VRCA: A Clustering Algorithm for Massive Amount of Texts

Ming Liu
HIT, China
liuming1981@hit.edu.cn

Lei Chen
BNUZ, China
chenlei@bnuz.edu.cn

Bingquan Liu
HIT, China

Xiaolong Wang
HIT, China
{liubq, wangxl}@insun.hit.edu.cn

Abstract

There are lots of texts appearing in the web every day. This fact enables the amount of texts in the web to explode. Therefore, how to deal with large-scale text collection becomes more and more important. Clustering is a generally acceptable solution for text organization. Via its unsupervised characteristic, users can easily dig the useful information that they desired. However, traditional clustering algorithms can only deal with small-scale text collection. When it enlarges, they lose their performances. The main reason attributes to the high-dimensional vectors generated from texts. Therefore, to cluster texts in large amount, this paper proposes a novel clustering algorithm, where only the features that can represent cluster are preserved in cluster's vector. In this algorithm, clustering process is separated into two parts. In one part, feature's weight is fine-tuned to make cluster partition meet an optimization function. In the other part, features are reordered and only the useful features that can represent cluster are kept in cluster's vector. Experimental results demonstrate that our algorithm obtains high performance on both small-scale and large-scale text collections.

1 Introduction

Facing the large-scale texts in the web, it is difficult for users to manually dig useful information from websites. This situation forces many text based analysis tools to appear. In general, clustering is a popular tool for text analysis due to its unsupervised characteristic. Unfortunately, most of traditional clustering algorithms can only handle small-scale text collection. They lose their high performances on large-scale text collection. The reason to this situation mostly attributes to the fact that most of traditional clustering algorithms apply *vector space model* to organize texts and clusters. When text collection is small, this model is all right, whereas when it enlarges, *vector space model* will generate high-dimensional and sparse vectors. This kind of vectors brings in three problems. The first is that the weights of many entries are close to zero, which depresses performance as indicated by [Xu and Wunsch, 2005]. The second is that many useless features are adopted to represent cluster. The last is that high-dimensional vectors make similarity calculation expensive.

To cluster large-scale text collection efficiently, this paper proposes a ***v**ector **r**econstruction based **c**lustering **a**lgorithm*, abbreviated as *VRCA*, where cluster's vector is formed by only choosing a few features that are useful to represent cluster. Two alternately repeated sub-processes are carried out for this purpose. In one sub-process, feature's weight is fine-tuned to gradually reduce one predefined optimization function via iterative tuning process. In the other sub-process, two measures, respectively measuring cluster's intra agglomeration and cluster's inter distinctness, are calculated, and the useful features that are not only capable to represent one cluster but also capable to separate different clusters apart are chosen to reconstruct cluster's vector.

Two large-scale text collections, ClueWeb9 and TRC2, are adopted to test the performance of our algorithm. Besides, another two popular text collections (Reuters and Newsgroup) are also adopted to prove our algorithm's high performance on small-scale text collection.

2 Related Work

Until now, many text clustering algorithms have been proposed. They can be partitioned into five categories: *a*) partition based; *b*) density based; *c*) hierarchy based; *d*) grid based; *e*) model based. *K-means* [MacQueen, 1967], *DBSCAN* [Martin *et al*., 1996], *BIRCH* [Zhang *et al*., 1996], *STING* [Wang *et al*., 1997], *Neuron Network* [Kohonen *et al*., 2000] are their typical exemplars. To further improve clustering performance, some algorithms are even derived from the other subjects, such as *Spectral Clustering* from Physics [Cai *et al*., 2008], *Non-Regression Matrix Factorization* from Mathematics [Wang *et al*., 2007], *LDCC* based on *Latent Dirichlet Allocation* from *Topic Models* [Shafiei and Milios, 2006]. Nevertheless, to our best knowledge, though it has been proposed so many clustering algorithms, the organization model applied by them seldom changes. That is *vector space model*. It will generate high-dimensional and sparse vectors in the situation that text collection becomes too large. Therefore, this model makes traditional clustering algorithms fail to cluster large-scale text collection. For this reason, many ways are designed to improve this model. The most popular ones are feature selection and dimension reduction.

For feature selection, it is often conducted to preserve the features that are useful for separating input texts apart. Liu *et al*., [2013] applied probability model to choose features. Zhao *et al*., [2010] added an entropy based choosing process for further filtering useless features. With time passing, more and more plans are proposed, e.g. mutual information based in [Kwak and Choi, 2002], correlation parameter based in [Sakar and Kursun, 2012]. The common point shared by them is that they treat feature selection as the preprocessing step of clustering. The two processes (selection and clustering) run for two distinct objectives.

For dimension reduction, it is often conducted to cut down vectors from high dimension to low dimension. Latent semantic indexing (*LSI*) proposed by Todd and Michael [1997] is one of the most prevalent methods, whereas the defect of it is that the features in the reduced vector lose their semantic meanings. Another powerful tool is principal component analysis (*PCA*) utilized in graph processing. Gomez and Moens [2012] imported it in text classification and obtained high performance. Unfortunately, the same issue happens to them as to the methods for feature selection, that is, the two processes (reduction and clustering) are isolated from each other. Thus, the way that treats clustering and dimension reduction or feature selection as two isolated processes does not definitely obtain prominent results.

Aiming at solving the drawbacks of traditional algorithms, this paper proposes an algorithm by combing vector reconstruction and text clustering together.

3 Clustering Process of VRCA

Due to its simple structure, *vector space model* is extensively used to organize texts and clusters as vectors. For simplicity, this model is also used in our algorithm. However, this model imports many useless features and depresses clustering performance very much. Therefore, if we can define a threshold, and only choose features that are useful to represent cluster to reconstruct cluster's vector, clustering performance should be improved. Our algorithm designs two processes to achieve this goal. In detail, text clustering is conducted in partial tuning sub-process, where texts are partitioned into clusters to reduce one predefined optimization function, and cluster's vector is reconstructed in overall tuning sub-process, where no more than three hundred features that are useful to represent cluster are chosen to reconstruct cluster's vector.

3.1 Partial Tuning Sub-Process

In this sub-process, texts are partitioned into clusters to reduce one predefined optimization function as

$$op\_func = \sum_{d_j \in D; c_i \in C} \frac{\sum_{l=1}^{v} V_{d_j}(l) V_{c_i}(l)}{\sqrt{\sum_{l=1}^{v} V_{d_j}(l)^2} \sqrt{\sum_{l=1}^{v} V_{c_i}(l)^2}} \quad (1)$$

in which D denotes text collection and d_j denotes one text in D; C denotes cluster set and c_i denotes one cluster in C; V_{dj} denotes the text vector of d_j and V_{ci} denotes the cluster's vector of c_i; v denotes vector dimension; $V_{dj}(l)$ and $V_{ni}(l)$ denote the weights of lth entry in V_{dj} and in V_{ci}. Eq.1 is extensively used to control clustering process. As indicated later that our algorithm employs iterative tuning process from *self-organizing-mapping* (SOM) algorithms to partition texts, we also adopt the convergence condition used in [Alahakoon *et al*., 2000] to stop our clustering algorithm. That condition is when the difference between two values obtained by Eq.1 through two successive steps falls below 0.0001, cluster partitions obtained by these two successive steps are very close. Then, we can stop clustering algorithm.

As indicated in [Kohonen *et al*., 2000], in SOM algorithms, neuron can be seen as cluster's center (in *vector space model*, cluster's center is represented by vector, so it is also called cluster's vector). This kind of algorithms organizes neuron (or cluster's vector) into topology, and via its iterative tuning process, neurons (or clusters) are partitioned well and one neuron is more similar to its adjacent neurons than to the other neurons. This characteristic is similar to neuron in human's brain. Therefore, in SOM, cluster's vector is called neuron. In our algorithm, we just borrow the iterative tuning process from SOM algorithms to fine-tune cluster partition. However, traditional SOM algorithms employ concurrence based similarity measurements to calculate similarity between text and neuron, and also never consider removing useless features. Therefore, their performances are lower than our algorithm (*VRCA*), which can be clearly seen from experimental results (Table 2 and Table 3).

In text clustering, it is difficult to predefine cluster number. Therefore, we employ the method used in [Andreas *et al*., 2002]. If convergence condition is not met after running iterative tuning process once, each neuron's cohesion is then measured by Eq.2 and one neuron will be created by Eq.3 to make cluster partition approach to convergence condition.

$$op\_func(n_i) = \sum_{d_j \in n_i(c_i)} \frac{\sum_{l=1}^{v} V_{d_j}(l) V_{n_i}(l)}{\sqrt{\sum_{l=1}^{v} V_{d_j}(l)^2} \sqrt{\sum_{l=1}^{v} V_{n_i}(l)^2}} \quad (2)$$

in which n_i denotes the neuron corresponding to the cluster c_i; V_{ni} denotes the neuron vector of n_i ; it is also the cluster's vector of c_i; d_j denotes one text included by the cluster c_i.

After measuring neuron's cohesion, the neuron of the least cohesion (n_e) and its one neighbor (n_d) are chosen to create one new neuron (n_r) as

$$V_{\substack{n_r \\ l=1..v}}(l) = \frac{V_{n_e}(l) + V_{n_d}(l)}{2} \quad (3)$$

in which $V_{nr}(l)$ is the mean between $V_{ne}(l)$ and $V_{nd}(l)$.

Traditional SOM algorithms apply concurrence based similarity measurements to calculate similarity between text and neuron. In this kind of measurements, similarity result is affected by each feature's weight. However, in text clustering, the features that can better represent the topic of one cluster are really useful in similarity calculation. The other features not only prolong running time but also depress clustering performance. Due to this reason, we use intersection between text and neuron to measure similarity between them. In this measurement, only the intersected features that are assigned to non-zero weights in both text vector and neuron vector are considered as

$$Sim(d_j, n_i) = \frac{\sum_{l=1}^{\max(q,p)} V_{d_j}(l) V_{n_i}(l)}{\sqrt{\sum_{l=1}^{q} V_{d_j}(l)^2} \sqrt{\sum_{l=1}^{p} V_{n_i}(l)^2}} * \log(r) \quad (4)$$

in which n_i denotes ith neuron in neuron set; d_j denotes jth text in text collection; f_l denotes lth feature and resides at lth entry; V_{ni} denotes the neuron vector formed from n_i; V_{dj} denotes the text vector formed from d_j; $V_{dj}(l)$ denotes the weight of lth entry in V_{dj}, evaluated by TF-IDF; $V_{ni}(l)$ denotes the weight of lth entry in V_{ni}, updated by Eq.5; r denotes the quantity of non-zero entries between V_{dj} and V_{ni}; q and p denote the quantities of non-zero entries in V_{dj} and in V_{ni}.

In overall tuning sub-process (Section 3.2), our algorithm uses two measures to weigh features and assigns the features that are capable to represent the topic of one cluster to non-zero weights. If many features are of non-zero weights in one text vector and one neuron vector, it indicates that the text and the neuron share the similar topic. They may be similar at a high probability. Eq.4 just forms according to this idea.

As shown in Eq.4, the number of entries needed to be scanned by our similarity measurement equals to $max(q,p)$. It means that time complexity of Eq.4 is $O(max(q,p))$. For q, it is quite small, since each text has few features among vector space. For p, if it is also small, our measurement will own high running speed. Fortunately, via the threshold that is set to limit the quantity of non-zero entries in overall tuning sub-process introduced in Section 3.2, the characteristic of neuron vector generated by our algorithm is sparseness.

To apply iterative tuning process of SOM, we need to tune feature's weight in neuron vector. Because our algorithm replaces concurrence based similarity measurement by intersection based similarity measurement, its neuron adjustment function also needs to be changed as

$$V_{n_i}(l)(t+1) = \begin{cases} V_{n_i}(l)(t) + a(t) * \frac{NH(t)}{dist} * \left(\frac{V_{d_j}(l) - V_{n_i}(l)}{V_{d_j}(l) + V_{n_i}(l)} \right); & If\ V_{d_j}(l) \neq 0 \,\&\&\, V_{n_i}(l) \neq 0 \\ 1.0; & If\ V_{d_j}(l) \neq 0 \,\&\&\, V_{n_i}(l) = 0 \end{cases} \quad (5)$$

in which $V_{ni}(l)(t+1)$ and $V_{ni}(l)(t)$ are the weights of the feature f_l in neuron vector V_{ni} after and before neuron adjustment; $a(t)$ denotes learning rate and descends along with tuning process; $dist$ and $NH(t)$ denote distance function and neighborhood range, which form the adjusting range. The neurons in that range all need to be adjusted [Alahakoon *et al*., 2000].

For Eq.5, if two weights of one feature (e.g. f_l) are both non-zero in two vectors formed from one text and one neuron (e.g. d_j and n_i), the upper sub-equation is applied to adjust the weight of f_l in n_i. It lets the neuron n_i approach to the text d_j. Because our algorithm reconstructs cluster's vector, many features are assigned to zero weights. Then, it is easy to occur that some features' weights are non-zero in text vector and zero in neuron (or cluster's) vector. These features are assigned to an initial weight by the lower sub-equation.

3.2 Overall Tuning Sub-Process

In partial tuning sub-process, neuron vector is tuned by an iterative tuning process. However, this process may bring in some features that are useless to represent cluster. To remove them, our algorithm adds an overall tuning sub-process.

In text clustering, the quantity of features that are capable to represent the topic of one cluster is much smaller than the dimension of vector space [Hammouda and Kamel, 2004]. For example, if there is one cluster whose topic is about "*sport*", to represent it, it only needs the features whose meanings are relevant to "*sport*". Thus, we can define a threshold to limit the number of features in cluster's vector to represent this cluster. Based on the analysis in [Martin *et al*., 2004; Liu *et al*., 2014] and our experimental results (Figure 1 and Table 1), this threshold is no more than 300.

Since partial tuning sub-process may bring in some features that are useless to represent cluster, we design two measures to weigh feature's ability as shown in Eq.6 and Eq.8. These two equations respectively measure feature's ability to represent one cluster and the ability to separate one cluster from the others. The features of less ability will be assigned to zero weights to be removed from cluster's vector.

Feature's intra-cluster representative ability is measured by Eq.6 to show how one feature can represent one cluster. In other words, this ability stands for feature's capacity to aggregate similar texts into a compact cluster.

$$InTraC\_M(f_l, n_i) = \sum_{u=1}^{|c_i|} \sum_{m=1 \&\& m \neq u}^{|c_i|} \left(\frac{V_{d_u}(l) - V_{d_m}(l)}{V_{d_u}(l) + V_{d_m}(l)} \right)^2 + \sum_{u=1}^{|c_i|} \left(\frac{V_{d_u}(l) - V_{n_i}(l)}{V_{d_u}(l) + V_{n_i}(l)} \right)^2 \quad (6)$$

in which c_i denotes one text cluster, which includes the texts that are more similar to the neuron n_i than to the other neurons; $|c_i|$ denotes the amount of texts included by c_i. The other symbols are already explained in Eq.1.

There are two parts in Eq.6 separated by a plus. The left part is calculated based on the fact that if one feature (e.g. f_l) is assigned to similar weights among the texts included by one cluster (e.g. c_i), f_l can help aggregate similar texts into c_i. For the right part, since our algorithm is just based on traditional SOM algorithms, neuron in our algorithm also represents cluster's center. Then, if the weight of f_l in each text included by c_i is similar to the weight of f_l in n_i, f_l can help assemble similar texts around the center of c_i.

In fact, if one feature has larger intra-cluster representative ability, it will be assigned to smaller value by Eq.6. Thus, Eq.6 is reversed to

$$InTraC\_Max = \max_b \left(InTraC\_M(f_l, n_b) \right);$$
$$InTraC(f_l, n_i) = InTraC\_Max - InTraC\_M(f_l, n_i) \quad (7)$$

The equation used to measure feature's inter-cluster discriminable ability to separate one cluster from the others is shown as

$$InTerD(f_l, n_i) = \sum_{b=1 \&\& b \neq i}^{k} \left(\frac{V_{n_i}(l) - V_{n_b}(l)}{V_{n_i}(l) + V_{n_b}(l)} \right)^2 \quad (8)$$

in which k denotes the quantity of neurons.

In traditional SOM algorithms, the weight of one feature in one neuron vector is the mean weight of that feature among the texts mapped to that neuron [Kohonen *et al*., 2000]. Since our algorithm employs iterative tuning process from SOM algorithms, our algorithm also obeys this principle. Then, if one feature has distinct weights among most of neuron vectors, this feature can help separate clusters apart. Eq.8 just forms according to this idea.

3.3 Algorithm Workflow and Related Analyses

Algorithm Workflow

***Input*:** text set D; maximum neuron (or cluster) number k; current neuron (or cluster) number ck; convergence condition $m_{ax}th$; iterative index t; the quantity of steps to enter into overall tuning sub-process $m_{ax}t$; the threshold to limit the selected features $m_{ax}p$.
***Output*:** neuron (or cluster) set N (or C, in our algorithm, one neuron corresponds to one cluster).
***Initialization*:**
1. Initialize t=0;
2. Select ink random texts from D, ink<<k, and treat these ink texts as ink initial neurons;
3. ck=ink.
***Partial Tuning Sub-Process*:**
4. Select one random text d_j from D;
5. Use Eq.4 to calculate similarity between d_j and each neuron in N;
6. Select the neuron with the maximum similarity, and mark it as n_i;
7. Use Eq.5 to adjust n_i and its adjacent neurons in N;
8. Check whether the difference between two values obtained by Eq.1 at step t and step t-1 is below $m_{ax}th$ or not; if yes, stop;
9. Calculate the cohesion of each neuron by Eq.2;
10. Select the neuron that has the least cohesion, and mark it as n_e;
11. Select the neuron that has the least cohesion in the neighbors of n_e, and mark it as n_d;
12. Insert one new neuron between n_e and n_d by Eq.3;
13. ck=ck+1;
14. Check whether ck is larger than k or not, if yes stop.
15. t=t+1;
16. Check whether t is the integral times of $m_{ax}t$ or not; if yes, go to step 17 to enter into overall tuning sub-process; else, go to step 4.
***Overall Tuning Sub-Process*:**
For each neuron n_i
  For each feature f_l in n_i
    17. Use Eq.7 to measure intra-cluster representative ability of f_l;
    18. Use Eq.8 to measure inter-cluster discriminable ability of f_l;
    19. Add these two abilities together;
  End For
End For
20. Sort features in terms of their abilities, and choose $m_{ax}p$ features of larger abilities to reconstruct cluster's vector.
21. Go to step 4 to enter into partial tuning sub-process.

Parameter Setting

The previous workflow has five initial parameters needed to be fixed in advance. They are k, ink, $m_{ax}th$, $m_{ax}t$, $m_{ax}p$.

For k, it denotes the maximum cluster number. We set it as the square root of n. As indicated in [Liang *et al*., 2012], cluster number is often smaller than the square root of the size of text set. When k exceeds this number, cluster partition becomes incredible. We can stop clustering process. For ink, it denotes the initial neuron number. We set it as 2 without loss of generality. For $m_{ax}th$ and $m_{ax}t$, they respectively denote convergence condition and the quantity of steps during partial tuning sub-process. We set them according to the parameter setting in [Alahakoon *et al*., 2000]. The reason is that our algorithm is just based on SOM. Then, we can borrow the parameter setting from SOM algorithms. Besides, as stated in [Alahakoon *et al*., 2000], the setting on $m_{ax}th$ and $m_{ax}t$ does not deeply affect clustering results. For $m_{ax}p$, it denotes the threshold to limit the number of selected features to reconstruct cluster's vector. We set it as 300 according to the analysis in [Martin *et al*., 2004; Liu *et al*., 2014] and our experimental results in Section 4.1 (Figure 1 and Table 1).

Time Complexity

The workflow is partitioned into three parts: *initialization*, *partial tuning sub-process*, and *overall tuning sub-process*.

With respect to *initialization*, time complexity of steps 1, 3 is $O(1)$; time complexity of 2 is $O(ink)$; because ink is much smaller than k, time complexity of 2 is at most $O(k)$. Therefore, time complexity of initialization is $O(1+k)=O(k)$.

With respect to *partial tuning sub-process*, time complexity of steps 4, 12 to 16 is $O(1)$; time complexity of 5, 7 is $O(kmax(m_{ax}q,m_{ax}p))$, in which $m_{ax}q$ denotes the maximum quantity of non-zero entries in text vector, $m_{ax}p$ denotes the maximum number of selected features to reconstruct cluster's vector; because text vector is inevitably sparse, $m_{ax}p$ is often larger than $m_{ax}q$; then, $O(kmax(m_{ax}p,m_{ax}q))$ equals to $O(km_{ax}p)$; time complexity of 8, 9 is $O(knm_{ax}p)$; time complexity of 6, 10 is $O(ck)$; since ck<k, time complexity of 6, 10 is at most $O(k)$; time complexity of 11 is $O(nb)$, in which nb denotes the number of neighbors around one neuron; because nb is also smaller than k, time complexity of 11 is at most $O(k)$. Therefore, time complexity of partial tuning sub-process is $O(1+k+km_{ax}p+knm_{ax}p)=O(knm_{ax}p)$.

With respect to *overall tuning sub-process*, time complexity of step 17 is $O(m_{ax}c_i)$, in which $m_{ax}c_i$ denotes the maximum quantity of texts included by one cluster (e.g. c_i); since $m_{ax}c_i$<n, time complexity of 17 is at most $O(n)$; time complexity of 18 is $O(k)$; since k<n (cluster number k is smaller than the size of text set n), time complexity of 18 is at most $O(n)$; time complexity of 19 is O(1); these three steps roll for $O(km_{ax}p)$ times; then, time complexity of this loop is $O(knm_{ax}p)$; step 20 is ranking step, thus time complexity of it is $O(m_{ax}plogm_{ax}p)$; time complexity of step 21 is $O(1)$. Therefore, time complexity of overall tuning sub-process is $O(knm_{ax}p+m_{ax}plogm_{ax}p+1)=O(knm_{ax}p+m_{ax}plogm_{ax}p)$.

In conclusion, for *initialization*, it only needs to run once; for *partial tuning sub-process*, supposing that it runs for about l times to converge, then for overall tuning sub-process, it runs for $l/m_{ax}t$ times. Thereby, the total time complexity is $O(k+l*(knm_{ax}p)+l/m_{ax}t*(knm_{ax}p+m_{ax}plogm_{ax}p))=O(klnm_{ax}p+(lm_{ax}plogm_{ax}p)/m_{ax}t)$.

Taking traditional efficient clustering algorithms, such as *K-means*, *GSOM*, and *GHSOM* (their high-speed abilities can be seen from Table 3), for comparison, their time complexity is linear, $O(klnm_{ax}p)$ [Shahpurkar and Sundareshan, 2004]. The differences between them and our algorithm are two points. One is that $m_{ax}p$ in our algorithm is quite smaller than that in them, since our algorithm limits the number of selected features. The other is that our algorithm has an additional part, $O(lm_{ax}plogm_{ax}p)$. However, because $m_{ax}p$ in our algorithm is set as constant, this part can be neglected. Due to these two reasons, our algorithm should possess higher running velocity than traditional clustering algorithms, which can also be observed from experimental results (Table 3).

4 Experiments and Analyses

The following experiments are partitioned into three parts. The first part discusses how to set threshold (denoted as $m_{ax}p$) to choose features to reconstruct cluster's vector. The second and the third parts are conducted to compare our algorithm with several popular baseline algorithms in time performance and clustering precision to demonstrate our algorithm's high quality. The baseline algorithms include *K-means*, *STING*, *DBSCAN*, *BIRCH*, *GSOM*, *GHSOM*, *Spectral Clustering*, *Non-Regression Matrix Factorization* (*NRMF*), and *LDCC*. Four testing collections are employed in the experiments, including two prevalent testing collections, Reuters (21,578) and Newsgroup (2,000), and two large-scale text collections, TRC2 (1.8 millions), and released ClueWeb9 after removing empty and short texts (100 millions).

In order to compare the precisions of different algorithms, two metrics are employed. They are *NMI* (Normalized Mutual Information) and *ARI* (Adjusted Random Index) respectively introduced in [Zhao and Karypis, 2002] and [Hubert and Arabie, 1985].

NMI is calculated in Eq.9. It is calculated based on entropy. Due to the reason that *NMI* does not need to predefine cluster number and utilizes inter-cluster distinctness and intra-cluster agglomeration to measure clustering precision, it is extensively applied in clustering evaluation scenario.

$$NMI = \frac{\sum_{r=1}^{k}\sum_{q=1}^{k} n_r^q / n * \log\left(n * n_r^q / n_r * n^q\right)}{\left[\sum_{r=1}^{k} n_r / n * \log\left(n_r / n\right) + \sum_{q=1}^{k} n^q / n * \log\left(n^q / n\right)\right] / 2} \quad (9)$$

in which S_r denotes rth cluster obtained by clustering algorithm; n_r denotes the quantity of texts included by S_r; C_q denotes qth cluster predefined by user; n^q denotes the quantity of texts included by C_q; $n^q{}_r$ denotes the quantity of texts included by the intersection between S_r and C_q; n denotes set size; k denotes cluster number.

Different from *NMI* that needs to relate the labels indicating the clusters obtained by clustering algorithm to the labels predefined by user, *ARI* frees from predefining cluster partition. It combines two texts as one pair-point, and uses four situations to identify clustering results:

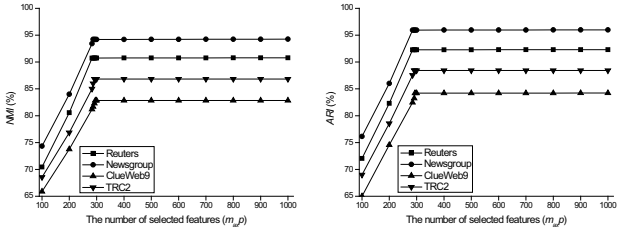
a) Two texts in one pair-point are manually labeled in the same cluster, and in clustering results they are also in the same cluster. *b*) Two texts in one pair-point are manually labeled in the same cluster, and in clustering results they are in different clusters. *c*) Two texts in one pair-point are manually labeled in different clusters, and in clustering results they are also in different clusters. *d*) Two texts in one pair-point are manually labeled in different clusters, and in clustering results they are in the same cluster.

Apparently, *a* and *c* stand for the rightly partitioned pair-points, and can be used to measure clustering precision as

$$ARI = \frac{\binom{n}{2}(a+c) - \left[(a+b)(a+d)+(b+c)(c+d)\right]}{\binom{n}{2}^2 - \left[(a+b)(a+d)+(b+c)(c+d)\right]} \quad (10)$$

4.1 The Number of Selected Features

In our algorithm, we need to set a threshold, labeled as $m_{ax}p$, and choose features less than $m_{ax}p$ to reconstruct cluster's vector. Figure 1 shows how $m_{ax}p$ affects clustering precision on the four testing collections. For clarity, Table 1 shows the value of $m_{ax}p$ when precision in Figure 1 becomes stable.



(*a*) *NMI* (*b*) *ARI*

Figure 1. Clustering precisions of different $m_{ax}p$

Table 1. $m_{ax}p$ in Figure 1 when precision becomes stable

	Reuters	Newsgroup	ClueWeb9	TRC2
NMI (*a*)	283	287	293	292
ARI (*b*)	285	285	294	291

One phenomenon can be observed from Figure 1. That is all the precision curves in Figure 1 own the same track. They all climb along with the increase of $m_{ax}p$ at the beginning, and when $m_{ax}p$ is beyond certain value (a little smaller than 300) they become stable. The reason to this phenomenon is that in our algorithm, we measure features via their abilities to represent cluster. When we increase the number of selected features ($m_{ax}p$), more useful features will be imported to reconstruct cluster's vector and therefore increase clustering precision. However, when this number exceeds certain value, the imported features are useless to represent cluster. Fortunately, in our algorithm, the useless features will be assigned to zero weights. Therefore, precision curves become stable, but importing many useless features obviously prolongs running time. By combining Figure 1 and Table 1, it can be observed that when clustering precision becomes stable, different collections correspond to similar $m_{ax}p$. Among these four collections, the size of TRC2 is about one thousand times of the size of Newsgroup, while $m_{ax}p$ on them is very close. The reason to this phenomenon is that the texts included by one cluster should reflect one similar topic. Then, only the semantically similar features that are related to the topic are really useful to represent this cluster. The number of those semantically similar features only occupies a small proportion of vector space, and does not increase much when text collection enlarges. Based on Table 1 and several previous literatures [Martin *et al*., 2004; Liu *et al*., 2014], this number seldom exceeds 300. Therefore, we set $m_{ax}p$ as 300 to be the threshold to limit the number of selected features to reconstruct cluster's vector.

4.2 Clustering Precision

We record clustering precisions of different algorithms measured by *NMI* and *ARI* on Reuters, Newsgroup, ClueWeb9, and TRC2 in Table 2.

Table 2. Clustering precisions of different algorithms

		Reuters	Newsgroup	ClueWeb9	TRC2
K-means	*NMI*	74.65	86.65	55.43	66.58
	ARI	76.53	88.13	57.02	68.11
STING	*NMI*	75.11	88.13	57.15	69.67
	ARI	76.92	89.54	58.87	71.33
DBSCAN	*NMI*	73.82	85.03	56.16	68.17
	ARI	75.33	86.43	57.73	69.84
BIRTH	*NMI*	75.34	87.91	50.36	65.05
	ARI	77.29	89.31	52.17	66.92
GSOM	*NMI*	80.32	91.47	68.29	74.51
	ARI	82.16	93.34	70.04	76.35
GHSOM	*NMI*	81.47	92.80	69.28	75.79
	ARI	83.57	94.27	70.88	77.12
SPECTRAL	*NMI*	83.72	93.67	70.45	77.24
	ARI	85.44	95.12	71.87	78.71
NRMF	*NMI*	86.50	93.68	73.73	78.82
	ARI	88.48	95.26	75.35	80.58
LDCC	*NMI*	88.33	94.06	74.22	81.34
	ARI	90.43	95.67	75.78	82.93
VRCA	*NMI*	90.75	94.22	82.83	86.84
	ARI	92.27	95.95	84.21	88.43

By observing Table 2, one conclusion can be drawn. That is clustering precisions of *VRCA* on small-scale and large-scale text collections are both outstanding, while the other clustering algorithms all fail to cluster large-scale text collection. They cannot copy their high quality from small-scale text collection to large-scale text collection. Due to space transition executed by SOM, clustering precisions of *GSOM* and *GHSOM* do not drop greatly on large-scale text collection. Besides, together with dimension reduction, *Spectral Clustering*, *NRMF*, and *LDCC* can achieve much higher precisions than *GSOM* and *GHSOM*. Nevertheless, in comparison with *VRCA*, their precisions are also too weak, since they keep many useless features in cluster's vector. These features decrease clustering precision. For *K-means*, *STING*, *DBSCAN*, and *BIRTH*, they not only run in uncompressed space, but also do nothing to remove useless features from cluster's vector. Their precisions are the lowest.

4.3 Time Performance

Due to the high-dimensional vectors generated by *vector space model*, traditional clustering algorithms are time-consuming on clustering large-scale text collection. In order to demonstrate our algorithm's high efficiency, we record running time of different algorithms in Table 3.

Two observations can be found from Table 3: (*a*) *VRCA* spends the least time on ClueWeb9, TRC2, and spends the second least time on Reuters, while *VRCA* only ranks the fourth on Newsgroup. By contrast, on Reuters and Newsgroup, *K-means* has the best time performance, and ranks the second on ClueWeb9 and TRC2; (*b*) all the algorithms spend similar amount of time when text collection is small, while when it enlarges, *VRCA* spends extremely less time. The reason to this phenomenon mainly attributes to *VRCA*'s reconstructive plan and its fast intersection based similarity measurement, where only the features that are useful to represent cluster are adopted in similarity calculation. In comparison with *VRCA*, *K-means*, *STING*, *DBSCAN*, *BIRTH*, *GSOM*, and *GHSOM* all generate high-dimensional vectors, and consequently run much slower. Since *K-means*, *GSOM*, *GHSOM* own linear time complexity, and *STING*, *DBSCAN* run by space partition, their running time is less than the others. However, due to the high-dimensional vectors, their time performances are lower than *VRCA*. In comparison with the pervious baseline algorithms, the vectors generated by *Spectral Clustering*, *NRMF*, and *LDCC* are much denser and shorter, whereas they need to perform an additional dimension reduction process. Thus, they are also time-consuming. *BIRTH* is one kind of hierarchical clustering algorithms, thus it has square time complexity and spends the most time.

Table 3. Running time of different algorithms

	Reuters	Newsgroup	ClueWeb9	TRC2
	time (sec)	time (sec)	time (min)	time (min)
K-means	16.4	4.5	359.7	42.3
STING	32.2	7.7	648.6	78.1
DBSCAN	37.8	10.2	1156.2	106.4
BIRTH	91.3	20.1	4223.0	394.6
GSOM	34.0	7.3	994.5	89.1
GHSOM	40.5	12.6	1336.5	134.6
SPECTRAL	58.3	11.8	1541.4	157.9
NRMF	67.6	12.1	1825.8	211.3
LDCC	80.9	18.2	2147.1	254.8
VRCA	19.7	7.9	124.2	22.5

5 Conclusions

Because of the high-dimensional problem aroused by *vector space model* and time-consuming issue brought from concurrence based similarity measurement, most of traditional clustering algorithms fail to cluster large-scale text collection. Facing the massive amount of web texts, only if clustering algorithms can efficiently handle large-scale text collection, they can be used in real applications. Thus, this paper proposes a clustering algorithm particularly for large-scale texts. This algorithm consists of two sub-processes and alternately repeats them until convergence. The two sub-processes are: partial tuning sub-process and overall tuning sub-process. In the former sub-process, an iterative tuning process based on SOM is adopted to tune neuron vector. Besides, an intersection based similarity measurement is also implemented. In the latter sub-process, two measures are calculated to choose features to reconstruct cluster's vector. Through comparing our algorithm with the other nine popular clustering algorithms, it demonstrates that our algorithm performs well on both small-scale and large-scale text collections.

Acknowledgments

This research is supported by National Natural Science Foundation of China (No. 61300114), Specialized Research Fund for the Doctoral Program of Higher Education (No.20132302120047), the Special Financial Grant from the China Postdoctoral Science Foundation (No.2014T70340), CCF-Tencent Open Fund.

References

[Alahakoon et al., 2000] Alahakoon D., Halgamuge S. K., and Srinivasan B. Dynamic self-organizing maps with controlled growth for knowledge discovery. *IEEE Transactions on Neural Networks*, 11: 601-614, 2000.

[Andreas et al., 2002] Andreas R., Dieter M., and Michael D. The growing hierarchical self-organizing map: Exploratory analysis of high-dimensional data. *IEEE Transactions on Neural Networks*, 13: 1331-1341, 2002.

[Cai et al., 2008] Cai D., He X. F., and Han J. W. SRDA: An efficient algorithm for large-scale discriminant analysis. *IEEE Transactions on Knowledge and Data Engineering*, 20: 1-12, 2008.

[Gomez and Moens, 2012] Gomez J. C., and Moens M. F. PCA document reconstruction for email classification. *Computational Statistics & Data Analysis*, 56: 741-751, 2012.

[Hubert and Arabie, 1985] Hubert L., and Arabie P. Comparing partitions. *Journal of Classification*, 2: 193-218, 1985.

[Hammouda and Kamel, 2004] Hammouda K. M., and Kamel M. S. Efficient phrase-based document indexing for web document clustering. *IEEE Transactions on Knowledge and Data Engineering*, 16: 1279-1296, 2004.

[Kohonen et al., 2000] Kohonen T., Kaski S., Lagus K., Salojärvi J., Honkela J., Paatero V., and Saarela A. Self organization of a massive document collection. *IEEE Transactions on Neural Networks*, 11: 574-585, 2000.

[Kwak and Choi, 2002] Kwak N., and Choi C. H. Input feature selection by mutual information based on parzen window. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24: 1667-1671, 2002.

[Liang et al., 2012] Liang J. Y., Zhao X. W., Li D. Y., Cao F. Y., and Dang C. Y. Determining the number of clusters using information entropy for mixed data. *Pattern Recognition*, 45: 2251-2265, 2012.

[Liu et al., 2013] Liu M., Liu Y. C., Liu B. Q., and Lin L. Probability-based text clustering algorithm by alternately repeating two operations. *Journal of Information Science*, 39: 372-383, 2013.

[Liu et al., 2014] Liu M., Wu C., and Liu Y. C. Weight evaluation for features via constrained data-pairs. *Information Sciences*, 282: 70-91, 2014.

[MacQueen, 1967] MacQueen J. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281-297, 1967.

[Martin et al., 1996] Martin E., Hans P. K., Jörg S., and Xu X. W. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, Pages 226-231, 1996.

[Martin et al., 2004] Martin H. C. L., Mario A. T. F., and Anil K. J. Simultaneous feature selection and clustering using mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26: 1154-1166, 2004.

[Shahpurkar and Sundareshan, 2004] Shahpurkar S. S., and Sundareshan M. K. Comparison of self-organizing map with k-means hierarchical clustering for bioinformatics applications. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1221-1226, 2004.

[Shafiei and Milios, 2006] Shafiei M. M., and Milios E. E. Latent dirichlet co-clustering. In *Proceedings of the 6th IEEE International Conference on Data Mining*, pages 542-551, 2006.

[Sakar and Kursun, 2012] Sakar C. O., and Kursun O. A method for combining mutual information and canonical correlation analysis: Predictive mutual information and its use in feature selection. *Expert Systems with Applications*, 39: 3333-3344, 2012.

[Todd and Michael, 1997] Todd A. L., and Michael W. B. Large-scale information retrieval with latent semantic indexing. *Information Sciences*, 100: 105-137, 1997.

[Wang et al., 1997] Wang W., Yang J., and Richard R. M. STING: A statistical information grid approach to spatial data mining. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 186-195, 1997.

[Wang et al., 2007] Wang F., Zhang C. S., and Li T. Regularized clustering for documents. In *Proceedings of the 30th International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 95-102, 2007.

[Xu and Wunsch, 2005] Xu R., and Wunsch D. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16: 645-678, 2005.

[Zhang et al., 1996] Zhang T., Raghu R., and Miron L. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 103-114, 1996.

[Zhao and Karypis, 2002] Zhao Y., and Karypis G. *Criterion functions for document clustering: Experiments and analysis*. Technical Report, #01-40, University of Minnesota, 2002.

[Zhao et al., 2010] Zhao F., Jiao L. C., Liu H. Q., Gao X. B., and Gong M. G. Spectral clustering with eigenvector selection based on entropy ranking. *Neurocomputing*, 73: 1704-1717, 2010.