J Supercomput (2014) 70:1249–1259
DOI 10.1007/s11227-014-1225-7

Optimized big data K-means clustering using MapReduce

Xiaoli Cui · Pingfei Zhu · Xin Yang · Keqiu Li · Changqing Ji

Published online: 19 June 2014

Abstract Clustering analysis is one of the most commonly used data processing algorithms. Over half a century, K-means remains the most popular clustering algorithm because of its simplicity. Recently, as data volume continues to rise, some researchers turn to MapReduce to get high performance. However, MapReduce is unsuitable for iterated algorithms owing to repeated times of restarting jobs, big data reading and shuffling. In this paper, we address the problems of processing large-scale data using K-means clustering algorithm and propose a novel processing model in MapReduce to eliminate the iteration dependence and obtain high performance. We analyze and implement our idea. Extensive experiments on our cluster demonstrate that our proposed methods are efficient, robust and scalable.

Keywords K-means · MapReduce · Sampling · Performance

1 Introduction

Clustering data is a fundamental problem in a variety of areas of computer science and related fields, such as machine learning, data mining, pattern recognition, etc.

X. Cui (✉) · X. Yang · K. Li
School of Computer Science and Technology, Dalian University of Technology, Dalian 116024, China
e-mail: microcui@foxmail.com

X. Yang
e-mail: xinyang@dlut.edu.cn

P. Zhu
Beijing China-Power Information Technology Co., Ltd., State Grid Electric Power Research Institute, Beijing 100192, China

C. Ji
School of Physical Science and Technology, Dalian University, Dalian 116600, China

Meanwhile, many complex algorithms also use clustering to do preprocess such as data partitioning, index creating, etc.

In recent years, cluster analysis has undergone vigorous development. There are several types of clustering [1,2], hierarchical clustering, density-based clustering, grid-based clustering, model-based clustering and partitional clustering. Each clustering type has its own style and optimization methods. In this paper, we major in the efficiency optimization of partitional clustering algorithm, K-means, to get high clustering performance.

As we know, solving clustering problem exactly is NP-hard, even with just two clusters [3]. Regarding the expansion of the data size and the limitation of a single machine, a natural solution is to consider parallelism in a distributed computational environment. MapReduce [4] is a programming framework for processing large-scale datasets by exploiting the parallelism among a cluster of computing nodes. MapReduce gains popularity for its simplicity, flexibility, fault tolerance and scalability soon after its birth. It is appreciable that some researchers use MapReduce for big data clustering [5,6].

In [7], Weizhong Zhao and his colleagues proposed parallel Kmeans clustering using MapReduce and gave a detailed description for the algorithm, and Alina Ene et al. [8] give the first analysis that shows several partitional clustering algorithms in MapReduce. However, not all big data processing problems can be efficient by parallelism; a research shows that partitional clustering algorithm requires exponentially many iterations [9]. Meanwhile, job exponential creation time and time of big data shuffling are hard to swallow especially when data size is huge, so just parallelism is not enough, only by eliminating the partitional clustering algorithms dependence on the iteration can we implement high performance.

In this paper, we propose a novel processing model in MapReduce, use sampling to eliminate the iteration dependence of K-means and obtain high performance. In particular, we summarize the contributions of the paper as follows.

Firstly, we indicate that the iteration dependence of K-means is the biggest bottleneck when we process big data in MapReduce.

Secondly, we present a novel method to optimize K-means clustering algorithms using MapReduce, which eliminates the dependence of iteration and reduces the computation cost of algorithms. The implementation defines the mapper and reducer jobs and requires no modifications to the MapReduce framework.

Thirdly, we propose two sample merging strategies for our processing model and conduct extensive experiments to study the effect of various parameters using two real datasets and one synthetic dataset. The results show that our proposed methods are efficient and scalable.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 gives details of K-means clustering in MapReduce framework and strategies for center merging. Section 4 reports the experimental results and Sect. 5 concludes the paper.

2 Related work

K-means algorithm is based on specifying an initial number of groups, and iteratively reallocates objects among groups to convergence. We are given an integer k and a set of n data points $D \subset R^d$. We wish to choose the collection of k centers C, so as to minimize the potential function

$$\varphi = \sum_{x \in D} \min_{c \in C} ||x - c||^2 . \quad (1)$$

The algorithm assigns each point to the cluster whose center is nearest. The center's coordinates are the arithmetic mean for each dimension separately over all the points in the cluster. Suppose itr is the convergent boundary, the pseudo code of Algorithm 1 is to explain how it works.

Algorithm 1: K-means(D,k)

```
1 Let $i$=Float.MAXVALUE; $j$=1
2 Choose $k$ centers from $D$, let $C^{(0)} = c_1^{(j)}, c_2^{(j)}, ..., c_k^{(j)}$
3 **while** $i > itr$ **do**
4     form $k$ clusters by assigning each points in $X$ to its nearest center
5     find new centers of the $k$ clusters $c_1^{(++j)}, c_2^{(++j)}, ..., c_k^{(++j)}$
6     $i \leftarrow \sum_{m=0}^{k} ||c_m^j - c_m^{j-1}||^2$
7 output $C^{(j)}$
```

Some researchers aim at initial phase to optimize clustering accuracy. David Arthur and Sergei Vassilvitskii [10] obtain an algorithm that is named K-means++ and O(logk)-competitive with the optimal clustering by carefully seeding, and Zhou Aiwu et al. [11] also aimed at optimizing initial clustering center to get better accuracy. let $D(x)$ denote the shortest distance from a data point x to the closest center we have already chosen, $p_x = \frac{D(x')^2}{\sum_{x \in D} D(x)^2}$, then the following algorithm is the K-means++ initialization.

(a) Choose an initial center c_1 uniformly at random from D.
(b) Choose the next center c_i, selecting $c_i = x' \in D$ with probability p_x.
(c) Repeat Step b until we have chosen a total of k centers.

Furthermore, other researchers seek various optimization methods to speed up the clustering, these method can be divided into two categories. The methods which produce approximate solution, and the methods which produce the same solution as obtained using the conventional K-means clustering method.

There are several approximate methods. One approach to speed up the K-means clustering method is by bootstrap averaging [12]. Later Farnstrom et al. [13] improved this idea to speed up K-means method. Domingos et al. [14] proposed a fast K-means algorithm which gives a better approximate solution, with statistically bounded loss in the clustering quality.

There are other improvements to speed up the K-means method without compromising the quality. Fahim et al. [15] used a simple structure to keep some information in each iteration to be used in the next iteration and enhanced K-means clustering algorithm. While Kanungo et al. [16] made use of kd-tree to get a filtering algorithm over K-means.

In [17], Bahmani proposed kmeans|| algorithm, which parallelized the initial phase of Kmeans++ to get higher performance and implemented it using MapReduce. kmeans|| initial algorithm uses an oversampling factor l and proceeds in constant iterations, for each iteration it samples each point x with probability lp_x, while p_x is the same as Kmeans++, then set the number of points in original dataset D closer to x than any other point as the weight of x, and recluster the weighted points into k clusters to get k initial centers.

In this paper, we focus on improving the speed of clustering of large-scale dataset using MapReduce.

3 Handing K-means using MapReduce

The major research problems for clustering big data with MapReduce are: (a) how to minimize the I/O cost; (b) how to minimize the network cost among processing nodes. In this section, we introduce our implementation of K-means optimization using MapReduce, which can reduce both I/O cost and network cost.

In the following pages, we set D as the original dataset, k as the number of the clusters we want to get, i and j are all integers, while i from 1 to $2k$ and j from 1 to k, c_i is the ith center, C_i is the collection of points belong to center c_i.

As shown in Fig. 1, MapReduce job will read disk where the large-scale datasets stored repeatedly and the large-scale data will be shuffled over the whole clusters each time. Therefore the I/O cost and network cost are very expensive.

To estimate the iteration, we come up with an idea that use sampling to get some subsets of the big data. By processing these subsets, we obtain the center sets which can be used to cluster original datasets. Figure 2 illustrates the working flow of our clustering algorithm, which consists of three MapReduce jobs. fc_i represents the final ith center, while n_i is the number of points belong to center c_i. Next, we will introduce details of it.

3.1 Probability sampling

The first MapReduce job is sample selection. We sample on the original large dataset using k and probability $p_x = \frac{1}{\varepsilon^2 N}$, where $\varepsilon \in (0,1)$ and controls the size of the sample

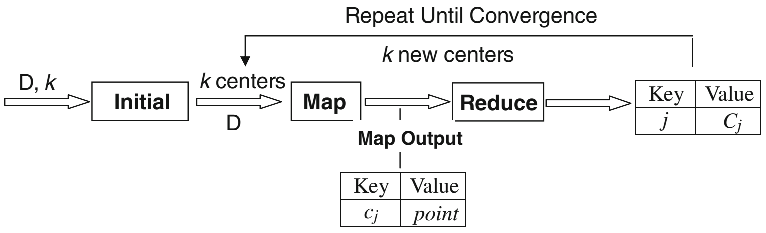


Fig. 1 Traditional K-means clustering in MapReduce

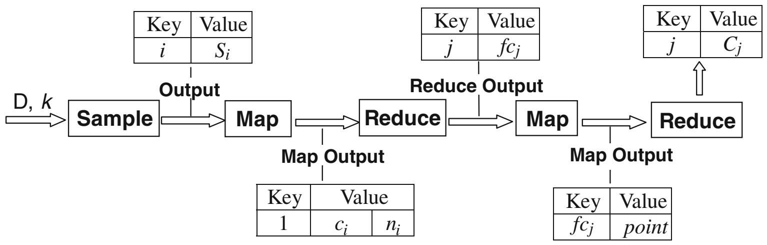


Fig. 2 An overview of optimized clustering in MapReduce

while N is the number of points, and then we generate some samples small enough to be processed in a single machine. In experiment, we set $\varepsilon = 0.005$.

Algorithm 2: Probability Sampling

```
Input  : $D, k$
Output: key: $k_i$, the sample file id;
        value: $C_i$, the collection of points for sample file $i$;
        $i$ from 1 to $2k$
1 map($k_1$,$v_1$)
2 for each mapper, for $i = 1$ to $2k$ do
3     sample each point $x$ in $D$ with probability $p_x = \frac{1}{\varepsilon^2 N}$
4     context.write($i$, $x$)
5 reduce($k_2$,$v_2$)
6 for each $v \in$ values do
7     $C_i$.add($v$)
8 output($k_i$,$C_i$)
```

One simple rule of thumb sets the number to k is with n as the number of objects, where $n \approx 2k^2$. So we prefer to produce $2k$ small-sized sample, then after clustering each sample, respectively, using k, we can get $2k^2$ center points in total and use them to generate final k centers. It is worth mentioning that if k is so large that $2k^2$ points cannot be processed in a single machine, the number of samples should be reduced according to the actual conditions. Algorithm 2 shows how to sample the input datasets using MapReduce. For each mapper we do sampling and then shuffle these samples to corresponding reducers, and finally we can obtain $2k$ samples, each of them can be processed by a single machine.

3.2 Sample clustering and merging

After sampling, we can get $2k$ small-sized sample files. In the second job, mappers cluster each sample file using k and get $2k^2$ centers in total. We shuffle these centers to only one reducer and merge them into k final centers, which are used to obtain final clustering result.

Algorithm 3 describes the detailed procedure. C_i is the collection of ith sample's cluster centers, N_i is the collection of points' number of C_i, P_i is the collection of points for each center of C_i. Specially, in the center initial step of the 2 for each map, we first choose two specified points to make sure the distance between them is largest so that the clustering result of each map is global optimal [10].

```
Algorithm 3: Sample Clustering and Merging
  Input  : $k$
           the samples of the original dataset;
  Output: $c_i$, $i$ from 1 to $k$, $k$ final centers;
1 map($k_1$,$v_1$)
2 do clustering for each sample by Kmeans++ initial strategy;
3 output(1,($C_i$, $N_i$, $P_i$))
4 reduce($k_2$,$v_2$)
5 merge sequence ($C_i$, $N_i$, $P_i$)
6 output($c_1$,$c_2$,...,$c_k$)
```

To merge $2k^2$ center points into k clusters with statistics, we present two novel merging methods which can make our algorithm more stable and general.

3.2.1 Weight-based merge clustering (WMC)

Given a situation S in which we will merge two clusters, A and B, whose centers are point c_a and point c_b, respectively, and A has m points while B has n points. If we use simple merging strategy, the new center of the merging result is the midpoint of c_a and c_b, but this is inaccurate, as a common sense, if $m > n$, the real new center of A and B will be more closer to the range of A and c_a, according to this theorem, we come up with WMC strategy.

In the second MapReduce job, after sample clustering in each mapper, we collect the number of points assigned to each center points, which will be used to weight the centers. Algorithm 4 shows the details on how to merge intermediate clustering results using WMC strategy in the reducer of the second MapReduce job.

```
Algorithm 4: Weight-based merge clustering
  Input  : $2k^2$ intermediate centers and their weight $W$;
           $c_i$, the $i$th center; $w_i$, the weight of $c_i$, $i = 1$ to $2k$
  Output: $k$ final centers;
1 reduce($k_2$,$v_2$)
2 choose $k$ centers using Kmeans++ initial strategy
3 assign input points to appropriate centers
4 compute the weighted arithmetic mean of points for each cluster using $c_i$ and $w_i$
5 repeat 2,3,4 until satisfied the convergence condition.
6 output final centers ($c_1$,$c_2$,...,$c_k$)
```

3.2.2 Distribution-based merge clustering (DMC)

As we know, with rational sampling, each sample file can represent the whole dataset very well. Meanwhile, the clustering result of each map in 2 of Algorithm 3 is global optimal by carefully seeding. So we come up with a merge idea to separate the $2k^2$ centers into k groups, each group has $2k$ centers and consists of one and only one center from each sample clustering results. Here we first select an intermediate clustering result randomly whose k centers are recognized as the first member of k different groups, and then we choose members from the rest of intermediate clustering results for each group according to the distances between centers. Algorithm 5 shows the details on our DMC strategy.

In step 5, we have to make sure that each group has one and only one center from C_i, the collection of centers from sample i, while G_i is the ith group of centers, and g_i is the center of points in G_i.

Algorithm 5: Distribution-based merge clustering

```
Input  : $2k^2$ intermediate centers and their statistical information;
Output: $k$ final centers;
1 reduce($k_2$,$v_2$)
2 choose $C_1$ randomly from samples
3 assign $k$ centers in $C_1$ to $k$ different groups
4 for $i = 2$ to $2k$ do
5     assign centers of $C_i$ to the nearest group to them
6 computer the center for each group
7 for $i = 1$ to $k$ do
8     compute the central point $g_i$ of points for $G_i$
9 output final centers ($g_1$,$g_2$,...,$g_k$)
```

Finally, we start the third MapReduce job to generate Voronoi diagram using k points we have obtained from Sect. 3.2, partition the original dataset and finally obtain the clustering result.

4 Experiments and results

In order to evaluate our optimized algorithm in practice, we present the experimental setup for evaluating. We set up a cluster of 16 Server PC machines, each of them has a Dual Core AMD Opteron 2.00 GHz CPU, 73GB hard drive, Inter 82551 10/100 Mbps Ethernet Controller, 2GB memory and Ubuntu10.10 server OS. All nodes are connected to a 100 Mbps Ethernet switch. We use Cloudera Hadoop 0.20.2 and compile the source codes under JDK 1.6.0-24 in Eclipse 3.6.2. One TaskTracker and DataNode daemon run on each slave. A single NameNode and JobTracker run on the master.

Next, we describe the datasets and baseline algorithms that will be used for comparison.

4.1 Data sets

We use three datasets to evaluate the performance of our optimized partitional clustering algorithm. The first dataset, Gauss Distribution Set, is synthetic. We generate 10,000 three-dimensional points with gauss distribution. The mean of generated points from this Gauss distribution is 350 and the sigma is 100.

The other two datasets considered are from real-world settings and are publicly available from the UC Irvine Machine Learning repository. The Bag of Words Data Set(BoW) consists of 2,351,710,420 points in 3 dimensions and represents features available to (docID, wordID, count). The Individual household electric power consumption Data Set (House) consists of 4,296,075,259 points in 9 dimensions.The Gauss dataset is studied in a single machine and the other two datasets are studied with the parallel implementation in the Hadoop framework.

4.2 Baselines

We compare the performance of our optimized K-means algorithm against the traditional K-means algorithm and Kmeans|| algorithm using MapReduce and stand-alone Kmeans++ algorithm. we use Gauss dataset in a single machine to explore the iteration times for K-means, Kmeans++ and Kmeans||, and for large-scale datasets BoW and House, it becomes infeasible and parallelization becomes necessary, the two baseline algorithms and our WMC/DMC algorithms are implemented using MapReduce in cloud environment and we tested kmeans|| with $l = 2k$ and $r = 5$.

4.3 Performance evaluation

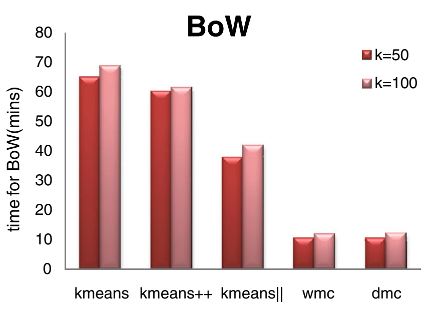
In this subsection, we describe the experimental results based on the setup in the beginning of Sect. 4. To evaluate the clustering cost of our optimized algorithm model, we compare it against the baseline approaches. For Gauss, we set $k \in \{20,50,100\}$, and for BoW and House, we set $k \in \{50,100\}$.

We first turn our attention to the convergence time of the iteration for different initializations in K-means, Kmeans++ and Kmeans||, we implement baselines in a single machine to get the number of iterations till convergence of the baselines. Table 1 shows our experimental result. From the table, we can come to a conclusion that the number of iteration for partitional algorithms is very large.

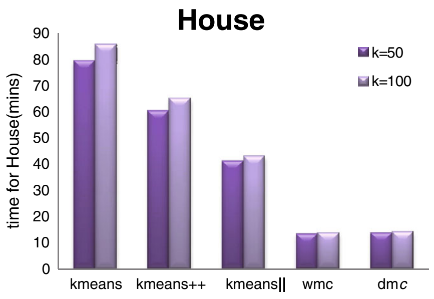
We now illustrate that our WMC/DMC model can get higher performance than K-means and Kmeans|| when implemented to run in parallel. Figure 3a, b shows the experimental result for BoW and House dataset, and we can see the total running time of these algorithms. Both WMC and DMC are much faster than baselines.

Table 1 Number of iterations till convergence for Gauss (averaged over 10 runs)

Algorithm	$k = 20$	$k = 50$	$k = 100$
K-means	32.4	63.2	54.2
Kmeans++	20.2	31.5	41.7
Kmeans||	25.7	41.3	50.6

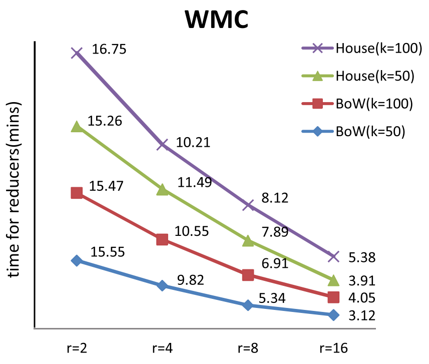


(a) Running time for BoW

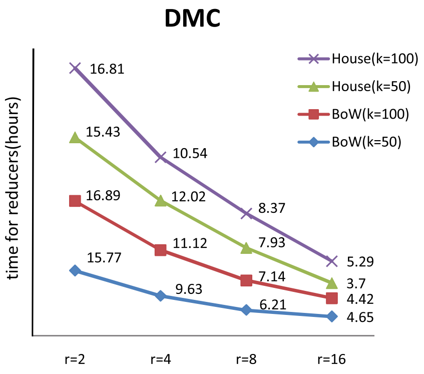


(b) Running time for House

Fig. 3 Comparison of time measured by various methods



(a) Running time forWMC



(b) Running time for DMC

Fig. 4 Running time with reducers from 2 to 16

Next, we study the efficiency of WMC/DMC clustering and baselines by varying the number of reducers, from 2 up to 16. Figure 4 illustrates the effect of node number on the time of clustering on BoW and House datasets for $k = 500$. The result shows that our optimized model is scalable.

4.4 Clustering validation

Cluster validity issued by and large concerned with determining the optimal number of clusters and checking the fineness of clustering results. Assessment of clustering results is commonly referred to as cluster validation [18]. Many different indices of cluster validity have been already proposed [19]. In this section, we discuss briefly the Davies–Bouldins index (DBI) which we have used in our proposed clustering algorithm for examining the soundness of our clusters.

Davies–Bouldins index [20] is a function of the ratio of the sum of within-cluster distribution to between-cluster separation. The distance function, dispersion measure, and characteristic vector were chosen

Table 2 Comparison of K-means, Kmeans++, Kmeans||, WMC and DMC clustering algorithms by considering DBI on different sized data sets

Algorithm	Bow ($k = 50$)	Bow ($k = 100$)	House ($k = 50$)	House ($k = 100$)
K-means	0.0185	0.0167	0.0175	0.0124
Kmeans++	0.0161	0.0152	0.0166	0.0113
Kmeans||	0.0182	0.0161	0.0176	0.0116
WMC	0.0193	0.0171	0.0186	0.0133
DMC	0.0187	0.0166	0.0179	0.0128

$$S_i = \left\{ \frac{1}{T_i} \sum_{j=1}^{T_i} |x_j - A_i|^q \right\}^{1/q}. \tag{2}$$

$$M_{ij} = \left\{ \sum_{k=1}^{N} |a_{ki} - a_{kj}|^p \right\}^{1/p}. \tag{3}$$

where T_i is the number of vectors in cluster i, A_i is the centroid of cluster i, a_{ki} is the kth component of the n-dimensional vector a_i, which is the centroid of cluster i.

Here we set $p = q = 2$, then S_i is the standard deviation of the distance of samples in a cluster to the respective cluster center, M_{ij} is the Euclidean distance between centroids and $R_{ij} \equiv \frac{S_i+S_j}{M_{ij}}$ is the reciprocal of the classic Fisher similarity measure calculated for clusters i and j. The partition with smaller $R \equiv \frac{1}{N}\sum_{i=1}^{N} R_i$ is the superior, where R_i is maximum of R_{ij} when $i \neq j$.

Table 2 gives a comparative analysis of the DBI compared result for BoW dataset and House dataset. It shows that our clustering methods produce good quality of clustering as compared to traditional K-means.

5 Conclusion

In this paper, we presented that the iteration of K-means algorithm was the important factor which affected the performance of clustering, and proposed a novel efficient parallel clustering model. Experimental results on large real-world datasets and synthetic dataset demonstrate that our optimized algorithm is efficient and performs better compared with parallel K-means, Kmeans|| and stand-alone Kmeans++ algorithms. Clustering validation shows that the quality of our clustering methods are as well as K-means.

Acknowledgments The work of this paper is supported by the National Science Foundation for Distinguished Young Scholars of China (Grant No. 61225010), NSFC under Grants 61173160, 61173161, 61173162, 61103234, 61272417, 61300084 and 61300189.

References

1. Madhulatha TS (2012) An overview on clustering methods[J]. arXiv preprint arXiv:1205.1117
2. Xu R, Wunsch D (2005) Survey of clustering algorithms[J]. IEEE Trans Neural Netw 16(3):645–678
3. Drineas P, Frieze A, Kannan R et al (2004) Clustering large graphs via the singular value decomposition[J]. Mach Learn 56(1–3):9–33
4. Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters[J]. Commun ACM 51(1):107–113
5. Ekanayake J, Pallickara S, Fox G (2008) Mapreduce for data intensive scientific analyses[C]. eScience, eScience'08. IEEE fourth international conference on. IEEE 2008, pp 277–284
6. Liu T, Rosenberg C, Rowley HA (2007) Clustering billions of images with large scale nearest neighbor search[C]. Applications of Computer Vision, WACV'07. IEEE Workshop on. IEEE 2007, p 28
7. Zhao W, Ma H, He Q (2009) Parallel k-means clustering based on mapreduce[M]., Cloud computing-Springer, Berlin Heidelberg
8. Ene A, Im S, Moseley B (2011) Fast clustering using MapReduce[C]. In: Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 681–689
9. Vattani A (2011) K-means requires exponentially many iterations even in the plane[J]. Discret Comput Geom 45(4):596–616
10. Arthur D, Vassilvitskii S (2007) k-means++: the advantages of careful seeding[C]. In: Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms. Society for industrial and applied mathematics, pp 1027–1035
11. Wang J, Su X (2011) An improved K-means clustering algorithm[C]. Communication software and networks (ICCSN), 2011 IEEE 3rd international conference on. IEEE, pp 44–46
12. Davidson I, Satyanarayana A (2003) Speeding up k-means clustering by bootstrap averaging[C]. IEEE data mining workshop on clustering large data sets
13. Farnstrom F, Lewis J, Elkan C (2000) Scalability for clustering algorithms revisited[J]. ACM SIGKDD Explor Newsl 2(1):51–57
14. Domingos P, Hulten G (2001) A general method for scaling up machine learning algorithms and its application to clustering[C]. ICML, pp 106–113
15. Fahim AM, Salem AM, Torkey FA et al (2006) An efficient enhanced k-means clustering algorithm[J]. J Zhejiang Univ Sci A 7(10):1626–1633
16. Kanungo T, Mount DM, Netanyahu NS et al (2002) An efficient k-means clustering algorithm: analysis and implementation [J]. IEEE Trans Pattern Anal Mach Intell 24(7):881–892
17. Bahmani B, Moseley B, Vattani A et al (2012) Scalable k-means++[J]. Proc VLDB Endow 5(7):622–633
18. Mishra BK, Rath A, Nayak NR et al (2012) Far efficient K-means clustering algorithm[C]. In: Proceedings of the international conference on advances in computing, communications and informatics. ACM, pp 106–110
19. Friedman HP, Rubin J (1967) On some invariant criteria for grouping data[J]. J Am Stat Assoc 62(320):1159–1178
20. Davies DL, Bouldin DW (1979) A cluster separation measure[J]. IEEE Trans Pattern Anal Mach Intell 2:224–227