Implementing K-means and K-medoids in Apache Spark

Ripul Jain
r6jain@uwaterloo.ca

ABSTRACT
This report describes the implementation of K-means and K-medoids algorithms in Apache Spark. The algorithms are tested on small 20 news group dataset and large Wikipedia dataset and results are discussed.

CCS Concepts
• Computing methodologies~Cluster analysis

Keywords
Clustering; Apache Spark; K-means; K-medoids

1. INTRODUCTION
Clustering is an unsupervised learning problem whereby the goal is to partition the data into groups based on their similarity. It has many applications in information retrieval, image analysis, pattern recognition and machine learning. In this report we look at the most common clustering algorithm K-means and its variant K-medoids. The algorithm is presented for both K-means and K-medoids and implemented in Apache Spark.

2. BACKGROUND
There are various open source implementations available for both K-means and K-medoids algorithms. But most of them are written for handling data that fits on one machines memory. For large scale distributed data, Apache Spark provides an implementation of K-means algorithm with its Mllib library. At the moment there is no support in Apache Spark for K-medoids algorithm as the standard K-medoids algorithm presents a scalability issue with runtime complexity to the order of $O(N^2)$ [3]

3. ALGORITHM
3.1 K-means
In this report, Lloyd's algorithm for K-means clustering is implemented which is the universally accepted standard algorithm for K-means. Given a set of N data points and K clusters, the goal of K-means algorithm is to partition the N data points into K clusters. It starts by randomly selecting K centroid points from the data and iteratively updates the centroids till the difference between the updates is less than some small defined value

1) Select K initial centroids from N data points, c_1, c_2, c_k
2) Repeat until convergence
    a. Assign each data point to the nearest centroid
    b. For each cluster, compute new centroid by taking the mean of all data points in that cluster

The algorithm is guaranteed to converge every time regardless of the choice of initial centroids. However, the results of clustering are very much dependent on the choice of initial centroids.

3.2 K-medoids
K-medoids algorithm is similar to K-means. Instead of computing new centroid, K-medoids chooses a new data point as the new medoid. A medoid is a data point in the set of data points, whose average dissimilarity to all the data points in the cluster it belongs to is the lowest. The goal is to select a data point which is most centered as the medoid. Typically, K-medoids is more robust to noise and outliers than K-means.

The algorithm is as follows:

1) Initialize: Select K initial medoids from N data points, m_1, m_2, m_k
2) Repeat until convergence
    a. Assign each data point to the nearest medoid
    b. For each medoids m and for each document d associated with medoids m, calculate the average distance of d to all other document in m. Replace d with lowest average distance as new medoid m

Similar to K-means, K-medoids is guaranteed to converge every time regardless of the choice of initial medoids. However, the results of clustering are very much dependent on the choice of initial medoids.

4. IMPLEMENTATION
For both the algorithms, input is provided in the form of Tf-idf vector of a text document. Tf-idf stands for term frequency – inverse document frequency. It is a measure used to evaluate the importance of each word in a document over a collection of documents. For a given list of document, Tf-idf weights are calculated for each document using Apache Spark's Mllib library. The distance metric required for computing the similarity between the documents is obtained using Cosine Similarity method. In this method, the cosine of the angle between the two document vectors is used to calculate the similarity between two documents.

$$\text{Cossim}(x, y) = \frac{\sum_{i=0}^{n} x_i * y_i}{\sqrt[2]{\sum_{i=0}^{n} x_i^2} * \sqrt[2]{\sum_{i=0}^{n} y_i^2}}$$

$$\text{Distance}(x, y) = 1 - \text{Cossim}(x, y)$$

Two highly similar documents are expected to have cosine similarity 1, while two highly dissimilar documents will have cosine similarity 0. Consequently, the distance between two highly similar documents would be close to 0, while the distance between two highly dissimilar documents would be 1. This distance metric is used in both K-means and K-medoids implementation.

4.1 K-means
The algorithm starts by choosing K random sample documents from the data as initial centroids. It then maps over the entire dataset and emits tuples of (data point, 1) to group the data points according to the nearest centroid.

Then in reducer phase it merges the data points to create new centroids and sum the 1s to get a count of all documents in each cluster. At the end of the iteration it gets new centroids at the driver and updates them for the new iteration. The algorithm continues until the difference between the old centroids and new centroids is greater than a predefined delta. In case the algorithm does not converge, it runs until the predefined number of iterations has elapsed.

The execution flow of K-means algorithm in Spark is shown below.

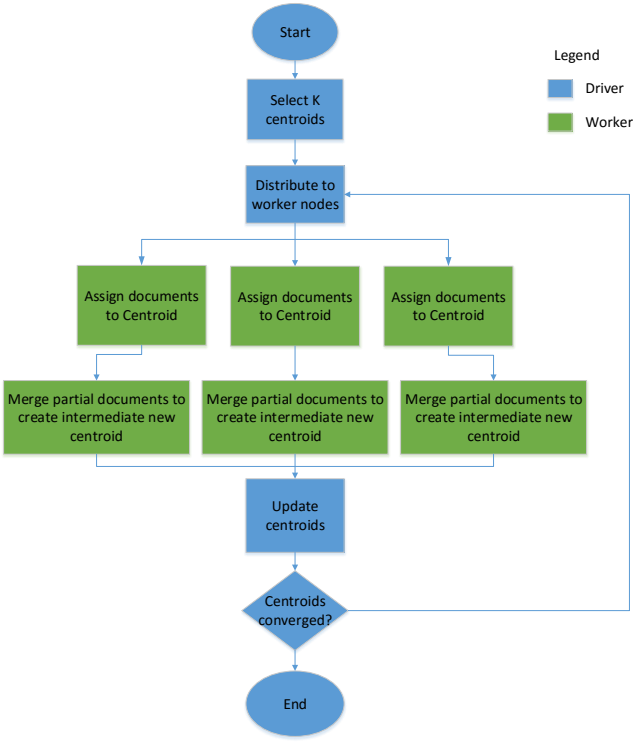

Figure 1: K-means execution

As shown in Figure 1, the only data flow that occurs per iteration is of centroids between driver and workers. This data flow becomes the performance bottle neck for high dimensional centroids as in the case of text document clustering

The runtime complexity of K-means algorithm is O (N*K*I), where N is number of documents in a dataset, K is number of desired clusters and I is the number of iterations required for convergence

4.2 K-medoids

The algorithm starts by choosing K random sample documents as initial medoids. It then maps over the entire dataset and emits tuples of (medoid, List[document1, document2 ..]) to form clusters centered around each medoid. For each medoid, it then maps over list of documents in its cluster and finds the document that is the most centered as the new medoid. The new medoids are collected at the driver and updated for the next iteration. The algorithm continues until the difference between the old medoids and new medoids is greater than a predefined delta. In case the algorithm does not converge, it runs until the predefined number of iterations has elapsed.

Unlike K-means implementation, a lot of intermediate data gets shuffled between the worker nodes in K-medoids implementation while trying to find the new medoid as each data point in a cluster need to be tested with all other data points.

The runtime complexity of K-medoids algorithm is O (K*(N-K)^2*I), where N is the number of documents in cluster, K is the number of desired clusters and I is the number of iterations required for convergence.

The execution flow of K-medoids in Spark is shown in Figure 2.

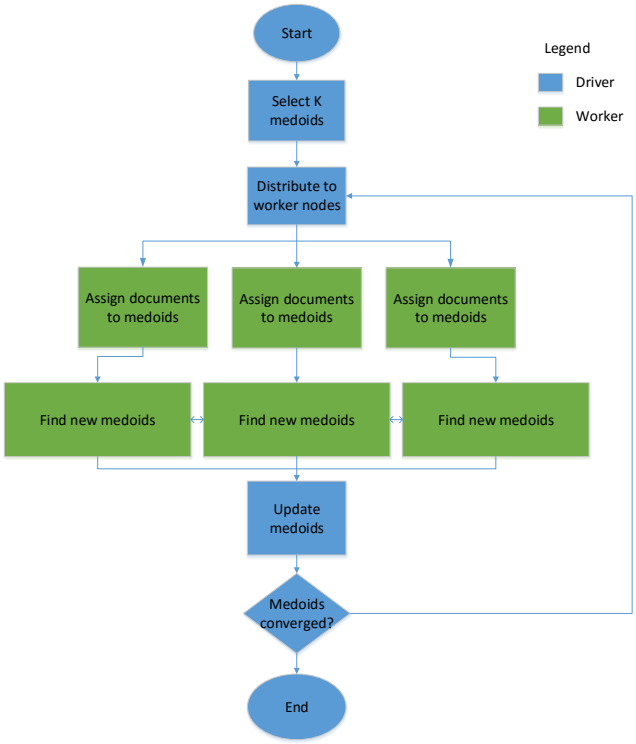

Figure 2: K-medoids execution

5. EXPERIMENT

To test K-means and K-medoids, the experiment uses the 20 newsgroup dataset containing 11,293 documents and a difficult Cade12 dataset containing 27322 documents and runs it on the Altiscale cluster. The 20 newsgroup dataset is organized into 20 different newsgroups, each corresponding to a different topic. Some of the newsgroups are very closely related to each other (e.g. comp.sys.ibm.pc.hardware / comp.sys.mac.hardware), while others are highly unrelated (e.g. misc.forsale / soc.religion.christian). The Cade12 dataset corresponds to a subset of webpages extracted from the CADE web directory [1]. To evaluate the algorithms, we calculate Purity of the clustered documents. The idea is to assign each cluster to a class that is the most frequent in the cluster. Then Purity is calculated as the ratio of the number of correctly assigned documents and total number of documents. Bad clustering has purity values close to 0, a perfect clustering has a purity of 1[2].

The convergence criterion was found by trial and error and both algorithms were run until convergence.

Table 1: Analysis Results

Algorithm	Dataset	Number of Clusters	Purity
K-means	20 news group	20	0.48
	Cade	12	0.31
K-medoids	20 news group	20	0.40
	Cade	12	0.39

The algorithm was tested against datasets for increasing size and the results from those experiments are as follows

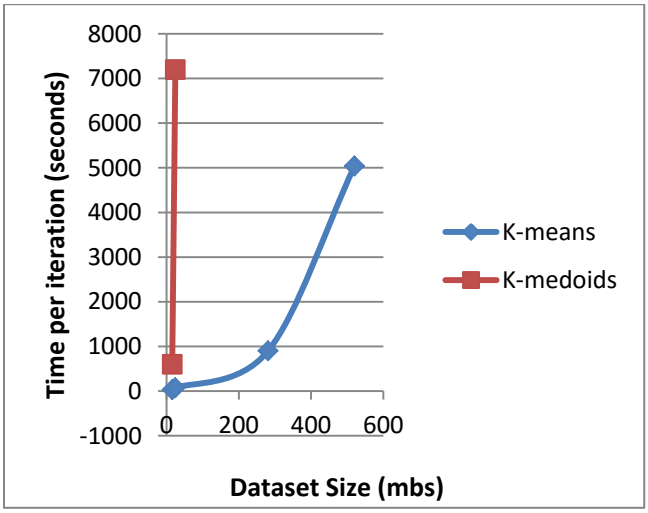

Figure 3: Iteration times for different dataset sizes

As seen from Figure 3, the time per iteration for K-medoids grows very quickly when compared to K-means. This is in line with the algorithmic complexity discussed above.

For K-means algorithm running on Wikipedia dataset (520 Mb), the time for one iteration was 84 mins. This is due to large size centroids being shuffled across the network.

6. CONCLUSION

From the above experiments, it can be shown that K-means can be efficiently implemented using Apache Spark, while more work is needed to find out a faster algorithm for K-medoids implementation.

7. REFERENCES

[1] Ana Cardoso-Cachopo. Improving Methods for Single-label Text Categorization. PhD Thesis, Instituto Superior Tecnico, Universidade Tecnica de Lisboa, 2007.

[2] Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. *An Introduction to Information Retrieval*. Cambridge: Cambridge UP, 2008. Web. 16 Apr. 2016.

[3] [SPARK-4510] Add K-Medoids Partitioning Around Medoids (PAM) Algorithm - ASF JIRA". *issues.apache.org*. N.p., 2014. Web. 16 Apr. 2016.