CSE 601 – Data Mining and Bioinformatics (Fall 2017)

Project 2:

Clustering Algorithms

Team Members:

1. Siddharth Pateriya spateriy@buffalo.edu Person #: 50206348
2. Hrishikesh Saraf hsaraf@buffalo.edu Person #: 50205927
3. Ajay Gandhi agandhi3@buffalo.edu Person #: 50207403

Introduction:

What has been expected:

Part1: Three Clustering Algorithms

- The main objective in this project is to implement three clustering algorithms to find clusters of genes that exhibit similar expression profiles: K-Means, Hierarchical Agglomerative clustering with Single Link (Min), and one from (density-based, mixture model, spectral).
- We have to compare these three methods and discuss their pros and cons, also we have to validate our results using an external index (either Rand Index or Jaccard Coefficient).
- We also have to visualize data sets and clustering results by Principal Component Analysis (PCA).

Part2: Hadoop Cluster (K-Means)

- We have to set up a single-node Hadoop cluster on our machine and implement MapReduce K-means.
- We also have to compare with non-parallel K-Means on the given datasets and improve the running time.

What has been done:

Part1: Three Clustering Algorithms

- We have implemented three clustering algorithms to find clusters of genes that exhibit similar expression profiles: K-Means, Hierarchical Agglomerative clustering with Single Link (Min), and Density-Based Scan.
- We have compared these three methods on both input files (cho.txt and iyer.txt) and discussed their pros and cons, we have validated our results using both Rand Index & Jaccard Coefficient.
- We have visualized data sets and clustering results by Principal Component Analysis (PCA).
- We have also discussed results for different user input values and provided a table of the output received.

Part2: Hadoop Cluster (K-Means)

- We have set up a single-node Hadoop cluster on our machine and implemented MapReduce K-means which works efficiently.
- We also compared with non-parallel K-Means on the given datasets and improved the running time.

Part 1: Clustering Algorithms

1. K-Means Clustering:

- K-Means algorithm basically works by partitioning data base into separate disjoint clusters. The properties of these clusters is such that the distance of points between different clusters is high but distance of points within each clusters is low.
- This helps in getting, **low inter-cluster** similarity and **high intra-cluster** similarity.
- For our implementation, we ask the user to enter 'k' the total number of clusters to be equal to the number of ground truth labels in the input file (k=5 for cho.txt & k=10 for iyer.txt).

A) Steps for K-Means:

1. First we input the number of clusters required, initial cluster centroids and maximum number of iterations allowed.

2. Each data point is then assigned to the nearest centroid based on minimum distance from the centroid.

3. The cluster centroids are then updated by taking the mean of all the data points belonging to that cluster.

4. Repeat steps 2 and 3 until convergence i.e. new cluster centroids is the same as old cluster centroids or number of iterations has reached maximum number of iterations allowed.

B) Validation Table:

1. Cho.txt

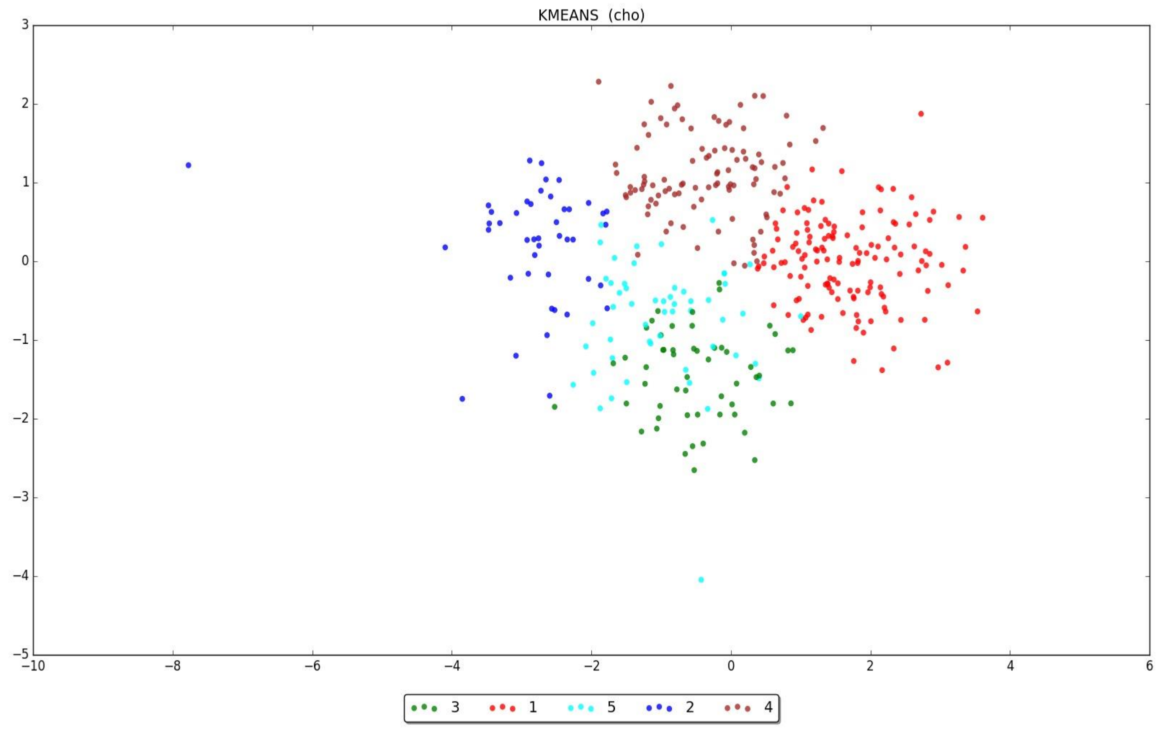
No. of Clusters	Jaccard Co-efficient	Rand Index	Iterations taken	Time taken
3	0.4088574886337618	0.7574028832988805	16	15ms
5	0.40661758822117305	0.8009208300893984	27	16ms
8	0.28179131620822917	0.7879540390346049	11	16ms
10	0.264342740438173	0.787256033719026	11	16ms
15	0.16706670667066706	0.7764101049692609	16	32ms

2. Iyer.txt

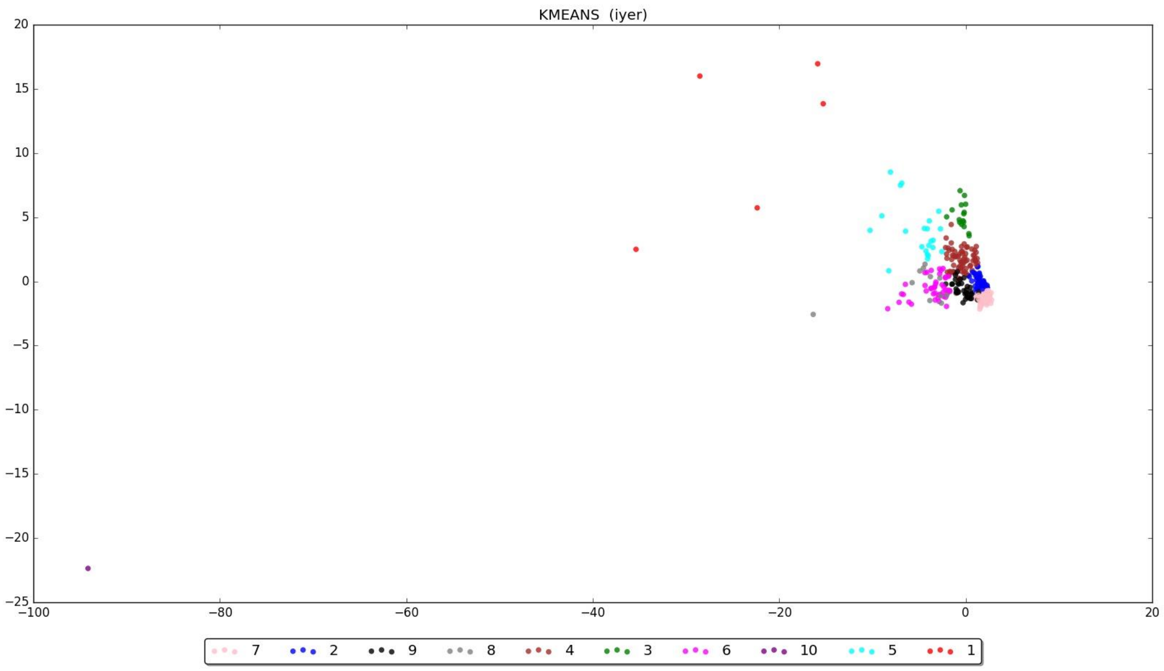
No. of Clusters	Jaccard Co-efficient	Rand Index	Iterations taken	Time taken
3	0.2199408701534227	0.4995304707638549	26	16ms
5	0.27680630499518616	0.6374710519325524	17	4ms
8	0.3081339795294705	0.6993067428887833	30	70ms
10	0.34934458699525234	0.7554257750973665	34	38ms
15	0.2779913059532401	0.8390580981634111	28	54ms

C) Visualization (using PCA):

1. Cho.txt



2. Iyer.txt



D) Result Analysis:

- On trying different starting cluster centroids, we received different results.
- In some cases, the Rand Index or Jaccard Coefficient gave better values and in some other its gave lesser values; implying that K-Means algorithm is somehow sensitive to initial cluster centroid values.
- But any specific relation between initial cluster centroid values and output doesn't exist.

E) Pros:

- Easy to implement.
- Time complexity is of the order O(nki); where 'n' are the number of data points, 'k' number of clusters and 'i' number of iterations.
- Gives best results if the dataset is distinct or well separated from other values.

F) Cons:

- Number of clusters 'k' has to be specified.
- Random initial clusters means non repeatability for results.
- K-Means is not a good choice for data of non-spherical type.
- K-Means fails to handle categorical data.
- Global minimum is not always achieved.

---

2. Hierarchical Agglomerative Clustering (with Single Link/Min):

- The Hierarchical Agglomerative Clustering (HAC) with Single Link / Min, is a bottom-up approach to clustering of data points into k clusters.
- HAC starts with taking each point as a cluster, and then merging the closest pair of clusters (measuring the euclidean distance between the points), until only k number of clusters remain.

A) Steps for HAC:

1. First, we read the input file and store it as a list of points and their co-ordinates.
2. Next, we compute the distance matrix, which contains the distance of each point 'a' to all the other points. This forms an NxN matrix where N is the total number of points.

3. Then, taking each individual point as a cluster, find the cluster closest to the current cluster by using the distance matrix and store this value.
4. Now, we need to merge the closest clusters together. For example, in the previous step if Cluster A is closest to Cluster B, then we merge the two clusters to form a single cluster.
5. After merging the clusters, we update the distance matrix with the distances between the new clusters.
6. We keep on iterating till the total number of clusters remaining is k, which is the number of clusters we require eventually.
7. The most critical operation is the computation of distances between two clusters.
8. After the k clusters are determined, we compute the Jaccard Co-efficient and the Rand Index by comparing the cluster values with the ground truth values given in the input file.

B) Validation Table:

1. Cho.txt

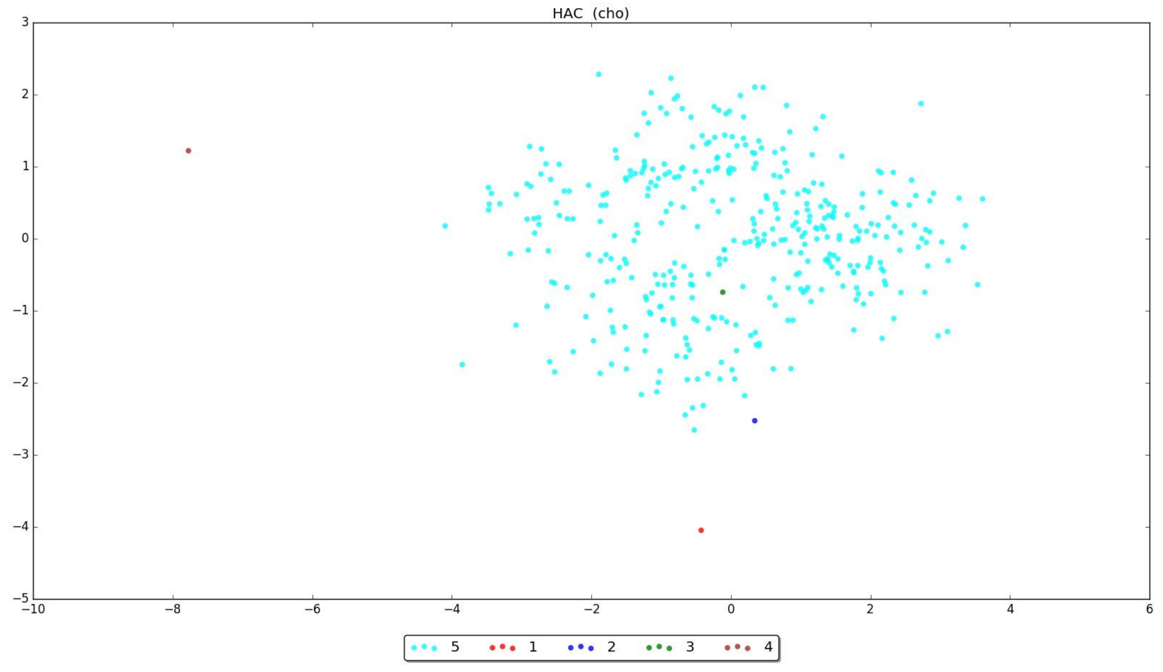
No. of Clusters	Jaccard Co-efficient	Rand Index	Iterations	Time taken
3	0.22933831189036352	0.2353485999624151	383	299ms
5	0.22839497757358454	0.24027490670890495	381	318ms
8	0.2287336160103863	0.2504228301430911	378	300ms
10	0.22778520126941706	0.25529544417299793	376	289ms
15	0.22503156968742463	0.266839378238342	371	302ms

2. Iyer.txt

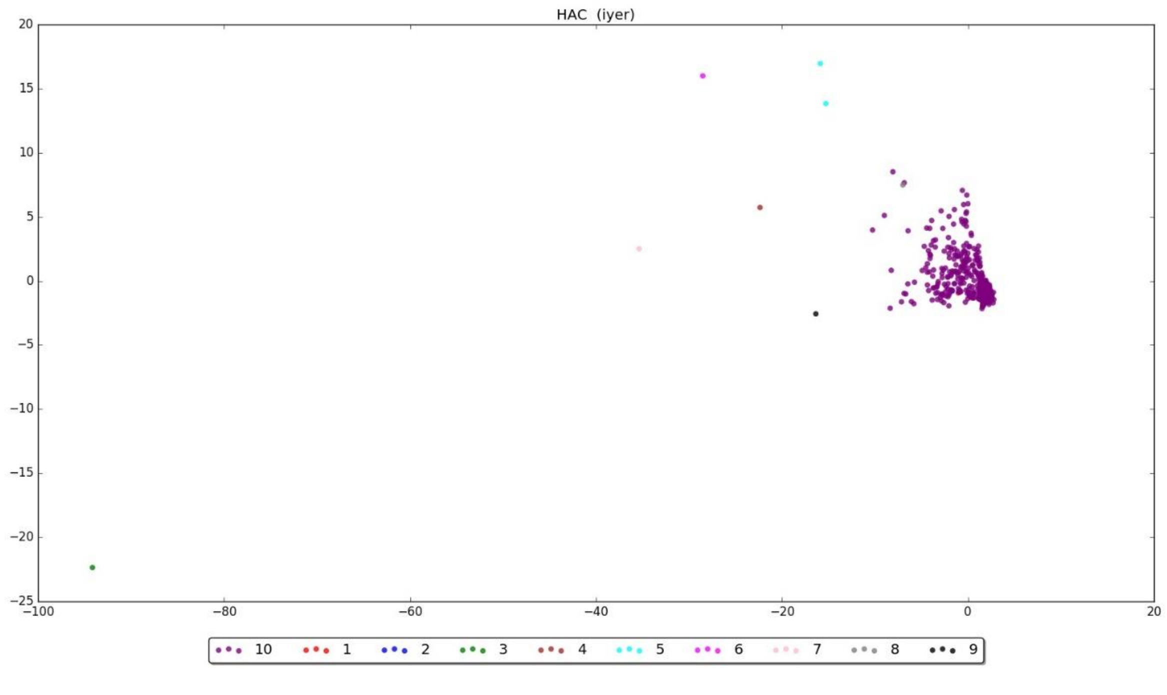
No. of Clusters	Jaccard Co-efficient	Rand Index	Iterations	Time taken
3	0.15647222380925174	0.17144364339722173	514	346ms
5	0.1559309385706048	0.1621204015129691	512	350ms
8	0.15710452461338867	0.1808566757330081	509	347ms
10	0.15824309696642858	0.1882868355974245	507	347ms
15	0.15945154219073931	0.20667891308658418	502	354ms

C) Visualization (using PCA):

1. Cho.txt



2. Iyer.txt



D) **Result Analysis:**

- Since we used MIN based distance metric for HAC, eventually we find that for k = 5 running on cho.txt, all the points except 4 single points are assigned to one single cluster.
- The rest of the 4 points are assigned one cluster each, making a total of 5 clusters. This is an expected result due to using Single Link/Min HAC.
- HAC does not distinguish between outlier and non-outlier points and thus all points are assigned eventually to one of the k clusters.

E) **Pros:**

1. Easy to implement.

2. No apriori information about number of clusters is required.

3. Produces better quality clusters compared to k-means or DBScan.

F) **Cons:**

1. Computationally expensive - O(n^2).

2. Has larger space complexity.

3. All merges are final, which can cause problems with noisy, high-dimensional data.

---

3. Density-Based Scan:

- DBSCAN is a density-based clustering algorithm such that given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away).

A) Steps for DBScan:

1. Pick a point at random that has not been assigned to a cluster or been designated as an outlier.
2. Compute its neighborhood to determine if it's a core point. If yes, start a cluster around this point. If no, label the point as an outlier.
3. Once we find a core point and thus a cluster, expand the cluster by adding all directly-reachable points to the cluster. Perform "neighborhood jumps" to find all density-reachable points and add them to the cluster. If an an outlier is added, change that point's status from outlier to border point.
4. Repeat these two steps until all points are either assigned to a cluster or designated as an outlier.

B) Validation Table:

1. Cho.txt

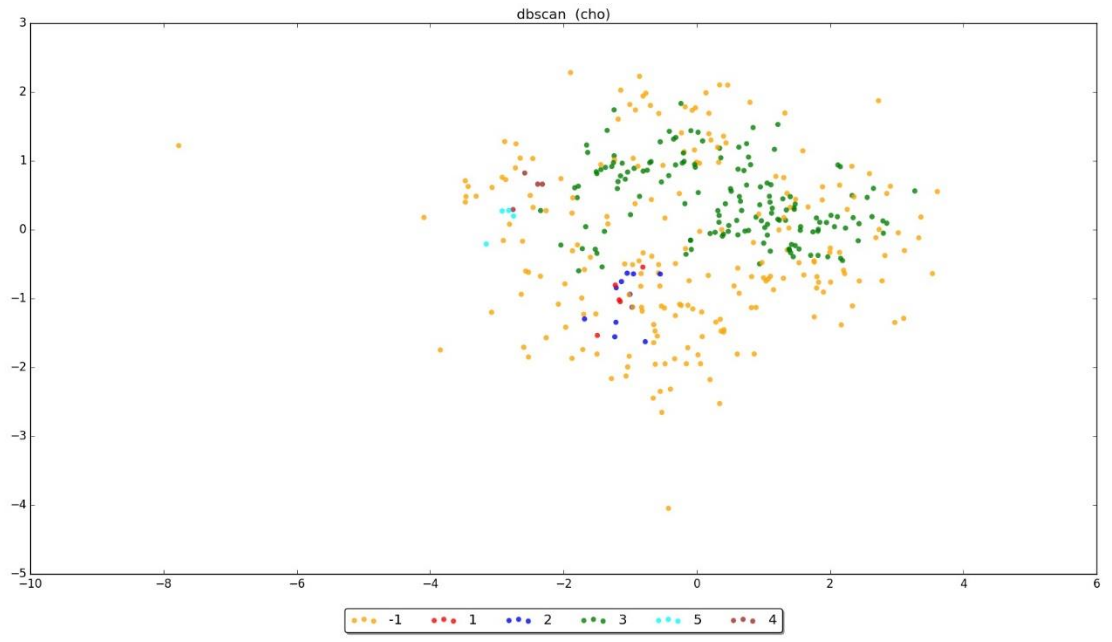
Epsilon (eps)	Number of min pts (minpts)	Rand Index	Jaccard Co-efficient
1.08	4	0.5678	0.2079
2.2	6	0.2550	0.2277
4.4	6	0.2337	0.2303
6.6	6	0.2300	0.2300
6.6	8	0.2300	0.2300

2. Iyer.txt

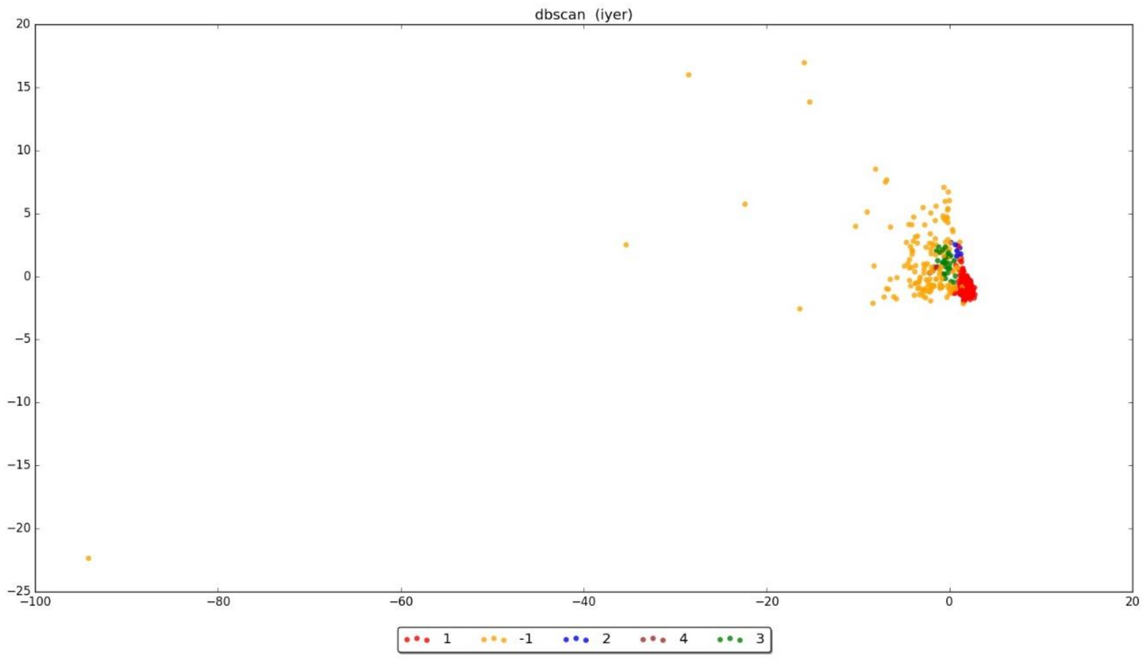
Epsilon (eps)	Number of min pts (minpts)	Rand Index	Jaccard Co-efficient
1.03	4	0.6523	0.2840
2.2	6	0.3041	0.1744
4.4	6	0.1968	0.1586
6.6	6	0.1772	0.1566
6.6	8	0.1807	0.1571

C) Visualization (using PCA):

1. Cho.txt



2. Iyer.txt



D) <u>Result Analysis:</u>

- From the validation above we can deduce that the accuracy is better for parameters eps = 1.03, minpts = 4 for iyer.txt and eps = 1.08, minpts = 4 for cho.txt.
- We observed that by increasing the parameters eps and minpts, Rand Index and Jaccard Co-efficient decreased in most of the cases. Also, the number of clusters formed decreased with increase in eps and minpts.
- This simply justifies the fact that clusters are difficult to form with increase in distance between points (eps) and with increase in number of minimum points (minpts).
- In some cases, after a certain point there’s no change with increase in the input parameters.

E) <u>Pros:</u>

1. Unlike K-means DBSCAN does not require one to specify number of clusters.
2. DBSCAN can find arbitrarily shaped clusters
3. DBSCAN is robust to outliers as it doesn’t cluster “noise”.
4. DBSCAN requires just two parameters and is mostly insensitive to the ordering of the points in the database
5. The parameters minPts and ε can be set by a domain expert, if the data is well understood.

F) <u>Cons:</u>

1. DBSCAN does not do a good job on border points as they can be put in any cluster by the order in which they are processed.
2. Since most of the times Euclidean Distance is used for DBSCAN, in case of high-dimensional data, this metric can be rendered almost useless due to the so-called "Curse of dimensionality", making it difficult to find an appropriate value for ε.
3. DBSCAN cannot cluster data sets well with large differences in densities, since the minPts-ε combination cannot then be chosen appropriately for all clusters.
4. If the data and scale are not well understood, choosing a meaningful distance threshold ε can be difficult.

Part 2: MapReduce K-Means

- K-Means implemented in MapReduce (Hadoop) can leverage the advantages of parallelism if we have the resources, that is, multiple number of computing nodes, rather than a single computer, to efficiently run k-Means over a large data set and give the results in a shorter amount of time than it would take with non-parallel k-Means.

A) Steps for K-Means (MapReduce):

The MapReduce implementation of k-Means can be divided into **three** parts:

- **Job Control** - which controls the running of the job and knows when to stop the job, or until converged
- **Mapper** - which receives input line by line and assigns each row (point) to one of the clusters by finding the cluster which is nearest to the point.
- **Reducer** - which receives a cluster centroid and it's corresponding points and then uses these points to calculate the new centroid, and does this for all the centroids. It also checks if the new centroids and old are same or not, and if they are same, it notifies the Job Control to stop, else it increments the counter by 1, so that the job runs again.

i) Mapper:

1. In the setup() function of the Mapper, we first determine the initial centroids and write them to a file. If initial centroids already exist, then we check for the previous centroids which were given by the last run of the reducer.
2. The map() function reads each line and computes it's distance from each of the k-centroids, and finds the nearest one.
3. map() emits the centroid assigned and the rowID.

ii) Reducer:

1. The setup() of reducer reads the current centroids from the centroids file and stores them.
2. The reduce() function takes in a centroid and it's corresponding rowIDs, and then computes the new centroid by averaging the sum of each of the data points.
3. reduce() outputs the new centroid and it's corresponding rowIDs.

4. The cleanup() function checks if the new centroids and centroids are equal or not. If they are not equal, the iterationCount is incremented by 1. If they are equal, that means the algorithm has converged and we can now write the output.
5. The cleanup() function then computes the Jaccard co-efficient and Rand Index for the given dataset and writes them to a file along with the output.

iii) Job control (main() function):

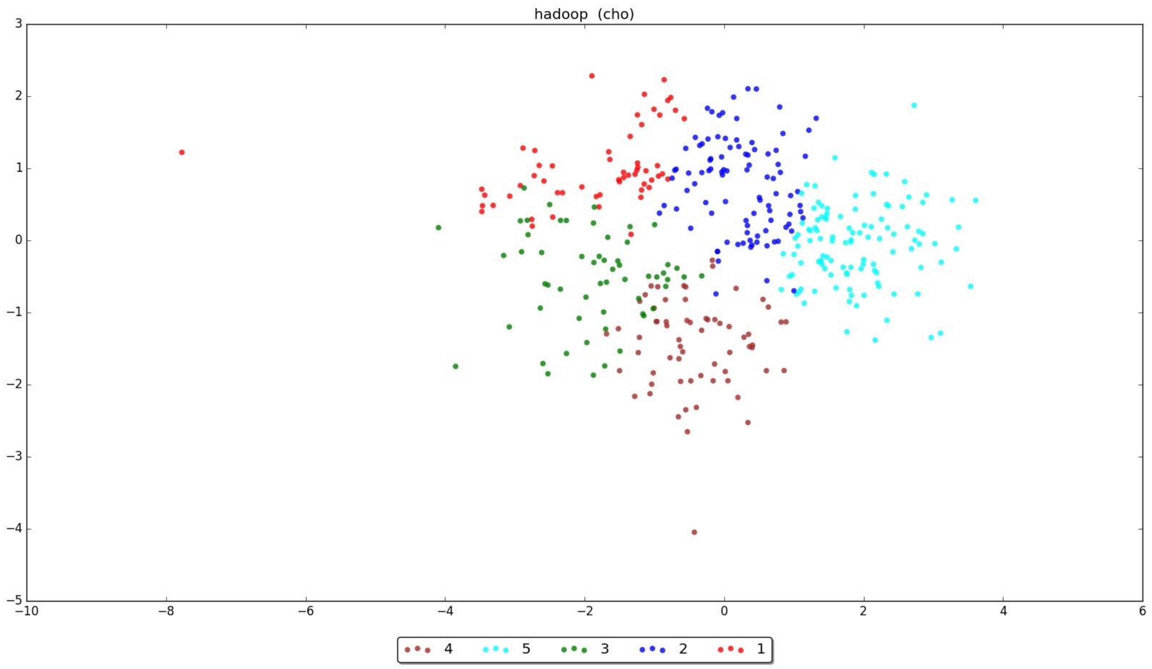
1. The main function takes from user input and output locations as arguments
2. It sets the number of clusters = k. By default, k = 5, if no argument is passed.
3. While(result>0), the job is run, again and again. Eventually, when the job converges, result would be zero, and we will exit the loop.

B) Validation:

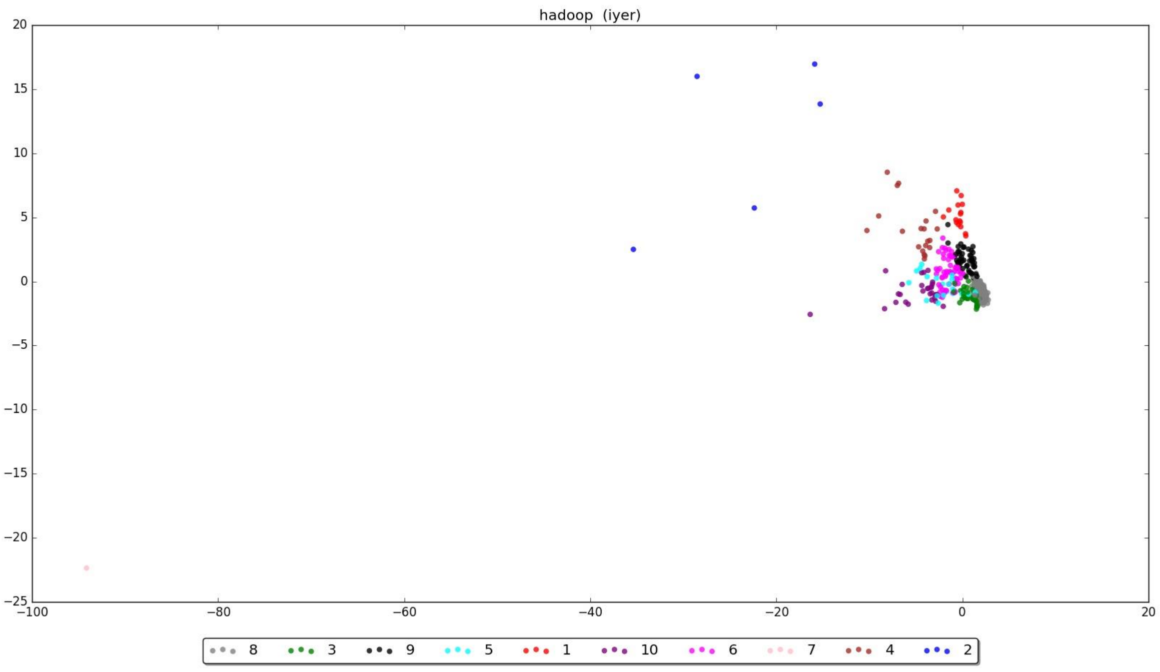
File	Clusters	Jaccard Co-efficient	Rand Index	Iterations	Time taken
cho.txt	3	0.4292561158262606	0.7698193240086982	6	11885ms
cho.txt	5	0.4073040427398134	0.8004510188192971	17	27242ms
cho.txt	8	0.284204423831609	0.7845445515315848	22	34281ms
iyer.txt	3	0.2199408701534227	0.4995304707638549	24	36930ms
iyer.txt	5	0.27680630499518616	0.6374710519325524	15	24877ms
iyer.txt	10	0.393427481426946	0.830468144966684	21	32868ms

C) Visualization (using PCA):

1. Cho.txt



2. Iyer.txt



D) <u>**Result Analysis**</u>:

- The results for these datasets for both non-parallel and parallel k-Means yeild similar results, with not too much difference in the external index validations.
- But for larger dataset, parallel k-Means will show a significant improvement in the performance compared to non-parallel k-Means.
- We see observe that for this small data-set, and single node setup, MapReduce takes lot of iterations to converge, compared to normal k-Means.
- However, if the dataset is large enough to warrant usage of parallelism, and we have enough computing resources, MapReduce k-Means should run faster than non-parallel k-Means.

E) <u>**Pros:**</u>

1. It is easily implemented
2. Significant performance boost over large datasets when compared to non-parallel k-Means.
3. Parallelism details and node configuration are handled by Hadoop itself.

F) <u>**Cons:**</u>

1. No significant difference in performance compared to non-parallel k-Means over smaller datasets.
2. Hadoop setup can be daunting for newer users.