International Journal of Latest Trends in Engineering and Technology
Vol.(8)Issue(1), pp.423-430
DOI: http://dx.doi.org/10.21172/1.81.054
e-ISSN:2278-621X

USING BAGGING AND MAPREDUCE IN K-MEANS CLUSTERING FOR DETECTION OF CANCER

Leena I Sakri[1] and Smruti Joshi[2]

Abstract- Cancer is a group of diseases which involves abnormal growth of cell and spread to other body parts. As the cause of the disease is unknown, it is leading to the cause of death of many people. Early detection of cancer is a challenging thing. Many methods of detection have been developed and are being used, but the selection of a proper method of detection helps in early treatment and recovery of a patient. This paper proposes the clustering of k-means using MBK Algorithm, MapReduce technique. So that large data sets will run efficiently and also it will help in early detection of cancer.
Keywords – K-means, MapReduce, Bagging, MBK Algorithm

I. INTRODUCTION

Cancer is the most extreme complex disease; every single tumor consists of more than 100 billion cells, and each of these cells can develop separate mutations. This disease is always altering, spreading and adapting. One of the key problems in the treatment of cancer is the early detection of the disease. Often, it is detected at its later stages, when it is widespread throughout the body. Methods for the early detection of cancer are the most important and an active area of research. Current cancer treatments include surgical therapy, radiation therapy, and chemotherapy; however, these treatments also damage normal tissues [5]. A long term goal of cancer research is to identify the molecular mechanisms by which cancer is being developed, and later to recognize the molecular markers of cancers earlier so that those recognized molecules can be targeted with the drugs specifically designed to attack them. Many researchers are focused on developing targeted molecular therapies also damage normal tissues [5].

The word Big Data refers to a huge amount of data which is increasing day by day. It plays vital role in all the sector, and also used in developing applications of big data which includes weather forecasting, agriculture, bioinformatics, education, social networks, health care, e-commerce, e-governance and so on. Due to the vast increase in size of the gene and complex structure, it is difficult to analyze and explain a large amount of data [4]. In gene data analysis, they will be evaluated under different situations, such as separate developmental stage of treatments

[1] *Department of Information Engineering SDMCET, Dharwad, Karnataka, India*

[2] *Department of Inforamtion Engineering SDMCET, Dharwad, Karnataka, India*

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 [6]. 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 [7]. 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.

II. RELATED WORK

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 [8]. 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 [9].
Given a set of objects (*x*1, *x*2,…, *xn*), where each object is a *d*-dimensional vector, the k-means clustering aims to partition the *n* objects into *k* sets ($k < n$) $S = \{S1, S2,\ldots, Sk\}$. The k-means algorithm is composed of the following steps:

1) Given an initial set of *k* means *m*1 (1),…, *mk* (1), which may be specified randomly or by some heuristic.
2) Assignment step: Assign each object to the group that has the closest centroid. ( ) ( ) ( ) * *t* { :|| *t* || || *t* || for all *=1,..., } *i j j i j i S* = *x x* − *m* ≤ *x* −*m i k*
3) Update step: When all objects have been assigned, recalculate the positions of the *k* centroids.

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 $\boldsymbol{O}(ndk+1\log n)$, 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 [10]. 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 [7].

2.4. MBK Algorithm

2.4.1. Data Splitting and Distance Measure

Given a standard training set *D* of size *n*, bagging generates *m* new training sets *Di* of size *n*' ≤□*n* (*i*=1,…,
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:

$$Distance(obj_i, obj_j) = \sum_{k=1}^{d} |x_{ik} - x_{jk}|^q$$

where *obji*=(*xi*1, *xi*2,…, *xid*) and *objj*=(*xj*1, *xj*2,…, *xjd*) 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 $\boldsymbol{O}(d)$.

2.4.2. 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 effect 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 1 and details are as follows:

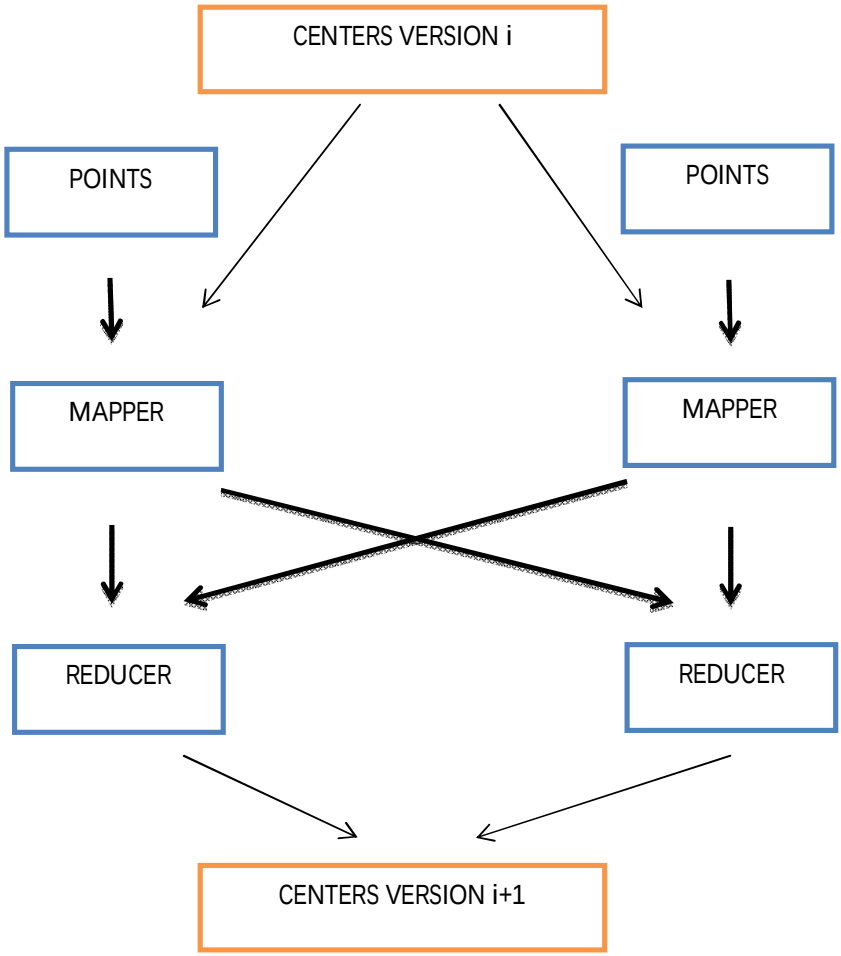

Figure1. K-means framework with MapReduce

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

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.

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 *reCalCentroid* 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_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*);

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**

5: *centroid* ← *ReCalCentroid*(*v*);

6: *output*. *collect*(*input. key*, *centroid*);

2.4.3 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 *Centroidsk*. Therefore, the *k* centroid sets must be merged into a finally centroid set.

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 *centroidsi* (*i*=1,…, *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,…, *ck*) with the minimum inner distance. *ID* (inner distance) is defined as follow:
1 1 (,...,) (,) *k k i i ID c c Distance c c* □□□□□ where *ci* comes from centroid set *centroidsi* (*i*=1… *k*) and *c* is the centroid. Then remove *ci* from *centroidsi* (*i*=1,…, *k*), and add *c* into *centroids.* Go to step b.
There are *k* * *k* centroids, shown in Figure 5.

According to the combination principle, there are *kk* different vectors of centroids (*c*1,…, *ck*), of which *ci* comes from centroid set *centroidsi* (*i*=1… *k*). Therefore, step c) in the merging process, which calculates the inner distances of *kk* 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.

The *map* function of the *ID* algorithm, calculates the centroid of the *k* centroids (*input. value*) recorded as *c* , and then calculates the inner distance. This function employs a *Distance* method to calculate and accumulate the distance between *c* and each *centroid* in step 4 to obtain the inner distance. Then, it sets the centroid vector (*c*1,…, *ck*) and the inner distance in the *output* and passes them out.
The *reduce* function of the *ID* algorithm using MapReduce 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,…, *ck*), where *ci* comes from centroid set *centroidsi* (*i*=1… *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: *c* ← *ReCalCentroid*(*input key*);
2: *distance* ← 0;
3: **for each** *obj*∈*input. value* **do**
4: *distance* ← *distance* + *Distance* (*obj*, *c*);
5: **end for**
6: *output*. *collect*(*input. key*, *distance*);

Function ID_in_Reduce

Input: *input*. // The *key* of the *input* is the centroid vector (*c*1,…, *ck*), 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*);

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.

III. (COMPARISON) PROS and CONS

K-Means Clustering

Advantages:

- If variables are huge, then K-Means most of the times computationally faster than hierarchical clustering, if we keep k smalls.
- K-Means produce tighter clusters than hierarchical clustering, especially if the clusters are globular.

Disadvantages:

- Difficult to predict K-Value.
- With global cluster, it didn't work well.

- Different initial partitions can result in different final clusters.
- It does not work well with clusters (in the original data) of Different size and Different density

MAP REDUCE

Advantages:

- can (potentially) query live data
- can (conceptually) be highly efficient at joining data sets that are identically sharded on the join key (the joins can be pushed down into the key-value store itself)

Disadvantages:

- Full scans (the most common pattern for map-reduce) is most likely to be much faster with raw file system access
- Because of the better decoupling of computation and storage in the GFS+Map-Reduce model - tolerating hot spots (resulting from MR jobs) is much easier
- Key-value stores are rarely arranged to have schemas optimized for analytics

IV. CONCLUSION

In this paper, we find 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.

REFERENCES

[1] Z. Weizhong, M. Huifang, and H. Qing, “Parallel kmeans clustering based on mapreduce,” Proceedings of the CloudCom ’09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 674–679.
[2] L. Guang, W. GongQing, H. XueGang, Z. Jing, L. Lian, and W. Xindong, “K-means clustering with bagging and mapreduce,” in Proceedings of the 2011 44th Hawaii International Conference on System Sciences. Washington, DC, USA: IEEE Computer Society, 2011.
[3] Sawyers C. Targeted cancer therapy. Nature. 2004; 432:294–297. [PubMed]
[4] Cancer Biomarkers: The Promises and Challenges of Improving Detection and Treatment (2007) book by Sharyl J. Nass and Harold L. Moses, editors, Institute of Medicine of the National Academies, The National Academies Press, Washington, D.C. https://www.nap.edu
[5] Predicting cancer-relevant proteins using an improved molecular similarity ensemble approach by bin Zhou, Qi Sun and De-Xin Kong, Oncotarget May 2016.
[6] T. G. Dietterich, “Machine Learning Research: Four Current Directions”, AI Magazine, 1997, Vol. 18, No. 4, pp. 97-136.
[7] J. Dean, S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, Communications of the ACM, January, 2008, Vol. 51 No. 1, pp. 107-113.
[8] S. P. Lloyd, “Least Squares Quantization in PCM”, IEEE Transactions on Information Theory, March, 1982, Vol. 28, No. 2, pp. 129-137.
[9] A. K. Jain, M. N. Murty, and P.J. Flynn, “Data Clustering: A Review”, ACM Computing Surveys, 1999, Vol. 31, No. 3, pp. 264-323.
[10] J. R. Quinlan, “Bagging, Boosting, and C4.5”, Proceedings of the National Conference on Artificial Intelligence, 1996, pp. 725-730.