Query Suggestion by Constructing Term-Transition Graphs

Yang Song, Dengyong Zhou, Li-wei He

Microsoft Research,
One Microsoft Way,
Redmond, WA 98052, USA
{yangsong, denzho, lhe}@microsoft.com

ABSTRACT

Query suggestion is an interactive approach for search engines to better understand users information need. In this paper, we propose a novel query suggestion framework which leverages user re-query feedbacks from search engine logs. Specifically, we mined user query reformulation activities where the user only modifies part of the query by (1) adding terms after the query, (2) deleting terms within the query, or (3) modifying terms to new terms. We build a term-transition graph based on the mined data. Two models are proposed which address topic-level and term-level query suggestions, respectively. In the first topic-based unsupervised Pagerank model, we perform random walk on each of the topic-based term-transition graph and calculate the Pagerank for each term within a topic. Given a new query, we suggest relevant queries based on its topic distribution and term-transition probability within each topic. Our second model resembles the supervised learning-to-rank (L-TR) framework, in which term modifications are treated as documents so that each query reformulation is treated as a training instance. A rich set of features are constructed for each (query, document) pair from Pagerank, Wikipedia, N-gram, ODP and so on. This supervised model is capable of suggesting new queries on a term level which addresses the limitation of previous methods. Experiments are conducted on a large data set from a commercial search engine. By comparing the with state-of-the-art query suggestion methods [4, 2], our proposals exhibit significant performance increase for all categories of queries.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - Search process

General Terms

Algorithms, Experimentation

WSDM'12, February 8–12, 2012, Seattle, Washington, USA.

Keywords

query suggestion, page rank, learning to rank

1. INTRODUCTION

While more and more sophisticated algorithms have been applied to web search engines to increase user search experiences, it is still admittedly difficult for search engines to fully understand user's search intent in many scenarios. Consequently, many researches have been focused on interaction with users to find out what the users really want [12, 13]. Among all, query suggestion [5, 23] has established as one of the most promising interactive techniques as we have witnessed its wide usage in most commercial web search engines. Generally speaking, query suggestion techniques aim at recommending a list of relevant queries to user's input, by mining correlated queries from previous knowledge, e.g., search engine logs. Indeed, the quality of query suggestion can be affected by many factors. For example, depending on the underlying algorithms, query suggestion techniques can be categorized into graph-based models (e.g., random walk on click graphs)[16, 7] and probabilistic models (e.g., language models)[24, 11]. Moreover, depending on the length of suggested queries, we can further divide existing query suggestion techniques into query expansion [17, 6], query reduction [12] and query reformulation [24, 8].

Among all existing techniques, random walk-based models [16, 7, 23] have exhibited their superiority over others on both efficiency and effectiveness. Due to its simplicity and mathematically-proven fast convergence, random walk-based query suggestion models are capable of scaling up to large data sets for commercial search engines. Essentially, most models in this category construct a bipartite graph of queries and clicked URLs by assuming the goodness of clicked URLs. Queries are treated relevant to each other if they share common clicks. While both theoretically sound and empirically usable, this approach appears to have a drawback that needs attention. To be specific, random walk models only focus on query and clicks but ignore the rich information which is embedded in the entire user session. For example, considering four user activities in a session: $\{q_1, URL_1, q_2, URL_2\}$. The user clicked URL_1 after issuing q_1. Not satisfied by the result, the user immediately reformulates the query to q_2 and clicked URL_2. Apparently, URL_1 is not as a good click as URL_2, which is unfortunately treated equally in random walk models.

In this paper, we try to address two fundamental questions in query suggestion research. (1) How to extract high-quality user clicks in the user session and use the clicks prop-

erly? Since the quality of the clicks directly affects the recommendation performance of models which leverage clicks as signals, it is critical to accurately infer user intent by examining the entire user search session. (2) Given a specific query, which query suggestion method should be used? e.g., can query reduction method be applied to short queries? Is query expansion always better than query reformulation? In our opinion, answers to these questions are non-trivial. And in practice, it usually depends on each individual queries as well as the user intent at that moment.

To briefly answer these questions, we first derive high-quality user clicks by extracting specific user activity patterns which conveys implicit user preferences. Secondly, instead of using the clicks directly, we construct a term-transition graph model from the data. We use random walk as a guideline to find user preferences within each topic and choose the best suggestion method according to our model.

To be concrete, we propose a framework that combines the strength of graph-based models and probabilistic models. Our model naturally unifies the advantages of query expansion, reduction and reformulation techniques while successfully avoids their relative shortcomings. More specifically, we make the following contributions in this paper:

- From the search engine session logs, we mine a large amount of user preference data. We consider the following tuple $\{q_1, q_2, u\}$ where a user abandoned a query q_1 and immediately reformulated it into q_2 then made a click on URL u, during the same session. These activities strongly indicate a user's preference on query q_2 over q_1, which often differs by only a few terms.

- We construct a term-preference graph from the above data where each node is a term in the query and each directed edge a preference. We then train a topic-biased Pagerank model for each of the query topics by extracting topics from clicked URLs. Given a query, this model guides the decision of (1) expanding relevant terms to the original query, (2) removing terms from the original query, or (3) replacing existing terms with relevant terms. This unsupervised model learns the user preference from the log which does not require any specific user labels.

- We further propose a supervised learning-to-rank model to interpret user preference at term level. This model leverages a set of rich features, which includes N-gram features, external knowledge (e.g., Wikipedia, ODP) as well as the topic-biased Pagerank features learnt in the previous model. No human labeling was conducted. Instead, we automatically infer the training labels from the logs by leveraging a sophisticated probabilistic preference model.

To summarize, our system provides a principled formulation that is capable of addressing both topic and term-level suggestions. Both of our methods exhibit much better recommendation performance than the state-of-the-art methods [16, 7] on a real-world large data set mined from a commercial search engine.

2. RELATED WORK

The complexity of dealing with query suggestion related problems has led to a myriad of proposals of methods in literature. In general, this problem has been addressed by different techniques including query reduction, query expansion, query reformulation and query suggestion using search engine logs [22, 12, 17, 4, 19].

In [12], the authors proposed to reduce long queries into shorter ones by removing redundant terms and extract key phrases. Three methods for selecting sub-queries were introduced: mutual information, maximum spanning tree and extracting named entities. Results indicated that with user interaction, the precision of retrieval improved. However, this method is only useful for long queries. When applying to short queries, it only exhibits marginal improvement.

Automatic query expansion was first proposed in [17], where the authors leveraged term correlations in documents as a metric to recommend similar terms. Similarly, authors in [6] measured the difference between the distribution of terms in relevant documents and in collections to score similar terms. In general, query expansion techniques were able to introduce useful terms while also bring in unhelpful terms.

Differing from the above two approaches, query reformulation replaces the existing terms in the query by more relevant ones. Essentially, it tries to map the query language model into document language model with term replacement. In [24], the authors used context distribution to extract term associations from the logs. Given a new query, their method replaced the terms with new terms of higher matching probability in the context. Similarly, using anchor text as the context for reformulation was recently studied [8].

The three approaches mentioned above primarily focused on improving retrieval relevance. On the other hand, query suggestion aims at interaction with users to improve search experience by showing relevant queries on the search engine result pages (SERP) and allowing users to decide whether to click or not. Among all related techniques for query suggestion, log-based random walk methods have shown good results [7, 16, 9]. Most random walk models are performed on query-document bipartite graphs. For example, in [7], the authors introduced a backward random walk model which alleviates the bias towards documents with more clicks. The model normalizes the correlation matrix on the document clicks rather than query counts, which is why the transition from a document to query is not biased to popular queries. In [16], a parameter-free random walk model is proposed which addressed the efficiency of random walk. The model constructed a subset of query-url nodes in the bipartite graph and performed depth-first search for relevant queries. By estimating the transition probabilities between queries using similarity-based method, this model was capable of suggesting semantically related queries. In [4], the authors proposed a query-flow graph that calculates the probability of going from one query to another, by leveraging a set of features. This approach can be treated a special case of random walk on the query-URL graph where the URLs are removed but used to calculate the transition probability. Although looks similar to our method, it only considered query-level transition while our model deals with much finer granularity by addressing term level transitions.

Recently, the authors in [23] introduced an optimal way of query suggestion using random walk. The authors addressed two important factors in the paper. First, how to tune the jumping parameter in the random walk model and second, how to optimally combine the clicked and skipped URLs in the log for relevance estimation. The authors proposed to

perform two random walks on click graph and skip graph respectively, and optimally combine the results by minimizing the categorization error on the URL correlations, which was estimated using ODP data. Experimental results indicated a significant improvement over previous random walk models.

3. GET USER PREFERENCE FROM LOG

In this section we describe how we acquire a large collection of user preference data from the logs. The primary source of data used in this paper is from logs of a widely-distributed browser toolbar, where a large sample of users opted-in for providing their online browsing activities. The logs are anonymized so that no user personal information is recorded. A typical log entry contains the following fields: a unique user identification number generated by a one-way hash function, the URL that the user visited, the timestamp of the visit, and the total dwell time on that URL. Since we are interested in the user queries when visiting search engines, we also post-processed the data to generate the user queries and the name of the search engine. The logs are organized based on sessions, where each session contains a series of URL visits from a particular user, ordered by the timestamp. If the user becomes inactive for more than 30 minutes, that session ends.

We are particularly interested in spontaneous query refinement behaviors by the users, which can be summarized in a sequence of three activities:

1. A user enters a query q_1 to a search engine.
2. After examining the search result page, the user did not get the right information, so he/she refines the query to be q_2.
3. This time the user is satisfied with the results and clicked one of the URLs, therefore ends the search session.

For example, the user first issued "nyc traffic cameras" and refined the query by adding one term at the end "nyc traffic cameras live", followed by a click on the satisfied result. This sequence of activities conveys two important pieces of information: first, the user prefers the query q_2 to q_1 since he performed no click after issuing q_1 but then becomes positive about the results after q_2. Second, the user chooses to refine the query rather than clicking the suggested queries by the search engine, which may indicate poor suggestion quality for that particular query.

Although the users may refine the queries in whatever way they want, our statistics have shown that more than 76% of times users only modify one of the terms in the queries. It has also shown the users are much more likely to revise the last term in the query, which covers more than 80% of all cases[1]. Consequently, we only focus on mining user preferences that fall into these two categories in this paper, although the proposed methods can be easily generalized to handle more complex scenarios.

Formally, given a word vocabulary $W = \{\epsilon, w_1, ...w_n\}$ where ϵ is used to denote the empty string, we define three cases of user query refinements as follows:

[1]These statistics are drawn from one month search logs from a commercial search engine with over 100 million queries

Type	User Activity	Pattern
Modification	1. q:{single ladies song} 2. q:{single ladies lyrics} 3. URL click	song→lyrics
Expansion	1. q:{sports illustrated} 2. q:{sports illustrated 2010} 3. URL click	ϵ→2010
Deletion	1. q:{ebay auction} 2. q:{ebay} 3. URL click	auction → ϵ

Table 1: Three types of user refinement examples.

- Modification: user modifies the last term of the query: $\{w_{i1}, ...w_{im}\} \rightarrow \{w_{i1}, ...w'_{im}\}$, e.g., "single ladies song" → "single ladies lyrics".
- Expansion: user adds one term to the end of the query: $\{w_{i1}, ...w_{im}\} \rightarrow \{w_{i1}, ...w_{im}, w_{i(m+1)}\}$, e.g., "sports illustrated" → "sports illustrated 2010".
- Deletion: user removes the last term of the query: $\{w_{i1}, ...w_{i(m-1)}, w_{im}\} \rightarrow \{w_{i1}, ...w_{i(m-1)}\}$, e.g., "ebay auction" → "ebay".

Table 1 summarizes the three types of refinements used in this paper. Overall, we extracted the aforementioned information from 3-month Toolbar logs between May 2010 and August 2010. In total, over 4 million user refinement activities were discovered in the log. By aggregating the refinements, a total of 350,000 pairs of refinements were found. Table 2 summarizes the top user refinements in each case.

Next, we show how these implicit user feedbacks can be leveraged for query suggestion in our models.

4. MODEL 1: AN UNSUPERVISED APPROACH

Given the pair-wise transition between each pair of terms, we represent the transitions between all terms into a directed graph. Since every transition from t_i to t_j indicates a preference of t_j over t_i, we assume this is an analogy of webpage inlinks and outlinks, where in general the more inlinks a webpage has, the higher its authority score is. The most famous algorithm for calculating such kind of score is the Pagerank algorithm [18], which is widely used in commercial search engines for webpage static rank estimation. In

Modification	Expansion	Deletion
login→home	ϵ→free	the→ϵ
2009→2010	ϵ→2010	search→ϵ
facebook→myspace	ϵ→online	mail→ϵ
lyrics→video	ϵ→download	2→ϵ
google→yahoo	ϵ→lyrics	facebook→ϵ
music→songs	ϵ→video	scam→ϵ
earth→maps	ϵ→login	best→ϵ
check→access	ϵ→pictures	cheats→ϵ
jobs→employment	ϵ→games	cheap→ϵ
depot→max	ϵ→reviews	wiki→ϵ

Table 2: Summary of top user query refined queries in each category.

our scenario, we assume that terms exhibit different level of importance in a set of pre-defined topics. In what follows, we describe a topic-based Pagerank approach which is inspired by [10].

The Pagerank is a random walk-like algorithm where higher Pagerank is assigned to pages with greater importance. In the Pagerank algorithm, the rank of each page can be estimated iteratively. Specifically, at the $t+1$ step, the Pagerank is updated as follows:

$$Rank^{(t+1)} = (1-\alpha)M \times Rank^{(t)} + \alpha p, \quad (1)$$

where M is a stochastic matrix where $M_{ij} = 1/N(j)$ if there is a transition from node j to node i, with $N(j)$ being the total number of outlinks of node j. At each step, the random walk has a probability of $\alpha \in (0,1)$ that jumps to an arbitrary node. In the original Pagerank algorithm, $p = [\frac{1}{N}]N \times 1$ so that the probability of jumping to any node is the same.

In our topic-based Pagerank model, for each topic t_j, the jumping vector p is calculated as follows:

$$p_{ji} = \begin{cases} \frac{1}{|U_j|} & \text{if } i \in U_j \\ 0 & \text{otherwise} \end{cases}$$

where U_j is the set of terms belonging to topic j. Given t topics, this approach essentially constructs t topic graphs where each graph G_j only contains nodes which is categorized in topic j.

Figure 1 illustrates an example of such graph that contains 9 nodes (our actual graph is much bigger). After Pagerank algorithm converges, each term w is assigned with a vector of scores denoting its Pagerank score in all t topics.

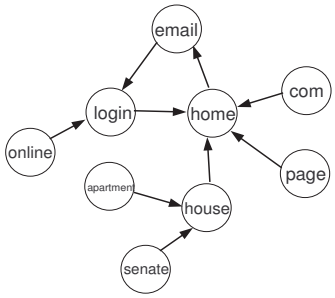

Figure 1: An illustration of graph construction from the term transition data. Each arrow indicates a user preference, e.g., page → home.

4.1 Extracting Topics from Clicks

Recall that in our definition, a query refinement essentially contains three elements: $\{q_1, q_2, click_{url}\}$. Since getting the topics directly from query is difficult, we propose to infer the topics from the clicked webpage which naturally reflects the query intent. In this paper, we leverage a sophisticated content-based classifier that categorizes webpages into ODP categories [3]. The ODP categories are organized hierarchically where higher level topics are more abstract. For example, www.microsoft.com belongs to "Computers"→"Companies" category. We leverage the 14 top-level categories from ODP and construct 14 topic graphs in our models. Note that it is common for a transition to appear in more than one category according to the URLs clicked. For example, coupon→free happens very often in both Computers and Shopping categories. Table 3 shows the top-ranked terms in several categories after running the topic-based Pagerank model.

4.2 Query Suggestion

After calculating topic-based Pagerank, the next step is to suggest query-dependent terms when a new query comes. Given a query q_i with m terms $\{w_{i1}, ..., w_{ij}, ..., w_{im}\}$, we want to suggest a set of related queries by reformulating the query using modification, expansion or deletion methods. Although for training we only consider the modification of the last term in the query, it should be noted that at query suggestion time, our model can be adapted to make modifications for any terms in the query.

Overall, any term can be either modified or deleted from the query. The query itself in the meanwhile can be expanded where the expanded term is considered to be appended at the end. Specifically, for any term w_{ij}, the term refinement probability is defined as

$$P(w_{ij} \to w'_{ij}|q_i) = \sum_k P(w_{ij} \to w'_{ij}|T_k)P(T_k|q_i),$$
$$w'_{ij} \in W = \{\epsilon, w_1, ...w_n\}, j \in 1, ..., m. \quad (2)$$

Here $P(w_{ij} \to w'_{ij}|T_k)$ is a query-dependent score, i.e., the most likely transition in topic T_k given w_{ij}, which is estimated by controlling the jumping factor in Pagerank:

$$p_{ji} = \begin{cases} 1 & \text{if } i \in U_j \\ 0 & \text{otherwise} \end{cases}$$

Meanwhile, $P(T_k|q_i)$ denotes the topic distribution of the new query. For simplicity, we leverage the pseudo-relevance feedback from the search engine [21, 25] which assumes that the top-returned results are usually relevant. We thus classify the top-10 URLs returned by a search engine into ODP categories and treat it as the topic distribution of q_i. Table 5 shows examples of the most possible transitions.

To choose the best candidates among all refinements for all terms in the query, we leverage the Pagerank of the terms to rank the refinements:

$$P(q'_i|q_i) = \arg\max P(w_{ij})P(w_{ij} \to w'_{ij}|q_i), \quad (3)$$

where q'_i equals to q_i except for term j being replaced: $w_{ij} \to w'_{ij}$.

In addition, query q_i has the following probability to be expanded by adding a new term w', according to our model,

$$P(w'|q_i) = \sum_k P(w'|T_k)P(T_k|q_i), \quad (4)$$

where $P(w'|T_k)$ is the static Pagerank of term w' in topic T_j.

The results from term refinement and query expansion are then combined to suggest new queries based on the descending order of their probabilities. Table 4 shows a concrete example for the query "stanford university map". The final winners are "stanford university location", "stanford university address", "stanford university", "harvard university map" and "yale university map". In this case, suggestions from using expansion gets a very low score hence are not suggested, indicate that our model is capable of suggesting queries that are different from a syntactic point of view.

Computers	Health	Home	Science	Shopping	Sports	Business
download	symptoms	recipes	channel	walmart	sports	jobs
free	hospital	recipe	zoo	parts	espn	bank
verizon	treatment	coupons	salary	furniture	games	ashley
wireless	pain	cooking	coupons	store	results	dmv
software	diet	food	weather	buy	schedule	locations
iphone	health	baby	ohio	accessories	news	free
email	calculator	network	calculator	shoes	soccer	calculator
microsoft	surgery	easy	craigslist	supplies	cup	continental
windows	doctor	reviews	toyota	target	rumors	america
support	dosage	grilled	jetblue	clothing	nba	chase

Table 3: Top-ranked terms in each category by the topic-biased Pagerank method. A total of 14 ODP categories are assigned to queries.

Query		Prob(expansion)	
stanford university map		stanford university map online (0.0005) stanford university map download (0.0003) stanford university map printable (0.0001)	
Term	**Pagerank Score**	**Prob(replacement)**	**Prob(deletion)**
stanford	0.04	harvard (0.03), yale (0.02), columbia (0.02)	0.005
university	0.04	college (0.02), school (0.01), institute (0.002)	0.01
map	0.05	location (0.04), address (0.03), history (0.005)	0.015
Final Suggestions		**Source**	**Score**
stanford university location		replacement	0.04
stanford university address		replacement	0.03
stanford university		deletion	0.015
harvard university map		replacement	0.03
~~stanford university map online~~		expansion	0.0005
~~stanford university map download~~		expansion	0.0003

Table 4: Examples of query suggestion for "stanford university map". Top: query level expansion probability. Bottom: term level replacement and deletion probabilities for each term in the query. The final suggestion is the highest-scored queries from both models. Candidates from expansion are not suggested in this case.

Walmart	Microsoft	Steelers	Weather
target	msn	penguins	hurricane
sears	office	pirates	radar
costco	hp	panthers	zoo
wal-mart	microsoft.com	hurricanes	news
walgreens	live	ravens	earthquake

Table 5: Some examples of the most relevant terms given specific terms. Calculated using term-specific Pagerank algorithm.

5. MODEL 2: A SUPERVISED APPROACH

The Pagerank-based query suggestion model presents a novel way to efficiently reformulate queries on the topic level. Nevertheless, the model ignores the underlying relationship between the query and the suggested term after it aggregates the term transitions into topics. Indeed, within the same topic the term transition probability still varies from queries. In this section, we propose a query-level learning-to-rank approach which leverages a set of language-modeling features to address the limitation of the previous model.

We briefly introduce the idea of learning-to-rank here. For a better understanding of learning-to-rank, we refer readers to [15]. Given a set of queries and URLs as well as labeled relevance score (usually in 5-scale) for each query-URL pair, learning-to-rank tries to optimize for the ranking loss for all queries in the training set. Generally, three sets of features are often used during training, (1) query features (e.g., query frequency), (2) URL features (e.g., the Pagerank score of the URL), and (3) query-URL features (e.g., BM25 [20]). In our model, we mimic these principles and construct three sets of features for each triplet of transition: $\{q_1, q_2, click\}$. To be specific, the common part of q_1 and q_2 are extracted to represent the query part, whereas the refinement is treated similar as the URL role. For example, for the first example shown in Table 1, "single ladies" is the query, while "song" and "lyrics" are treated as refinements.

5.1 Feature Construction

For each of the query-refinement pair, we construct three sets of features as shown in Table 7.

The query features generally fall into two categories. The first category is related to Wikipedia. We dumped over 8 million Wikipedia articles in English, which contains the Title, Body content and Category information. In total, the corpus contains 7,181,315 titles and 10,170,187 articles. We match the query to the Title, Body and Category to estimate its popularity in the Wikipedia corpus. Secondly, we con-

struct N-gram language modeling features from a publicly available N-gram services built from Billions of documents [1]. The N-gram contains four streams: title, body, anchor and query streams, available in unigram, bigram, trigram, N-gram with $N = 4, 5$. For each query, each stream returns an N-gram probability for the query within that stream.

The term features are quite similar to URL/document features in information retrieval systems. The Pagerank scores of the terms are directly calculated from previous model with different jumping rates from 0.55 to 0.85. The entropy of the Pagerank score is calculated as $-\sum_i P(t_i)\log P(t_i)$ for term t_i, which estimates the term's specificity across all topics. In addition, a good indicator of term importance is the number of times users expand the original query by adding the term, which is specified as "# of times derived from _EMPTY_ node".

Finally, the query-term features consist of Wikipedia, N-gram and several other features. The N-gram conditional probability measures $P(term|\text{N-gram})$ where the N-gram equals to all phrases (N up to 5) in the original query. e.g., for query "stanford university map" and term "online", there are three probability scores: $P(\text{online}|\text{stanford university map})$, $P(\text{online}|\text{university map})$ and $P(\text{online}|\text{map})$. Meanwhile, the KL Divergence score measures the closeness of two probability distributions: ODP-topic distribution for the query and for the term. Additionally, the inverted query frequency also measures the specificity of the term given a query, defined as $\log N(t_i)/N(t_i|q_j)$, where $N(t_i)$ is the total number of appearances of term t_i and $N(t_i|q_j)$ measures how many times t_i is used to modify query q_j.

Overall, we constructed 30 features for each training instance. The importance of individual features will be discussed in the experiment section.

5.2 Inferring Training Labels from Log

The next step is to generate ranking labels for each group of query refinements. Instead of asking human judgers to manually label a large set of training data, we infer training labels automatically from implicit user feedbacks in the log. For each of the training instance, we extract its total number of appearances (impressions) and total number of user clicks from the log. In literature, it is quite common to use either the total clicks or the click through rate (defined as clicks/impressions) to approximate the labels. Nevertheless, these two metrics sometimes conflict with each other (as shown in the following examples). Instead, we use a more principled way to derive rankings. Since in our training tuple $\{q_1, q_2, click_{url}\}$, we consider both clicks and skips (impressions-clicks) as good signals, it is important to find a model that takes both numbers into consideration. Specifically, given a refined query q' for the original query q, we model its probability of click distribution using a function $\Phi(\alpha, \beta)$ where α corresponds to the total clicks and β is the total skips. In our model, we fit this distribution using a $Beta$ function by following [14], i.e., $Beta(\alpha, \beta)$. Consequently, to compare two query refinements q_i and q_j, it has been shown in [14] that the probability of q_j better than q_i can be computed as

$$P(q_j \succ q_i) = \int_0^1 Beta_j(R_j)\Phi_i(R_j)dR_j, \qquad (5)$$

where R_i and R_j corresponds to the ranking of q_i and q_j respectively, Φ_i the cumulative distribution function of R_i. Numerical integration is used to estimate the solution. The rankings of the refined queries are then derived directly from the probabilities which show the pair-wise comparison results. Figure 2 shows an example of comparing two cases.

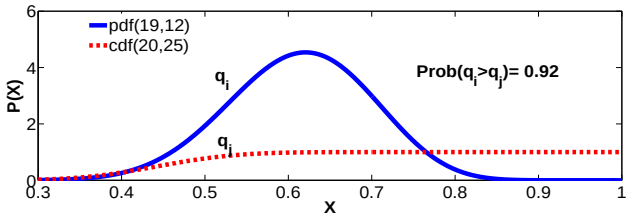

Figure 2: An example of compare two query refinements q_i:(19, 12) and q_j:(20, 25). Although q_i has less clicks than q_j (19 vs. 20), it has a very high probability (0.92) of being a better refinement due to its significantly less number of skips (12 vs. 25).

More specifically, table 6 shows two examples from our log. Using our model, we show that

$$P(\text{stanford admission} > \text{stanford map}) = 0.61$$
$$P(\text{stanford map} > \text{stanford history}) = 0.08$$

The query "stanford admission" is a clear winner over "stanford map" with higher number of clicks and relative less number of skips. On the other hand, "stanford map" is ranked lower than "stanford history". According to our method, the former query has more clicks but also significantly more skips that the latter, thus "map" should be treated as a less relevant term than "history" for the query "stanford".

Query	Term	# of clicks	# of skips	rank
stanford	admission	1913	1235	1
	map	568	438	4
	history	324	148	2
	employment	1273	934	3
microsoft	career	2274	1589	2
	office	1089	222	1
	jobs	1748	1733	3
	games	523	670	4

Table 6: Two examples of constructing query-term training data. Each query consists of a list of terms with their statistics. Labels are assigned according to the probability from eq.(5).

5.2.1 Online Query Suggestion

Given a new query q that contains k terms: $\{t_1, ..., t_k\}$, we first create a candidate set of queries by removing one terms from the original query. e.g., for query with terms $\{t_1, t_2, t_3\}$, the candidate set contains $\{t_1, t_2\}$, $\{t_1, t_3\}$ and $\{t_2, t_3\}$, as well as the query itself. Then for each query in the candidate set, we construct features for all terms available in our training set and calculate its score using our model. The highest scored terms are suggested for the query. In practice, since the potential number of candidate suggestions is quite big (n suggested terms for each candidate, where n is the number of unique terms in the training data), we prefiltered candidates by using ODP categories, where only the

Query Features	Is the query a Wikipedia title? $\in \{0, 1\}$ Is the query a Wikipedia category? $\in \{0, 1\}$ # of times query contains in Wikipedia title $\in \mathbb{R}$ # of times query contains in Wikipedia body $\in \mathbb{R}$ N-gram probabilities in the title, body, anchor and query streams $\in \{0 \sim 1\}$
Term Features	Pagerank score of the term # of inlinks & outlinks entropy of Pagerank score in 16 ODP topics: $-\sum_i P(t_i) \log P(t_i)$ # of times derived from _EMPTY_ node
Query-Term Features	All Wikipedia features from query features N-gram conditional probabilities, $p(w_n \mid w_{n-m+1}, ..., w_{n-1})$ KL Divergence of query and term probability distributions Inverted Query Frequency: $\log N(t_i)/N(t_i \mid q_j)$

Table 7: Some of the features used in the learning-to-rank method.

terms within the same category as the candidate query are considered.

6. EXPERIMENTS

In this section, we describe the evaluation methods. Our experiments consist of primarily two steps. During the first step, we compare the performance of the two proposed models based on the labels we derived from the log. We also calibrate the parameters of our methods according to the labels. During the second step, we conduct user study and ask judgers to evaluate the query suggestion performance of our proposals as well as several state-of-the-art methods.

6.1 Parameter Tuning

This section describes how to calibrate the parameters in our algorithms. For the Pagerank algorithm, we only need to tune one parameter which is the jumping rate α. Since we use the SVM-rank package[2] for our learning-to-rank approach, we calibrate the c parameter which specifies the trade-off between training error and the margin[3].

For evaluation, we use two well-known metrics: normalized discounted cumulative gain (NDCG) and zero/one-error.

NDCG is widely used in information retrieval systems. In our study, we treat the training labels as the golden standard. We use the top-5 suggested queries for evaluation. From the training label, we use a five-scale relevance score (rel): Perfect (5), Excellent (4), Good (3), Fair (2) and Poor (1). The Ideal DCG (IDCG) is calculated from the training set itself. For the suggestions, if the query is in top 5 of the training data, we assign the score according to its original position or 0 otherwise. The NDCG is calculated as:

$$NDCG(q) = \frac{\sum_i (rel_i)/\log(i+1)}{IDCG(q)}. \quad (6)$$

The NDCG of the test set is averaged over all queries.

The zero/one-error metric, on the other hand, is a more rigorous metric. It calculates the fraction of perfectly correct rankings between training set and test result. The output is binary for each ranking and averaged over all queries.

Figure 3 depicts the results of two metric according to different jumping rates of Pagerank algorithm. The two horizontal lines correspond to the learning-to-rank approach which is not affected by α. It can be noticed that when $(1-\alpha)$ increases, Pagerank generally performs better and better. The best NDCG score is achieved at 0.85 for Pagerank with the value of 0.61, whereas learning-to-rank achieves 0.82 NDCG — a 25% better performance. Similarly, Pagerank has the best zero/one loss score of 0.52, comparing to 0.75 for learning-to-rank algorithm.

In Figure 4, we show the results of the Pagerank algorithm across 14 ODP topics. Comparatively, our algorithm achieves the best performance in the business category, a 0.7 NDCG score. The news category, on the other hand, exhibits the worst performance, which has only 0.47 NDCG and 0.35 zero/one loss scores.

Next, we split the training data into different proportions and examine the performance of the learning-to-rank approach. Figure 5 shows the results of two metrics with training data ranging from 20% to 90%. Even with only 20% of training data, the learning-to-rank method is capable of achieving 0.63 NDCG score. The highest NDCG score is around 0.82 when most of the data is used for training.

Finally, we performed a 5-fold cross validation to select the optimal parameter c in SVM-rank in the range of [-3, 3]. The performance generally increases when c becomes larger. The optimal performance was found when c equals 2.8.

Figure 6 lists the top-weighted features used in learning-to-rank model. The Wikipedia title feature indicates critical importance which is top-ranked. The Pagerank score with 0.85 jumping rate also shows superiority which predicts the term relevance well. 4 of the N-gram features made to the top-10 feature list where the uni-gram feature ranks the highest among them.

6.2 User Study

After calibrating the parameters for our models, we conducted a user study by comparing our methods to others. Since the user study is unable to cover all queries in our data set, we applied a similar approach as in [7, 23] by randomly selecting 450 queries from the search log for judgment.

To compare with state-of-the-art algorithms, we implemented two random walk algorithms which have shown good performance in literature. The first algorithm involves using pseudo relevance feedback [23] which builds the query-url bipartite graph by considering all top URLs returned by a search engine as relevant (RW-Pseudo). This algorithm has shown significantly better performance than the traditional random walk method. The second algorithm considered

[2]http://www.cs.cornell.edu/People/tj/svm_light/svm_rank.html
[3]We used linear kernel in our experiments. Due to the data size, all other kernels reported out-of-memory errors.

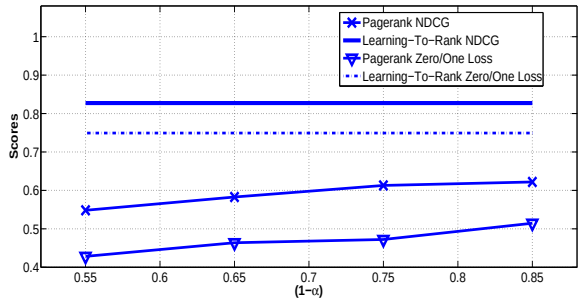

Figure 3: The NDCG & zero/one loss scores affected by the jumping rate α.

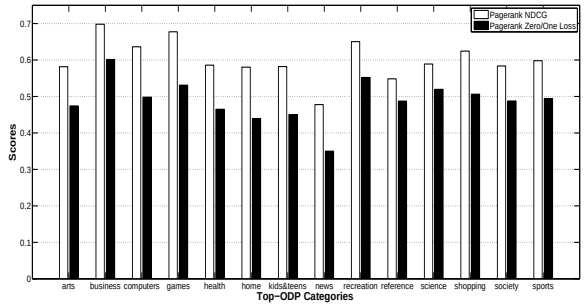

Figure 4: The performance of Pagerank algorithm in all top-14 ODP categories in the training set.

is a backward random walk model [7](RW-Back), which addresses the bias towards URLs with more clicks in traditional forward walk models. This model assumes a uniform prior on all URLs by normalizing the graph on URL clicks instead of query counts, which turns out to be much more efficient in retrieval relevant queries and URLs.

To be more convincing, we also compare with a recently developed query-flow approach [4] that represents the transition between queries into directed graphs and learns the edge weight from a set of features. For query suggestion, we use one of their methods which also leveraged random walk that shown better performance than maximum weight and history-based methods. Furthermore, we compare with an optimized model of the query-flow approach that is described in [2], which introduced query-recommendation links that improve the quality of the random walk model.

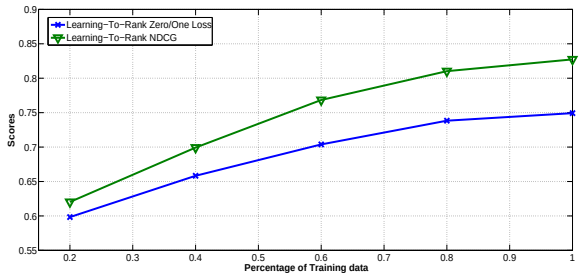

Figure 5: The performance of learning-to-rank algorithm given different portion of training data.

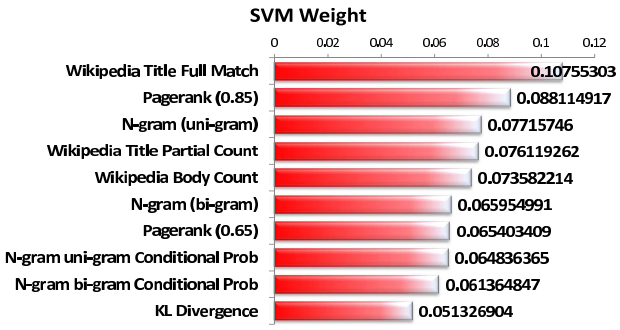

Figure 6: Top-ranked features from SVM-rank used in learning-to-rank model.

Finally, we implemented a baseline model. This simplified model only takes query frequency into consideration. Specifically, for any query q, this model recommends q' which appears most frequently after users issued q, and leads to a successful click in the session.

Following [7, 23], we conducted the user study approach as follows. For each sampled query, the four algorithms in comparison independently generate top-5 suggested queries. We then mixed the suggestions by removing the duplicates and randomly sort the remaining, which helps eliminating the positional bias during user study. The suggestions, along with the original query, are given to judgers to evaluation. We asked the judgers to evaluate whether each suggestion is (1) relevant, (2) irrelevant or (3) no opinion (hard to judge). Every query is triple-judged by three different judgers and the majority vote is used as the final label. During the user study, we also give the judgers some context by showing the results from search engine besides the queries, for a better understanding of what the query means.

Since the responses from judgers are binary, the NDCG metric used in previous section no longer applies. Instead, we evaluate by using Precision at rank N (P@N) and Mean Average Precision (MAP). The precision at rank N for a specific query is defined as the percentage of relevant queries:

$$P(N) = \frac{\#\ rel.\ queries}{N}. \tag{7}$$

P@N is defined as the aggregated precision for all queries,

$$P@N = \frac{\sum P(N)}{\#\ total\ queries}. \tag{8}$$

While P@N addresses the precision, MAP takes both precision and recall into consideration. Specifically, for each query q_i

$$AverageP(q_i) = \frac{\sum_j (P(j) \cdot \mathcal{I}(j))}{\#\ rel.\ queries},$$
$$MAP = \frac{1}{K} \sum_k AverageP(q_k), \tag{9}$$

where $P(j)$ is the precision at rank j as defined in eq.(7) and $\mathcal{I}$ equals 1 when query at j is relevant and 0 otherwise.

Query	Pagerank	Learning-to-Rank
battlefield bad company 2	battlefield bad company 1 battlefield bad company 2 ringtones battlefield bad company 2 slots battlefield bad company 2 realms battlefield bad company 2 games	battlefield bad company 2 price battlefield bad company 1 battlefield bad company 2 download battlefield bad company 2 walkthrough battlefield bad company 2 games
best exercise for abs	best exercise for abs workout best exercise for abs exercise best exercise for abs lower best exercise for abs exercises best exercise for abs workouts	best exercise for abs home best exercise for abs six best exercise for abs workouts best exercise for abs cardio best exercise for abs diet
dante's inferno xbox360	dante's inferno xbox360 wiki dante's inferno ps3 dante's inferno xbox360 cheats dante's inferno xbox360 walkthrough dante's inferno	dante's inferno xbox360 wiki dante's inferno xbox360 video dante's inferno xbox360 walkthrough dante's inferno xbox360 cheats dante's inferno xbox360 guide

Table 8: Examples of query suggestions by Pagerank and learning-to-rank methods. Our methods work best for short and medium queries in most scenarios.

Algorithm	MAP	P@5	P@1
LTR	0.6709452	0.6346841	0.668909
Pagerank	0.6483460	0.600000	0.630982
RW-Back [7]	0.5805054	0.5480981	0.57902
RW-Pseudo [23]	0.5789023	0.5408738	0.573906
Baseline	0.458293	0.428911	0.452865
Query-Flow [4]	0.570428	0.539200	0.609072
Query-Flow-Opt [2]	0.583297	0.543886	0.570965

Table 9: Overall scores of the seven algorithms in MAP, P@5 and P@1. LTR performs the best in all metrics. Query-Flow is the second worst for $P@5$ after the Baseline. Query-Flow-Opt shows marginal improvement over Query-Flow but not significantly.

	Deletion		Modification		Expansion	
	Pagerank	LTR	Pagerank	LTR	Pagerank	LTR
S	0.47	0.56	0.59	0.62	0.64	0.68
M	0.54	0.61	0.57	0.6	0.63	0.66
L	0.47	0.57	0.54	0.56	0.63	0.67
Avg	0.5	0.58	0.58	0.6	0.63	0.67

Table 10: Comparison of Pagerank and LTR in terms of suggested methods: deletion, modification and expansion. Results are broken down into query lengths. Short: $\leq$ 2, medium: $[3, 5]$ and long: ≥ 6.

6.2.1 The Results

Table 9 summarizes the overall performance of the four algorithms in terms of MAP, P@5 and P@1. The learning-to-rank and Pagerank approaches significantly outperformed random walk methods in all three metrics. Comparatively, LTR improves 2.2% of MAP, 2.4 % P@5 and 3.8% P@1 upon the Pagerank algorithm, which shows the usefulness of the Wikipedia and N-gram features in addition to the Pagerank features. Note the query-flow approach performs better than both random walk models (but worse than both of ours) for $P@1$, but the performance significantly decreased for precision at 5. We looked into the data and figured out the reason. Since the query-flow method aggregates node at query level, the graph loses the power of distinguishing ambiguous queries with different user intent. Therefore lots of noises are introduced to random walk which also gradually increases with the length of the walk. On the other hand, the optimization framework proposed in [2] only achieves marginal improvement over the query-flow method. Both query-flow approaches are significantly worse than our proposals, which supports the importance of topical-level term-transition graph.

Figure 7 presents the P@5 scores in different ODP categories. Learning-to-rank outperforms other approaches in almost all categories except for the business and games categories. On the other hand, Pagerank exhibits better performance than random walks and query-flow graphs in 8 out of 13 categories. In general, the home and kids&teens are the most difficult categories to make query suggestions, while games and computers are relatively easier than others.

Furthermore, we analysis the effectiveness of our algorithms by breaking down the queries into different query lengths as well as the three techniques (term deletion, modification and expansion) in terms of P@5. As shown in Table 10, long queries are generally the most difficult to make suggestions. While for short and medium queries, the expansion technique indicates better suggestion performance than deletion and modification. The Pagerank algorithm exhibits relatively poor performance for long queries since it is usually harder to accurately infer the topics from long queries, which justifies the superiority of term-level suggestion method over the topic-level method.

Table 8 shows four randomly-selected queries and the suggestions by our algorithms. As it can be seen, most of the suggestions are quite relevant to the query itself yet some of the irrelevant ones are suggested due to the ambiguity of the query itself (e.g., "eclipse" in the query "eclipse 2010").

7. CONCLUSION AND FUTURE WORK

In this paper, we presented a novel query suggestion framework which extracted user preference data from user sessions in search engine logs. We then used the user pat-

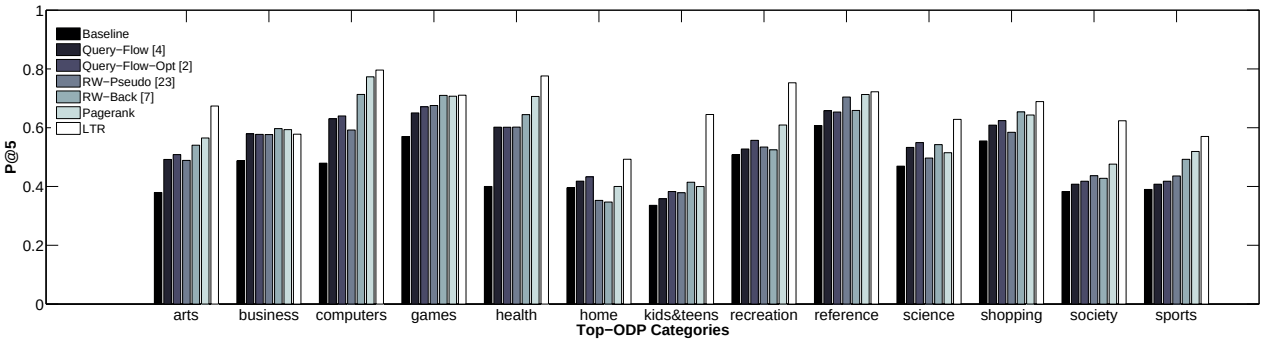

Figure 7: Breakdown of P@5 for 13-ODP categories in the testing set consisting of 450 queries.

terns to build two suggestion models. The first model was a topic-based Pagerank model where the suggested terms were treated as nodes in directed graphs. Pagerank algorithm was used to calculate the relative importance of each term within a topic. The model suggested new queries to the original query by using one of the three techniques: term deletion, term expansion and term modification. The second model was meant to address the limitation of the topic-based model, which constructed features from Wikipedia, N-gram, Pagerank and other resources, and performed query-level suggestions by training a learning-to-rank model.

Our models were compared with two state-of-the-art random walk methods and query-flow models on randomly-sampled queries. A rigorous user study was conducted where all queries are triple-judged by human judgers. Experimental results indicated significant improvement on our models. We also broke down the queries into different ODP categories as well as query lengths. It was shown that our models worked the best for short and medium-length queries.

There is a rich body of work to be extended. As a preliminary study, our model only considered changing one term from the queries. It would be interesting to see how the performance will increase/decrease by leveraging a more sophisticated user preference extraction model which could consider multi-term alteration. In our topic-biased Pagerank model, only the first-level ODP categories are used to classify queries, while suggesting queries in a lower-level of category which further narrows down user intent, might be more relevant than queries on the same topic category.

8. REFERENCES

[1] Microsoft web n-gram services. http://research.microsoft.com/en-us/collaboration/focus/cs/web-ngram.aspx.

[2] A. Anagnostopoulos, L. Becchetti, C. Castillo, and A. Gionis. An optimization framework for query recommendation. In WSDM '10, pages 161–170, 2010.

[3] P. N. Bennett and N. Nguyen. Refined experts: improving classification in large taxonomies. In SIGIR '09, pages 11–18, New York, NY, USA, 2009. ACM.

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

[5] 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 '08, 2008.

[6] K. Collins-Thompson and J. Callan. Query expansion using random walk models. In CIKM '05, pages 704–711.

[7] N. Craswell and M. Szummer. Random walks on the click graph. In SIGIR '07, pages 239–246, 2007.

[8] V. Dang and B. W. Croft. Query reformulation using anchor text. In WSDM '10, pages 41–50, 2010.

[9] H. Deng, I. King, and M. R. Lyu. Entropy-biased models for query representation on the click graph. In SIGIR '09, pages 339–346, 2009.

[10] T. H. Haveliwala. Topic-sensitive pagerank. In WWW '02, pages 517–526, 2002.

[11] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In WWW '06, pages 387–396, 2006.

[12] G. Kumaran and J. Allan. A case for shorter queries, and helping users create them. In Human Language Technologies 2007, pages 220–227, April 2007.

[13] G. Kumaran and J. Allan. Effective and efficient user interaction for long queries. In SIGIR '08, pages 11–18, New York, NY, USA, 2008. ACM.

[14] C. Liu, M. Li, and Y.-M. Wang. Post-rank reordering: resolving preference misalignments between search engines and end users. In CIKM '09, pages 641–650, 2009.

[15] T.-Y. Liu. Learning to Rank for Information Retrieval. Now Publisher, 2009.

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

[17] M. Mitra, A. Singhal, and C. Buckley. Improving automatic query expansion. In SIGIR '98, pages 206–214, 1998.

[18] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, 1999.

[19] F. Radlinski and T. Joachims. Query chains: learning to rank from implicit feedback. In KDD '05, pages 239–248.

[20] S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. pages 109–126, 1996.

[21] J. Rocchio. Relevance Feedback in Information Retrieval, pages 313–323. 1971.

[22] F. Silvestri. Mining query logs: Turning search usage data into knowledge. Foundations and Trends in Information Retrieval, 4(1-2):1–174, 2010.

[23] Y. Song and L. He. Optimal rare query suggestion with implicit user feedback. In WWW '10, pages 901–910.

[24] X. Wang and C. Zhai. Mining term association patterns from search logs for effective query reformulation. In CIKM '08, pages 479–488, 2008.

[25] C. Zhai and J. Lafferty. Model-based feedback in the language modeling approach to information retrieval. In CIKM '01, pages 403–410. ACM, 2001.