2017 Fall

CSE 601

Project-2 Report

Xuan Han 50168962

Laiyun Wu 50061617

Ting Zhou 50169020

PART1 – Implementation of Three Clustering Algorithms

First algorithm—K-Means

1. Algorithm description

Standard K-means

- Choose k observations uniformly at random from the data set X as the initial means.
- For each point $x \epsilon X$, get $argmin_i \|x - \mu_i\|_2^2$ and add x to a set S_i.
- For $i = 1, \ldots, k$, set μ_i to be the centroid of the points in S_i.
- Repeat steps 2 and 3 until the means have converged.

Standard K-means algorithm takes time *O(tkmn),*

Where,

t – the times of iteration

k – the number of predefined clusters

m – the number of observations of the input data

n – the number of attributes of the input data

Due to K-means is highly influenced by the initial centroids, one way to improve the performance of K-means is just to run the algorithm multiple times with different initial seed and taking the results with the smallest SSE. In this way, the program is going to take *O(ltkmn),* where *l* is the times to repeat the K-means.

For cho.txt data, we set k = 5 and the maximum iteration times as 100, however, on real datasets the algorithm typically converges in at most a few dozen iterations. And we repeat the K-means 10 times in order to reduce the impact by the bad initial centroids. The following result is from the one with smallest SSE. Same procedures with the iyer.txt data, iyer.txt data have some outliers. Thus, we implement the k-means to the iyer.txt data and the iyer.txt data without outliers in order to discuss the influence of the outliers to results from kmeans.

2. Visualization

For the data file cho.txt, using sklearn.adjusted_rand_score to compare the clustering results and the ground truth. The Rand Index is 0.46. Implementing PCA to reduce the dimension from 16 to 2 and visualize the results, as shown in Figure1.

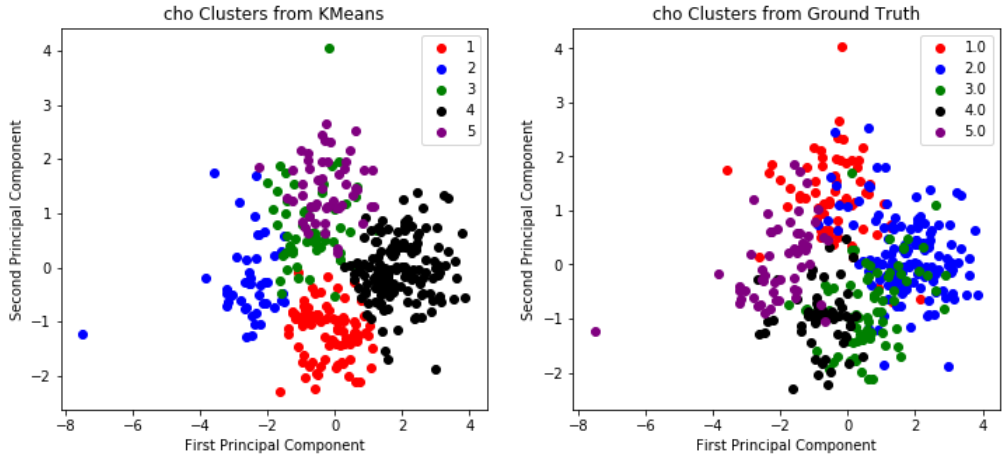

Figure1. Comparison between K-means and Ground Truth of cho.txt

For the data file iyer.txt, using sklearn.adjusted_rand_score to compare the clustering results and the ground truth. The Rand Index is 0.33. Implementing PCA to reduce the dimension from 12 to 2 and visualize the results. Also, after removing the outliers in iyer.txt, the Rand Index between the clustering results and the ground truth is 0.44. The Rand Index improves for the iyer data without outliers. The visualization of the results are shown in Figure 2.

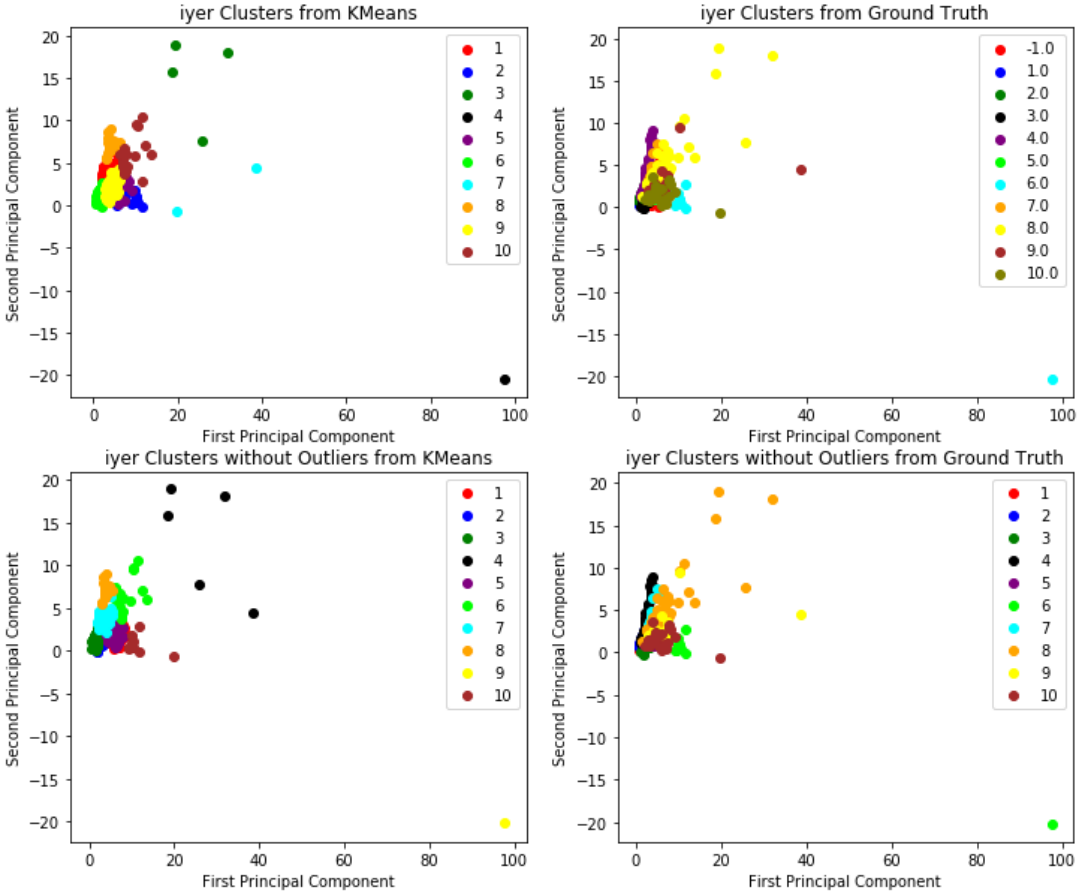

Figure2. Comparison between K-means and Ground Truth of iyer.txt

3. Result Evaluation

Due to the initial centroids are randomly taken from the data, the times of iteration are not fixed. Thus the results may be different for each execution of the algorithms as well as the run time.

For cho.txt dataset, k-means works not well, because the clusters are not relatively spherical like and the density of each cluster is different shown in Figure1. Furthermore, the dataset cho.txt has 16 attributes, these attributes are impossible independent with each other. That's why the performance of K-means on cho.txt is not good.

For iyer.txt dataset, k-means works poorly. Same as the cho.txt, the clusters are not relatively spherical like and the density of each cluster is different shown in Figure2. And there are some outliers in iyer.txt, K-means is sensitive to outliers. We remove the outliers and do the K-means to the data without outliers, the adjusted rand index improve from 0.33 to 0.45. The results are shown in Table 1.

Table 1. Results of K-means

Data	Max_Iter	n_init	ARI	Run Time (sec)
cho.txt	100	10	0.46	6
iyer.txt	100	10	0.33	20.38
iyer without outliers	100	10	0.45	24.74

Pros and Cons of K-means:

Before talking about the pros and cons of k-means, we need to know the assumptions of k-means:

1. Assuming balanced cluster size within the dataset;
2. Assuming the joint distribution of features within each cluster spherical, which means that features within a cluster have equal variance and are independent to each other;
3. Clusters have similar density.

Pros:

1. K-means is fast and easy to use. The complexity of k-means, as mentioned before, is *O(ltkmn)*. K-means usually converges within a few dozens of times;
2. Practically works well even some assumptions are broken;
3. K-means has objective function, min(SSE).

Cons:

1. The size of the clusters from k-means is uniform, even the input data have different cluster size;
2. If the spherical assumption is broken, because the features are usually correlated, the performance would be unexpected. Other cluster algorithm should be considered, such as EM for features with covariance or DBSCAN for irregular shape data.
3. K-means also work poorly if the clusters have different density even they are spherically distributed.
4. Need to assume the K, the number of clusters. For unsupervised learning, we don't know the ground truth. How to solve the K? We need to try different K values maybe from 2 to 100 and for each K value we also need to run lots of time or use K-means++. Then we can pick the K with the lowest SSE. In this way, K-means is not efficient anymore.
5. K-means is sensitive to outliers, because the outliers would stretch the centroids out of the map, which has been shown in the results of the iyer.txt data and iyer without outliers in this project. The rand index improve from 0.33 to 0.52 when the outliers are removed.
6. K-means is sensitive to initial centroids, for different initial centroids K-means might converge at local optimal.

Second algorithm—Agglomerative

1. **Algorithm Description:**
    1. Hierarchical clustering technique starts with each data point to be a cluster, by calculating the distance between any two data points, we could have the first distance matrix of all data points.
    2. Then we have to repeat the following process:
        a. Merge the two closest clusters. It is worth to mention that there are many ways to calculate distance between two clusters (single link, complete link, average link, centroid distance, ward's method), but we always choose the two closest clusters to merge.
        b. Update the distance matrix by updating the distance row and column of the new cluster
    3. If there is only one cluster remain, stop the iteration. Clustering result will be generated by cutting the dendrogram at the proper level.

2. Visualization

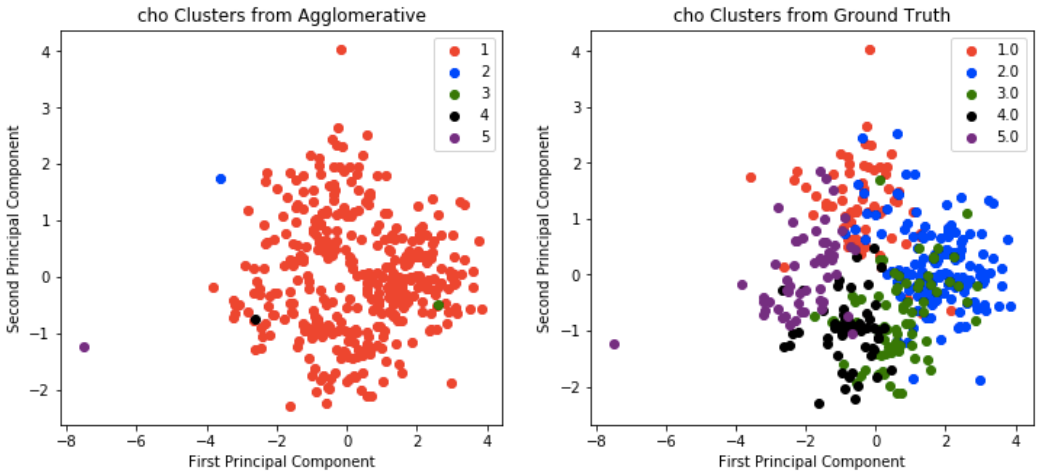

Figure3. Comparison between Agglomerative and Ground Truth of cho.txt

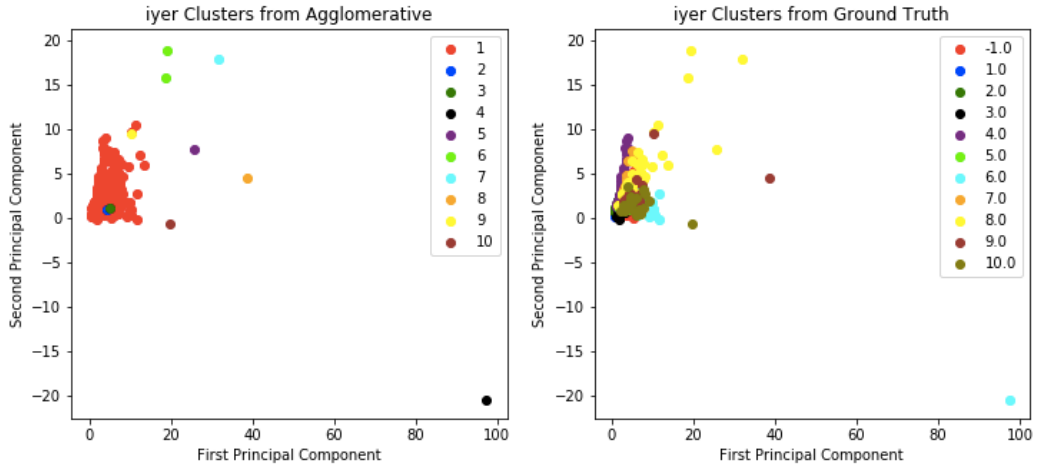

Figure 4. Comparison between Agglomerative and Ground Truth of iyer.txt

3. Result Evaluation

This algorithm performs very poorly on both cho.txt and iyer.txt datasets. It's not hard to find that there are some points are far from the dense points cluster in Figure3 and Figure4. Due to we need to use Agglomerative clustering with single link(min), these dense points would be clustered firstly and those outlier-like points would get in the hierarchical company late. Due to the ground truth is available in this project, when we cut the hierarchical company to get 5 clusters for cho.txt and 10 clusters for iyer.txt, each point in those outlier-like points would become a single cluster even though these points are not real outliers.

Table 2. Results of Agglomerative

Data	Clusters	ARI	Run Time (sec)
cho.txt	5	0.0037	4.21
iyer.txt	10	0.0081	8.45

Pros and Cons:

Strengths: number of clusters are not necessary to perform hierarchical clustering algorithm.

Limitations: it is sensitive to noises and outliers

decisions of combining two clusters cannot be undone

Third algorithm—Density-based DBSCAN

1. Algorithm description

1) DBSCAN

In DBSCAN, a cluster is defined as a maximal set of density-connected points.

Algorithm:

For each point p in dataset,

a. If not visited, mark it as visited;
b. Then find its neighborhoods in radius of eps;
c. If the number of its neighborhoods is smaller than the minimum number of points required to form a dense region, which is MinPts, then mark p as noise;
d. If the number of its neighborhoods is equal to or larger than MinPts, mark p as a new cluster and then expand this cluster by the following steps:
   For each point p' in the neighborhoods of p,
   a) If not visited, mark it as visited;
   b) Then find its neighborhoods in radius of eps;
   c) If the number of its neighborhoods is equal to or larger than MinPts, join these neighborhoods of p' to the neighborhoods of p;
   d) Now if p' is not yet member of any cluster, label p' as the cluster we are expanding.

2) Parameter estimation

When conducting DBSCAN, we need to calibrate the parameters, eps and MinPts, to deliver a reasonable output. For points in a cluster, their k-th nearest neighbors are at roughly the same distance.

So we optimize our parameters by the following steps:

a. Given a MinPts value, sort points by their respective distance to their respective k-th (k = MinPts - 1) nearest neighbor;
b. Then we use the indices of these points as x-axis, their respective distances to k-th nearest neighbor as y-axis, to do scatter plot and find the gap distance in the plot. This distance is the best eps for the corresponding MinPts value;
c. Then we repeat the above two steps for different MinPts values and we can obtain plot like in Figure 1;
d. Now we have the best eps for each MinPts value, we take their average as eps value;
e. Then we can determine the best MinPts through iteration of different MinPts values on DBSCAN, choose the one that has the largest rand index.

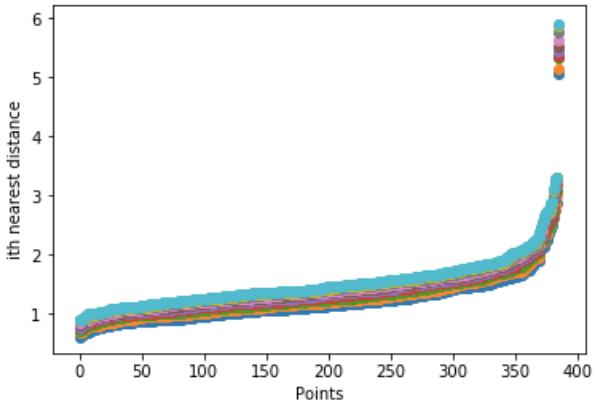

Figure 5. Plot of sorted distance of every point to its k-th nearest neighbor (data-cho, k-2:1:19)

2. Visualization

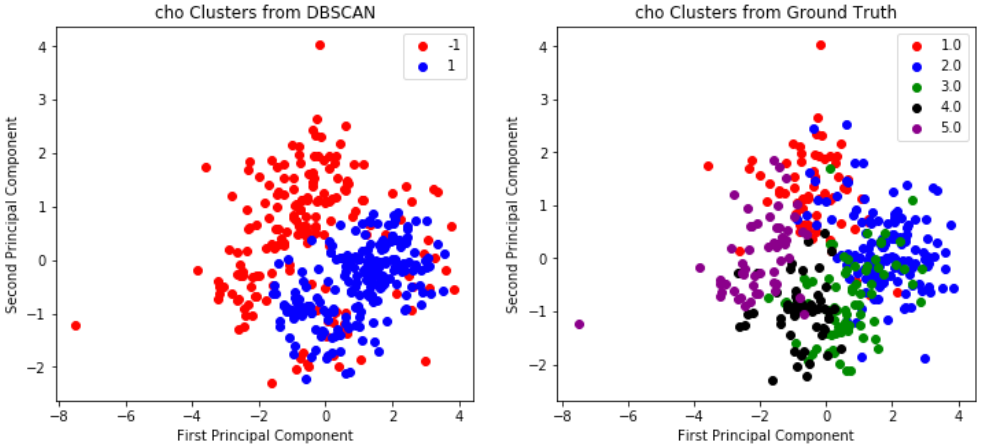

Figure 6. cho.txt Clusters from DBSCAN (eps = 1.3, MinPts = 16)

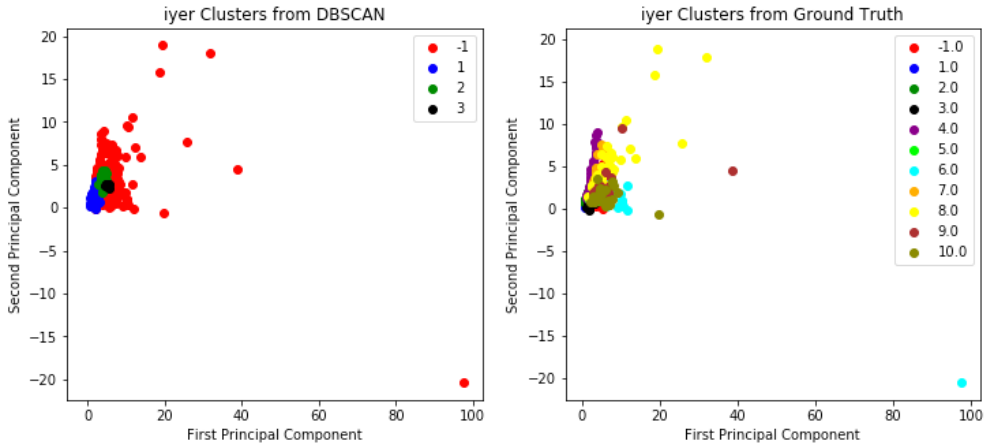

Figure 7. iyer.txt Clusters from DBSCAN (eps = 1.1, MinPts = 7)

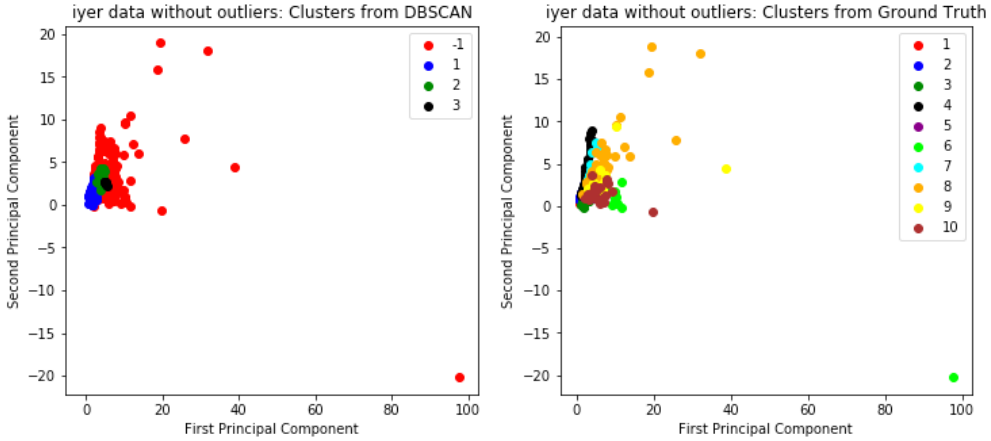

Figure 8. iyer.txt without Outliers from DBSCAN (eps = 1.1, MinPts = 7)

3. **Result evaluation**

1) Result analysis

Due to the points in cho.txt have varying densities and the boundary of the clusters are overlapped as shown in Figure 6, DBSCAN can not detect all the clusters correctly and perform poorly.

For iyer.txt dataset, same as the cho.txt, the points have varying densities and the boundary of the clusters are overlapped, DBSCAN still works poorly.

Table3. Results of DBSCAN

Data	Eps	MinPts	ARI	Run Time (sec)
cho.txt	1.3	16	0.13	1.61
iyer.txt	1.1	7	0.27	3.01
iyer without outliers	1.1	7	0.31	2.87

2) Pros:

- No need to specify the number of clusters
- Resistant to Noise
- Good to use when geometries are very non-flat
- Can handle uneven clusters of different shapes and sizes

3) Cons:

- Cannot handle varying densities
- Sensitive to parameters—extra effort to determine parameters
- Not deterministic at cluster boundaries
- Worst case run time complexity is $O(n^2)$, inefficient compared to K-means

PART2 - MapReduce K-means

1. Algorithm Description

Following the standard K-means, we need to separate the algorithm into two phases, Map and Reduce. The Map phase is to compute the sets S_i of points closest to centroid μ_i, and the Reduce phase is to compute the new means as the centroids of these sets.

1. Choose k observations uniformly at random from the data set X as the initial means.
2. In the Map phase
   a. Read the cluster centroids into memory from a SquenceFile.
   b. Operating on each point *x* in the dataset and computing the distance between *x* and each mean and find the μ_i which minimize this distance. We then emit a key-value pair with this mean's index *i* as key and the value as *x*. The function is,
   K-meansMap(centroids, *x*):
   emit $(argmin_i \|x - \mu_i\|_2^2, x)$
   c. Write the key-value pair (index, *x*) to the filesystem.
3. In the Reduce phase
   Pairwise summation over the values for each key. The function is,
   K-meansReduce(outputs from Map phase):
   combine(*i*, $(\sum_{x \in S_i} x, |S_i|)$)
   Compute the new means $\mu_1, \ldots, \mu_k$ from the results of the MapReduce.
   return($\mu_i = \frac{1}{|S_i|} \sum_{x \in S_i} x$)
4. Broadcast the new means to each machine on the cluster.
5. Repeat steps 2-4 until the means have converged.

2. Visualization

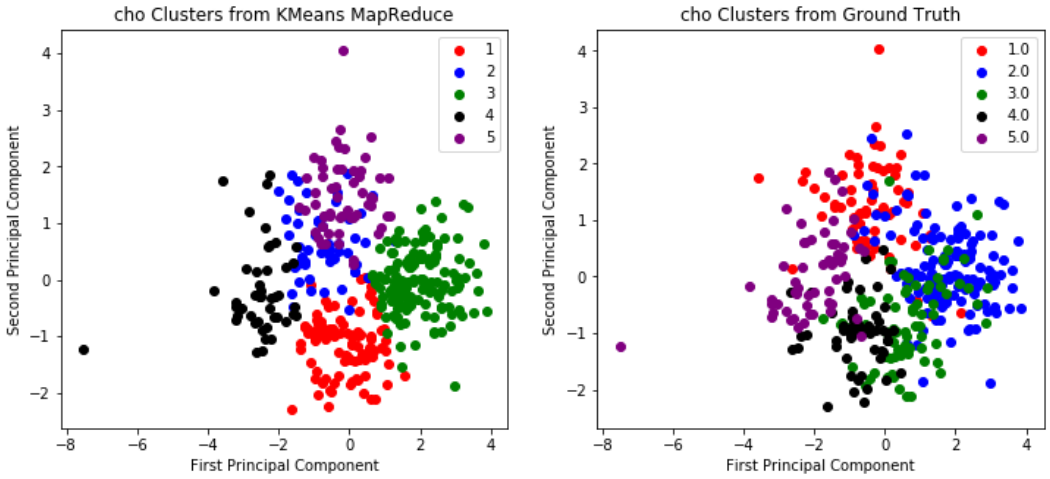

Figure 9. cho.txt from K-means MapReduce

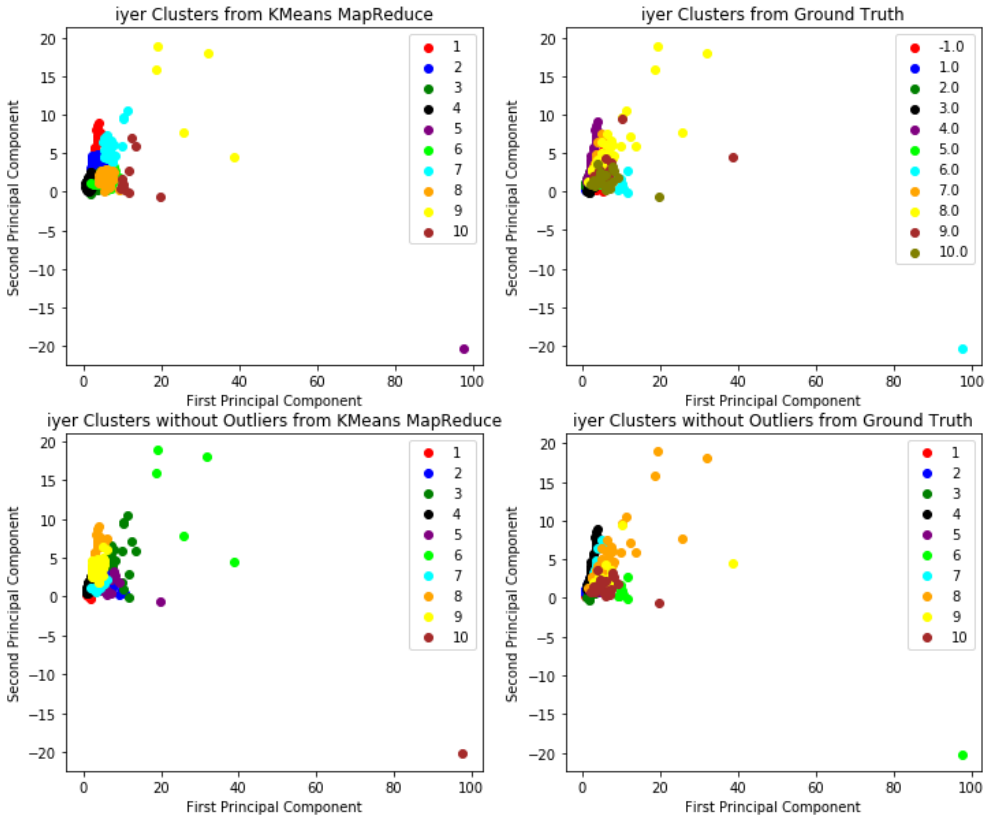

Figure 10. iyer.txt from K-means MapReduce

3. Results Evaluation

In this project, the data size is very small, thus using Hadoop Mapreduce is not efficient. It costs seconds for each iteration, because of the overhead and disk access. In order to improve the performance, we use the initial centroids from the k-means++ to reduce the times of iterations. The results of cluster centroids are almost same as the sequential k-means.

Table 4. K-means MapReduce

Data	Iteration Times	ARI	Run Time (sec)
cho.txt	16	0.44	136.3
iyer.txt	8	0.301	102.9
iyer without outliers	3	0.38	31.2

Pros and Cons:

Pros:

1. Reliable, no worry about machine failure.
2. Dividing the problem up into small chunks, then you can process huge amount of data very efficiently.
3. Processing in parallel allows for the timely processing of massive amounts of data.

Cons:

1. Not fit for small data, because HDFS lacks the ability to efficiently support the random reading of small files.
2. Disk access is relatively slow.
3. Map-reduce as implemented in Hadoop is not suitable for iterative processing. There is an overhead because of running the same job again and again till convergence of the cluster points is achieved, like k-means.

References:

1. https://www.quora.com/What-are-the-advantages-of-K-Means-clustering
2. https://en.wikipedia.org/wiki/DBSCAN
3. files.meetup.com/18503497/K_Means.pdf
4. https://stanford.edu/~rezab/classes/cme323/S16/projects_reports/bodoia.pdf
5. https://web.stanford.edu/class/cs246/homeworks/hw2/hw2sol.pdf
6. https://www.researchgate.net/post/How_can_I_implement_an_iterative_map_and_reduce_in_Hadoop_environment
7. http://www.j2eebrain.com/java-J2ee-hadoop-advantages-and-disadvantages.html