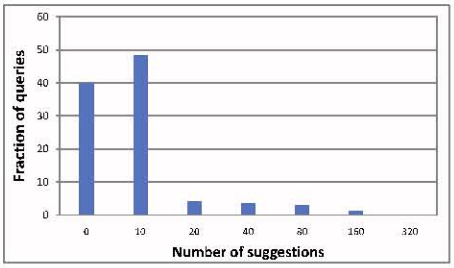

Figure 2: Distribution of number of suggestion candidates per query.

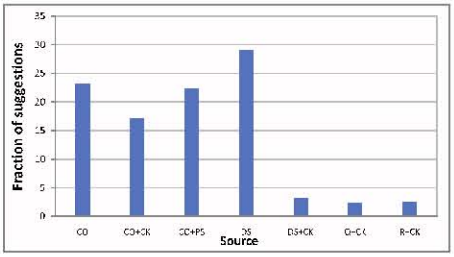

Figure 3: Distribution of number of suggestions for each candidate generation source.

- **Recall:** Given a list L of suggestions for a query, we compute recall as $\frac{|Number\ of\ correct\ suggestions\ in\ L|}{All\ correct\ suggestions\ for\ query|}$ where all correct suggestions for query is the union of the correct suggestions across all methods.

6. EXPERIMENTAL RESULTS

6.1 Coverage of extended suggestion pool

Our main goal is to increase the coverage of the queries for which useful suggestions can be provided to the users. Thus, our first experiment studies the increase in this coverage along various directions. Note that our experiments consisted only of queries for which a commercial search engine does not provide *any* suggestions and therefore results reported in this section naturally translate to an increase in the coverage of queries with suggestions. (Implementation details of this suggestion service are proprietary and out of scope of this paper.) Figure 2 shows the distribution of the total number of suggestion candidates explored by our candidate generation methods per query. As we can see, our proposed query suggestion generation methods provide a large set of candidates. We further break down the suggestions by the source in Figure 3, and see that all methods have some substantial contributions, and some of them like `CO`, `CO+CK`, `CO+PS` and `DS` are contributing much more than the others.

To investigate the quality of the suggestion candidate pool, we look at the number of good suggestions per query in Figure 4. Before the pruning and ranking stage, 24% of the queries have at least one good suggestion candidate. This is encouraging since the query set was sampled from those with no suggestions, which constitutes 66% of the unique query traffic. We again break down the good suggestion distribution into individual sources. Among the most contributing sources there is the `CO` method, investigated earlier by Jones et al. [20], and three other candidate sources that we proposed in this paper provide good suggestions almost as much as that one. Also, all of the sources have some substantial contribution.

6.2 Quality of extended suggestions

We now evaluate the quality of the ranked lists for each source. Each list is run through the well-formedness, relevance, and utility

query	suggestions
free auto parts manual	auto body parts, discount auto parts, auto part stores
minneapolis days inn	minneapolis comfort inn, minneapolis motel 6, minneapolis super 8, minneapolis best western
muscular system diagram	human skeletal system diagram, circulatory system diagram, human muscular system, skeletal anatomy
puppy training 101	dog training basics, puppy housetraining, dog whisperer, puppy training tips
csi crime scene investigation cast	csi miami cast, csi new york cast, csi episode guide, cbs csi miami
honda insight review	ford fusion review, 2010 toyota prius, toyota prius review, honda civic hybrid
obsessive compulsive disorder symptoms	ocd test, ocd symptoms, causes of ocd, ocd treatment, symptoms of ocd

Table 2: Examples for query suggestion pairs

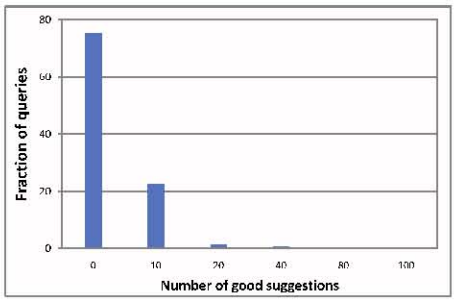

Figure 4: Number of good suggestions per query.

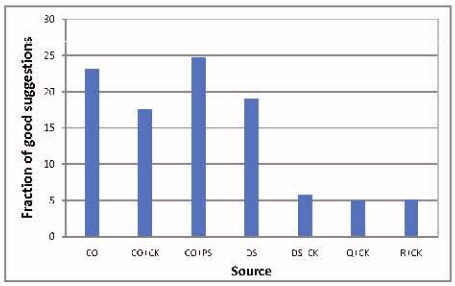

Figure 5: Distribution of good suggestions for each candidate generation source.

filters and for fairness, all sources share the same CRF relaxation, relevance thresholds.

Table 2 lists a few sample suggestions generated for our test queries. Figures 6 and 7 show the average precision and recall, respectively, of the ranked list produced by the GBDT model. We see that some of the sources like `Q+CK` are quite precise, but their recall as well as the depth of the ranked list is very low. `CO` and `CO+CK` are the ones with the highest recall, also they can provide up to 8-10 suggestions per query.

It is noteworthy that the precision for `HYB` is computed over more samples than those for the baseline since `HYB` produces the longest list of suggestions (see Figure 7). This, in turn, results in less variance of the precision values as compared to the baselines. In Figure 7, we observe that the recall for `HYB` is additive over the baselines, i.e., individual sources. The overlap between the suggestion candidates generated by each source is low with each source bringing in different and valuable candidates. Figure 8 gives the average f-measure over the suggestions set at each rank for all sources. As expected, `HYB` outperforms all individual sources including the earlier methods `CO` [20] and `Q+CK`, `R+CK` [29].

`HYB` in Figure 8 generates a relatively long list with 10 or more suggestions; in practice typically $k = 5$). If the desired size k is known a priori, one can tune the ranking stage to optimize for the precision and recall up to that k value. For example, if a deep list is not needed, the aim might be to *display a single suggestion per query, and increase the quality as much as possible.* This is perfectly reasonable. The exact number of suggestions to be displayed is a design choice: depending on where on the page they will be displayed, too many suggestions might distract the user.

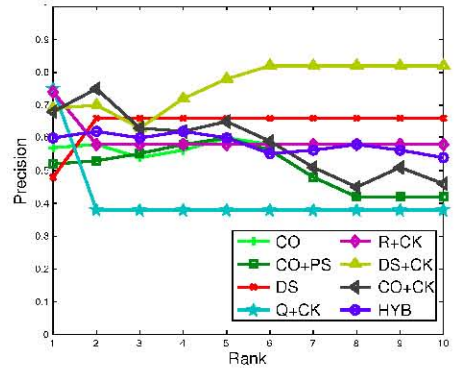

Figure 6: Mean precision at varying ranks.

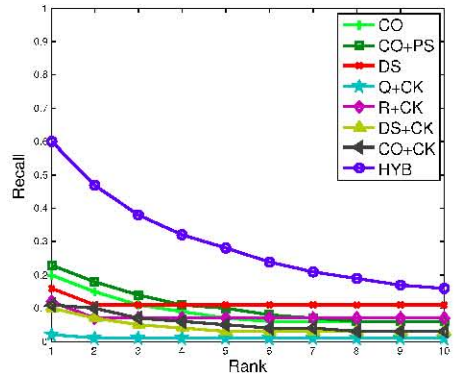

Figure 7: Average recall at varying ranks.

For this scenario, to optimize the f-measure at the top position, one can tune the GBDT confidence threshold, and compromise for the precision and recall at the lower ranks to improve the precision and recall at rank 1. See Figure 9 for a comparison of the f-measure for top 5 ranks for varying relevance confidence threshold levels. For example with the threshold 0.8, one can increase the f-measure at rank 1 very significantly to 0.82, by compromising the depth of the list; hence, the recall (and therefore the f-measure) decays much more sharply for increasing ranks as compared to the values with the threshold 0.5 -the black curve in Figure 8.

In summary, if a deep list is not needed, with a more strict relevance confidence threshold `HYB` can provide a ranking with 0.83 recall at rank 1, 0.56 recall at rank 2, 0.83 precision at rank 1, 0.85 precision at rank 2 and an average list length of 1.4. Precision value at rank two is 0.85, which is at the same level of quality of the suggestions that the search engine can provide for frequent queries via existing techniques -some of which we use as sources. Recall the distribution over the suggestion coverage provided in the Introduction section; 26.6% of unique queries have more than 5 suggestions, and 7.4% have between 1 and 4 suggestions, and 66% of the unique query distribution has no suggestions. It is remarkable that `HYB` with relevance threshold level 0.8 can increase the coverage of queries with at least one good suggestion from 34% to 47.2% of the unique query distribution, hence brings a coverage increase of 39% *without compromising precision.*

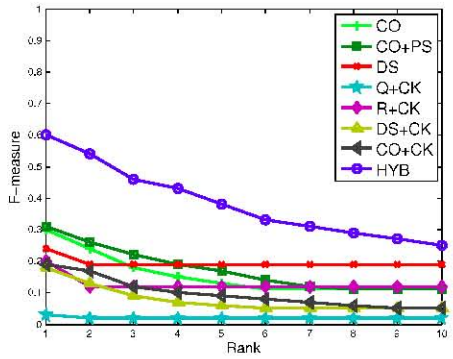

Figure 8: Average f-measure at varying ranks

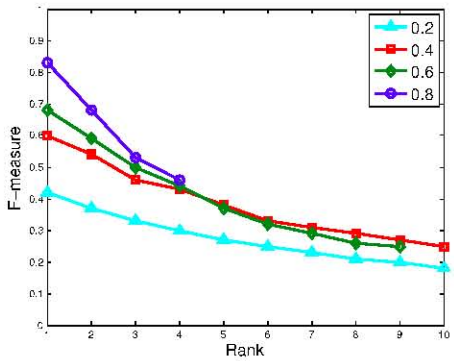

Figure 9: Average f-measure for varying relevance confidence threshold.

6.3 Query relaxation performance

After comparing each source, we now evaluate individual components in the pipeline starting with the CRF model from Section 2. Our evaluation of the proposed CRF model is two-fold. For our first evaluation, we ran 10-fold cross-validation and compared our model against a baseline method that eliminates the last term in the query. Table 6.3 compares these methods and shows that using a CRF model substantially improves the precision (15% gain) with a small amount of drop in recall (2%). Our second evaluation

method	precision	recall
CRF	0.82	0.87
Baseline	0.71	0.89

Table 3: Performance of query relaxation method.

involved a user study where annotators were provided the queries for which the CRF had dropped atleast one term. Annotators were asked to label (a) whether a query contained a non-critical term, (b) whether the CRF selected the correct term to drop. Of these queries 70% of queries did have a non-critical term, and for these queries CRF selects the correct term 81% of the time. Finally, a feature analysis of the model showed that the stand-alone ratio is consistently the most important feature in all 10 runs.

6.4 Feature analysis of suggestion ranker

To better understand the importance of various features used in our ranking model (Section 4.3), we examined for each feature the coverage as well as the importance ranking noted by the GBDT model. An encouraging result is that the GBDT feature importance ranking was consistent and thus stable across individual runs in our 10-fold cross-validation. Specifically, in all 10 runs Sim_{about} is the most important feature. Also, web and combined language model scores $P_{\mathcal{W}}$ and $P_{\mathcal{C}}$, and Sim_{click} are consistently among the top five most important features. Not surprisingly, the most important features are the ones with higher coverage, Sim_{about} with full coverage, combined (71%) and web (66.9%) language model scores. The one with the smallest coverage among the most important features is Sim_{click} with 17.5%. Also, lexical features seem more important than the source features overall.

6.5 Utility model performance

Finally, we discuss the performance of our utility model (Section 4.4). To select an optimal γ value, the threshold by which we decide to remove the query suggestion from the set due to redundancy, we carried out a user study where annotators were given 500 randomly selected pairs of queries and their suggestions. Annotators were asked if the suggestion is useful or redundant for the given query. Figure 10 shows $U(q_s|q_p)$ value versus the editor grade (a noise jitter is added into the editor grade for a better visualization), along with the precision-recall curve of $U(q_s|q_p)$ for varying γ values. For this model, we decided to use $\gamma = 0.70$, which corresponds to 0.94 precision and 0.67 recall on this dataset, as marked in Figure 10. In our experiments, the utility model removes low utility suggestions such as "cooking channel tv" → "cooking channel television", "tampa bay fisheries" → "tampa bay fishery".

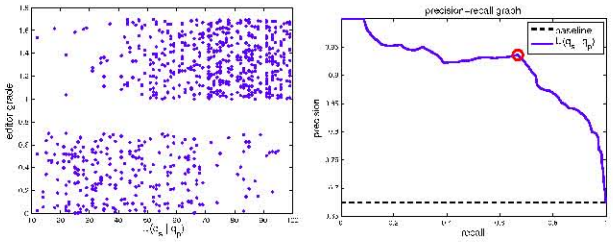

Figure 10: Utility estimate $U(q_s|q_p)$ versus the editor judgment (left), and precision recall curve of $U(q_s|q_p)$ with the operating point at $\gamma = 0.70$ and the precision recall curve of the baseline (majority) classifier.

7. RELATED WORK

Providing query suggestions: Research on query suggestion methods have largely relied on observed queries and developed similarity measures based on these observations. In [11] association rules are used to mine query recommendations from queries in individual users' search sessions which are defined as fixed length periods of interaction. In contrast, [36] builds a graph representation of the of the sequential search queries and combines it with a content based similarity method to account for the sparsity of the query logs. These methods simply mine query reformulations from observed queries. Other methods leverage the click information, as well. In [29], a query similarity measure is developed based on term-weight vector space representations of the queries and clicked URLs. Query term and click pattern overlap is used in [35] to develop query similarity measures in order to cluster user queries. Random walks on query-click graph is used by [8]. Slightly differently, Markov Random Field models are used over the query-click graph to generate bid terms in [13]. In [25], query-click graph based random walk techniques are further refined by considering the time to first visit to a query node. Main shortcoming of the query recommendation methods listed above is the limitation of suggestions to observed queries. This works fine for the frequent queries but not so well for the rare queries in the tail.

Earlier methods of generating alternative queries also included query expansion by pseudo-relevance feedback where additional query terms are obtained from the documents retrieved by the original query. These terms are used to expand the original query to retrieve more specific documents. Limitations of this method such as query drift are well documented [30]. Query term deletion has been tried in [19] but it leads to loss of specificity. In [32], query

term substitution is considered using the retrieved documents, and in [2] user input is sought for selecting appropriate related terms, but these methods add an additional step to query reformulation. Automatic term substitution using query logs is considered in [20], however a non-contextual query segmentation is used leading to many marginally useful reformulation candidates. Their filtering method also lacks a language model, so it does not have the right information to separate well-formed queries from illegible ones. This is probably acceptable for bid term generation for ad matching but not so for query suggestions to be offered to a search user.

Earlier work has explored comparing information from query logs and other textual sources [15, 9]. Closest to our work is the method in [9] which examines anchor text in web pages solely for the task of query suggestions. Interestingly, Dang and Croft show that suggestions generated using anchor text are comparable to those generated using query logs. This work builds an alternative to query logs using web pages whereas our work uses web and additional textual information to *extend* query session logs, especially for rare queries. Further, our proposed method explicitly handle the case of lateral query reformulations. Recently, Szpektor et al. [17] showed how templates can be used to provide suggestions for rare queries; our work considers the generic problem of providing suggestions for *any* rare query.

Re-writing long queries To improve the performance of search engines on long queries (or tail queries), several prior research efforts have looked into query re-writing via term substitution or term reduction [3, 9, 16, 22, 26, 33, 34]. While these problems are well-studied and have shown significant improvements in performance on TREC data, their utility on web environment is not well understood. Long query re-writing is conceptually related to our candidate generation step; however, a critical difference is that we focus on generating query reformulations to be offered *as an ordered set* to the user instead of improving search results. In our setting, we need to consider non-lexically overlapping yet semantically related query reformulations as well as consider the quality and utility of a 'user-facing' suggestion set as a whole.

Identifying key concepts in queries: Finally, identifying key concepts in the context of search queries has also been extensively studied over TREC data and over search engine logs to some extent [3]. Conceptually, this is related to our query relaxation step, however our end-to-end approach considers the web search setting and uses complements such methods with algorithms to generate and rank candidate suggestions.

8. CONCLUSIONS

In this paper, we presented a supervised learning-based end-to-end framework to synthesize query suggestions offered to search engine users as follow-up queries. Our technique uses query logs as a guide to capture user search and intent behavior and leverages information available in web pages, search results, as well as click logs to derive suggestions. We systematically decompose a query and apply various transformations to automatically generate suggestion candidates and propose a ranking method that incorporates language models and other useful features to select high quality suggestions. We show that our system increases the coverage of unique queries with at least one good suggestion from 34% to 47.2%, leading to a 39% increase by high quality suggestions.

9. REFERENCES

[1] S. Abney and S. P. Abney. Parsing by chunks. In *Principle-Based Parsing*, 1991.

[2] P. Anick. Using terminological feedback for web search refinement: a log-based study. In *SIGIR '03*, 2003.

[3] M. Bendersky and B. Croft. Discovering key concepts in verbose queries. In *SIGIR '08*, 2008.

[4] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna. The query-flow graph: model and applications. In *CIKM '08*, 2008.

[5] E. Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4), 1995.

[6] P. D. Bruza and T. W. Huibers. A study of aboutness in information retrieval. In *AI '96*, 1996.

[7] K. W. Church and P. Hanks. Word association norms, mutual information, and lexicography. In *ACL '89*, 1989.

[8] N. Craswell and M. Szummer. Random walks on the click graph. In *SIGIR '07*, 2007.

[9] V. Dang and B. Croft. Query reformulation using anchor text. In *WSDM '10*, 2010.

[10] L. Fitzpatrick and M. Dent. Automatic feedback using past queries: social searching? In *SIGIR '97*, 1997.

[11] 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 '03*, 2003.

[12] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29, 2000.

[13] A. Fuxman, P. Tsaparas, K. Achan, and R. Agrawal. Using the wisdom of the crowds for keyword generation. In *WWW '08*, 2008.

[14] Z. Harris. Distributional structure. *Word*, 10(23), 1954.

[15] J. Huang, J. Gao, J. Miao, X. Li, K. Wang, and F. Behr. Exploring web scale language models for search query processing. In *WWW '10*, 2010.

[16] S. Huston and B. Croft. Evaluating verbose query processing techniques. In *SIGIR '10*, 2010.

[17] Y. M. Idan Szpektor, Aristides Gionis. Improving recommendation for long-tail queries via templates. In *WWW '11*, 2011.

[18] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20, 2002.

[19] R. Jones and D. C. Fain. Query word deletion prediction. In *SIGIR '03*, 2003.

[20] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *WWW '06*, 2006.

[21] R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. *ICASSP 1995*, 1, 1995.

[22] G. Kumaran and V. R. Carvalho. Reducing long queries using query quality predictors. In *SIGIR '09*, 2009.

[23] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML '01*, 2001.

[24] D. Lin. Automatic retrieval and clustering of similar words. In *COLING '98*, 1998.

[25] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *CIKM '08*, 2008.

[26] G. K. Niranjan Balasubramanian and V. R. Carvalho. Exploring reductions for long web queries. In *SIGIR '10*, 2010.

[27] P. Pantel, E. Crestan, A. Borkovsky, A.-M. Popescu, and V. Vyas. Web-scale distributional similarity and entity set expansion. In *EMNLP '09*, 2009.

[28] V. V. Raghavan and H. Sever. On the reuse of past optimal queries. In *SIGIR '95*, 1995.

[29] C. H. Ricardo Baeza-Yates and M. Mendoza. Query recommendation using query logs in search engines. In *Trends in Database Technology - EDBT 2004 Workshops*, 2005.

[30] I. Ruthven. Re-examining the potential effectiveness of interactive query expansion. In *SIGIR '03*, 2003.

[31] G. Salton. On the use of term associations in automatic information retrieval. In *COLING*, 1986.

[32] E. Terra and C. L. Clarke. Scoring missing terms in information retrieval tasks. In *CIKM '04*, 2004.

[33] M. B. Van Dang and W. B. Croft. Learning to rank query reformulations. In *SIGIR '10*, 2010.

[34] X. Wang and C. Zhai. Mining term association patterns from search logs for effective query reformulation. In *CIKM '08*, 2008.

[35] J.-R. Wen, J.-Y. Nie, and H.-J. Zhang. Clustering user queries of a search engine. In *WWW '01*, 2001.

[36] Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. In *WWW '06*, 2006.