K-Means Clustering with Bagging and MapReduce

Hai-Guang Li1, Gong-Qing Wu1, Xue-Gang Hu1, Jing Zhang1, Lian Li1, Xindong Wu1,2

1 School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230009, China
2 Department of Computer Science, University of Vermont, Burlington, VT 50405, U.S.A.
lee.haiguang@gmail.com, wugongqing@gmail.com, jsjxhuxg@hfut.edu.cn, hfzj870608@126.com, llian@hfut.edu.cn, xwu@cs.uvm.edu

Abstract
Clustering is one of the most widely used techniques for exploratory data analysis. Across all disciplines, from social sciences over biology to computer science, people try to get a first intuition about their data by identifying meaningful groups among the data objects. K-means is one of the most famous clustering algorithms. Its simplicity and speed allow it to run on large data sets. However, it also has several drawbacks. First, this algorithm is instable and sensitive to outliers. Second, its performance will be inefficient when dealing with large data sets. In this paper, a method is proposed to solve those problems, which uses an ensemble learning method bagging to overcome the instability and sensitivity to outliers, while using a distributed computing framework MapReduce to solve the inefficiency problem in clustering on large data sets. Extensive experiments have been performed to show that our approach is efficient.

1. Introduction
Clustering is one of the most important unsupervised learning methods [1]. A loose definition of clustering could be “the process of organizing objects into groups whose members are similar in some way” [2]. So, the goal of clustering is to determine the intrinsic groups in a set of unlabeled data. A cluster is therefore a set of objects which are “similar” between them and are “dissimilar” to the objects belonging to other clusters. Clustering has many applications in many fields such as market research, image analysis, bioinformatics, computational biology, machine learning, data mining, knowledge discovery, pattern recognition, and pattern classification.

K-means is a famous clustering algorithm, which is one of the simplest unsupervised learning algorithms, and it is usually very fast [3]. It is also one of the top-10 algorithms in data mining [4]. However, there is no guarantee that it will converge to the global optimum, since it is a heuristic algorithm, and different choices of initial clusters may produce different results.

In recent years, cluster analysis has undergone vigorous development. There are several variation algorithms of k-means: The EM algorithm maintains probabilistic assignments to clusters, instead of the deterministic assignments in k-means; K-means++ seeks to choose better starting clusters --a way to avoid the sometimes poor clusters found by the standard k-means algorithm; the filtering algorithm speeds up each k-means step by using kd-trees [5], while some other methods adopt coresets [6] or the triangle inequality [7]; and even swapping objects between clusters is adopted to escape local optima [8].

Meanwhile, there remain several problems of those variation algorithms. An important problem of wide concern is k-means’s instability and sensitivity to outliers. Furthermore, dealing with high dimensions and large data sets can be problematic because of time complexity.

In machine learning, ensembling is a technique which uses a collection of models in order to obtain better results than any model in the collection [9]. Bagging is one of the most popular ensemble techniques. MapReduce is a programming model and an associated implementation for processing and generating large data sets [10].

In order to solve those problems mentioned above, a method named MBK-means is proposed in this paper, which adopts bagging to improve the stability and accuracy of k-means, and uses MapReduce to solve the inefficiency problem in clustering on large data sets.

The main contributions of this paper can be summarized as follows:

a) To overcome the instability and the sensitivity to outliers of k-means, an ensemble learning method bagging is introduced to improve the stability and accuracy.
b) The inefficiency problem in clustering on large data sets is solved by using a distributed computing framework MapReduce.

c) Extensive experiments have been performed to show that our method is efficient in solving the problems mentioned above.

The rest of the paper is organized as follows. Section 2 discusses related work. The MBK-means algorithm is presented in detail in section 3. Section 4 shows the performance evaluation and comparative experimental results. Finally, this paper is ended with a conclusion in section 5.

2. Background

In this section, introductions about the algorithms and technologies adopted in this paper are given. An outline on k-means is given in section 2.1, bagging in section 2.2 and MapReduce in section 2.3.

2.1. K-means

James MacQueen first used the term “k-means” in 1967, though the idea can be traced back to Hugo Steinhaus in 1956 [3]. Stuart Lloyd first proposed the standard algorithm in 1957, though it was not published until 1982 [11].

K-means clustering is a cluster analysis method which aims to partition \( n \) objects into \( k \) clusters, in which each object belongs to the cluster with the nearest centroid. It attempts to find the centers of natural clusters in the data set [12].

Given a set of objects \((x_1, x_2, \ldots, x_n)\), where each object is a \( d \)-dimensional vector, the k-means clustering aims to partition the \( n \) objects into \( k \) sets \((k < n) \ S = \{S_1, S_2, \ldots, S_k\} \). The k-means algorithm is composed of the following steps:

1) Given an initial set of \( k \) means \( m_1^{(1)}, \ldots, m_k^{(1)} \), which may be specified randomly or by some heuristic.

2) Assignment step: Assign each object to the group that has the closest centroid.

\[ S_i^{(t)} = \{x_j : ||x_j - m_i^{(t)}|| \leq ||x_j - m_p^{(t)}|| \text{ for all } p = 1, \ldots, k \} \]

3) Update step: When all objects have been assigned, recalculate the positions of the \( k \) centroids.

\[ m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j \]

4) Repeat Steps 2 and 3 until the means no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

The k-means algorithm is usually effective, particularly when the clusters are compact and well-separated, but it is less effective when clusters are nested. The results produced depend on the initial values for the means, and it frequently happens that suboptimal partitions are found.

Regarding computational complexity, if \( k \) and \( d \) are fixed, the problem can be exactly solved in time \( O(n^{d+1} \log n) \), where \( n \) is the number of objects to be clustered and the \( d \) is the vector dimension of objects.

2.2. Bagging

Ensemble methods employ multiple learners and combine their predictions to solve the problem together, which can obtain better performance than any of the constituent models [3].

Bagging, a name derived from bootstrap aggregation, was proposed by Leo Breiman in 1994. It is the first effective method of ensemble learning to improve models in terms of stability and accuracy, and reduce variance and help to avoid overfitting [13]. And it also is one of the simplest methods of arching.

The method uses multiple versions of a training set by using the bootstrap, and each data set is used to train a different model. The outputs of these models are combined by averaging or voting to obtain a single output.

Bagging is a special case of model averaging, was originally designed for classification, and can be used with any type of model, although it is usually applied to decision tree models.

2.3. MapReduce

MapReduce is a programming model and an associated implementation for processing and generating large data sets, which was introduced by Google to solve certain kinds of distributable problems. Programs written in MapReduce’s functional style can be automatically parallelized and executed on a high performance computing cluster of a large number of low-cost ordinary personal computers. It is inspired by the \( map \) and \( reduce \) functions, Users specify a \( map \) function that processes a (\( key, value \)) pair to generate a set of intermediate (\( key, value \)) pairs, and a \( reduce \) function that merges all intermediate values associated with the same intermediate key [10].
The main advantage is that the map and reduce functions of MapReduce are allowed for distributed processing. Each map operation is independent from others. Therefore, all map operations can be performed in parallel, and a set of reduce operations can be performed in parallel. Similarly, a set of reduce operations can perform the reduction phase independently.

The map function, written by the user, takes one pair of data with a type in a data domain, and returns a list of pairs in the same or a different domain. It is applied to every \((k_1, v_1)\) pair in parallel, and produces a list of intermediate \((k_2, v_2)\) pairs per map invocation.

\[ Map(k_1, v_1) \rightarrow \text{list}(k_2, v_2) \]

After that, the MapReduce framework collects all intermediate \((k_2, v_2)\) pairs with the same key from all lists and groups them together, thus creating one group for each one of the different generated keys, and passes them to the reduce function.

The reduce function, also written by the user, is then applied to each group in parallel, and produces a collection of values in the same domain. It accepts an intermediate key \(k_2\) and a list of values \(v_2\) for \(k_2\). It merges together these values to form a possibly smaller list of values \(v_1\). Each reduce call typically produces either one or an empty return. The returns of all calls are collected as the desired result list.

\[ Reduce(k^2, \text{list}(v^2)) \rightarrow \text{list}(v_1) \]

Thus the MapReduce framework transforms a list of (key, value) pairs into a list of values.

Apache Hadoop is a software platform that allows the user to write and run applications that process vast amounts of data easily [15]. Hadoop implements MapReduce with the HDFS (Hadoop Distributed File System). HDFS is designed to run on commodity hardware. It has many similarities with existing distributed file systems.

3. MBK-means Algorithm

The MBK-means framework is shown in Figure 1. In step 1, it generates \(k\) new training sets \(D_i\) \((i=1,\ldots, k)\) from the given standard training set \(D\) (Section 3.1). Step 2 clusters \(D\) using k-means with the MapReduce framework (Section 3.2). After getting the centroids \(\text{centroids}_i\) of \(D_i\), we combine \(\text{centroids}_i\) \((i=1,\ldots, k)\) and update the centroids in step 3 (Section 3.3).

![Figure 1. The MBK-means framework](image)

### 3.1. Data Splitting and Distance Measure

Given a standard training set \(D\) of size \(n\), bagging generates \(m\) new training sets \(D_i\) of size \(n' \leq n\) \((i=1,\ldots, k)\), by sampling examples from \(D\) uniformly and with replacement. In this paper, the \(n'\) is set as \(n\), while \(m\) is set as \(k\), which is the number of clusters in k-means.

Since similarity is fundamental to the definition of a cluster, a measure of the similarity between two objects is essential to most clustering procedures. It is common to calculate the dissimilarity between two objects using a distance measure. Distance is a numerical description of how far apart objects are. In physics or everyday discussion, distance may refer to a physical length or estimation based on other criteria. In mathematics, a distance function or metric is a generalization of the concept of physical distance. In this paper, the Minkowski distance is used to measure the similarity or dissimilarity between two objects, which is defined as:

\[
\text{Distance}(\text{obj}_i, \text{obj}_j) = \left( \sum_{d=1}^{d} |x_{d1} - x_{d2}|^q \right)^{1/q}
\]

where \(\text{obj}_i=(x_{i1}, x_{i2}, \ldots, x_{id})\) and \(\text{obj}_j=(x_{j1}, x_{j2}, \ldots, x_{jd})\) are two \(d\)-dimensional data objects, and \(q\) is a positive integer. In this paper, \(q\) is set as 2. Therefore, the Minkowski distance evolves into the Euclidean distance. The Distance function can be calculated in \(O(d)\).
3.2. K-means with MapReduce

The k-means algorithm spends most execution time on calculating the distances between objects and cluster centroids. For very large data sets, the iterative computation of distances between objects often causes a system to overload. This process is, however, necessary for the algorithm itself, and thus we must consider about improving the efficiency of the algorithm from other aspects. Therefore, enhancing the performance of the distance calculation is the key to improve the time performance of the algorithm. It is easy to note that the execution orders of distance calculation of objects will not affect on the final result of clustering. Therefore, the distance-calculating process can be executed in parallel by using the MapReduce framework.

The k-means framework with MapReduce is shown in Figure 2 and details are as follows:

1) Place k points into the space represented by the objects that are being clustered. These points represent the centroids of initial groups.
2) Assign each object to the cluster that has the closest centroid: This step is operating on mappers showing in Figure 2, and Figure 3 outlines the k-means in_Map algorithm.
3) When all objects have been assigned, recalculate the positions of the k centroids: This step is operating on reducers showing in Figure 2, and Figure 4 outlines the k-means_in_Reduce algorithm.

Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into clusters from which the metric to be minimized can be calculated.

```
Function K-means_in_Map
Input: centroids, input. // The key of the input is the offset of the input segment in the raw text (data set), and the value of the input is an object for clustering. The centroids are raw centroids, and are given to the map function by the JobConf of MapReduce.
Output: output. // The key of output is the nearest cluster to the object, and the value of output is the object.
1: nstCentroid ← null, nstDist ← ∞
2: for each c ∈ centroids do
3:    dist ← Distance(input. value, c);
4:    if nstCentroid == null || dist < nstDist then
5:        nstCentroid ← c, nstDist ← dist;
6:    end if
7: end for
8: output.collect(nstCentroid, object);
```

```
Figure 3. K-means_in_Map algorithm
```

The map function of the k-means algorithm using MapReduce is shown in Figure 3, and this function runs on each mapper in the cluster. This function assigns the object to the nearest centroid.

This function uses a distance method to calculate the distance between the object and each centroid in clusters in step 3. After finding the nearest centroid to the object, this function puts the object into the cluster which contains the centroid in step 8.

The reduce function of the k-means algorithm using MapReduce is shown in Figure 4, and this function runs on each reducer in the cluster. This function calculates the new centroid of the cluster.

This function first gets all objects in this cluster in step 3, and then uses a recCalCentroid function to recalculate the centroid with the objects in the cluster in step 5. After getting the new centroid of the cluster, this function returns the centroid in step 6.

```
Function K-means_in_Reduce
Input: input. // The key of input is the centroid of a cluster, and the value of input is a list of objects who are assigned to the cluster.
Output: output. // The key of output is the old centroid of the cluster, and the value of output is the new centroid of the cluster.
1: v ← ∅;
2: for each obj ∈ input. value do
3:    v ← v ∪ {obj};
4: end for
```

```
Figure 4. K-means_in_Reduce algorithm
```
5: centroid ← ReCalCentroid(v);
6: output. collect(input. key, centroid);

Figure 4. K-means_in_Reduce algorithm

3.3. Ensemble

Each data set can use the k-means algorithm to get k centroids. Since our MBK-means generates k new data sets from the original data set \( D \), we can get k centroid sets from \( Centroids_{1} \) to \( Centroids_{k} \). Therefore, the k centroid sets must be merged into a finally centroid set.

Centroids\(_{1}\)  |  Centroids\(_{2}\)  |  Centroids\(_{k}\)  |  Finally Centroids
---|---|---|---
\( c_{11} \)  |  \( c_{12} \)  |  \( c_{1k} \)  |  \( c_{1} \)
\( c_{21} \)  |  \( c_{22} \)  |  \( c_{2k} \)  |  \( c_{2} \)
\( \ldots \)  |  \( \ldots \)  |  \( \ldots \)  |  \( \ldots \)
\( c_{i1} \)  |  \( c_{i2} \)  |  \( c_{ik} \)  |  \( c_{i} \)
\( c_{k1} \)  |  \( c_{k2} \)  |  \( c_{kk} \)  |  \( c_{k} \)

Figure 5. Merging process

In this paper, a simple merging algorithm is used to ensemble, which can be processed with the following steps using a greedy style.

a) Initialize the finally centroid set centroids, which records the finally k centroids of the given original data set \( D \), as empty;

b) If any centroid set centroids\(_{i}\) (\( i=1, \ldots, k \)) is empty, then this algorithm exits and the centroids are the final. Otherwise, go to step c;

c) Find a vector of centroids \( (c_{1}, \ldots, c_{i}) \) with the minimum inner distance. ID (inner distance) is defined as follow:

\[
ID(c_{1}, \ldots, c_{i}) = \sum_{c_{i}} Distance(c_{i}, \bar{c})
\]

where \( c_{i} \) comes from centroid set centroids\(_{i}\) (\( i=1, \ldots, k \)) and \( \bar{c} \) is the centroid. Then remove \( c_{i} \) from centroid sets \( (i=1, \ldots, k) \), and add \( \bar{c} \) into centroids. Go to step b.

There are \( k \times k \) centroids, shown in Figure 5. According to the combination principle, there are \( k^{k} \) different vectors of centroids \( (c_{1}, \ldots, c_{i}) \), of which \( c_{i} \) comes from centroid set centroids\(_{i}\) (\( i=1, \ldots, k \)). Therefore, step c) in the merging process, which calculates the inner distances of \( k^{k} \) centroid vector, spends the most execution time.

It is obvious that the calculation of each inner distance is independent from others. Therefore, all inner distance calculations can be performed parallelly.

Figures 6 and 7 show the details of the ID algorithm, which employs the MapReduce framework for calculation.

The map function of the ID algorithm (Figure 6), calculates the centroid of the k centroids (input. value) recorded as \( \bar{c} \), and then calculates the inner distance.

This function employs a Distance method to calculate and accumulate the distance between \( \bar{c} \) and each centroid in step 4 to obtain the inner distance. Then, it sets the centroid vector \( (c_{1}, \ldots, c_{i}) \) and the inner distance in the output and passes them out.

The reduce function of the ID algorithm using MapReduce (Figure 7), does nothing but passes the input out.

Function ID_in_Map

Input: input. // The value of the input is a vector of centroids \( (c_{1}, \ldots, c_{i}) \), where \( c_{i} \) comes from centroid set centroids\(_{i}\) (\( i=1, \ldots, k \)).

Output: output. // The key of the output is the same as input. key, and the value of the output is the inner distance.

1: \( \bar{c} \leftarrow \text{ReCalCentroid}(\text{input. key}); \)
2: \( \text{distance} \leftarrow 0; \)
3: for each obj ∈ input. value do
4: \( \text{distance} \leftarrow \text{distance} + \text{Distance}(\text{obj}, \bar{c}); \)
5: end for
6: output. collect(input. key, distance);

Figure 6. ID_in_Map algorithm

Function ID_in_Reduce

Input: input. // The key of the input is the centroid vector \( (c_{1}, \ldots, c_{i}) \), and the value of the input is the inner distance of the centroid vector.

Output: output. // The same as the input.

1: output. collect(input. key, input. value);

Figure 7. ID_in_Reduce algorithm

The merging process merges \( k^{2} \) centroids into new \( k \) centroids; each new centroid comes from \( k \) centroids respectively.

The \( k \) centroids from this process are the final centroids. Then we finish the clustering process by assigning each object in the data set to its nearest centroid.
4. Experimental Results

In section 4.1, the experimental environment is outlined. The corpus used for experiments is described in section 4.2 in detail. In section 4.3, the evaluation criterion and methodology are presented, and section 4.4 analyzes the experimental results.

4.1. Experimental Environment

In this paper, experiments are based on a PC with the following hardware configuration: Intel (R) Core (TM) 2 Duo CPU T6570 @ 2.10GHZ, 2.10GHZ, 2.00GB RAM and 250GB hard disk. The software configuration uses Windows Vista™ Home Basic.

Based on the platform mentioned above, VMware workstation 6.0.2 is used to construct a virtual cluster, with the same hardware platform: one CPU processor; 128MB RAM; and 8GB hard disk. The software environment uses the same configuration: the Debian Linux 5.0 operating system; the distributed computing platform of Hadoop-0.20.0; and Java development platform JDK 1.6.

4.2. Data Sets

All experiments are conducted on a set of standard data sets downloaded from the UCI Machine Learning Dataset Repository [16]. Some brief information of data sets considered is listed in Table 1.

![Figure 8. Time cost in different clusters](image1)

![Figure 9. The squared-error of clustering results](image2)

To validate that the employing of the ensemble learning method bagging is able to overcome the instability and the sensitivity to outliers efficiently, we have compared MBK-means (with bagging) with a modified version of MBK-means called MK-means (without bagging). The MK-means algorithm, which just skips the bagging data splitting phase and the cluster ensemble phase of MBK-means, is a traditional k-means algorithm with MapReduce.

The two algorithms, both with the MapReduce framework for clustering, are comparable. In this paper, the squared-error criterion is used to measure the result of clustering, defined as:

\[ E = \sum_{i=1}^{k} \sum_{p \in C_i} | p - m_i |^2 \]

where \( E \) is the square-error sum for all objects in the data set, \( p \) is a given object in cluster \( C_i \), and \( m_i \) is the mean of cluster \( C_i \). The comparative evaluation of the square-error between with bagging and without bagging is shown in Figure 9.

4.3. Evaluation Criterion and Methodology

To validate that the MapReduce framework is able to fix the computational inefficiency of the traditional k-means algorithm, the MBK-means algorithm has been applied for different data sets (listed in table 1) on different virtual clusters.

With the increasing number of nodes of the Hadoop cluster, the time cost of the clustering job using our MBK-means is decreasing. The time cost is shown in Figure 8.

4.4. Analysis

Figure 1 shows that the MBK-means algorithm consists of three phases: 1) Bagging data splitting; 2) K-means with MapReduce; 3) Cluster ensembling.
Given a data set, the number of instances is \( n \), the dimension of the instance (also known as the number of attributes) is \( d \), and the parameter \( k \) of k-means is given too (\( n >> k \), \( n >> d \)). Based on the parameters, we roughly analyze the time complexity of the three phases to validate the time efficiency of MBK-means.

Phase 1) creates \( k \) new data sets from the given data set, each of which contains \( n \) instances from the given data set. Therefore, its time complexity is \( O(n * k) \).

The distance calculation between objects and centroids, which is the basis for assigning an object to the group that has the closest centroid, spends the most execution time of phase 2). Assume the k-means algorithm repeats \( m \) times (\( m > 1 \)) on average to reach convergence; each round has \( n \) instances to calculate their distances to the \( k \) centroids; the distance calculation employs a Distance function, which happens in \( O(d) \). Therefore, the time complexity of the distance calculations of a k-means process is \( O(d * n * k * m) \). There are \( k \) k-means processes in phase 2) as shown in Figure 1, therefore, the most time-consuming phase happens in \( O(d * n * k * m * k) \).

As the inner distance calculation consumes the most execution time of phase 3), it is acceptable to ignore other minor time consuming steps. There are \( k \) different centroid vectors employing the ID function for inner distance calculation, and each ID calculation requires \( k \) Distance calculations. Therefore, the time complexity of the most time-consuming part of phase 3) is \( O(d * k * k^2) \).

Obviously, the time complexity of phase 2) is much larger than that of phase 1), and that of phase 3) depends on the parameter \( k \) of k-means. Therefore, the time complexity of phase 1) is negligible. Phases 2) and 3) have been parallelized with MapReduce, therefore, the time complexity of the most time-consuming part of MBK-means is \( O(d * k * k^2 / numNode) + O(d * n * k * m * k / numNode) = O((d * k * k^2 + d * n * k * m * k) / numNode) \), where the \( numNode \) is the number of nodes in the cluster. Obviously, given a data set and the parameter \( k \), the time cost decreases linearly when the \( numNode \) increase.

As can be seen from Figure 8, the time cost of a clustering job using our MBK-means is decreasing almost in linearity with the increasing number of nodes of the cluster. So, if we want to analyze a huge data set with our MBK-means, we can reach a good time performance easily by adding nodes in a Hadoop cluster.

The bagging ensemble method is employed to overcome the instability and sensitivity of the traditional k-means method to outliers. Firstly, the MBK-means algorithm creates \( k \) data sets from the given data set. Then, it is able to reach \( k \) versions of centroids (each version contains \( k \) centroids). Finally, the \( k \) version centroids are merging into a final version. The MBK-means algorithm employs multiple versions of centroids together to overcome the instability and sensitivity to outliers.

From Figure 9, it is easy to notice that the square-error with bagging is significantly lower than that without bagging. The squared-error decreases from 7.83% (Image segmentation) to 41.3% (Waveform-21). So, it shows that ensembling can improve the stability of the k-means algorithm, and our MBK-means can partly solve the problem of the instability and sensitivity to outliers of k-means.

5. Conclusions

In this paper, we proposed a method using an ensemble learning method bagging to overcome the instability and the sensitivity to outliers of k-means, while using a distributed computing framework MapReduce to solve the inefficiency problem in clustering large data sets. Extensive experiments have been performed to show that our approach is efficient.

Acknowledgements

This work is supported by the National Basic Research Program of China (973 Program) under grant 2009CB326203; the International Collaboration grant from the Chinese Academy of Sciences under grant 2F05N01; the National Natural Science Foundation of China (NSFC) under grants 60828005, 61005044 and 60975034; and the National Science Foundation of Anhui Province of China under grant 090412044.

References


