Fast Query Recommendation by Search

Qixia Jiang and Maosong Sun
State Key Laboratory on Intelligent Technology and Systems
Tsinghua National Laboratory for Information Science and Technology
Department of Computer Sci. and Tech., Tsinghua University, Beijing 100084, China
qixia.jiang@gmail.com, sms@tsinghua.edu.cn

Abstract

Query recommendation can not only effectively facilitate users to obtain their desired information but also increase ads' click-through rates. This paper presents a general and highly efficient method for query recommendation. Given query sessions, we automatically generate many similar and dissimilar query-pairs as the prior knowledge. Then we learn a transformation from the prior knowledge to move similar queries closer such that similar queries tend to have similar hash values. This is formulated as minimizing the empirical error on the prior knowledge while maximizing the gap between the data and some partition hyperplanes randomly generated in advance. In the recommendation stage, we search queries that have similar hash values to the given query, rank the found queries and return the top K queries as the recommendation result. All the experimental results demonstrate that our method achieves encouraging results in terms of efficiency and recommendation performance.

Introduction

Query Recommendation (QR) plays an important role in IR and advertising. QR suggests some related queries to users in their search process, which can effectively facilitate users to clarify their information needs and obtain their desired results. In advertising, advertisers always bid on few terms that tend to have high search volumes. Such "hot" bidterms are always expensive. Alternatively, we could recommend some related, much cheaper but relatively low search volume bidterms to advertisers for achieving comparable advertising effects (Abhishek and Hosanagar 2007). Moreover, the scarcity usually leads to the difficulty of ad matching. Therefore suggesting some additional bidterms can significantly increase ads' click-through rates (Chang et al. 2009).

Figure 1 presents the process of QR: (1) query representation, (2) finding the queries that are similar to the current query, (3) ranking the found queries, and (4) post-processing based on the ranked query list such as query expansion etc. Existing QR methods differ from each other in terms of these steps. We divide them into two major technologies: context-free and context-aware. Context-free methods (Liu and Sun 2008; Ma, Lyu, and King 2010) only use linkage information (i.e., query-URLs) and they are usually carried out on bipartite networks constructed from user logs. Such methods suffer from the data scarcity thus are inappropriate for unpopular queries. Context-aware methods recommend queries by incorporating various search context information such as query terms, clicked URLs, query sessions, Web pages, etc (Cao et al. 2008; Jiang, Zilles, and Holte 2009; He et al. 2009). However, in these works, different query representations always lead to the different recommendation methods. So, a general framework is required to adopt various types of query information for QR.

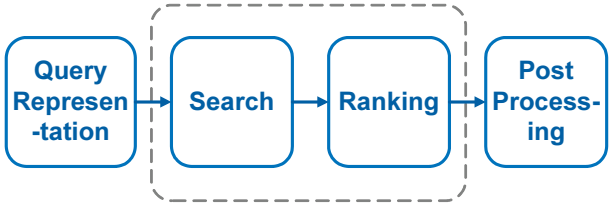


Figure 1: The process of query recommendation. This paper focuses on the part in the dashed box.

Inspired by (Yang et al. 2008; Cai et al. 2007), a direct solution is to convert the recommendation problem to a scalable search problem – recommendation-by-search. Specifically, each query is represented as a vector which consists of various types of information such as terms, clicked URLs, etc. Then, it turns to a similarity search problem. However, real-world query data is very large, which requires highly-efficient retrieval methods. This paper, as described in Figure 1, focuses on how to efficiently recommend queries by search. We should emphasize that it is obviously that enriching the query representation can improve the recommendation performance but it is not in the scope of this paper.

Hashing-based methods have gained great success in various applications such as image retrieval (Wang, Kumar, and Chang 2010), music recommendation (Cai et al. 2007), near-duplicate detection (Manku, Jain, and Das Sarma 2007), etc. They map high-dimensional objects to compact binary hash values and make similar objects have similar hash values. The similarity search according to these hash values is much more efficient than in the original attribute space (Charikar

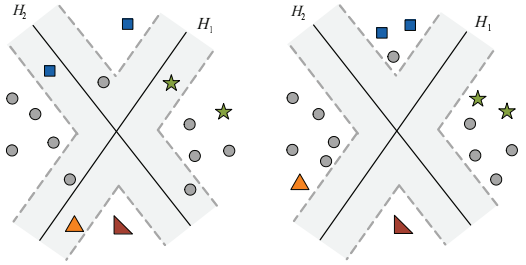


Figure 2: An illustration of Query Hashing. In this example, two similar pairs are denoted by stars and squared boxes respectively, and two dissimilar points are denoted by two different triangles (equilateral and right-angle). The prior knowledge guides us to hash queries more reasonable, while maximizing the gap between the data and the partition hyperplanes makes our method have better generalization.

2002). Inspired by this, we propose a novel method, Query Hashing (QH), for QR. Given query sessions, we generate the prior knowledge of queries in terms of pairwise similarity and dissimilarity via Hierarchical Agglomerative Clustering (HAC) (Zamir et al. 1997). Then we learn a transformation to move similar queries closer such that similar queries tend to have similar hash values. This is formulated as minimizing the empirical error on the prior knowledge while maximizing the gap between the data and some hyperplanes randomly generated in advance (see Figure 2). To hash a query, we firstly move it based on the learned transformation. Its hash value is then determined by the sides of hyperplanes it lies on. Recommendation for a query consists of two steps: (1) searching queries that have similar hash values to the given query, and (2) ranking the found queries and returning the top K queries as the result.

To evaluate the performance, we collect 12.5 million queries from a commercial search engine, *Sogou*[1]. We compare QH with several methods. All the experiments demonstrate that our method achieves encouraging results in terms of recommendation performance and efficiency.

Query Hashing

Assume we are given a set of queries $\mathcal{X} = \{\mathbf{x}_n\}_{n=1}^N, \mathbf{x}_n \in \mathbb{R}^D$ and a set of query sessions. A query session is a sequence of queries submitted by a specific user to a search engine in a certain period of time. Moreover we define two sets, similarity set Θ_s and dissimilarity set Θ_d. Any query-pair in Θ_s denotes that they should have the same hash value, while any query-pair in Θ_d indicates that they should have different ones. QH is to automatically generate the prior knowledge Θ_s and Θ_d and then learns a transformation matrix $\mathbf{T} \in \mathbb{R}^{D\times D}$ from the prior knowledge such that after transformation similar queries tend to have similar hash values.

Prior Knowledge Generation

This section describes the method of generating the prior knowledge Θ_s and Θ_d from query sessions. To reduce the influence of the noisy within query sessions, we use the HAC algorithm to cluster the queries. The basic HAC is a bottom-up clustering algorithm. It needs to calculate the adjacency matrix in advance and dynamically update this matrix, which is time consuming and computationally intractable for large-scale data. Hence, some heuristic rules are introduced to reduce the computation complexity.

Adjacency Matrix Calculation. The queries in a same query session are always strongly semantic-related, since they are issued by a user for a certain information need. Inspired by this observation, we only calculate the cosine similarity between any two queries that occur in the same session, while the similarity between any two queries that never occur in a session together is set to 0. Finally, we sort the query pairs in terms of their similarity and obtain a ranked query-pair list.

Clustering Algorithm. The clustering algorithm is a recursive procedure. At beginning, each query is considered as a singleton cluster. In each iteration, we select the query-pair with the highest similarity and remove it from the query-pair list. Then, we test whether the two clusters containing the selected queries can be merged. If they do not satisfy the merge conditions, go to the next iteration. The algorithm stops when (1) the number of obtained clusters is smaller than a desired number, or (2) no clusters can be further merged. There are two merge conditions: (1) the size of the after-merged cluster should be smaller than a threshold. This threshold should be set relatively small, i.e., 50. This is because that HACs are very sensitive to merge conditions (Zamir et al. 1997). When two "good" clusters are merged by mistake, the resulting cluster may be meaningless to users. (2) the similarity between two clusters should be larger than a threshold. The cluster similarity is measured as between two "super queries". Specifically, each cluster is represented by simply concatenating all the queries that it contains. And the similarity between two clusters is just the similarity between such two "super queries". Note that we never update the adjacency matrix during the clustering procedure which can greatly speed up clustering.

Generating Θ_s and Θ_d. After clustering, we randomly sample some pairs of queries within the same clusters as Θ_s and some queries within two different clusters as Θ_d.

Formulation

This section presents our hashing-based QR method. Hashing aims to map data to a Hamming space[2] for compact representation. For a query $\mathbf{x}$, QH firstly moves it via $\mathbf{T}$, then its hash value is determined by the sides of L random hyperplanes $\mathbf{h}_l, l = 1, \ldots, L$, that it lies on, i.e.,

$$f_l(\mathbf{x}) = \text{sign}(\mathbf{h}_l^T \mathbf{T}\mathbf{x}). \quad (1)$$

Intuitively, we hope that Θ_s and Θ_d could guide us to hash queries more reasonable – after transformation similar queries could be mapped into the same or adjacent hash buckets. Furthermore, it is reasonable that similar query-pairs are not only to share the same hash value but also to

[1] www.sogou.com

[2] Hamming space is a set of binary strings of length L.

have large projection magnitudes. If they are projected to different hash buckets, we toggle an extra penalty. Analogously, after transformation, two dissimilar queries are expected not only to be assigned to different hash values but also to have large margins. We toggle an extra penalty if two dissimilar queries are mapped into the same hash bucket. Therefore, the empirical error on the obtained prior knowledge Θ_s and Θ_d is:

$$\begin{aligned}\mathcal{L}(\mathbf{T}) &= \sum_{l=1}^{L}\Big\{ \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\Theta_s} \vartheta[(\mathbf{h}_l^T\mathbf{T}\mathbf{x}_i)(\mathbf{h}_l^T\mathbf{T}\mathbf{x}_j)] \\ &\quad + \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\Theta_d} \vartheta[-(\mathbf{h}_l^T\mathbf{T}\mathbf{x}_i)(\mathbf{h}_l^T\mathbf{T}\mathbf{x}_j)]\Big\}, \quad (2)\end{aligned}$$

where $\vartheta(t)$ is defined as follows:

$$\vartheta(t) = \begin{cases} 0, & if\ t \geq 0 \\ -t, & otherwise \end{cases} \quad (3)$$

Moreover, we also expect that our method could have a good generalization property. Inspired by margin-oriented criterion (Burges 1998), we hope that the gap between the data and the partition hyperplanes is large enough. Therefore we utilize a hinge-loss to penalize those queries that are too close to the hyperplanes. For a query $\mathbf{x}$ and the hyperplane $\mathbf{h}_l$, the hinge-loss function is defined as:

$$\ell_l(\mathbf{x}) = \max\{0, 1 - |\mathbf{h}_l^T\mathbf{T}\mathbf{x}|\}. \quad (4)$$

Add an additional F-norm regularization and finally we should minimize the following objective function:

$$\begin{aligned}\mathcal{L}(\mathbf{T}) &= \frac{1}{2}\|\mathbf{T}\|_{\mathrm{F}}^2 + \frac{1}{N}\sum_{l=1}^{L}\Big\{\lambda_1 \sum_{n=1}^{N}\ell_l(\mathbf{x}_n) \\ &\quad +\lambda_2 \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\Theta_s} \vartheta[(\mathbf{h}_l^T\mathbf{T}\mathbf{x}_i)(\mathbf{h}_l^T\mathbf{T}\mathbf{x}_j)] \\ &\quad +\lambda_2 \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\Theta_d} \vartheta[-(\mathbf{h}_l^T\mathbf{T}\mathbf{x}_i)(\mathbf{h}_l^T\mathbf{T}\mathbf{x}_j)]\Big\}, \quad (5)\end{aligned}$$

where λ_1 and λ_2 are trade-offs which weight loss terms.

For high-dimensional data, it is computationally expensive to determine the optimal $\mathbf{T}$ ($D \times D$ parameters). Thus, we assume that the features are mutually independent. As a result, $\mathbf{T}$ is simplified as a diagonal matrix, i.e., $\mathbf{T} = \mathrm{diag}(\mathbf{t})$, where $\mathbf{t} = (t_1, \ldots, t_D)$.

Parameter Estimation

We determine the model parameters $\mathbf{T}$ via minimizing the Equation (5). For description simplicity, only similarity set Θ_s is considered[3]. Differentiating the loss function $\mathcal{L}(\mathbf{T})$ with respect to the parameter t_d gives

$$\begin{aligned}\frac{\partial\mathcal{L}(\mathbf{T})}{\partial t_d} &= t_d - \frac{1}{N}\sum_{l=1}^{L}\Big\{\lambda_1\sum_{n=1}^{N}\mathbf{h}_{l,d}\mathbf{x}_{n,d}\cdot\mathrm{sign}(\mathbf{h}_l^T\mathbf{T}\mathbf{x}_n) \\ &+ \lambda_2 \sum_{\substack{(\mathbf{x}_i,\mathbf{x}_j)\in\Theta_s,\\ \vartheta((\mathbf{h}_l^T\mathbf{T}\mathbf{x}_i)(\mathbf{h}_l^T\mathbf{T}\mathbf{x}_j))>0}} \Big[(\mathbf{h}_l^T\mathbf{T}\mathbf{x}_i)\cdot(\mathbf{h}_{l,d}\mathbf{x}_{j,d}) \\ &+ (\mathbf{h}_l^T\mathbf{T}\mathbf{x}_j)\cdot(\mathbf{h}_{l,d}\mathbf{x}_{i,d})\Big]\Big\}. \quad (6)\end{aligned}$$

[3]The extension to dissimilarity set Θ_d is straightforward.

From above equation, we can see that it is impossible to analytically determine the optimal parameters $\mathbf{T}$, which is mainly because simply setting the gradient of $\mathcal{L}(\mathbf{T})$ to zero fails to yield a closed form solution. Instead, we solve this optimization problem via an effective Quasi-Newton algorithm, L-BFGS (Liu and Nocedal 1989).

Optimization with Jenkins Hashing Function

Unfortunately, explicitly representing one high-dimensional random hyperplane still requires lots of random bits. Inspired by (Indyk 2006), for N queries, we use *Jenkins hash function*[4] to pick $O(\log_2 N)$ random bits such that we can restrict the random hyperplanes in a family of size $2^{O(\log_2 N)}$. Specifically, we maintain a $D\times L$ matrix $\mathbf{H}$. Each feature is firstly hashed into an L-dimensional hash value $\mathbf{f} \in \{-1, +1\}^L$ via Jenkins hash function. Then, the corresponding row of $\mathbf{H}$ is set to $\mathbf{f}^T$. After all features have been processed, each column $\mathbf{h}_l \in \{-1, +1\}^D$, $l = 1, \ldots, L$, of $\mathbf{H}$ indicates one random hyperplane. Since each query is represented by a very sparse vector, such a procedure can be carried out online. As a result, we can avoid explicitly storing these L random hyperplanes and the space complexity can be reduced from $O(LD + D)$ to $O(D)$.

Summery

In this section, we briefly summarize our method. QH consists of two strategies, pre-processing and recommendation. The pre-processing strategy involves following three steps:

- Generate the similarity and dissimilarity sets Θ_s and Θ_d.
- Determine the optimal $\mathbf{T}^*$ by minimizing Equation (5).
- For each query $\mathbf{x} \in \mathcal{X}$, move it by multiplying the leaned transformation matrix $\mathbf{T}^*$ and generate its L-bit hash value, i.e., $f_l(\mathbf{x}) = \mathrm{sign}(\mathbf{h}_l^T\mathbf{T}^*\mathbf{x})$, $l = 1, \ldots, L$.

In the recommendation stage, given a query $\mathbf{q}$, we need:

- Form the hash value of $\mathbf{q}$ according to Equation (1).
- Search the queries in the dataset $\mathcal{X}$ that have the similar hash values to the query $\mathbf{q}$'s.
- Rank the search results and return the top K queries as the recommendation result.

Experiment

This section evaluates our method. We firstly introduce the experimental setups. Then some evaluation metrics are presented. Finally, we report the results.

Experimental Setups

Our evaluations consist of two parts: (1) data without labels. This evaluation is carried out on a large query dataset collected from a commercial search engine. (2) data with labels. To exclude the influence of HAC to the final recommendation performance, we manually label some queries, train the QH model and evaluate the performance.

We compare our method with three state-of-the-art hashing-based methods, SimHash (SH) (Charikar 2002),

[4]http://www.burtleburtle.net/bob/hash/doobs.html

(a) (b)

Figure 3: Run-time complexity of different methods. (a)Averaged searched data within Hamming radius 2 with different number of hash bits. (b)Averaged running time for recommendation per 1,000 queries (sec./1000q).

(a) (b)

Figure 4: Recommendation performance for hash ranking on the data without labels. (a) MAP for hash ranking with different number of hash bits. (b) Precision at rank R (P@R) with 30-bit hash value.

Kernelized Locality Sensitive Hashing (KLSH) (Kulis and Grauman 2009) and Semi-Supervised Hashing (SSH) (Wang, Kumar, and Chang 2010), and a method commonly used in IR, Inverted-List (IL) (Manning, Raghavan, and Schütze).

SH uses random projections to hash data. KLSH constructs a random hyperplane as a weighted sum of some randomly-selected points. For a query, the hash value is determined by its relative location to these constructed hyperplanes. In our experiment, we use the RBF kernel $\exp(-\frac{\|\mathbf{x}_i-\mathbf{x}_j|_2^2}{\delta^2})$ where the scaling factor δ takes 0.5. The parameters in KLSH are set as: $p = 500$ and $t = 50$. See Ref.(Kulis and Grauman 2009) for more details. Both SH and KLSH are unsupervised method. SSH can hash with prior knowledge. It determines the optimal hashing functions by finding the top L eigenvectors of an extended covariance matrix via Principle Component Analysis (PCA) (Jolliffe 1986). For QH, we simply set the tradeoff coefficients λ_1 and λ_2 to 1,000 and 100 respectively.

We test all the methods on a PC with a 2.66 GHz processor and 12GB RAM. All experiments are repeated 10 times and the averaged results are reported.

Evaluation Metrics

Given a test query, we firstly generate its hash value using different methods. Then, similarity search is performed to recommend some queries. Note that, IL in fact returns all the queries that contain any term(s) within the given query. As in the literature, quality measurements under two different scenarios are utilized in this paper.

Hash Lookup. Given a test query, we search queries in the dataset within a specified Hamming radius r ($r = 2$ in our experiments) as recommendation, i.e., lookup by successively altering i bits, $i = 0, 1, \ldots, r$. The proportion of relevance queries (i.e., on the same topic as the test query) among the search results is calculated as recommendation precision. Similarly to (Wang, Kumar, and Chang 2010; Weiss, Torralba, and Fergus 2009), for a test query, if no neighbors within the given Hamming radius can be found, it is considered as zero precision.

Hash Ranking. Given a test query $\mathbf{q}$, we firstly lookup queries (just as hash lookup) in the dataset within the Hamming radius from 0 to L until no less than K (100 in our experiments) queries are collected. Then, we rank the found queries according to their Hamming distance[5] from $\mathbf{q}$. Finally, we return the top K queries as the result. The recommendation precision at rank R (P@R) is calculated as:

$$\text{P@}R = \frac{\text{the number of searched relevent queries}}{R}. \quad (7)$$

A query considered as a relevant one is either on the same topic as the test query or the true K-nearest-neighbors of the test query in the original attribute space. We also calculate Mean Averaged Precision (MAP) as follows:

$$\text{MAP}(\mathcal{Q}) = \frac{1}{|\mathcal{Q}|}\sum_{\mathbf{q}\in\mathcal{Q}}\frac{1}{K}\sum_{R=1}^{K}\text{P@}R\cdot\delta(\mathbf{q},R), \quad (8)$$

where $\mathcal{Q}$ is the test query set and $\delta(\mathbf{q}, R) = 1$ when the searched query at rank R is relevant to $\mathbf{q}$, otherwise 0.

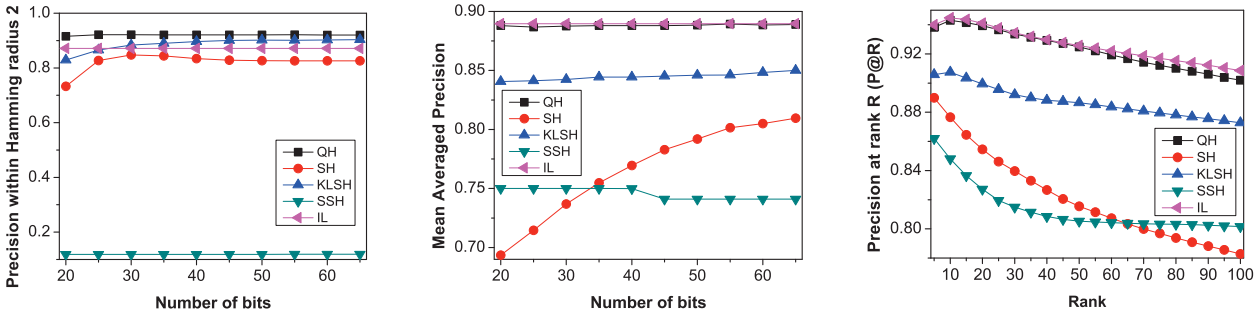
Data without Labels

We collect 4,039,322 query sessions and 12,502,641 Chinese queries from a commercial search engine. After word segmentation, this dataset contains totally 69,738,014 terms and 256,141 distinct features. Each query is represented as a vector of term frequency[6]. We randomly sample 20K queries as test set and the rest as training set. Both obtained Θ_s and Θ_d contain 5M query-pairs. Since no topic information is available in this section, a query is considered to be relevant to the test query when it is the true K-nearest-neighbors found by IL method. Moreover, considering there are 256K features, SSH totally fails from an effectiveness perspective. Thus, we only compare our method with SH, KLSH and IL.

Figure 3(a) shows the averaged searched data within Hamming radius 2. Obviously, KLSH is lack of discrimination and returns over 10 times more queries than IL. This

[5]Hamming distance is defined as the number of bits that are different between two binary strings.

[6]Note that QH is a general framework for QR. In fact, we can adopt any additional features in this framework as we like. For example, we can deal with the lexical gap problem by employing semantic features. But enriching query representation is not in the scope of this paper.



(a) (b) (c)

Figure 5: Recommendation performance on the data with labels. (a) Precision within Hamming radius 2 for hash lookup with different number of hash bits. (b) MAP for hash ranking with different number of hash bits. (c) Precision at rank R (P@R, $R = 1, \ldots, 100$) for hash ranking with 30-bit hash value.

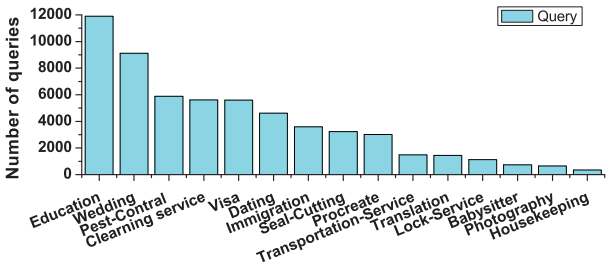


Figure 6: An illustration of the biggest 15 topics in the data.

is mainly because KLSH reconstructs the embedding space via only a few sample points, which is usually insufficient for high-dimensional especially sparse data thus lack of discrimination. As a result, KLSH is degraded to the linear scan method to a certain extent. We also find that IL returns much more queries than QH and SH, which leads to its poor efficiency for further post-processing.

Figure 3(b) presents the running time for recommendation. It shows that QH is much faster than the other methods. We find that the efficiency of SH significantly decreases with the increasing of hash bits. This is because when hash bits are long, SH always fails to return any neighbor even in a large Hamming radius, which is a potential problem of the basic SH. Therefore, SH has to linearly scan the whole dataset for recommendation. On the other hand, QH takes less than 30 minutes to determine the optimal transformation matrix $\mathbf{T}^*$.

Figure 4 reports the recommendation performance of different methods. Obviously, QH is superior to SH and comparable to KLSH (degraded to the linear scan). However, considering the efficiency, QH gets the best performance.

Data with Labels

This dataset contains 62,450 Chinese queries with totally 225,790 terms and 3,916 distinct features. Each query is represented as a vector of term frequency. We manually assign all these queries to one of 39 topics. The biggest topic contains 11,896 queries and the smallest one only contains 6. For space limitation, we only list the biggest 15 topics in Figure 6. Furthermore, we randomly sample 50K queries as training set and the rest 12,450 queries as test set. To train QH and SSH, we randomly sample 20K pairs of queries within the same topic as Θ_s and 20K pairs of queries within two different topics as Θ_d. In the following experiments, a query is considered to be relevant when it shares the same topic as the test query[7].

Recommendation performance of different methods is presented in Figure 5. Figure 5(a) shows the performance for hash lookup with different hash bits (L). It shows that SSH totally fails, which is mainly because SSH finds L optimal hash functions via PCA. However, the variance of directions obtained by PCA is mostly contained in the top few directions (Jolliffe 1986). Therefore, lower hash functions tend to have no discrimination. IL is inferior to QH and KLSH, which is because queries with different topics also share some features. Moreover, KLSH is superior to the basic SH. This benefits from the adoption of the kernel technique, which is proposed for nonlinear separated data. We can see that QH gets the best precision for hash lookup.

Figures 5(b) and 5(c) report the recommendation performance for hash ranking. IL ranks the queries according to their cosine similarity to the test query in the original attribute space. In fact, for hash ranking, MAP and P@R obtained by IL are the upper-bound of all the approximated similarity search methods. Such two figures show that the performance of QH is comparable to IL and is significantly superior to the other three hashing-based methods.

All the experimental results show that our method achieves encouraging results in terms of recommendation performance and efficiency.

[7]Previous works (Liu and Sun 2008; Yang et al. 2008) commonly evaluate the performance by asking editors whether the recommended queries are really retaliated to the original queries. Our experimental setup is analogous except for the procedure that editors label all the queries in advance and then we check whether the recommended queries are related. This maybe not sensitive enough but it is fair to all the methods.

Related Works

This section briefly introduces some related works on the search-based QR and the approximated similarity search.

Query Recommendation by Search

Jiang et al. (Jiang, Zilles, and Holte 2009) recommend for a query in the following two steps: (1) submitting the given query to a search engine, and (2) recommending by extracting some relevant queries from the documents in a specific rank range returned by the search engine. Yang et al. (Yang et al. 2008) enrich query representation by incorporating both search result context and query log session context. Given a query, they calculate the relevance scores for all the queries in the query set and recommend those queries have the highest relevance scores. The low efficiency of such two methods limit their usage in real-world applications.

Approximated Similarity Search

Existing approximate similarity search algorithms can be divided into two categories: unsupervised and supervised.

A notable unsupervised method is SimHash (SH) (Charikar 2002). SH performs random projections to map similar objects to similar hash codes. The kernel version of SH, Kernerlized Locality Sensitive Hashing (KLSH) (Kulis and Grauman 2009), is proposed to handle linearly-unseparated data. These methods cannot incorporate prior knowledge for better hashing.

Motivated by this, many supervised methods have been proposed. Spectral Hashing (Weiss, Torralba, and Fergus 2009) maintains similarity between objects in the reduced Hamming space by minimizing the average Hamming distance between similar neighbors in the Euclidean space. Unlike Spectral Hashing, Semi-Supervised Hashing (SSH) (Wang, Kumar, and Chang 2010) strive to learn hash functions that minimize the empirical error on prior knowledge. LAMP (Mu, Shen, and Yan 2010) enforces the consistency between hashing partitions and constraints implied in prior knowledge by adding a regularization term. All these supervised methods are computational expensive which limits their usage for high-dimensional data.

Conclusions

In this paper, we have proposed a general and efficient method for query recommendation, Query Hashing (QH). QH automatically generates the prior knowledge from query sessions. Then QH learns a transformation from the prior knowledge such that after transformation similar queries tend to have similar hash values. This is implemented by minimizing the empirical error on the prior knowledge while maximizing the gap between the data and some partition hyperplanes randomly generated in advance. All the experiments have shown that QH achieves the best results in terms of efficiency and recommendation performance.

Acknowledgements

This work is supported by the National Natural Science Foundation of China under Grant No. 60873174. We also thank *Sogou* for providing the data.

References

Abhishek, V., and Hosanagar, K. 2007. Keyword generation for search engine advertising using semantic similarity between terms. In *Proc. of ICEC'07*.

Burges, C. 1998. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery* 2(2):121–167.

Cai, R.; Zhang, C.; Zhang, L.; and Ma, W. 2007. Scalable music recommendation by search. In *Proc. of MM'07*.

Cao, H.; Jiang, D.; Pei, J.; He, Q.; Liao, Z.; E., C.; and Li, H. 2008. Context-aware query suggestion by mining click-through and session data. In *Proc. of KDD'08*.

Chang, W.; Pantel, P.; Popescu, A.; and Gabrilovich, E. 2009. Towards intent-driven bidterm suggestion. In *Proc. of WWW'09*.

Charikar, M. 2002. Similarity estimation techniques from rounding algorithms. In *Proc. of STOC'02*.

He, Q.; Jiang, D.; Liao, Z.; Hoi, S.; Chang, K.; Lim, E.; and Li, H. 2009. Web query recommendation via sequential query prediction. In *Proc. of ICDE'09*.

Indyk, P. 2006. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM (JACM)* 53(3):307–323.

Jiang, S.; Zilles, S.; and Holte, R. 2009. Query suggestion by query search: a new approach to user support in web search. *Proc. of WI'09*.

Jolliffe, I. 1986. Principal component analysis. *Springer-Verlag*.

Kulis, B., and Grauman, K. 2009. Kernelized locality-sensitive hashing for scalable image search. *Proc. of ICCV'09*.

Liu, D., and Nocedal, J. 1989. On the limited memory bfgs method for large scale optimization. *Mathematical programming* 45(1):503–528.

Liu, Z., and Sun, M. 2008. Asymmetrical query recommendation method based on bipartite network resource allocation. In *Proc. of WWW'08*.

Ma, H.; Lyu, M.; and King, I. 2010. Diversifying query suggestion results. *Proc. of AAAI'10*.

Manku, G.; Jain, A.; and Das Sarma, A. 2007. Detecting near-duplicates for web crawling. In *Proc. of WWW'07*.

Manning, C.; Raghavan, P.; and Schütze, H. An introduction to information retrieval.

Mu, Y.; Shen, J.; and Yan, S. 2010. Weakly-supervised hashing in kernel space. *Int. Conf. on CVPR'10*.

Wang, J.; Kumar, S.; and Chang, S. 2010. Semi-supervised hashing for scalable image retrieval. *Int. Conf. on CVPR'10*.

Weiss, Y.; Torralba, A.; and Fergus, R. 2009. Spectral hashing. *Proc. of NIPS'09*.

Yang, J.; Cai, R.; Jing, F.; Wang, S.; Zhang, L.; and Ma, W. 2008. Search-based query suggestion. In *Proc. of CIKM'08*.

Zamir, O.; Etzioni, O.; Madani, O.; and Karp, R. 1997. Fast and intuitive clustering of web documents. In *Proc. of KDD'97*.