Clustering Search Engine Query Log Containing Noisy Clickthroughs

Wing Shun Chan
vincentc@ust.hk

Wai Ting Leung
kwtleung@ust.hk

Dik Lun Lee
dlee@cs.ust.hk

Department of Computer Science
Hong Kong University of Science and Technology,
Clear Water Bay, Kowloon, Hong Kong

Abstract

Query clustering is a technique for discovering similar queries on a search engine. In this paper, we present a query clustering method based on the agglomerative clustering algorithm. We first present an overview of the agglomerative clustering algorithm proposed by Beeferman and Berger [1]. We point out a weakness of the method caused by noisy user clicks and propose an improved clustering algorithm. Our results show that in general the agglomerative clustering algorithm can cluster similar queries effectively and that our improved algorithm can successful eliminate noisy clicks and produce cleaner query clusters.

1. Introduction

The exponential growth of available information on the Internet has given rise to greater demands for efficient search engines. In order to satisfy the users' needs, a new generation of search engines has appeared that tries to understand users' queries and suggest queries that are often asked by other users or queries prepared by the editors. To find similar queries, the search engine has to exploit the *search engine log*, in which each record consists of a user query and the documents selected for the corresponding query (called *clickthrough*). Given millions of queries and clickthroughs, how can the search engine judge if queries are related or not? One popular method is to apply clustering to the search engine log.

In this paper, we present a method based on Beeferman and Berger's agglomerative clustering algorithm [1]. The algorithm is content-ignorant, which means that the algorithm does not make use of the actual content of the queries and the documents in clustering. We show that Beeferman and Berger's method is susceptive to "noisy" clicks. That is, user selections may not truly reflect the relevancy between the selected web page and the user query, because the user may make a click by mistake, be misled by a poor summary, or click on a result purely out of curiosity. These "noisy" clicks introduce errors into the query clusters, making the resulting clusters less precise and less cohesive.

We improved Beeferman and Berger's method to ignore the "noise" relationships from the search engine log. Experimental results show that our method can produce better clusters of similar queries.

2. Overview of Query Clustering

There are three main types of query clustering algorithm. The first assumes that if two queries share the same keywords, then they are similar. The more keywords the two queries share, the higher the similarity.

The second exploits query logs kept by the search engine. Given a large set of user query logs, we can extract out query sessions; each consisting of one query and the documents that the user clicked on (i.e., the clickthroughs of the query). If we find that a user clicks on a document, it is very likely that the document is relevant to the query, or at least the document is related to the query to some extent. Furthermore, if two queries lead to the same document, they are similar to some extent.

Finally, the third type of clustering algorithm is a combination of the first and second algorithm, it relying on both the query content and document

clicks during the clustering process.

In the clustering process, we first obtain the search engine log from a search engine. Then, we perform preprocessing processes such as stemming, stop words removal, phase recognition, etc., on the search engine log, convert the log into an internal representation, and pass the internal representation to the clustering algorithm.

Clustering methods had been developed for clustering query logs for query expansion [2] as well as for search engine selection and performance evaluation [3]. In the remainder of the paper, we will focus on the agglomerative clustering method proposed by Beeferman and Berger [1] for clustering similar queries from query logs.

3. Graph-Based Iterative Clustering

3.1 The Clustering Algorithm

Given a search engine log, Beeferman and Berger' clustering algorithm first constructs a bipartite graph with one set of vertices corresponding to queries, and the other corresponding to documents.

After the bipartite graph is obtained, we apply the agglomerative iterative algorithm to obtain the clusters. Notice that during the clustering process, the algorithm iteratively combines the two most similar queries, then the two most similar documents, and then the two most similar queries and so on. The main reason for not clustering all of the queries first and then all of the documents is that some queries may seem unrelated prior to document clustering but may become similar to each other after some documents are merged.

3.2 The Similarity Function

Assume that N(x) is the set of neighboring vertices of vertex x in a graph, and N(y) is the set of neighboring vertices of vertex y in a graph. Then we can assume that y is similar to x if N(x) and N(y) have a large overlap. Therefore, Beeferman and Berger adopted the following similarity function:

$$Sim(x,y) = \begin{cases} \dfrac{N(x) \cap N(y)}{N(x) \cup N(y)} & \text{If } |N(x) \cup N(y)| > 0 \\ 0 & \text{otherwise} \end{cases}$$

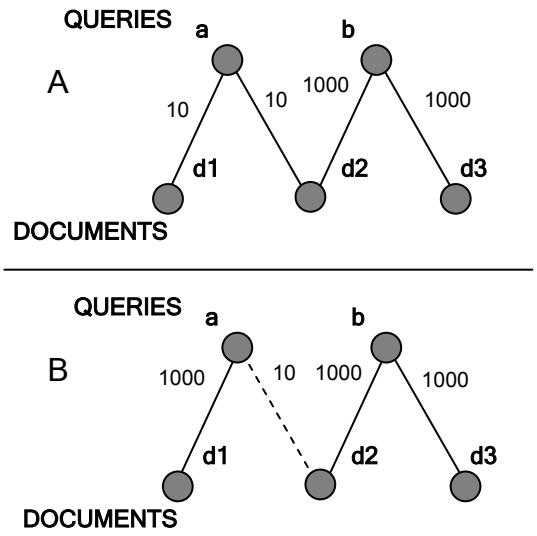

Figure 1. a) A bipartite graph without "noise" b) A bipartite graph with a "noise" link, where the solid edges are "real" links and the dash edge is a "noise" edge.

A major weakness of the similarity function proposed by Beeferman and Berger is that it does not take "noise" links into consideration in the clustering process. Consider the three examples shown in Figure 1, where the number besides a link is the total number of clicks on the document. In Fig. 1(a), we can say that "d1" and "d2" are relevant to "a" and "d2" and "d3" are relevant to "b". Thus, we can conclude that "a" and "b" are similar queries because they share the same relevant document "d2". However, in Fig. 1(b), we cannot say that "d1" and "d2" are both relevant to "a" since only a small fraction of the clickthroughs (10 out of 1010) supports that conclusion. Consequently, we cannot conclude that "a" and "b" are similar queries. Beeferman and Berger's algorithm can not detect the "noise" link in Figure 1(b). It will give the same similarity score of 1/3 to both examples.

To solve this problem, we proposed the following similarity function. Assume that L(x,y) is the set of links connecting x and y to the same documents, L(x) and L(y) are all the links connecting to x and y, respectively, and | L() | is the cardinality of L(). Then our similarity function is defined as:

$$Sim(x,y) = \begin{cases} \dfrac{|L(x,y)|}{|L(x) \cup L(y)|} & \text{If } |L(x) \cup L(y)| > 0 \\ 0 & \text{otherwise} \end{cases}$$

Applying our similarity function, we get a similarity score of 1010/2020 = 1/2 for (a), and similarity score of 1010/3010 ≈ 1/3 for (b). Notice that the score for (a) is higher than that of (b), because most people are selecting document "d1" in (b), and the links between "a" and "d2" can be considered as "noise". Therefore, it makes sense to assign a lower score to (b).

Using our similarity function, the similarity between two vertices always lies between [0,1]. The similarity for two vertices is 0 if they share no common neighbor, and the similarity between two vertices is 1 if they have exactly the same neighbor vertices. Our results show that our similarity function can effectively group similar queries into the same cluster.

4. Implementation

Since we need to collect clickthrough data, we implemented a middleware through which users submit queries to a real search engine. In our experiment, queries are redirected to the YAHOO! search engine. Results from YAHOO! are returned to the user through the middleware. The user clicks on relevant results, and the clickthroughs are recorded by the middleware and stored in a database. The middleware was implemented using C++, MFC, and MySQL.

After obtaining the clickthrough data, the data is passed to the clustering program which is implemented using C++ and MySQL. To obtain the clickthrough data, the clustering program simply connects to the database and obtains the information needed for clustering.

5. Evaluation

During testing, we used the following nine queries:

computer, computer network, notebook computer, hardware, accounting, laptop, summer course, winter course, course

We obtained our clickthrough data by searching each query on YAHOO! via our middleware. The search domain was fixed to www.ust.hk, which is the web site for our university, so that only web pages within our university are returned. The testers are asked to look for web pages within our university that are relevant to the queries. This makes such that the testers have a common focus and generate clickthroughs that have enough overlaps for clustering similar queries together, even with a limited amount of clickthrough data.

By using the similarity function and the terminating condition proposed by Beeferman and Berger which iterates until the resulting graph consists of connected components, we found that most of the queries were grouped into one single cluster even when some were unrelated to the cluster. Therefore, instead of suggesting all the queries in the cluster, we suggest to consider queries which have similarity scores greater than a certain threshold. For our middleware, we will only keep those terms with similarity score higher than 0.001.

computer (original similarity function)	computer	1
	computer network	0.170412
	notebook computer	0.106742
	hardware	0.095819
	accounting	0.059546
	summer course	0.010949
	course	0.006590
	winter course	0.003717
	laptop	0.001883
computer (new similarity function)	**computer**	**1**
	computer network	**0.002178**
	notebook computer	**0.001972**
	hardware	**0.001142**
	accounting	0.000976
	laptop	0.000802
	summer course	0.000401
	winter course	0.000321
	course	0.000079

Figure 2. Similarity scores for the queries in the "computer" cluster obtained by using the original similarity function and our proposed similarity function. And the bolded queries are the queries which will be suggested to the user based on our scheme.

Figure 2 shows the resulting cluster for the query "computer" by using Beeferman and Berger's similarity function and our proposed similarity function. Notice how the position of the query "laptop" changes after using our newly proposed similarity function. Using Beeferman and Berger's similarity function, it is least relevant to "computer". However, using our own similarity function, it becomes the sixth most relevant query to "computer".

To see how the similarity function can help lower the similarity score for query with "noise" links, we will look at another cluster for the query "course" in Figure 3. Using the original

similarity function, the similarity scores for the other two queries in the same subject domain as "course," namely, "summer course" and "winter course" are 0.039604 and 0.022472, respectively. Furthermore, the query "computer" (the query just below "winter course"), which is not in the same domain as "course," has a similarity score of 0.006590 which is around 30% of the similarity score of the query "winter course".

course (original similarity function)	course	1
	summer course	0.039604
	winter course	0.022472
	computer	0.006590
	accounting	~=0
	laptop	~=0
	notebook computer	~=0
	computer network	~=0
	hardware	~=0
course (new similarity function)	**course**	**1**
	winter course	**0.002469**
	summer course	**0.002058**
	computer	0.000079
	accounting	~=0
	laptop	~=0
	notebook computer	~=0
	computer network	~=0
	hardware	~=0

Figure 3. Similarity scores for the queries in the "course" cluster obtained by using the original similarity function and our proposed similarity function. And the bolded queries are the queries which will be suggested to the user based on our scheme.

Using the similarity function we propose, we can see that the similarity scores for the two queries in the same subject domain as "course" are all around 0.002. However, for the query "computer" (the query just below "summer course"), which is not in the same domain, the similarity score is only 0.000079 which is around 3% of the similarity score of the query "summer course". As a result, the queries which will be suggested to the user based on our schemes are "winter course" and "summer course." Hence, we can see that the scores obtained by using our new similarity function are more stable. Moreover, our new similarity function can also detect "noise" links more efficiently compared to the original similarity function.

6. Conclusion

In this paper, we successfully implemented the agglomerative query clustering algorithm proposed by Beeferman and Berger. However, if we iterate the algorithm until the resulting graph consists of connected components, most of the queries are grouped into one single cluster even if some are unrelated to the cluster. Therefore, we modified the algorithm so that only queries with similarity scores higher than a certain threshold would be suggested to the user. In addition, we found that the similarity function proposed by Beeferman and Berger is inadequate for detecting and filtering out "noise" links in the bipartite graph. Therefore, we proposed a new similarity which can help filtered out "noise" links, and our results show that our proposed similarity function outperforms that proposed by Beeferman and Berger.

Even though our method can effectively cluster similar queries, it is not incremental. Further research is needed to extend our algorithm to incrementally update the clusters when new clickthrough data are available.

7. Acknowledgment

This work was supported in part by grant from the Research Grant Council of Hong Kong, Grant No HKUST6079/01E.

8. References

[1] Doug Beeferman, Adam Berger, Agglomerative clustering of a search engine query log. *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, p.407-416, August 20-23, 2000, Boston, Massachusetts, United States.

[2] Hang Cui, Ji-Rong Wen, Jian-Yun Nie, Wei-Ying Ma, Probabilistic query expansion using query logs. *WWW 2002*: 325-332.

[3] T. Joachims Optimizing search engines using clickthrough data. *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*, 2002.

[4] Ji-Rong Wen, Jian-Yun Nie, Hong-Jiang Zhang, Query clustering using userlogs. *TOIS* 20(1): 59-81 (2002).