Research Article

Consolidating User Search Histories using Query Groups

Varshali Cholake\textsuperscript{ab}, Modraj Bhavsar\textsuperscript{a} and P.M. Chawan\textsuperscript{a}

\textsuperscript{a}Department of Computer Engineering & IT, Veermata Jijabai Technological Institute, Mumbai-19, India


Abstract

With the exponential growth in web users, search history is also growing exponentially. To manage the web search, search engine uses different techniques. It gives users an easy feel to search their interest by providing page ranking, web personalization and other tools. Most of the search engines store search history in search query and respective click format. We are using query grouping mechanism to store user’s history so that she will get her previous related searched data easily.

Keywords: clicks, query groups, query reformulations, user search history

1. Introduction

Web search is the main source of getting large desired information so users are pursuing variety of tasks on the web. Even complex tasks are also done online to get the intended information. As today’s computers are multitasking users can accomplish different jobs at a time in multiple tabs. Queries issued in this period are not related to each other. Within a single job user searches for different information thus issues multiple queries to search engine. A complex task such as admission process has to be divided into different tasks. For example, a user may first search for different colleges for particular course, schedule, process etc. After deciding college she will search for cutoff, ranking and all needed information about the college. This process will span for few months. So if users search history is stored according date it will be difficult for her to get previous searched queries.

To enable better services for users search engines have introduced new Search history feature to track their searches and clicks. Though this feature is useful it will be cumbersome to manage history over long period of time. Identifying query groups will help users to keep track of their queries in search history. They just need to find the group which they want to continue. Query grouping will help search engines to identify the users session and their interest. According to group search engine will recommend pages for the user or rank results.

In this paper, we discuss the problem of organizing user search history into multiple groups in an automated and dynamic fashion. Each query group is a collection of related queries of the same user. When user searches for the particular query system will suggest her related searches and other related searches. If the keyword is matched with existing keyword then that page will be navigate to corresponding query group. Here we are proposing algorithm for getting related searched results.

Algorithm for selecting related queries

<table>
<thead>
<tr>
<th>Input:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Current query q</td>
</tr>
<tr>
<td>2) Category I</td>
</tr>
<tr>
<td>3) Subcategory Is</td>
</tr>
<tr>
<td>4) Stored category set $S = (S_1, S_2, \ldots, S_m)$</td>
</tr>
<tr>
<td>5) Stored subcategory set $Sb = (Sb_1, Sb_2, \ldots, Sb_n)$</td>
</tr>
<tr>
<td>6) Stored queries $Q = (q_1, q_2, \ldots, q_p)$</td>
</tr>
</tbody>
</table>

| Output: Related queries from stored categories $= Rs$ |

\[
\text{Step 1: input category I} \\
\text{for(int j=1 to m) if(I=SIM(S_j)) AND} \\
\text{Step 2: input subcategory Is} \\
\text{for(int i=1 to n) if(Is=SIM(Sb_i)) AND} \\
\text{Step 3: Input query q} \\
\text{for(int k=1 to p) if(q=LIKE(q_j))} \\
\text{Step 4: Rs final resultset Rs=R1,R2,\ldots,Rn} \\
\text{Where R1,R2,\ldots,Rn are queries similar to the input query which satisfies step 1, 2 and 3.} \\
\text{Here, category S are main query groups which are sub divided into sub category. SIM()} function finds category and subcategory which matches to input category and subcategory respectively. LIKE() function will return queries which has common words and queries related to input query. Organizing user search histories within query groups is a challenging task because related queries may not appear close to one another, as a search may span days or weeks. Here is sample example for query group.

\footnotetext{*Corresponding author: Varshali Cholake}
2. Query Relevance

2.1 Aim

Our aim is to find the related query group for the given query in an automated and dynamic fashion and to gain results which are related to our query. Each query group has subcategory and query with its click.

2.2 Query Relevance

There are different metrics for selecting relevant queries. We can measure query relevance by using textual similarity, temporal similarity. It is very important to have related queries within a single group for this we need relevance measure \( sim \) between the current query \( S_c \) and query group \( S_i \in S \). There are different measures to determine the relevance.

**Time:** We can consider two queries relevant if they appear close to each other in time in user search history.

**Text:** We can consider two queries relevant if they appear textually similar i.e., some keywords are found same in other query then we assume that both queries are similar. But as discussed in section 1 user is issuing multiple queries simultaneously which may not be relevant so time metrics is not useful in such a situation. There is possibility that two queries \( S_c \) and \( S_i \) share common words but they are not relevant to each other.

3. Query Relevance using Search Logs

Here we will discuss two important properties of relevant queries. 1) query reformulation graph 2) query click graph.

3.1 Search Behavior Graphs

We design three types of graphs from the users search log. Query reformulation graph \( QRG \) is reformulations of different queries. And query click graph \( QCG \) shows the relationship between two queries which leads to clicks on same links. Query fusion graph is merging of \( QRG \) and \( QCG \).

3.1.1 Query Reformulation Graph

\( QRG \) is designed as follows. We look for the queries \( q_i \) and \( q_j \) which appear together for multiple users in their search logs. We select threshold value \( T_r \) to choose \( qi \) and \( qj \) as a reformulation. There is a directed edge between \( qi \) and \( qj \) in EQR if \( \text{count}(qi,qj) >= T_r \). The edge weight \( W_r(qi,qj) \), is defined as the normalized count of the query transitions

\[
W_r(q_i,q_j) = \frac{\text{count}(qi,qj)}{\Sigma(q_i,q_j) \in EQR \Sigma(q_i,q_k)}
\]

3.1.2 Query Click Graph

Query click graph is a bipartite graph whose edge \( ECG(qi,qj) \) is formed if both \( qi \) and \( qj \) induce same clicks for the multiple users. Edge between \( qi \) and \( qj \) is formed if count of similar clicks is greater than threshold value \( T_c \). The weight of edge \( (qi,qj) \) in QCG, \( W_c(qi,qj) \), is defined as follows.

\[
w_c(q_i,q_j) = \frac{\Sigma_{uk} \min(\text{count}(qi,uk),\text{count}(qj,uk))}{\Sigma_{uk} \text{count}(qi,uk)}
\]

3.1.3 Query Fusion Graph

Query fusion graph \( QFG \) has both the properties of query search logs by merging \( QRG \) and \( QCG \). \( QFG \) contains the set of edges that are present in either \( QRG \) or \( QCG \). The weight of edge \( (qi,qj) \) in \( QFG \), \( W(qi,qj) \) calculated as follows.

\[
W(q_i,q_j) = \alpha * w_r(q_i,q_j) + (1 - \alpha) * w_c(q_i,q_j)
\]

Where \( \alpha \) is used to control relative contribution of the two weights.

3.2 Computing Query Relevance

The edges of \( QFG \) corresponds to edges present in \( QRG \) and \( QCG \) however it is not feasible to use pairwise relevance values from \( QFG \) as a final relevance score. Consider vector \( rq \) consisting values \( W(q_i,q_j) \), if there exists an edge from \( q \) to \( q_j \) in \( QFG \) otherwise 0. But this approach will not consider related queries which are not directly connected in \( QFG \).
Algorithm for calculating the query relevance by simulating random walks over the query fusion graph.

Relevance (q)

Input:
1) The query fusion graph, QFG
2) The jump vector, g
3) The damping factor, d
4) The total number of random walks, numRWs
5) The size of neighborhood, maxHops
6) The given query, q

Output: the fusion relevance vector for q, relFq

(0) Initialize relFq=0
(1) numWalks=0; numVisits=0;
(2) while numWalks<numRWs
(3) numHops=0; v=q
(4) While v ≠ NULL ^ numHops<maxHops
(5) numHops++
(6) relFq(v)++; numVisits++
(7) v=SelectNextNodeToVisit(v)
(8) numWalks++
(9) For each v, normalize relFq(v)= relFq(v)/numVisits

So for obtaining relevance Markov chain for query q, MCq is introduced over the given QFG and to compute the stationary distribution of the chain. We can find query relevance easily with the MC approach. This measure of query relevance is called as fusion relevance vector of q. The stationary probability distribution of MCq can be estimated using the matrix multiplication, where matrix corresponding to MCq is multiplied by itself iteratively until the resultant matrix reaches a fix point. But in real scenario large no. of users issuing queries makes QFG huge. It is infeasible to perform matrix multiplication for stationary distribution when new query comes in. instead we use Monte Carlo random walk simulation to calculate query relevance. The random walk proceeds as follows. We use the jump vector gq to pick the starting point for random walk. At each node v, for a given damping factor d, the random walk either starts at node with probability d, or stops or restarts at node with probability (1-d). The selection of the next node to visit is done by SelectNextNodeToVisit algorithm. This algorithm considers outgoing edges of current node v in QFG and damping factor d. Given query q and damping factor d, the marker chain for q, MCq, is defined as follows:

MCq(qi,qj) = d * Wf(qi,qj) if qj ≠ q,
MCq(qi,qj) = d * Wf(qi,qj)+( 1- d ) if qj = q

Algorithm for selecting the next node to visit

SelectNextNodeToVisit(v)

Input:
1) The query fusion graph, QFG
2) The jump vector, g
3) The damping factor, d
4) The current node, v

Output: the next node to visit, qi

(0) If random()<d
(1) V={qi| (v,qi) ∈ EQG }
(2) Pick a node qi ∈ V with probability Wf(v, qi)
(3) Else
(4) V={qi| g(qi)>0}

(5) Pick a node q, ε V with probability g(q)
(6) Return q.

3.3 Incorporating Current Clicks

In addition to query reformulations, user activities also include clicks on the URLs following each query submission. The clicks of user may show her search interest behind query q and thus identify queries and query groups relevant to q more effectively. We identify the set of URLs, clk, that were clicked by the current user after issuing q. Then we use clk and the click-through graph CG to expand the space of queries considered when we calculate fusion relevance vector of q. We employ a click jump vector, gclk that represents the queries which have also induced same clicks. Each entry in gclk, gclk(qi), corresponds to the relevance of query qi to the URLs in clk. Using CG gclk is defined as the proportion of the number of clicks to clk induced by qi(qi cVq(q)) to the total number of clicks to clk induced by all the queries in Vq(q)

\[
g_{\text{clk}}(qi) = \frac{\sum_{\text{urls}} \text{counte}(qi, uk)}{\sum_{\text{urls}} \text{counte}(q, uk)}
\]

Since the given query q is already captured in gq, we set the entry in gclk corresponding to q to 0 (gclk(q)=0).

4. Comparison of Query Relevance Metrics

We now compare the performance of our proposed method against the metrics discussed in section 2. For these metrics we use the same SelectBestQueryGroup algorithm with different relevance metrics.

We use time-based metric to find the query relevance. But as said earlier due to users multtasking time based relevance metric does not perform well. Next one is text based similarity. Query qj may have similar keywords as that of previous query qi but it does not show the actual relevance between the keywords. For example apple ipod and apple fruit share common words in between even though they are not related to each other.

Our approach of query fusion graph uses the search log to analyze the query group. As said in section 3 it also incorporates user clicks after query submission. Even if apple ipod and apple fruit share common words by analyzing query clicks in QFG we can understand the users interest. Thus using QFG as a query grouping metric we get intended solution.

Conclusion

In this paper we have shown that how query reformulation graph and query click graph are combined together to get the query fusion graph. Query fusion graph is used to find the query relevance thus we have shown its usefulness in query grouping. As a future work we intend to find the usefulness of information gained from query grouping. This information can be further used in applications such as page ranking, or in web personalization.
References


Craig Silverstein, Monika Henzinger, Hannes Marais, Analysis of a Very Large Web Search Engine Query Log.

Naresh Barsagade (Dec.8,2003), Web Usage Mining And Pattern Discovery: A Survey Paper, CSE 8331.

Zdravko Markov and Daniel T. Larose, Mining the Web: Uncovering Patterns in Web Content Structure, and Usage Data. Copyright 2007

Xuanhui Wang and Cheng-Xiang Zhai (2007), Learn from Web Search Logs to Organize Search Results, SIGIR.


Daxin Jiang, Jian Pei, Hang Li (2013), Mining Search and Browse Logs For Web Search: A Survey, ACM Transactions

Huanhuan Cao, Daxin Jiang, Jian Pei., Context-Aware Query Suggestion by Mining Click-Through and Session Data.