

# A Hash-based Hierarchical Algorithm for Massive Text Clustering

Yin Luo<sup>1</sup>, Yan Fu<sup>2</sup>

<sup>1</sup> Department of Software University of Electronic Science and Technology of China, Chengdu, China  
Email: luoyin7@yahoo.com.cn

<sup>2</sup> Department of Computer Science and Engineering  
University of Electronic Science and Technology of China, Chengdu, China  
Email: fuyan@uestc.edu.cn

**Abstract**—Text clustering is the process of segmenting a particular collection of texts into subgroups including content based similar ones. The purpose of text clustering is to meet human interests in information searching and understanding. This study proposes a new fast hierarchical text clustering algorithm HBSH (Hash-based Structure Hierarchical Clustering), which is suitable for massive text clustering. This algorithm uses hash table instead of numerical vectors as its input data. Compared with the other clustering algorithms, the HBSH performs the text clustering process without setting clustering center number and has minor space complexity in advance, which can achieve better performance. The experimental results illustrate that the average time of HBSH is faster than that of traditional text clustering algorithms.

**Index Terms**—hierarchical, text clustering, hash table

## I. INTRODUCTION

Text clustering has been adopted as an effective way for sorting numerous of documents to help users navigate, summarize, and organize text documents. There are two methods to organize a collection of texts [1]. One is text categorization and the other is text clustering. The former needs prior knowledge about the collection of texts to predefine categories texts manually to prepare sample texts. If we don't have such knowledge, we must use the latter. Text clustering has higher complexity in its algorithm circuits than text categorization, but it does not require prior knowledge [2]. By organizing a large amount of documents into a number of meaningful clusters, document clustering can be used to browse a collection of documents or to organize the results returned by a search engine in response to a user's query. It can significantly improve the precision and recall in information retrieval systems, and it is an efficient way to find the nearest neighbors of a document [3]. The problem of document clustering is generally defined as follows: given a set of documents, we would like to partition them into a predetermined or an automatically derived number of clusters, such that the documents assigned to each cluster are more similar to each other than the documents assigned to different clusters. In other words, the documents in one cluster share the same topic, and the documents in different clusters represent different topics.

In most existing document clustering algorithms, documents are represented using the vector space model [4], which treats a document as a bag of keywords. A

major characteristic of this representation is the high dimensionality of the vector space, which imposes a big challenge to the performance of clustering algorithms. They could not work efficiently in high dimensional vector spaces due to the inherent sparseness of the data. Although many studies proposed feature selection methods for reducing the dimension of feature vectors, such solutions do not solve two problems. With regard to the first problem, the huge dimensionality of the feature vectors, they reduced more than 10,000 dimensions to several hundreds dimensions, but we are not satisfied by such reductions which are still too small. They do not consider our second problem, the sparse distribution in feature space [5].

In this research, considering the vector space model is sparsely in most cases, we extended the hierarchical clustering algorithm by introducing hash tables rather than vector as its clustering input data structure, which store each cluster's center and its keywords count. Unlike the k-means and DBScan algorithm need huge space for storing vector data and needs the calculation of their semantic similarity [6]. The proposed HBSH just need store the index of the non-zero weight data and calculation of the non-zero weight data semantic similarity. Such similarity is based on their collocations within the same text and their weights indicating the degree of their content based importance in each text.

## II. TEXT REPRESENTATION WITH HASH TABLES

Since texts are unstructured data in natural languages and computers can not process them directly, they should be encoded into structured data. This section describes how HBSH do that.

In the process of text encoding, first step is cleaning texts include removing tags, removal of stop words, and stemming of words. Then the text is represented as vector space model (VSM) which introduced by Salton [7], in vector space model, each text is represented by a vector of words, each element of the vector reflects a particular word, or term, associated with the given text. In the term space,  $d_i = (w_{i1}, w_{i2}, \dots, w_{in})$ , where  $w_{ij}$ ,  $j=1, \dots, n$  is the weight of term  $j$  in document  $i$ . A well-known approach for computing term weights is the TF-IDF weighting [8]. The weight of term  $j$  in a document  $i$  is

given by  $w_{ij} = tf_{ij} \times \log\left(\frac{N}{df_j}\right)$ .

Where  $tf_{ij}$  is the term  $j$  frequency in the text  $i$ , or the number of occurrences of the term  $j$  in a text  $i$ .  $N$  is total number of texts and  $df_j$  is the text frequency, that is the number of texts in which term  $j$  occurs at least once. The inverse document frequency (idf) factor of this type is given by  $\log\left(\frac{N}{df_j}\right)$ .

Then we get each text's keywords vectors and its' weight. The average vector weight of document  $i$  can be defined as

$$AVE_i = \sum_{j=0}^{k_i} w_{ij} / k_i. \quad k_i > 0.$$

When  $k_i$  represents the amount of non-zero weight data in vector of document  $i$ . We remove the keywords which weight less than  $AVE_i$  in the vector of document  $i$ . After that we sort whole documents' keywords alphabetically. Then we create one hash table  $HT = \{HT_1, HT_2, \dots, HT_n\}$  for each text.

The hash table  $HT_i$  is an implementation of (partial) functions between two domains, here called *Address* and *Value*. The hash table thus implements a modifiable shared variable  $X$ :  $Address \rightarrow Value$ . We put the keywords' index number in the domain of *Address* and the default value 1 in the domain of *Value*. An equality  $X(a) = \text{null}$  means that no value is currently associated with the address.

Then put the index of the word in the vector of document  $i$  and the domain *Value* represent the frequency ratio of the word into the  $HT_i$ .

### III. OPERATIONS ON HASH-BASED HIERARCHICAL TEXT CLUSTERING ALGORITHM

#### A. Overview of HBSH

Hierarchical clustering constructs a hierarchy of clusters by either repeatedly merging two smaller clusters into a larger one or splitting a larger cluster into smaller ones [9]. The crucial step is how to best select the next cluster(s) to split or merge. Here we provide a comprehensive analysis of selection methods and propose several new methods. Hierarchical clustering methods are among the first methods developed and analyzed for clustering problems. There are two main approaches. (i) The agglomerative approach, which builds a larger cluster by merging two smaller clusters in a bottom-up fashion. The clusters so constructed form a binary tree; individual objects are the leaf nodes and the root node is the cluster that has all data objects. (ii) The divisive approach, which splits a cluster into two smaller ones in a top-down fashion. All clusters so constructed also form a binary tree [10].

The hierarchical cluster structure provides a comprehensive description of the data, which is quite useful for a number of applications [11]. Given a dataset, however, the hierarchical cluster structure is not uniquely [12]. It depends crucially on the

criterion of choosing the clusters to merge or splitting. This is the main subject of this paper. The HBSH algorithm can be divided into the following 2 major steps: (1) Compute the density for each point; (2) Update the core point's center; (3) Merge the neighbor points.

First we present some basic definitions in HBSH. The set of all correspondence Value  $HT_i$  is denoted as  $PIT_i$ ,  $|PIT_i| = \text{SizeOf}(HT_i)$ , and the distance(or dissimilarity) between point  $p$  and  $q$  is denoted as  $\text{dist}(p, q)$ . The value in  $PIT_i$  is denoted as  $PIT_{ik}$ ,  $k > 0$ ,  $k < |PIT_i|$ .

**Definition 1** (Cluster Core point) A core point  $Co_i$  is compose with the similar hash tables, in the other word, a cluster point is the combination of those hash tables which core distance less than  $\epsilon$ .

In this definition,  $\epsilon$  is a parameters set by the user.

**Definition 2** (Core distance)

Given the value of  $\text{MinPts}$ , the core distance of a point  $P$  is the smallest  $E$  that makes  $P$  as core point, the core distance and it is denoted as  $\text{coreDist}(P)$ .

The core distance between  $Co_i$  with  $Co_j$  is denoted as  $\text{CoreDis}(Co_i, Co_j)$ .

Formally,

$$\text{CoreDst}(Co_i, Co_j) = \frac{|Co_i \cap Co_j|}{|Co_i \cup Co_j - Co_i \cap Co_j|}$$

#### B. The Agglomerative Approach

Firstly, we consider the cluster core points to be an ordered sequence  $Co_1, \dots, Co_n$ , not a set.

The fundamental idea underlying this step is to merge the two core points which core distance is the least. For instance, in Fig. 1, according to (1), the core distance between  $Co_i$  and  $Co_j$  is 71.4%, thus they should be merged into a new core points. The new cluster is still denoted as  $Co_i$ , but the correspondence Value has updated.

Based on the information provided by (1) and Fig. 1, we have the corresponding program (listed in Fig. 2). Obviously, the complexity of the computation is  $O(n^2)$ .

In HBSH, the core points organize all the neighbor sub-core points in a hierarchical framework, and it also constitutes the basis of further process for other points. In the core point tree, each node is represented by a set of sub-core points, and the set of sub-core points represents the dense region around these core points. By constructing the core point tree, we gain the insight into the relation of these regions.

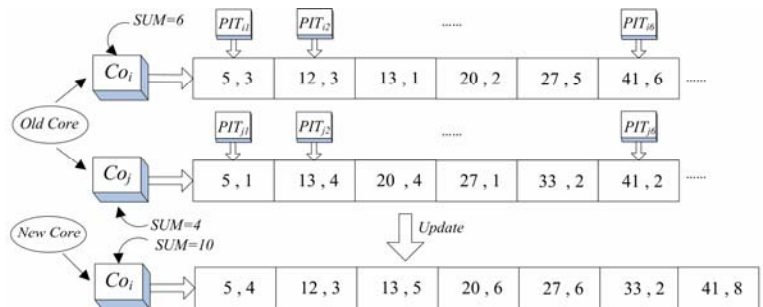


Figure 1. Merging of core points

```

//Initialization.  $Co_i$  is the  $i$ th core point
// $n$  is the number of core points
for  $i=1: n$ 
  for  $j = i+1: n$ 
    if  $CoreDst(Co_i, Co_j) < \epsilon$ 
      Merge ( $Co_i, Co_j$ )
      Update  $Co_i$ 's center
    endif
  endfor
endfor

```

Figure 2. The program to agglomerative clustering

TABLE I. F-MEASURES AND RUM TIME( $T$ ) COMPARISON

Dataset	K-means $T=12min$			DBScan $T=15min$			HCSH $T=4min$			
	$M_i$	$N_j$	$M(n_{ij})$	$M(F(i,j))$	$N_j$	$M(n_{ij})$	$M(F(i,j))$	$N_j$	$M(n_{ij})$	$M(F(i,j))$
Military	481	404	235	0.531	519	316	0.631	506	393	0.798
Sports	620	626	430	0.692	642	423	0.671	640	516	0.820
Art	415	361	279	0.722	368	294	0.752	419	340	0.817
Education	577	547	415	0.746	530	414	0.748	541	489	0.875
Computer	468	554	347	0.683	440	346	0.763	464	407	0.873
Medicine	312	339	201	0.626	256	208	0.734	293	252	0.834

TABLE II. ALGORITHM'S STABILITY COMPARISON

F-measure interval	K-means	DBScan	HCSH
[0.45, 0.55]	5	2	0
(0.55, 0.65]	9	5	3
(0.65, 0.75]	14	15	8
(0.75, 0.85]	2	6	13
(0.85, 0.95]	0	2	4
(0.95, 1.00]	0	0	2

#### IV. EXPERIMENTAL RESULTS

The experiments were performed on an Inter Core 2 Duo PC with 2 GHz clock speed and 1 GB of main memory. We chose java 1.5 as the programming language because it allows fast and flexible development. For a fair comparison, we implemented k-means DBScan and HBSH ourselves and used identical classes whenever possible. The dataset in our experiments take from [www.people.com.cn](http://www.people.com.cn) and [www.xinhuanet.com](http://www.xinhuanet.com).

Table I shows the comparison of efficiency and the accuracy of the three cluster algorithms. We denote  $N_j$  as the sum of the  $j$ th cluster.  $M(n_{ij})$  is the total number of the  $j$ th cluster contains the  $i$ th type when the  $i$ th type achieve the max value of F-measure.  $M(F(i,j))$  is the max value of the  $i$ th type with the different  $j$ th cluster.

The results showed that HBSH approach improved efficiency and accuracy in text clustering task and are significantly better than k-means and DBScan or some

other traditional text clustering methods. For verify HBSH and other algorithm's stability, we employ a numerous of dataset to compares those algorithms, we obtain 30 groups experiment results. Table II shows that HBSH has outstanding stability compares with k-means and DBScan, and HBSH's F-measure equalizing value is higher than the others.

#### V. CONCLUSION

In this paper, we presented a novel approach for massive text clustering. We introduced the algorithm HBSH for flat hierarchical frequent hash-based text clustering.

Compared with some traditional text clustering methods in a number of experiments on 30 groups experiment results, HBSH was significantly more efficient than its competitors.

Finally, we would like to outline a few directions for future research. We already mentioned that the integration of a more advanced algorithm for the generation of frequent term sets could significantly speed-up HBSH and increase HBSH's accuracy.

Hierarchical clustering are of special interest for many applications. However, the well-known measures of hierarchical clustering quality do not adequately capture the quality from a user's perspective. New methods should be developed for this purpose. The proposed clustering algorithms have promising applications such as a front end of a web search engine. Due to the similarity of text data and transaction data, our methods can also be used on transaction data, e.g. for market segmentation. We plan to investigate these applications in the future.

#### ACKNOWLEDGMENT

This work is supported by the National High-Tech Research and Development Plan of China under Grant No. 2007AA01Z440. The authors also wish to thank Dr. Hui Gao for his invaluable support and guidance.

#### REFERENCES

- [1] Zamir O., Etzioni O.: Web Document Clustering: A Feasibility Demonstration, *Proc. ACM SIGIR 98*, 1998, pp. 46-54.
- [2] A. K. Jain, R. C. Dubes, Algorithms for Clustering Data, Prentice Hall, Englewood Cliffs, NJ, 1988.
- [3] M. Suzuki, "Text classification based on the bias of word frequency over categories," Proceedings of the International Conference on Artificial Intelligence and Its Applications (AIA), pp.400-405, (2006)
- [4] K. Nigam, A.K. McCallum, S. Thrun, T.M. Mitchel, "Text classification from labeled and unlabeled documents using EM", *Machine Learning*, 39 (2/3), pp.103-134, 2000.
- [5] M.H.C. Law, M.A.T. Figueiredo, A.K. Jain, "Simultaneous feature selection and clustering using mixture models", *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 26(9), pp.1154-1166, 2004.

- [6] Y. Li, S. M. Chung, J. D. Holt, "Text document clustering based on frequent word meaning sequences", *Data & Knowledge Engineering*, 64(1), pp.381-404, 2008.
- [7] F. Beil, M. Ester, X. Xu, "Frequent term-based text clustering", in Proceedings of ACM SIGKDD International Conference on *Knowledge Discovery and Data Mining*, pp.436-442, 2002.
- [8] Fasulo D. An Analysis of Recent Work in Clustering Algorithms[R]. Technical Report UW-CSE-01-03-02, University of Washington, 1999: 176-186.
- [9] Zaiane O., Antonie M.-L.: Classifying Text Documents by Associating Terms with Text Categories, Proc. *Australasian Database Conference*, 2002, pp. 215-222.
- [10] Haojun Sun, Determining The Number Of Clusters And Distinguishing Overlapping Clusters In Data Analysis. University Of Sherbrooke, Canada Oct,2004.
- [11] P. Bradley, U. Fayyad, C. Reina, "Clustering very large database using EM mixture models", in Proc. *15th Intern. Conf. on Pattern Recognition*, pp.76-80, 2000.
- [12] Larsen B., Aone Ch.: Fast and Effective Text Mining Using Linear-time Document Clustering, *Proc. KDD 99*, 1999,pp.16-22.