Learning Similarity Function for Rare Queries

Jingfang Xu, Gu Xu
Microsoft Research Asia
4/F, Sigma Building, No.49, Zhichun Road
Haidian District, Beijing (100080), China
{jingxu, guxu}@microsoft.com

ABSTRACT

The key element of many query processing tasks can be formalized as calculation of similarities between queries. These include query suggestion, query reformulation, and query expansion. Although many methods have been proposed for query similarity calculation, they could perform poorly on rare queries. As far as we know, there was no previous work particularly about rare query similarity calculation, and this paper tries to study this problem. Specifically, we address three problems. Firstly, we define an n-gram space to represent queries with their own content and a similarity function to measure the similarities between queries. Secondly, we propose learning the similarity function by leveraging the training data derived from user behavior data. This is formalized as an optimization problem and a metric learning approach is employed to solve it efficiently. Finally, we exploit locality sensitive hashing for efficient retrieval of similar queries from a large query repository. We experimentally verified the effectiveness of the proposed approach by showing that our method can indeed enhance the accuracy of query similarity calculation for rare queries and efficiently retrieve similar queries. As an application, we also experimentally demonstrated that the similar queries found by our method can significantly improve search relevance.

Categories and Subject Descriptors

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

General Terms

Algorithms, Experimentation, Performance, Theory

Keywords

Query Similarity, Rare Query, Learning Similarity Function

1. INTRODUCTION

Many query processing tasks like query suggestion[34, 20, 19], query reformulation[17, 15, 27], and query expansion[32, 26] need to implicitly or explicitly calculate and utilize query similarity. Query suggestion manages to provide similar queries to the user, which represent the same or related search intent as the current query. Query reformulation attempts to transform the current query into a similar but better-formed query to help the user to find more relevant documents. In query expansion, similar terms (words) are added into the original query in order to improve the recall of search. In some sense, the expanded query can also be viewed as a similar query to the original query.

Various approaches to measuring query similarity have been proposed for different applications using different data sources. However, most of them were mainly focus on common or frequent queries. The distribution of web search queries follows a heavy-tailed power-law distribution. It implies that a major proportion of queries are in fact rare queries, which are issued infrequently. As far as we know, there is no previous work particularly about rare query similarity and this paper tries to study this problem.

User behavior data, including click-through and session data, has been widely accepted as an important data source to accurately calculate query similarity [20, 19, 3, 30, 34]. The basic intuition behind various methods, which utilize user behavior data for query similarity calculation, is that two queries should be similar to each other if they share similar behavior patterns, e.g. many co-clicked URLs or co-occurred sessions. These methods generally work well on common queries, but would fail on rare queries as there is insufficient user behavior data for rare queries. Several methods have been proposed to propagate similarities between queries by using user behavior data [7], but it still cannot essentially overcome the problem of data sparsity on rare queries. (In an extreme case, some queries may be missing in the user behavior data, or unseen to a search engine.)

Defining a similarity function merely based on the content of queries should be a more essential solution and generally applicable to both common and rare queries. Traditionally, cosine similarity and bag-of-words assumption are widely employed to measure the similarities between text strings. However, there are two fundamental issues that block the way to accurate similarity calculation: term ambiguity (literally the same but semantically different) and term dependency (literally different but semantically similar). These issues are exacerbated in query similarity calculation given that queries are normally very short. In this paper, we will address both issues and aim to solve the problem of rare query similarity with a novel similarity function defined on the content of queries.

In this paper, we propose a principled approach to learn a similarity function for queries, especially for rare queries. The basic idea of our method is to represent queries using

WSDM'11, February 9–12, 2011, Hong Kong, China.

n-grams and automatically learn a query similarity function defined in the n-gram space by leveraging the training data derived from user behavior data. In our model, rare queries are represented in the same way as common queries, and thus we can leverage the knowledge learned from common queries to improve the similarity calculation on rare queries.

Specifically, we address the following issues: (1) to define a n-gram space to represent query with its own content and a similarity function to measure similarities between queries; (2) to learn the similarity function by leveraging the query similarity information derived from user behavior data; (3) to index queries and provide an efficient way to finding similar queries from a large query repository with the learned similarity function.

First, we propose to use n-gram space for query similarity calculation, where a query is represented as a vector with each dimension corresponding to an n-gram. Furthermore, a similarity function is defined in the n-gram space, which is equivalent to the cosine similarity in a transformed n-gram space. The n-gram space can be viewed as an extension of the traditional vector space where each dimension corresponds to a unigram or word, while the conventional cosine similarity can be thought as a special case of the proposed similarity function.

The n-gram space and new similarity function respectively address the two challenges aforementioned. Compared with unigram, n-grams convey more context information and can partially solve the problem of term ambiguity. For instance, in the n-gram space, "apple iphone" can be represented as <"apple", "iphone", "apple iphone">. With this representation, it is clearer that the term "apple" in "apple iphone" is about the company rather than the fruit. To tackle the problem of term dependency, we employ the similarity function defined in a transformed space where terms are no longer orthogonal or independent. Then the similar query pairs without common terms(e.g., "NY" and "New York") can have high similarity scores.

Second, we employ a supervised learning approach to automatically learn the query similarity function. We can utilize user behavior data (e.g. click-through or session data) to derive similar or dissimilar query pairs, and learn a similarity function that is consistent with the input query pairs. This is formalized as an optimization problem whose objective is to maximize the similarities between similar query pairs and minimize the similarities between dissimilar pairs. Furthermore, a sparsity constraint is also introduced to guarantee a sparse solution to the problem. We also employ a very efficient metric learning algorithm to solve the optimization problem. The advantage of our learning approach is that we can learn n-gram dependencies from similar and dissimilar common query pairs in training data and then utilize them to improve query similarity calculation for all queries, especially helpful for rare queries.

Last, we adopt the locality sensitive hashing framework to index queries in the n-gram space. Efficiently finding similar queries from a large query repository is a common problem in various query processing tasks. A new hash family is derived to compute hash codes based on the learned similarity function. This method can accelerate the speed of similar query retrieval and effectively support online applications.

We have experimentally verified the effectiveness of our approach to query similarity calculation and the efficiency of similar query retrieval. The experimental results show that our method significantly improves upon the existing methods in accuracy of query similarity calculation for rare queries, such as Pearson coefficient on click-through data and cosine similarity on n-grams. Moreover, with the proposed locality sensitive hash function, we can efficiently retrieve similar queries from a large query repository. We have also applied our method to improve the search relevance for rare queries, which indicates that by using better similar queries defined by the learned similarity function we can significantly improve the performance of search.

The rest of the paper is organized as follows. Section 2 introduces related work. In Section 3, we show the distributions of Web search queries and verifies the importance of rare queries. Our method is described in Section 4, followed by experimental results given in Section 5. Finally conclusions are made in the last section.

2. RELATED WORK

Query processing is an important kind of tasks in web search, such as query suggestion [34, 20, 19], query reformulation [17, 15, 27], and query expansion [32, 26], etc. Recently, rare query processing has attracted much attention from IR community [25, 4, 5]. Most of previous work on query processing explicitly or implicitly calculate and utilize query similarity. Our work differs from the previous work in that it focuses on query similarity calculation, especially for rare queries. Our method is applicable to all the tasks that utilize query similarity.

So far many methods have been proposed and various types of data have been utilized to calculate query similarity. Our approach is unique because we make use of not only terms (n-grams) but also click-through data, and automatically learn a similarity function which is applicable to any queries.

By viewing queries as short documents, one can exploit the document similarity functions to measure query similarities. These include degree of term overlap between queries [2, 30], edit distance between queries [2, 30], TF-IDF weighted cosine similarity between queries [34] and other measures [21]. Since queries are too short (2–3 words on average), these term-based similarity functions are usually perform poorly. Our work is unique in that we learn term(n-gram) dependencies from click-through data to deal with the term mismatch problem.

Since queries are short and it is difficult to calculate their similarities, auxiliary information has been used as features to enrich query representations. Search results from a search engine emerge as one type of auxiliary information[23]. They have been verified to be an effective data for query expansion [32]. Sahami *et al.* [24] proposed using search result snippets to compute TF-IDF weighted cosine similarity in query expansion. As an extension, Yih [33] took a learning approach to calculating term weights using human labeled training data. The search-engine-based methods heavily rely on the performance of search engine and are hard to apply into large-scale and time-sensitive scenarios.

Click-through data is another data source for deriving query features. Mei *et al.* [20] used the 'first hitting time' on the query-URL bipartite graph as a similarity measure. Craswell & Szummer [7] built two random walk processes to propagate query similarity along the graph and obtained better similarity scores between queries. In [19], matrix factorization was employed to compute query similarity on both query-URL and query-user bipartite graphs. Besides, the

overlap of clicked URLs between two queries in a query-URL bipartite was also used as query similarity measure [3, 30]. Search session data is also a widely used data source. Zhang *et al.* [34] proposed using the number of co-occurrence of query pairs in search sessions as a query similarity measure. These methods for calculating query similarities based on user behavior data perform well on common queries where the wisdom of crowds can be fully leveraged. However, they are not applicable to rare or new queries. Our work differs from the previous work in that it utilizes click-through data to automatically generate training data, and then learns the query similarity function which is consistent with the training data.

Metric learning is a new area in machine learning, which aims to learn a metric space from supervision. We employ metric learning method to solve the optimization problem in learning similarity function. So far, many methods for metric learning have been proposed [31, 28, 12]. Most methods require to solve a semidefinite programming problem, which is computationally very expensive and can hardly be applied to large-scale or high-dimensional problems. Recently, some efficient algorithms for metric learning have been devised [29, 9, 11, 22]. We employ the algorithm proposed in [16, 9] in this paper. The basic idea of the algorithm is to introduce a prior with Bregman divergence plus sparsity constraints [11, 22].

High-dimensional indexing is also related to our work. How to efficiently find similar queries from a large collection of existing queries is another core problem in our scenario. Locality sensitive hashing (LSH) [13] is a randomized approach to high-dimensional indexing and does not suffer from "curse of dimensionality". It is a promising approach especially for online applications. So far, various hash families have been devised for different metric functions, e.g. Euclidean distance [8] and cosine similarity [6]. Bridging metric learning and locality sensitive hashing together turns out to be an interesting topic [14]. In this paper, we propose a method to index queries and efficiently retrieve similar queries.

Some existing methods on term vector space can also be interpreted as methods of learning similarity function in the vector space. For example, latent semantic analysis (LSA) [10] and supervised semantic indexing [1] transform the original vector space of bag-of-words into a low-dimensional space of topics. There are clear differences between our approach and the existing approaches, however. In our work, we introduce a sparsity constraint to the high-dimensional space and do not make dimension reduction. In contrast, in the existing approaches the "learning" process is driven by dimension reduction.

3. COMMON QUERY VS. RARE QUERY

In this section, to investigate the importance of rare queries, we study the distributions over queries with a search log data accumulated at a popular commercial search engine during a time period of one month. There are about 320 million unique queries and 967 million associated clicks in total.

We commonly use the number of impressions to classify common and rare queries. Specifically, we can consider the queries, whose impression frequencies are larger than a given threshold, as common queries, and the rest as rare queries. Fig. 1 shows the proportions of common queries and rare queries on different thresholds. One can easily see that most

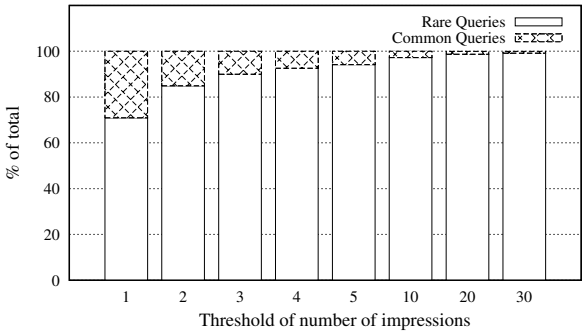

Figure 1: Proportions of common queries and rare queries with various thresholds of number of impressions

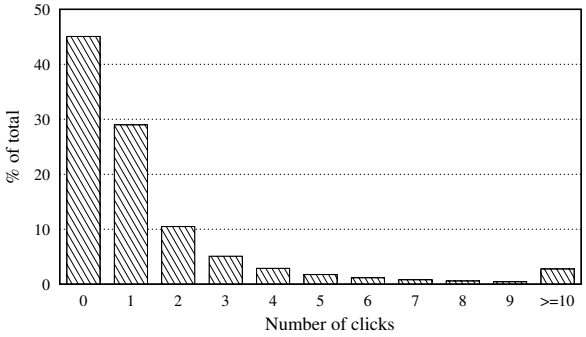

Figure 2: Number of clicks of Queries

of queries in the search log are actually infrequent or rare queries. For instance, there are more than 97% queries that are issued less than 10 times within one month. It confirms the importance of rare queries as well as the position of this paper.

Fig. 2 shows the availability of click-through data. We can see from the figure that most of queries do not have many clicks. For example, about 45% queries do not have any clicks and only about 2.7% queries have more than 10 clicks within one month. It concludes that the click-through data, user behavior data in general, is quite sparse which is not applicable for most search queries. Our approach is based on the own content of queries, i.e., n-grams, and applicable to all the queries including those do not have user behavior data.

4. OUR METHOD

In this section, we elaborate on our approach to rare query similarity calculation. We propose a solution for each of the issues aforementioned.

4.1 N-Gram Space of Queries

We define a n-gram space for query similarity calculation, where each query (including rare queries) is represented as a vector with each dimension corresponding to an n-gram. Furthermore, a similarity function is also defined upon the n-gram space to measure "the ideal similarity" between queries.

Table 1: Examples of n-gram Vectors

Query	Vectors in n-gram vector space (ny,new,york,times,ny times,new york,...)
NY times	(1, 0, 0, 1, 1, 0, ...)
New York times	(0, 1, 1, 1, 0, 1, ...)

4.1.1 N-Gram Space

We first give the definition of n-gram space.

Definition 1. The n-gram vector space is a vector space that represents a query as a vector in the space, and each dimension of the space corresponds to an n-gram. Suppose that query q has k terms $q = q_1 q_2 \cdots q_k$, then there are n-grams $q_i \cdots q_{i+n-1}$ in q, where $i \leq k - n + 1$.

$$q_1 q_2 \cdots q_k \rightarrow \{q_1, q_2, \cdots q_k, q_1 q_2, \cdots, q_{k-1} q_k, q_1 q_2 q_3, \cdots\}$$

Query q is then represented as a vector in the space. If an n-gram occurs in the query, then the value of the corresponding dimension of the vector is the frequency of the n-gram. The other values of the vector are zero. Examples of n-gram vectors are shown in Fig. 1.

The weighting schemes in IR, like TF-IDF, can be applied to n-gram space model. Furthermore, non-contiguous n-grams can also be considered, with different weights according to degrees of contiguity. To reduce the number of n-grams, in practice, we restrict n-grams to trigrams or bigrams and remove low frequency n-grams.

The vector space used in vector space model, which has been widely used in IR and other related fields, can be considered as a special case of the n-gram space here. It is actually unigram space model. Compared with unigram, n-grams convey richer context information, which is useful in similarity calculation, especially when query terms are ambiguous.

Different from most of previous work which represents query using user behavior data, our method chooses n-grams as the basic units of query representation. One advantage of it is that our representation only depends on the language, but does not depend on the availability and quality of external data, like user behavior data. Therefore, rare queries, which usually cannot be described by user behavior data, are represented in the same way as common queries in our method. (We actually use user behavior data for learning of the similarity function, as explained later).

One may notice some similarities between the n-gram space and the string kernel proposed in [18]. They are essentially different models, however. The n-gram space assumes that text strings are embedded into a finite dimensional vector space, while string kernel assumes that the text strings are embedded into an infinite dimensional vector space (Hilbert space). In the n-gram space, the "vocabulary" is a set of n-grams , while in the string kernel the vocabulary is usually a set of letters.

4.1.2 Similarity Function

There are multiple ways of defining similarity function upon n-gram space. Cosine similarity is a similarity measure widely used in IR and related fields. However, directly defining the cosine similarity in n-gram space for query similarity calculation will not work well, because in the n-gram space n-grams are assumed to be independent from each other. This assumption is too strong to represent semantic similarities. For example, "New York times" is similar to "NY times", but different from "movie times". However, cosine similarity will give the same similarity scores to the pairs, because "times" is the only common n-gram among the queries. Ideally we would want to make "NY" and "New York" as close as possible.

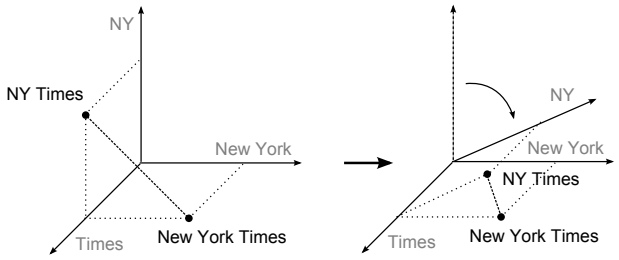

Figure 3: Transformation between n-gram spaces

In this paper, we take an extension of cosine similarity as the similarity function. We actually do not directly utilize cosine similarity in the original n-gram space. Instead, we make a linear transformation of the space to another one, and take the cosine similarity in the transformed space as the "ideal similarity function". In the transformed space, the n-grams could be dependent. For the above example, we try to find a transformation that can make the angle between "NY" and "New York" approach to zero, as shown in Fig. 3. In this way, the cosine similarity between "New York times" and "NY times" will be much larger than that between "New York times" and "movie times".

Let $\Phi \subseteq \mathcal{R}^m$ be the original n-gram space of queries and $\phi(q)$ be the vector of query q in Φ ($\phi(q) \in \Phi$), where m denotes the dimensionality of Φ. Further let $\Theta \subseteq \mathcal{R}^k$ be a new n-gram space, in which the dependencies between n-grams are considered, and $\theta(q)$ be the vector of q in Θ ($\theta(q) \in \Theta$), where k denotes dimensionality of Θ. We introduce a linear transformation L from Φ to Θ. Then, for each q, we have

$$\theta(q) = L\phi(q). \tag{1}$$

The cosine similarity between any two queries q_i and q_j in the new space Θ is:

$$\cos(\theta(q_i), \theta(q_j)) = \frac{\theta(q_i)^T \theta(q_j)}{\sqrt{\theta(q_i)^T \theta(q_i)} \sqrt{\theta(q_j)^T \theta(q_j)}}. \tag{2}$$

Therefore the similarity between q_i and q_j in the original space becomes

$$sim(\phi(q_i), \phi(q_i)) = \frac{\phi(q_i)^T M \phi(q_j)}{\sqrt{\phi(q_i)^T M \phi(q_i)} \sqrt{\phi(q_j)^T M \phi(q_j)}}, \tag{3}$$

where $M = L^T L$ is a positive semidefinite matrix.

Note that we actually do not need to explicitly define the linear transformation L, including its dimensionality k. The information about the transformed space is all encoded in M. The only requirement is that M is positive semidefinite. Furthermore note that the range of $sim(\cdot, \cdot)$ is between -1 and $+1$, even the elements of $\phi(q_i)$ and $\phi(q_j)$ are usually non-negative, because the elements of M can be negative.

The elements in the diagonal of M represent the weights of the corresponding n-grams, while others describe the dependencies between n-grams. Assuming M is an identify

matrix, our similarity function is equivalent to the cosine similarity function in original n-gram space. Furthermore, in the case that M is diagonal, the corresponding similarity function becomes the cosine similarity function in original space with n-gram weighting. In this paper, without making any assumption about matrix M, we automatically learn M from supervision for better representing the similarities between queries.

4.2 Learning Query Similarity Function

In this section, we employ a metric learning method to automatically learn the similarity function in n-gram vector space by leveraging training data derived from user behavior data.

4.2.1 Formulation of Learning Problem

We consider how to learn the matrix M defined in the n-gram vector space using supervised learning method. Suppose that there are two sets of training data (containing supervision information) S_+ and S_-, where S_+ includes similar query pairs (e.g. "New York times" and "NY times") and S_- includes dissimilar query pairs (e.g. "New York times" and "movie times"). Our objective is to learn M such that it can maximize the similarities between similar query pairs from S_+, and minimize the similarities of dissimilar query pairs from S_-. Formally, the objective function is written as follows:

$$\max_{M\succeq 0} \sum_{(q_i,q_j)\in S_+} \frac{\phi(q_i)^T M\phi(q_j)}{\sqrt{\phi(q_i)^T M\phi(q_i)}\sqrt{\phi(q_j)^T M\phi(q_j)}} - \sum_{(q_i,q_j)\in S_-} \frac{\phi(q_i)^T M\phi(q_j)}{\sqrt{\phi(q_i)^T M\phi(q_i)}\sqrt{\phi(q_j)^T M\phi(q_j)}} - \lambda\|M\|_1 \tag{4}$$

where $\|M\|_1$ denotes the sum of absolute values of elements in the positive definite matrix M.

The $\|M\|_1$ regularization will lead to a sparse solution to M. The use of such a regularization is called sparse methods and is a popular and powerful technique in machine learning. There are several advantages of taking the approach. First, it is possible to improve the efficiency of learning. The size of M is of order $O(m^2)$ and thus is extremely large, where m is the number of unique n-grams. With the regularization, the actual number of parameters will be drastically reduced. Second, a sparse solution of M implies that an n-gram should only be related to a small number of other n-grams, and independent from the rest of them. It is consistent with the characteristic of language, i.e., one term will only be dependent on a limited number of other terms.

4.2.2 Optimization

For efficient optimization, we rewrite the similarity function (3) into a distance function (5).

$$sim(\phi(q_i),\phi(q_i)) = 2 - 2\left(\phi(q_i)-\phi(q_j)\right)^T M\left(\phi(q_i)-\phi(q_j)\right) \quad s.t.\ \ \phi(q_i)^T M\phi(q_i) = \phi(q_j)^T M\phi(q_j) = 1 \tag{5}$$

Please note that $\left(\phi(q_i)-\phi(q_j)\right)^T M\left(\phi(q_i)-\phi(q_j)\right)$ is the Mahalanobis distance between $\phi(q_i)$ and $\phi(q_j)$.

Therefore, maximizing cosine similarity equals to minimizing the Mahalanobis distance on the "unit sphere". Accordingly, the objective function becomes

$$\begin{aligned}\min_{M\succeq 0} & \sum_{(q_i,q_j)\in S_+} \left(\phi(q_i)-\phi(q_j)\right)^T M\left(\phi(q_i)-\phi(q_j)\right) \\ & - \sum_{(q_i,q_j)\in S_-} \left(\phi(q_i)-\phi(q_j)\right)^T M\left(\phi(q_i)-\phi(q_j)\right) \\ & + \lambda\|M\|_1 \\ s.t.\ \ & \phi(q_i)^T M\phi(q_i) = 1,\ \ \phi(q_j)^T M\phi(q_j) = 1 \\ & \forall\ (q_i,q_j)\in\{S_+\cup S_-\}\end{aligned} \tag{6}$$

We could in principle employ semidefinite programming to optimize the objective function (6). However, semidefinite programming algorithms are usually computationally expensive and thus are almost intractable in our case. Fortunately, there are efficient algorithms proposed recently, and we adopt the one proposed in [22, 11] for our optimization problem.

We first normalize the input n-gram vectors to unit length, i.e., for any q_i we have $\phi(q_i)^T\phi(q_i) = 1$. Then, to satisfy the constraints in equation (6), it is easy to verify that matrix M should be as close to the identity matrix as possible. We relax the constraints in equation (6) by introducing a penalty term in the objective function.

$$\begin{aligned}\min_{M\succeq 0} & \sum_{(q_i,q_j)\in S_+} \left(\phi(q_i)-\phi(q_j)\right)^T M\left(\phi(q_i)-\phi(q_j)\right) \\ & - \sum_{(q_i,q_j)\in S_-} \left(\phi(q_i)-\phi(q_j)\right)^T M\left(\phi(q_i)-\phi(q_j)\right) \\ & + \lambda\|M\|_1 + D_{Burg}(M||I),\end{aligned} \tag{7}$$

where $D_{Burg}(\cdot||\cdot)$ denotes Burg matrix divergence (or LogDet divergence), which is a popular "distance" between matrices. Burg matrix divergence is specifically defined as:

$$D_{Burg}(X||Y) = tr(XY^{-1}) - \log\det(XY^{-1}) - c$$

where c is the dimension of matricis X and Y. With some mathematical derivations, we can rewrite the objective function (7) as a matrix form.

$$\min_{M\succeq 0}\ tr\left(\left(I+\eta\Psi^T\left(diag\left(S[1,\ldots,1]^T\right)-S\right)\Psi\right)M\right) - \log\det M + \lambda\|M\|_1 \tag{8}$$

where Ψ is a matrix stacked by n-gram vectors by column; S is a matrix whose element $s_{i,j}$ is 1 if the ith and jth queries are similar (in S_+), -1 if the ith and jth queries are dissimilar (in S_-), and 0 otherwise. Equation (8) can be efficiently solved using the block-wise coordinate descent algorithm.

If $I+\eta\Psi^T(diag(S[1,\ldots,1]^T)-S)\Psi$, denoted as M_{init}, is positive semidefinite, then matrix M is guaranteed to be positive semidefinite during the optimization. In our experiments, we only use similar pairs (i.e., only S_+). It is easy to prove that M_{init} is positive semidefinite if all the elements in S are non-negative.

4.2.3 Training Data Generation

For our method, we need training data (similar query pairs) to learn the similarity function. In principle, we could collect similar query pairs by human labeling. However, since the n-gram space is high-dimensional, a huge number of similar query pairs are needed and thus human labeling would not be practicable in our case. On the other hand, just

as we have discussed above, many approaches can precisely calculate the similarities between common queries using user behavior data. Therefore, we automatically generate training data for similarity learning in this paper. Without loss of generality, we employ Pearson coefficient method to derive high-quality similar query pairs from click-through data. By representing each query as a click distribution over all possible URLs, we can employ Pearson correlation coefficients as the similarity scores to find similar queries.

4.3 Searching Similar Queries

In practice, given one query, searching similar queries in a large collection of existing queries is a common task. Most of existing work does not formally address this critical problem. Without any special techniques, we have to scan all the candidates which is often not acceptable when the collection is quite large. In this section, we present a new technique for indexing queries and fast retrieving similar queries from a large set of seen queries on the basis of Locality Sensitive Hashing (LSH), a high-dimensional indexing technique.

The basic idea of LSH is to hash points into buckets with a family of locality sensitive hash functions, such that the points close to each other can fall into the same bucket instead of the points that are far apart. By combining the collisions on multiple hash codes, one may theoretically bound the error probabilities in similarity search. See [13] for more details.

In [6], a family of hash functions has been proposed for cosine similarity. With the hash family, efficient retrieval of similar instances based on cosine similarity becomes possible.

The hash family $\mathcal{H} = \{h : S \rightarrow 0, 1\}$ for cosine similarity is defined based upon the "random hyperplane technique", where S is the input space. Specifically, we randomly sample a vector r according to Gaussian distribution $N_n(0, I)$, where n is dimensionality of S. For each r, we define a hash function in S as follows:

$$h_r(u) = \begin{cases} 1 & r^T u \geq 0 \\ 0 & r^T u < 0 \end{cases}$$

The hash function can be interpreted as the sign of inner product between r and an input vector $u \in S$. One can prove that the collision probability between objects is equal to their cosine similarity.

In our case, we first consider using the hash family $\mathcal{H}$ in the transformed space Θ. Suppose that $r \sim N_n(0, I)$. The hash function h_r for $\theta(q) \in \Theta$ becomes:

$$h_r(\theta(q)) = \begin{cases} 1 & r^T(L\phi(q)) = (L^T r)^T \phi(q) \geq 0 \\ 0 & r^T(L\phi(q)) = (L^T r)^T \phi(q) < 0 \end{cases}$$

Letting $v = L^T r$, we obtain $v \sim N_n(L^T 0, L^T I L) = N_n(0, M)$. It concludes that we can calculate the values of hash function (hash codes) in the original space. Specifically, we sample a vector u from Φ according to Gaussian distribution $N_n(0, M)$ and use the following hash function.

$$h'_v(\phi(q)) = \begin{cases} 1 & v^T \phi(q) \geq 0 \\ 0 & v^T \phi(q) < 0 \end{cases}$$

There are several advantages of using the new hash family. Firstly, because $\phi(q)$ is more sparse than $\theta(q)$, computing hash functions in the original n-gram space is more efficient. Secondly, it means that we do not need to identify the transformation matrix L. Interestingly, what we do with the hash function in the original space is equivalent to indexing and retrieving queries in the transformed space. But we actually do not know what L looks like.

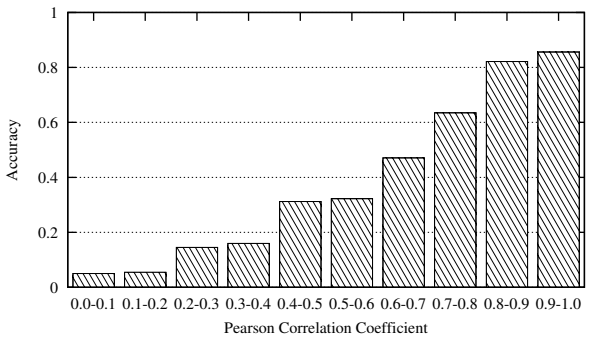

Figure 4: Accuracy of Pearson Coefficient

5. EXPERIMENTAL RESULTS

We conducted experiments to verify the effectiveness of our method. Our experiments were comprised of three parts. First, we evaluated the quality of similar queries found by our method, and compared the accuracy of our method with two baselines on query similarity calculation. Second, we evaluated the efficiency of our query indexing scheme. Last, we applied our method to relevance ranking.

5.1 Experiment Setting

We obtained a click-through dataset accumulated at a popular commercial search engine during a time period of six months, containing queries and their clicked URLs. Due to limitation of our computing resources, we only used the top 100,000 unigrams and bigrams in the click-through data in our experiments. Although we threw away a lot of low frequency n-grams, we were still able to handle 51.5% of unique queries in the data, including a large number of rare queries.

For training data generation, we calculated Pearson coefficients for query pairs which shared at least one co-clicked url in the click-through dataset. Then we randomly sampled 1.3 million query pairs whose Pearson coefficients are larger than 0.8, and took them as our training data. The threshold was decided based on the following experiment. We bucketed query pairs into different bins according to their Pearson coefficients, and asked human judges to evaluate 300 samples from each bin. The detailed judgment guideline has 3 pages and can be summarized as follows. Basically, we consider two queries similar if both of them have one dominant sense (search intent) and they share the same dominant search intent. Otherwise, we do not view the two queries similar. The results are shown in Fig. 4. We can see that when Pearson coefficient is larger than 0.8, more than 82.1% of pairs are indeed similar pairs.

In our similarity function, the parameter λ controls the sparsity of M matrix, while η effects the importance of supervision. Moreover, the training time of our similarity function, depends on the values of parameters(i.e. η and λ), as shown in Fig. 5. As we can see, the training time degrades when the value of λ increases and enlarges quickly with the increasing value of η. In order to balance the performance and training time, we experimentally determined the values

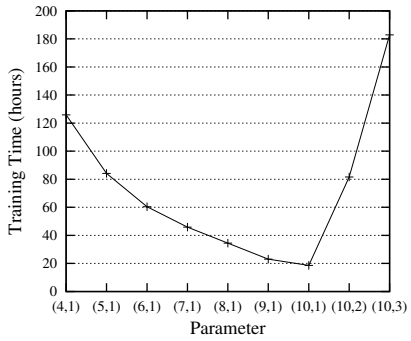

Figure 5: Training time on various values of parameters

of λ and η. We set $\lambda = 4$ and $\eta = 1$ through all the experiments in this paper, and the corresponding training time is about 125 hours.

As baseline methods for calculating query similarities, we used cosine similarity (TF-IDF weighted) on n-gram data and Pearson correlation coefficient on click-through data.

5.2 Accuracy of Query Similarity Calculation

5.2.1 Qualitative Evaluation

We calculated the similarity between any two n-grams with our method. Fig. 2 gives some examples of n-gram pairs whose similarity scores are larger than zero. We can see that various types of synonyms have been discovered by our method, including spelling errors, abbreviations and numbers. There are only 96,151 out of 10 billion n-gram pairs having non-zero similarity scores, indicating that the learned matrix M is sparse.

In Table 3, we also show some example query pairs (similar or dissimilar) with their click frequencies as well as the similarity scores calculated by the three methods. We can see that our method can find similar queries which do not share n-grams but are semantically similar(e.g. "email template" and "e-mail templates"), and dissimilar queries which share a large proportion of n-grams(e.g. "how to calculate gpa" and "how to calculate grades"). Pearson coefficient works very well for common queries but poorly on rare queries due to the sparsity of click-through data. On the other hand, cosine similarity can only calculate similarities between two queries based on overlaps of n-grams between them. In contrast, our method can take advantage of both cosine similarity and Pearson coefficient and thus can perform better.

5.2.2 Quantitative Evaluation

We quantitatively compared our approach with the two baselines, i.e., cosine similarity on n-grams and Pearson correlation coefficient on click-through data, in terms of P@N. We use the number of clicks to divide the queries in the click-through dataset into common queries and rare queries. Without loss of generality, we choose 30 as the threshold in this work, i.e., queries that have been clicked more than 30 times were regarded as common queries and the rest are viewed as rare queries. We randomly sampled 250 common queries and 250 rare queries from the dataset, and then employed the three similarity calculation methods respectively to retrieve top 3 similar queries for both common queries and rare queries.

Table 2: Examples of Similar n-grams

n-gram A	n-gram B	Similarity Score
Spelling Errors		
cannon	canon	0.7226
stimulus	stimulas	0.6891
chrome	crome	0.6859
protein	protien	0.6576
Abbreviations		
ga	georgia	0.6496
mn	minnesota	0.6581
vw	volkswagen	0.6744
bio	biography	0.6259
Stemming		
resumes	resume	0.7869
design	designs	0.6533
ferry	ferries	0.7087
kit	kits	0.7371
Synonyms		
mother's	mothers	0.7683
chevy	chevrolet	0.6797
return	refund	0.5902
macy's	macys	0.5159
Numbers		
1st	first	0.5039
2nd	second	0.6425
3rd	third	0.4292
4th	fourth	0.5386

We manually labeled the similar queries for each method, following the same guideline on query similarity aforementioned. The evaluation results are given in Fig. 6. We can see that, for common queries, Pearson correlation method performs the best, which confirms the usefulness of click-through data on similarity calculation on common queries. Moreover, the precision of our similarity function, which is learned using training data derived from Pearson coefficient method, is slightly lower than that of Pearson coefficient and is noticeably higher than that of cosine similarity method on common queries. The results indicate that our method can learn the dependencies of n-grams from click-through data and thus outperforms cosine similarity method. However, it is not as effective as using click-through data directly for common queries.

For rare queries, our method constantly outperforms the two baselines, and t-test indicates that all the improvements are statistically significant ($p < 0.01$). Moreover, since there is insufficient click-through data for rare queries, the precision of Pearson coefficient method drops a lot compared with that on common queries, and becomes the lowest among the three methods. Also, it cannot find any similar queries for about 85% queries in the rare query set because that these queries do not share any co-clicked urls with others. The results imply that click-through data based methods are not suitable for rare query similarity calculation. On the other hand, our method still outperforms cosine similarity method on rare queries, which indicates that n-gram dependencies learned from common query pairs in our method are useful in rare query similarity calculation.

We also analyzed the query pairs which were not judged as similar by humans but similar by our method. The major type of errors by our method was due to limitation of infor-

Table 3: Examples of Similar and Dissimilar Query Pairs

Query A	*Query B*	*Freq. A*[1]	*Freq. B*[1]	*Sim.*[3]	*Cos.* [4]	*P. Coef.*[5]
Similar Query Pair						
email template	e-mail templates	194	101	0.7961	**0.0000**	0.8153
stokes county north carolina	stokes county	180	697	0.9891	0.4607	0.9524
comal county isd	comal independent school district	189	517	0.9663	0.4441	0.8393
nutrition values	nutritional value	343	477	0.7012	**0.0000**	0.9015
runescape mining map	runescape mining maps	10	59	0.9963	0.7772	**0.0000**
las cruces coupons	city of las cruces coupons	23	0	0.9926	0.7364	**0.0000**
Dissimilar Query Pair						
Query A	*Query B*	*Freq. A*	*Freq. B*	*Sim.*	*Cos.*	*P. Coef.*
state of florida laws	state of florida unclaimed money	102	158	0.0498	0.6877	0.1931
fishing boats for sale	large boats for sale	1939	142	0.1036	**0.7008**	0.0261
how to calculate gpa	how to calculate grades	1706	201	0.1515	**0.9064**	0.0908
google	google gadget	1.22×10^8	557	0.0475	0.5495	0.0301
recording programs	recordpad	66	30	0.1073	**0**	**0.7984**
nike 6.0 store	nike college apparel	162	24	0.0673	**0.1827**	**0.8091**

[1]click frequency of query A; [2]click frequency of query B; [3]our similarity score; [4]cosine similarity score on n-grams; [5]Pearson coefficient.

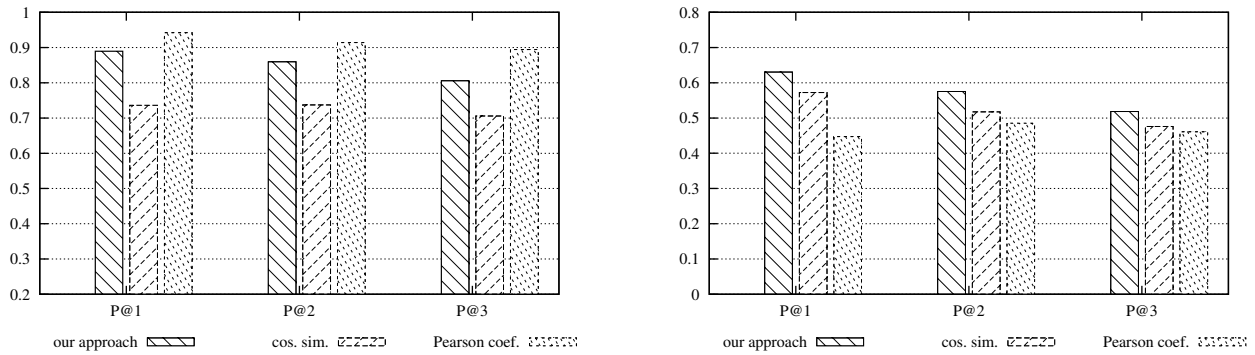

Figure 6: Precision of similar query calculation methods on (a) common query and (b) rare query

mation from either n-grams or supervision information. For example, for query "waterloo time", our model gave a wrong similar query "waterloo newspaper". The bigrams "waterloo time" and "waterloo newspaper" were not included in our model. Our model could only use unigrams "time" and "newspaper" to calculate the similarity without considering the context "waterloo", and gave the incorrect result.

5.3 Efficiency of Similar Query Search

We evaluated the efficiency of our method of similar query retrieval. We sampled some queries from the click-through data, removed the queries that could not be handled by the top 100k n-grams, and obtained in total 18.6 million queries. We indexed all of them to conduct similarity search. We randomly sampled 1,000 queries from them as *query points*.

Without an index of queries, the straightforward way for finding similar queries is to conduct linear scan, and thus we consider linear scan as baseline.

LSH offers approximated search, not exact search (there is always trade-off between accuracy and efficiency). In contrast, linear scan can give the exact results at the cost of time. In our method, we guarantee the precision is always 100% by filtering out the candidates whose similarity scores are lower than the given thresholds. Table 4 shows the recall of our method with different similarity thresholds. Table 4 also compares the time costs of our method and linear scan. We can see that our method is about 100 times faster than linear scan with an acceptable loss in recall.

Table 4: Performance of Indexing Scheme

Method		Time	Recall
Our Method	Indexing	668 minutes	
	Retrieval ($\geq$0.95)*	0.545 sec/query	98.59%
	Retrieval ($\geq$0.90)*	0.542 sec/query	92.11%
	Retrieval ($\geq$0.85)*	0.539 sec/query	85.97%
Linear Scan		43.2 sec/query	100%

* find all similar queries within a given radius.

5.4 Ranking Using Similar Queries

Query similarity calculation is useful for various tasks in search. As example, we applied our similarity function to relevance ranking. The idea is quite simple: if two queries are similar, we can use the clicks of one query to improve the relevance of the other query.

Give query q and document d, a simple ranking model can be a linear combination of the BM25 score between q and d, denoted as $bm25(q, d)$, and the PageRank score of d, denoted as $pr(d)$. We can add the number of clicks on

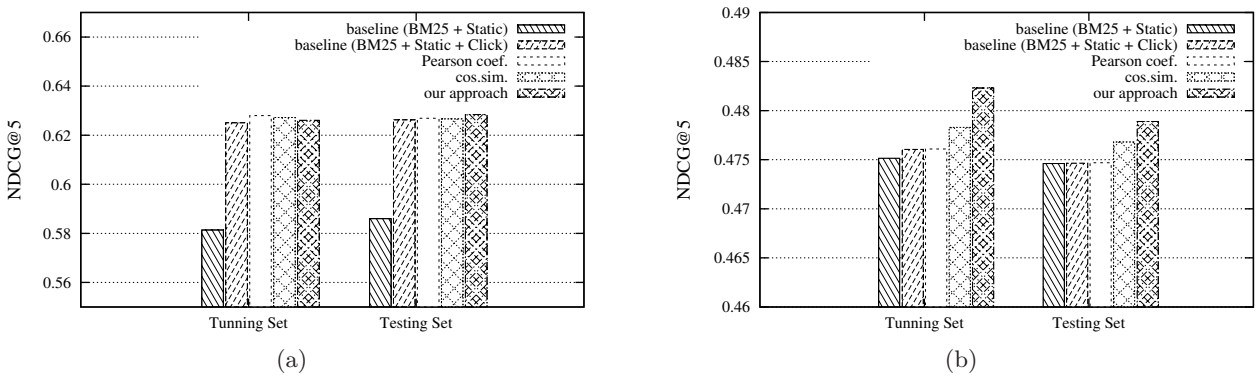

Figure 7: NDCG@5 of ranking functions on (a) common query and (b) rare query

document d by query q in the click-through data as another term of the linear combination ranking model:

$$r(q,d) = \alpha \cdot bm25(q,d) + \beta \cdot pr(d) + (1-\alpha-\beta) \cdot \log clk(q,d) \quad (9)$$

where $clk(q,d)$ denotes the number of clicks on d by q. We view both of them as baseline methods for relevance ranking.

We next consider a new method by incorporating similar queries into ranking. Specifically, we use the model

$$r(q,d) = \alpha \cdot bm25(q,d) + \beta \cdot pr(d) + \gamma \cdot \log clk(q,d) + (1-\alpha-\beta-\gamma) \cdot \sum_{q' \in S(q)} sim(q',q) \log clk(q',d) \quad (10)$$

where $S(q)$ is the set of queries similar to q and $sim(q',q)$ is the similarity score between queries q' and q. The rationale behind the model is that even if we do not have enough clicks by q and d, we can use the clicks of its similar queries to improve search relevance.

We used a search ranking data from the search engine (different from the click-through data), which contains about 13,000 queries and associated web pages. We classified the queries as common queries and rare queries with the same threshold on query similarity calculation aforementioned, which led to about 5,000 common queries and 8,000 rare queries. Moreover, for both common query set and rare query set, queries were split into two parts: 70% for parameter tuning and 30% for testing. The relevance of the web pages with respect to the queries are judged on a 5-point scale. There are on average 44 judged web pages for each query.

We employed the ranking function (10) to rank all the web pages in the test data after tuning the parameters (α, β, γ) using the training data. $S(q)$ was composed of the top 30 most similar queries to q. In addition to the proposed method, we also used the two baselines: cosine similarity on n-grams and Pearson coefficient on click-through data. The experimental results are shown in Fig. 7.

For common queries, as shown in Fig.7(a), the clicks of original query $clk(q,d)$ significantly improve the performance of ranking, which verifies the usefulness of click-through data in ranking. However, leveraging the clicks from similar queries $clk(q,d)$ just brings little additional gains in ranking, no matter what kinds of similarity calculation method are used. We think it is because that, for common queries, the clicks of the original query are sufficient to describe the relevances of clicked documents with respect to the query. Similar queries share similar click distributions over the documents with the original query, and thus adding similar queries' clicks has little effect on the performance of ranking.

In contrast, in the case of rare queries(Fig.7(b)), clicks of the original query contribute little to the performance of ranking as there is insufficient click-through data on rare queries. On the other hand, clicks of similar queries can improve ranking performance. Specifically, the improvement with our method is the largest and that of Pearson coefficient is the smallest. The order is consistent with that of similar query accuracy on rare queries. Furthermore, t-test results indicate that all the improvements of our method against other methods are statistically significant ($p < 0.01$). The results imply that similar queries' clicks can enrich the clicks of rare queries, and so leveraging similar queries' clicks will notably improve ranking performance. Moreover, more accurate similar query calculation method will leads to larger improvement in ranking.

6. CONCLUSIONS

In this paper, we have studied the problem of learning similarity function for rare queries. We define an n-gram space to represent queries and a similarity function to measure similarities between queries. By leveraging training data derived from click-through data, we propose to employ a metric learning method to automatically learn the similarity function. Moreover, locality sensitive hash is employed for indexing queries and fast retrieving similar queries in the n-gram space.

Experimental results on a large scale web search data show that our method is superior to the baseline methods of cosine similarity using n-gram data and Pearson coefficient using click-through data on rare queries. It is because that our method can effectively learn n-gram dependencies from common query pairs derived from click-through data and then apply them into query similarity calculation, especially for rare queries. Furthermore, the quality of learned similar query pairs is high, so is the efficiency of similar query search. The experimental results also demonstrate that the use of similar queries in search can improve the search relevance.

Our work in this paper is motivated by query processing, but the proposed methodology should not be limited to query processing. It is potentially applicable to other prob-

lems such as query-document matching and collaborative filtering. In query-document matching in search, the relevance between query and document can be represented by a similarity function in the n-gram space. Moreover, leveraging similar queries into query-document matching is another direction one can further explore.

7. REFERENCES

[1] B. Bai, J. Weston, D. Grangier, R. Collobert, K. Sadamasa, Y. Qi, O. Chapelle, and K. Weinberger. Supervised semantic indexing. In *CIKM '09*, pages 187–196, New York, NY, USA, 2009.

[2] E. Balfe and B. Smyth. An analysis of query similarity in collaborative web search. In *BCIR 2005*, pages 330–344, March 2005.

[3] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *KDD '00*, pages 407–416, New York, NY, USA, 2000.

[4] A. Broder, P. Ciccolo, E. Gabrilovich, V. Josifovski, D. Metzler, L. Riedel, and J. Yuan. Online expansion of rare queries for sponsored search. In *WWW '09*, pages 511–520, 2009.

[5] A. Z. Broder, M. Fontoura, E. Gabrilovich, A. Joshi, V. Josifovski, and T. Zhang. Robust classification of rare queries using web knowledge. In *SIGIR '07*, pages 231–238, 2007.

[6] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC '02*, pages 380–388, New York, NY, USA, 2002.

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

[8] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG '04*, pages 253–262, New York, NY, USA, 2004.

[9] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon. Information-theoretic metric learning. In *ICML '07*, pages 209–216, New York, NY, USA, 2007.

[10] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[11] J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostat*, 9(3):432–441, July 2008.

[12] A. Globerson and S. Roweis. Metric learning by collapsing classes. volume 18, pages 451–458, 2006.

[13] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC '98*, pages 604–613, New York, NY, USA, 1998.

[14] P. Jain, B. Kulis, I. Dhillon, and K. Grauman. Online metric learning and fast similarity search. pages 761–768, 2008.

[15] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *WWW '06*, pages 387–396, New York, NY, USA, 2006.

[16] B. Kulis, M. Sustik, and I. Dhillon. Learning low-rank kernel matrices. In *ICML '06*, pages 505–512, New York, NY, USA, 2006.

[17] M. Li, Y. Zhang, M. Zhu, and M. Zhou. Exploring distributional similarity based models for query spelling correction. In *ACL-44*, pages 1025–1032, Morristown, NJ, USA, 2006.

[18] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *J. Mach. Learn. Res.*, 2:419–444, 2002.

[19] H. Ma, H. Yang, I. King, and M. R. Lyu. Learning latent semantic relations from clickthrough data for query suggestion. In *CIKM '08*, pages 709–718, New York, NY, USA, 2008.

[20] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *CIKM '08*, pages 469–478, New York, NY, USA, 2008.

[21] D. Metzler, S. Dumais, and C. Meek. Similarity measures for short segments of text. In *ECIR '07*, pages 17–27, 2007.

[22] G.-J. Qi, J. Tang, Z.-J. Zha, T.-S. Chua, and H.-J. Zhang. An efficient sparse metric learning in high-dimensional space via l1-penalized log-determinant regularization. In *ICML '09*, pages 841–848, New York, NY, USA, 2009.

[23] M. Sahami and T. D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *WWW '06*, pages 377–386, 2006.

[24] M. Sahami and T. D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *WWW '06*, pages 377–386, New York, NY, USA, 2006.

[25] Y. Song and L.-w. He. Optimal rare query suggestion with implicit user feedback. In *WWW '10*, pages 901–910, 2010.

[26] E. M. Voorhees. Query expansion using lexical-semantic relations. In *SIGIR '94*, pages 61–69, New York, NY, USA, 1994.

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

[28] K. Weinberger, J. Blitzer, and L. Saul. Distance metric learning for large margin nearest neighbor classification. volume 18, pages 1473–1480, 2006.

[29] K. Q. Weinberger and L. K. Saul. Fast solvers and efficient implementations for distance metric learning. In *ICML '08*, pages 1160–1167, New York, NY, USA, 2008.

[30] J.-R. Wen, J.-Y. Nie, and H.-J. Zhang. Clustering user queries of a search engine. In *WWW '01*, pages 162–168, New York, NY, USA, 2001.

[31] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. In *NIPS 15*, volume 15, pages 505–512, 2002.

[32] J. Xu and W. B. Croft. Query expansion using local and global document analysis. In *SIGIR '96*, pages 4–11, New York, NY, USA, 1996.

[33] W.-t. Yih. Learning term-weighting functions for similarity measures. In *EMNLP '09*, pages 793–802, 2009.

[34] Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. In *WWW '06*, pages 1039–1040, New York, NY, USA, 2006.