A NEW CLUSTER MERGING ALGORITHM OF SUFFIX TREE CLUSTERING

Jianhua Wang, Ruixu Li
Computer Science Department, Yantai University, Yantai, Shandong, China

Abstract: Document clustering methods can be used to structure large sets of text or hypertext documents. Suffix Tree Clustering has been proved to be a good approach for documents clustering. However, the cluster merging algorithm of Suffix Tree Clustering is based on the overlap of their document sets, which totally ignore the similarity between the non-overlap parts of different clusters. In this paper, we introduce a novel cluster merging approach which will combines the cosine similarity and overlap percentage. Using this method, we can get a better clustering result and a comparative small number of clusters.

Key words: suffix tree clustering, cluster merging algorithm

1. INTRODUCTION

Document clustering has been studied intensively recently because of its wide applicability in areas such as web mining, search engines, information retrieval, and topological analysis. Clustering documents into groups can organize large bodies of text for efficient browsing and searching. A lot of different text clustering algorithms have been proposed in the literature, including Agglomerative Hierarchical Clustering (AHC) [5], Scatter/Gather [2] and K-Means [4].

Zamir and Etzioni presented a Suffix Tree Clustering(STC) algorithm on document clustering in[3]. STC is a linear time clustering algorithm that is based on a suffix tree which efficiently identifies sets of documents that share common phrases. STC treats a document as a string, making use of proximity information between words, at the same time, it is incremental and has an *O(n)* time complexity.

In Zamir and Etzioni's Suffix Tree Clustering algorithm, after the suffix tree construction, the overlap of the different clusters is calculated, and the clusters are merged if they have more than 50% overlap. This merging method is fast, however, it is too simple to yield the best merging result because it totally neglects the similarity between the non-overlap parts. Considering such situation: two different clusters have much related and similar documents, but none of the documents are contained in both clusters, that is, no overlap between the two clusters. According to Zamir and Etzioni's merging algorithm, the two clusters have no chance be merged. This obvious is not a good choice because we actually hope the two clusters can be merged due to their much related documents. Another problem is this simple merging algorithm can result in too many clusters, usually hundreds even thousands of clusters, with only a small amount of documents in each of it. This really frustrates the browsers to locate the desired information.

In this paper, we present a novel approach which introduces the well-known cosine similarity algorithm into the cluster merging process. In our algorithm, the similarity of the two clusters is not only decided by the overlap of their documents, but also by the similarity of the non-overlap parts. This is quite natural, because the related and similar documents should also contribute to the similarity between two clusters. The following experiments show that this algorithm has more accuracy that the original one. The new algorithm also has another advantage: with adjusting to some parameters, we can control the number of final clusters we get. This is very useful in some cases.

The rest of this paper is organized as follows. Section 2 briefly introduces Suffix Tree Clustering and its cluster merging algorithm. In section 3, we introduce our novel cluster

Please use the following format when citing this chapter:

Wang, J., Li, R., 2006, in IFIP International Federation for Information Processing, Volume 228, Intelligent Information Processing III, eds. Z. Shi, Shimohara K., Feng D., (Boston: Springer), pp. 197–203.

merging algorithm which combines the overlap percentage and cosine similarity. An experimental evaluation was conducted, and section 4 reports its major results. A summarization of the paper is presented in section 5.

2. RELATED WORKS

As a data structure, suffix tree has been studied a lot in information retrieval field [3], [13],[15]. A suffix tree is a trie data structure built over all the suffixes of the text. Each node of the suffix tree represents a group of documents and a phrase that is common to all of them. The label of the node represents the common phrase. In Zamir and Etzioni's paper[3], they found that using suffix tree to cluster web search results yielded better results than other clustering methods. STC algorithm has several steps:

(1) Document Preprocessing, including stemming, html tags and stopwords filtering out.
(2) Suffix Tree Construction. Construct a suffix tree for the document collections.
(3) Cluster Merging. Merge similar clusters according to their similarity measure.
(4) Clusters Ranking. Rank the cluster according to their relevance to the query.

In the cluster merging step, Zamir and Etzioni defined a similarity measure between clusters based on the overlap of their document sets. The two clusters are merged only when they have enough overlap. Despite its simplicity and fastness, this method is not accurate enough to merge all related clusters. In this paper, we propose a new similarity measure algorithm which takes more factors into account.

3. A NEW DOCUMENT CLUSTERING ALGORITHM

Before constructing the suffix tree and getting the documents clusters for merging, a preprocessing step must be done. First, any non-textual information such as HTML-tags and punctuation is removed from the documents. Stopwords such as "I", "am", "and" are also removed. Second, All capitals in the documents are converted to lowercase. Then, the remaining words are stemmed by removing prefixes/suffixes and reducing plural to singular. Through the above preprocessing, we get rid of all noise and produce a cleaned document for further processing.

The construction of a suffix tree can be viewed as the creation of an inverted index of phrases for the document collections. This process can be done in linear time with the size of the document set, and can be done incrementally as the documents are being read. At each node, after construction, we get a list of documents that correspond exactly to that particular node, as well as an index that allows us to locate phrases in the document.

After construction of the suffix tree, we should merge the related clusters based on its similarity. In [3], Zamir and Etzioni defined a similarity measure between clusters based on the document overlap. Given the two base clusters B_m and B_n, with sizes $|B_m|$ and $|B_n|$ respectively, and $|B_m \cap B_n|$ representing the number of documents common to both clusters. The two clusters are merged only when:

$$\frac{|B_m \cap B_n|}{|B_m|} > 0.5 \quad and \quad \frac{|B_m \cap B_n|}{|B_n|} > 0.5 \tag{1}$$

There are two disadvantage of Equation 1:

First, it is not suitable for two clusters when $|B_m|/|B_n|$ is too big or too small. Here is an example: B_m contains 100 documents and *Bn* contains 10 documents, all of which are totally overlapped by the documents in B_m. That is, B_n is a subset of B_m. Obviously we should merge

B_n into B_m. But if we follow Equation 2, | $B_m \cap Bn$|/| B_m | =0.1 <0.5, we cannot merge them together.

Here, we employ a better equation to calculate the overlap of two different clusters.

$$Overlap(B_m, B_n) = \frac{|B_m| \cap |B_n|}{Min(|B_m|, |B_n|)} \tag{2}$$

Second, it neglects the similarity between the non-overlapped parts. Consider the flowing situation: There are several different but very similar and related documents located in two clusters, and no overlap between the two clusters.

Due to all the documents in the two clusters are similar and related, we should merge them together. However, using the equation 1, they will never be merged because there is no overlap between them. That is, in Zamir and Etzioni's merging algorithm, only the same documents located in different clusters can contribute to the similarity measure. Similar but different documents cannot contribute anything to the merging decision. This is obvious inappropriate.

To solve this problem, we introduce the well-known cosine similarity to the merging algorithm and employ the cosine similarity to evaluate the similarity between different clusters. Given $B_m' = B_m$- | $B_m \cap B_n$| , $B_n' = B_n$ -| $B_m \cap B_n$ | , the similarity between B_m' and B_n' is:

$$Sim(B_m', B_n') = \frac{\overrightarrow{B_m'} \bullet \overrightarrow{B_n'}}{|\overrightarrow{B_m'}| \times |\overrightarrow{B_n'}|} = \frac{\sum_{i=1}^{t} (w_{i,m} \times w_{i,n})}{\sqrt{\sum_{i=1}^{t} w_{i,m}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,n}^2}} \tag{3}$$

Where $B_j = (w_{1,j}, w_{2,j}, \ldots w_{t,j})$ *is the term frequency vector representation of cluster* B_j *and* t is the total number of index term in the system. w_{ij} is the weight of term k_i in cluster j. We calculate w_{ij} using the *TFIDF algorithm.*

$$w_{ij} = f_{ij} \times idf_i = \frac{freq_{i,j}}{\max freq_j} \times \log \frac{N}{n_i} \tag{4}$$

Where N is the total number of clusters and n_i is the number of clusters in which the index term k_i appears. $freq_{i,j}$ is the number of times the term k_i appears in the text of the cluster B_j. Max$freq_j$ is maximum frequency of all terms which are mentioned in the text of the cluster B_j.

Using this representation, we can calculate the similarity between the non-overlap parts of the two clusters based on the similarity of their term vectors.

After we get the overlap of the two clusters from Equation 2 and the similarity of the non-overlap parts from Equation 3, we can calculate the overall normalized similarity of the two clusters:

$$S_{m,n} = a * Overlap(B_m, B_n) + (1-a) * Sim(B_m', B_n') \tag{5}$$

And merge them only if:

$$S_{m,n} > k \tag{6}$$

B_m' and B_n' is the non-overlap parts of the Cluster B_m and Cluster B_n, that is, $B_m' = B_m - |B_m \cap B_n|$, $B_n' = B_n - |B_m \cap B_n|$. In our experiments, we assigned α=0.6. Changing the value of k in Equaion 6 can adjust the number of merged clusters we get. A smaller k means the merging condition is easy to satisfy and can result in less merged clusters. While a bigger k set a more restrictive condition and can result in more merged clusters. So, we can adjust the parameter k to make the merging algorithm suitable to different fields.

The whole merging process is not a single step, but an iterative process described as follow:

1. We compute the pairwise similarity among all clusters according to Equation 5.
2. We select the maximum $S_{m,n}$ ($S_{m,n} > k$) and merge the cluster B_m and cluster B_n.
3. Step 1 and step 2 is iteratively executed until maximum $S_{m,n} \leq k$.

The initial time complexity of the cluster merging process is $O(n^2)$. To keep the cost of this process constant, we don't calculate the similarity of all base clusters, but only calculate the n top ranking base clusters. We found that n=500 was sufficient to ensure good performance. The ranking score of clusters is determined by the number of documents and their ranking position.

$$rs(B) = \sum_{i=1}^{p} \frac{1}{\log(R_i + e)} \tag{7}$$

Here p is the number of the documents contained in the cluster, and e is a positive parameter to adjust the ranking position in relation to the score. A bigger e can weaken the effect of the ranking position, while a smaller e can strengthen the ranking position's effect. We considered $e=2$ to be a good choice.

R_i is the position of the document in the documents list returned and ranked by the search engine. In Equation 7, the highly ranked documents in the ranking list always contribute more to the score than the lower ranked ones. We take the document ranking into account because we think the number of the documents contained in the cluster cannot reflect the importance of the cluster very well only by itself. For example, one cluster might contain the top 15 documents returned by the search engine, while another cluster contains the 1001th to 1020th documents in the ranking list. If we only consider the number of the documents, we can easily conclude that the second cluster is more important than the first one, which, in this example, is obviously wrong.

The cost of "cleaning" the documents is obviously linear with the collection size. The cost of organizing documents into the suffix tree is also linear with the collection size. And due to only a fix number of clusters is merged, the overall time complexity of this whole process is linear with regard to the collection size.

4. EXPERIMENTS

We conduct several experiments to validate the effectiveness of the new algorithm.

4.1 The number of merged clusters with different α and k

In this experiment, we submitted 10 different queries to Google[11] search engine and get 10 different searching results, each of which has 2000 document snippets in it. Then, for each result, we preprocessed it and constructed the suffix trees. After that, we merged the suffix tree clusters using the algorithm described in this paper. First, we chose 0.5 as the value of k in Equation 6 and merged the base clusters with different value of α. When the 10 cluster merging processes were finished, we calculated the average number of final merged clusters. The results are shown in Figure 1.

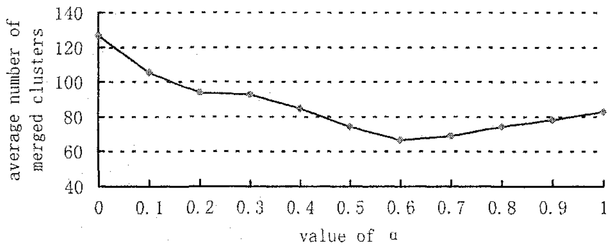

Figure 1. The average number of final merged clusters for different α

From Figure 1, we can see that we get the least number of merged clusters when the value of α is 0.6. In our experiments, 0.6 is a good value for α due to its effectiveness on merging similar clusters together.

We also calculated the average number of merged clusters for different k in Equation 6. The results are shown in Figure 2.

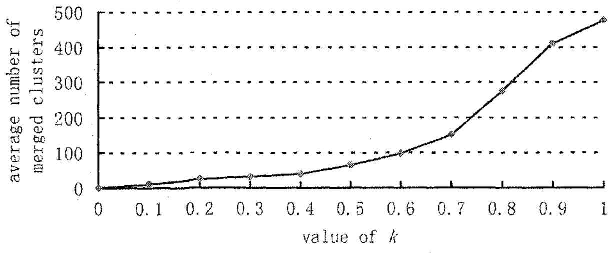

Figure 2. The average number of final merged clusters for different k

From Figure 2, we can see that the value of k can dramatically affect the number of the merged clusters. A smaller k can result in a small number of the merged clusters, but the similarity between the documents in the merged cluster is not strong. With a bigger k we can get very similar documents in the same merged cluster, however, the number of the merged clusters is usually large. Here, we think k=0.5 is a good choice.

4.2 Precision Evaluation

There are many approaches to precision evaluation. To compare with the traditional STC algorithm, we adopted the same evaluation process described in [3].

The process was as follows: "We first defined 10 queries by specifying their topics(e.g., "black bear attacks") and their descriptions(e.g. "we are interested in accounts of black bear attacks on humans or information about how to prevent such attacks"). The words appearing in each query's topic field were used as keywords for a web search using Google search engine. We generated 10 collections of 200 documents from the results of these queries. We manually assigned a relevance judgment (relevant or not) to each document in these collections based on the query's descriptions.

In our experiment we applied the various clustering algorithm to the document collection and compared their effectiveness for information retrieval."[3]

We used the results of the clustering algorithms to reorder the list of documents returned by the search engine. We ordered the clusters according to which labels seemed most relevant to the information needed. In this step we looked only at the cluster labels, ignoring the contents of each cluster. We then defined the ordering of the results as the ordering of the documents in the highest rank cluster, followed by the documents in the next ranked cluster, and so on through the lowest ranked cluster. Documents appearing in multiple clusters were removed from all but the highest ranked cluster in which they appeared, so as to avoid duplicates.

After reordering the documents with clusters, we considered only the top 20 documents in the reordered list and used them to calculate the percentage of relevant documents.

As seen in Figure 3, the STC algorithm with the cluster merging method described in this paper is more precise than all the others. The top 20 documents cover more than 40% relevant documents. The reason for this is mostly because we used a more accurate merging algorithm in this paper.

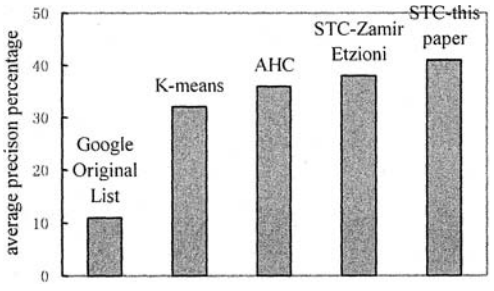

Figure 3. The average precision of the clustering algorithms and of the original ranked list returned by Google.

5. CONCLUSION

We reformed the Suffix Tree Cluster merging algorithm by combining the overlap percentage of two clusters and the similarity between the non-overlap parts of two clusters. First, we revise the overlap percentage calculation method to better reflect the overlap between two clusters. Then, we employ the cosine similarity to calculate the similarity between the non-overlap parts. Our preliminary results are encouraging and show a better result in helping the user to locate the desired documents more easily.

ACKNOWLEDGEMENTS

We thank Prof. Song Yibin and Bi Yuanwei for their advice about this paper. We also appreciate all the experiments done by Wang Zhaoguan, Yu Wei and Wang Bo. Without the help for all of them, this paper can hardly be done.

REFERENCES

1. Liu B., Chin C. W., and Ng, H. T. Mining Topic-Specific Concepts and Definitions on the Web. In *Proceedings of the Twelfth International World Wide Web Conference (WWW'03)*, Budapest, Hungary, 2003.
2. Cutting D.R., Karger D.R., Pedersen J.O., Tukey J.W. Scatter / Gather: A Cluster-based Approach to Browsing Large Document Collection, *Proc. ACM SIGIR 92*, 1992
3. Zamir O., Etzioni O. Web Document Clustering: A Feasibility Demonstration, In *Proceedings of the 19th International ACM SIGIR Conference on Research and Development of Information Retrieval (SIGIR'98)*, 1998.
4 J. J. Rocchio, Document retrieval systems – optimization and evaluation, Ph.D. Thesis, Harvard University, 1966.
5 P. Willet. Recent trends in hierarchical document clustering: a critical review. *Information Processing and Management*, 24:577-97, 1988.
6. Leuski A. and Allan J. Improving Interactive Retrieval by Combining Ranked List and Clustering. *Proceedings of RIAO, College de France*, pp. 665-681, 2000.
7. Smith, D.A. Detecting and Browsing Events in Unstructured Text. In *Proceedings of ACM/SIGIR'2002*.
8. Sergey Brin, and Larry Page. The anatomy of a large scale hypertextual web search engine. In *Proceedings of WWW7*, Brisbane,Australia, April 1998.
9. Hua-Jun Zeng Qi-Cai He Zheng Chen Wei-Ying Ma Jinwen Ma Learning to cluster web search results *SIGIR*'04, July 25 29, Sheffield, South Yorkshire, UK , 2004
10. X. Shen, B. Tan, and C. Zhai. Intelligent search using implicit user model. Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign, 2005.
11. Google search engine, http://www.google.com.
12. Yahoo search engine, http://www.yahoo.com
13. Ricardo Baeza-Yates. Berthier Ribeiro-Neto, *Modern Information Retrieval*, Addison Wesley Press, 1999
14. Ian.H.Written, Alistair Moffat, Timothy.C. Bell. *Managing Gigabyte*, Morgan Kaufmann publishing, 1999
15. P. Weiner. Linear pattern matching algorithms. In Proceedings of the 14th Annual Symposium on Foundations of Computer Science (FOCS), pages 1-11, 1973.