Data Mining
Project 2
Clustering Algorithms

Arvind Srinivass Ramanathan | **arvindsr** | **50205659**
Naveen Muralidhar Prakash | **naveenmu** | **50208032**
Senthil Kumar Laguduva Yadindra Kumar | **laguduva** | **50207553**

DATA MINING
PROJECT 2
CLUSTERING ALGORITHMS

IMPLEMENTATION:
K-Means clustering algorithm:

The K-Means algorithm produces clusters based on the distance of each point from a set of initially chosen centroids. The algorithm runs in multiple iterations and for every iteration, the clusters centroids are updated based on the clusters that are formed up to that point. This is repeated until the centroids in successive iterations remain the same, denoting that the clusters have converged.

- The input file was read and the ground truth clusters are found and saved into '*clusters'*.
- The gene features are loaded into a matrix called '*matrix'*.
- The initial set of centroids are passed as command line arguments to the program such that they correspond to specific data points (gene id's) in the input matrix.
- Iterate through every data point in matrix, and find its distance to each of the centroids, and assign the data point to the centroid/cluster that it is closest to.
- We now have a set of data points, each belonging to one of the clusters that we have. For every cluster, take all the data points belonging to that cluster and compute the new centroid. We have as many centroids as the number of clusters.
- Repeat this process iteratively until we get the same centroid in the current iteration as the previous iteration, i.e., repeat the process until the clusters merge.

Agglomerative Hierarchical clustering algorithm:

The agglomerative hierarchical clustering algorithms are based on a bottom up approach. Here each individual point is considered a cluster. Through subsequent iterations, the closest points in the cluster are joined to each other and they now form a single cluster. This process is continued and in the later stages clusters come together to form one cluster when two nearest points in different clusters become the nearest point among the data set. This is because the distance between two clusters is represented as the closest distance between two points and it is called single link clustering.

The input file was read and the ground truth clusters are found and saved into 'clusters'.

- The gene features are loaded into a matrix called '*matrix'*.
- The distance between each data point to all other data points is found and saved into '*disMatrix'*.
- Algorithm:
    - **i.** Create a dictionary and add every data point from the input matrix into its own list, such that there are n clusters, each of which is a singleton cluster.
    - **ii.** In the disMatrix, find the two data points that correspond to the minimum distance amongst all the data points we have.

iii. Now, the clusters containing these data points need to be merged into one. Apply Set Union operation on the clusters containing these two data points to merge them.
iv. Add the above merged cluster into the dictionary. Remove the two clusters that contained these two data points before they were merged.
v. In the disMatrix, update the distances to reflect that the distance between the two points we obtained is infinity, and also, as the algorithm we apply is Single-link Min, update the distances in the disMatrix to reflect the minimum distance from the new cluster to each of the input data points.
vi. Iteratively repeat the above process until you end up with a single cluster, or a predefined number of clusters that you require.

Density Based Scan algorithm:

The basic principle in this implementation is that in density based scan, the clusters are differentiated based on the regions of low density space of data points between regions with high density of data points. It is good at discovering clusters of arbitrary shapes. The algorithm allows the epsilon distance of a data point to be chosen and the number of points that needs to appear within it.

- The input file was read and the ground truth clusters are found and saved into '*clusters'*.
- The epsilon (eps) value and the minpts are given as command line arguments.
- The gene features are loaded into a matrix called '*matrix'*.
- The distance between each data point to all other data points is found and saved into '*disMatrix'*.
- **DBSCAN Algorithm:**
  i. For each unvisited data point P in the matrix, mark them as visited and the neighboring points are found using '*regionQuery*' and saved into '*neighbors*'.
  ii. Given a point R, '*regionQuery'* returns all points that are within epsilon distance or '*eps'* from point R.
  iii. If the total number of neighboring points of P found in the previous step is less than minimum points or '*minpts',* then point P is marked as noise.
  iv. Else initialize a new cluster C and expand the cluster to include neighbors of all points in list '*neighbors'*.
  v. *Point P* is added to the cluster.
  vi. For each point, Q in the list '*neighbors*' , if Q is not visited then we mark them as visited.
  vii. Find all neighbors of point Q using '*regionQuery'* and save into '*neighborsOfPoint'*.
  viii. If the total number of neighboring points of Q found in the previous step is greater than or equal to minimum points or '*minpts'* then point Q is '*neighborsOfPoint'* is merged with '*neighbors*'.
  ix. If Q does not belong to any cluster, then we add Q to the current cluster C

K-MEANS CLUSTERING USING MAP-REUDCE AND HADOOP:

K-Means implementation is done using Hadoop framework using Map-Reduce program. Here we are distributing the work to multiple servers and then bringing together their output to form different clusters.

i. Read the input file and gather all gene expressions into '*clusters'*.
ii. Get the total number of clusters into *x* .
iii. Load just the gene's attributes or features into '*matrix'*
iv. Based on the number of centroids needed, initialize and append the centroids to a list called '*centroidList'*
v. Save the entire content of the '*matrix'* and '*centroidList'* into '*finalList'*.
vi. Run the terminal commands running in python. In this we call the mapper and reducer programs and the input file is sent to mapper. The input file has the list of centroids followed by the input data matrix.

i. Mapper

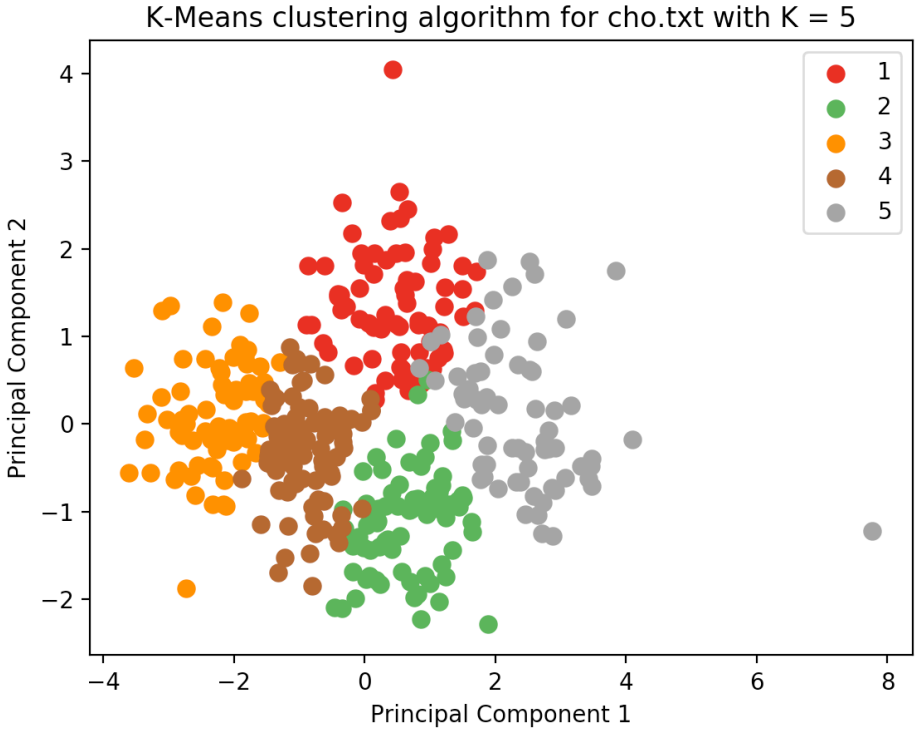
i. Reads the input file which we have sent as system input.
ii. For every point in the '*matrix'* , it finds the closest centroid to it and saves this into '*cluster'*.
iii. It then sends the output to the reducer.
Format: Cluster number \t gene ID in the data matrix represented as a string split by ','.

ii. Reducer

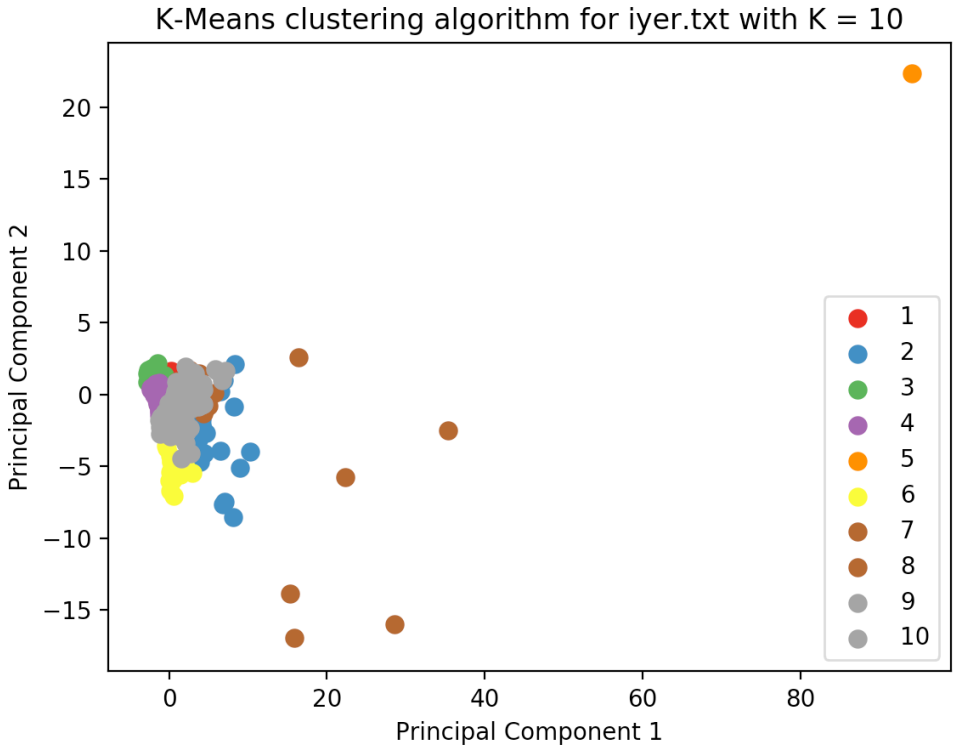
i. For every line which we receive as system input from the mapper, we split and save the details into '*cluster'*, '*index'* and '*matrixString'* in sorted order of clusters.
ii. Here for every cluster we calculate the centroid by continually adding the corresponding gene expressions of all data points in a cluster.
iii. This is sent in the format cluster number \t its gene id's \t list of gene ID belonging to that center. The reducer writes this output to part files.

Now the control switches back to driver. We read the output using sub process command and get the new centroid and compare if they have converged with the centroids from the previous iteration. At this point we have map for every gene id to its cluster. This process iterates continuously till we generate the same centroids through a couple of iterations i.e., when the clusters merge. Jaccard coefficient is calculated for our cluster classification against the ground truth. PCA is implemented to visualize the gene expressions in a graph.

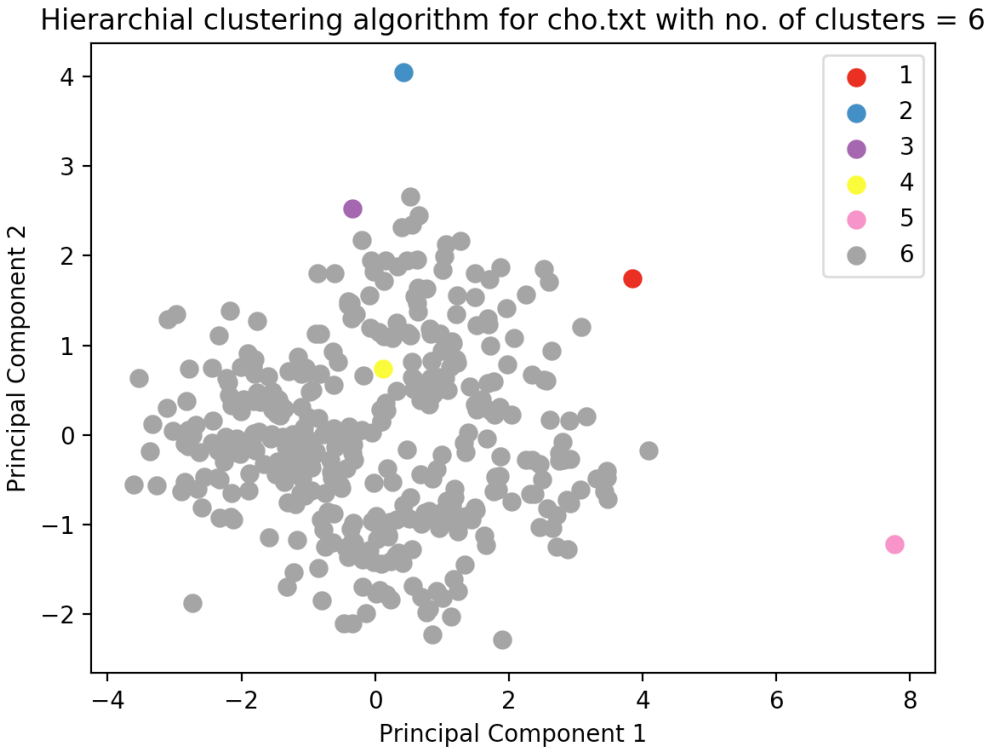
COMPARISON OF CLUSTERING ALGORITHMS USING VISUALIZATION AND JACCARD COEFFICIENT:



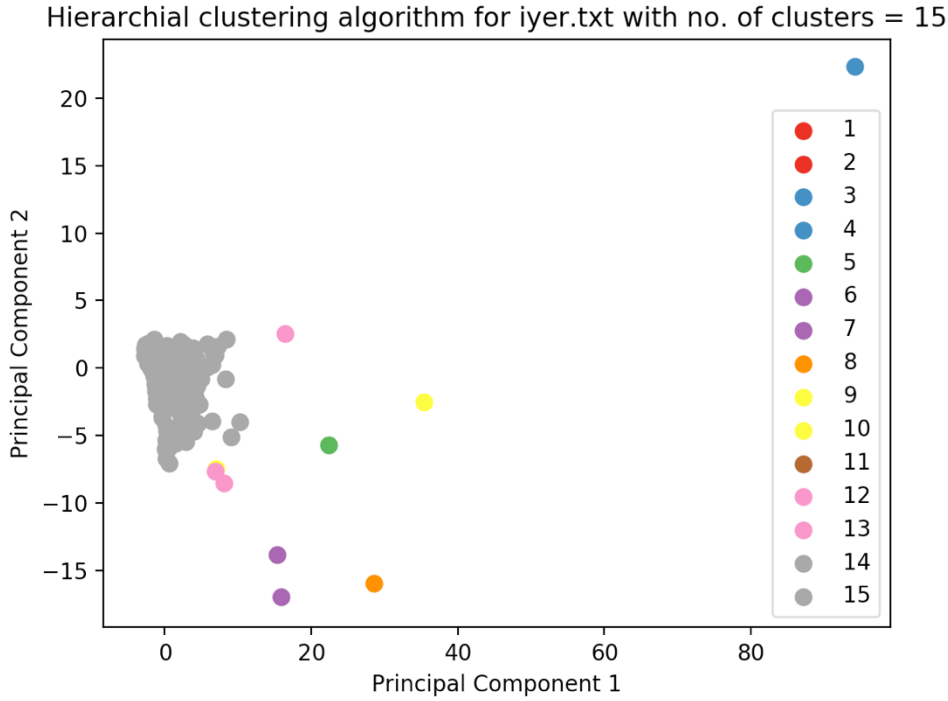
JACCARD COEFFICIENT = 0.3388



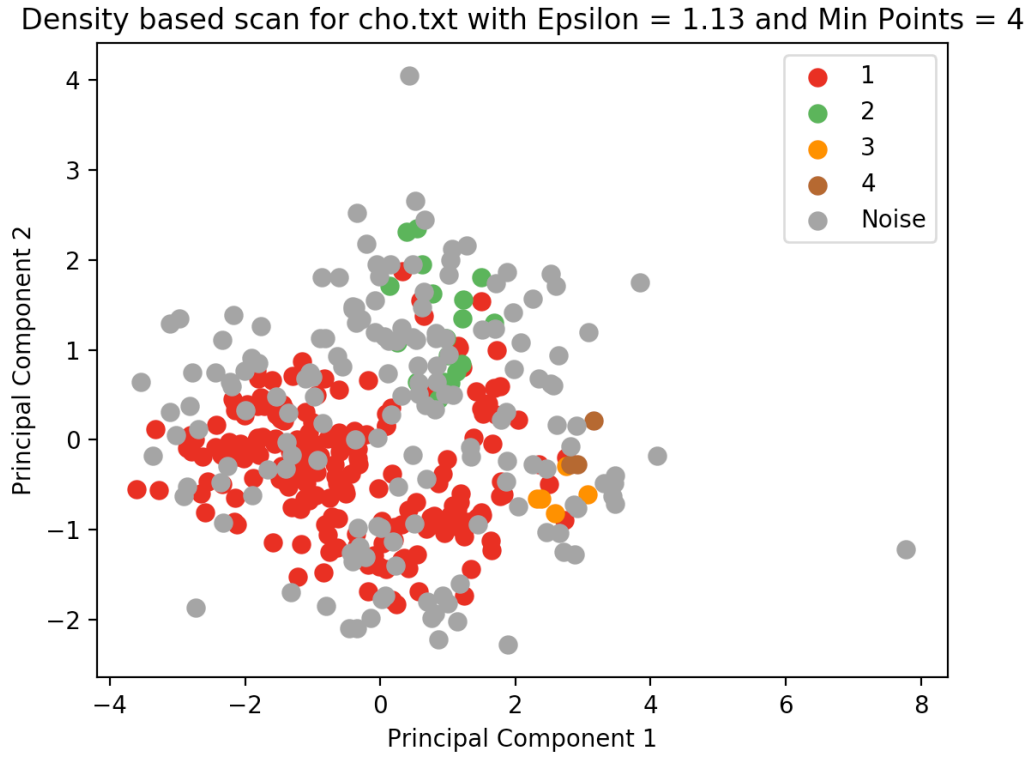
JACCARD COEFFICIENT = 0.3007



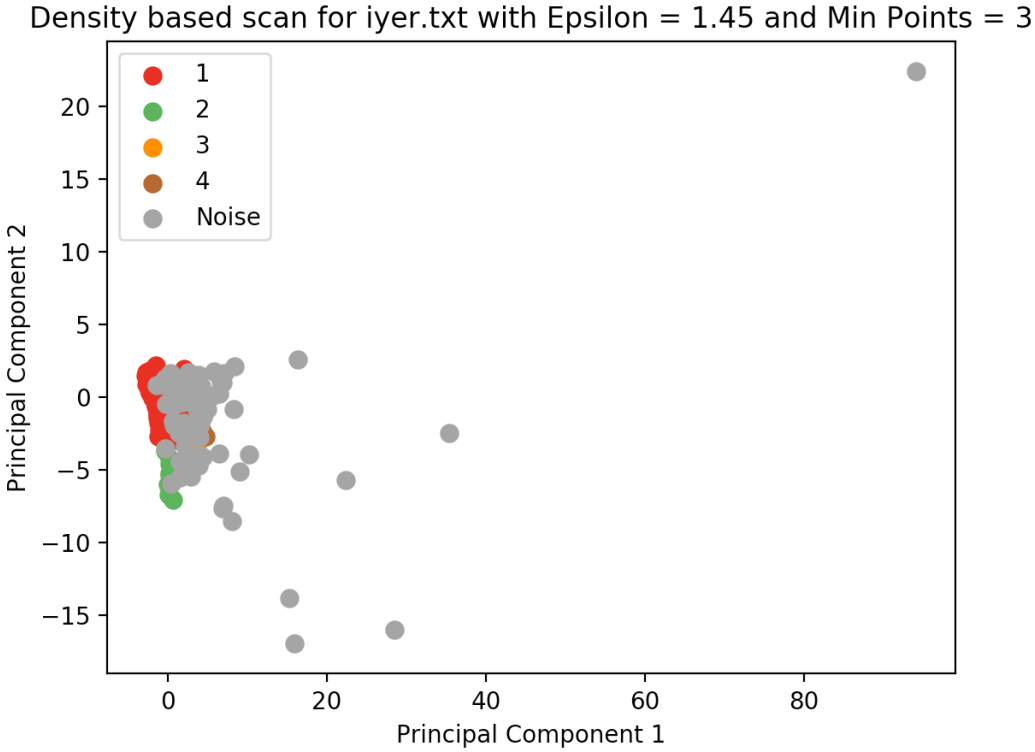
JACCARD COEFFICIENT = 0.2284



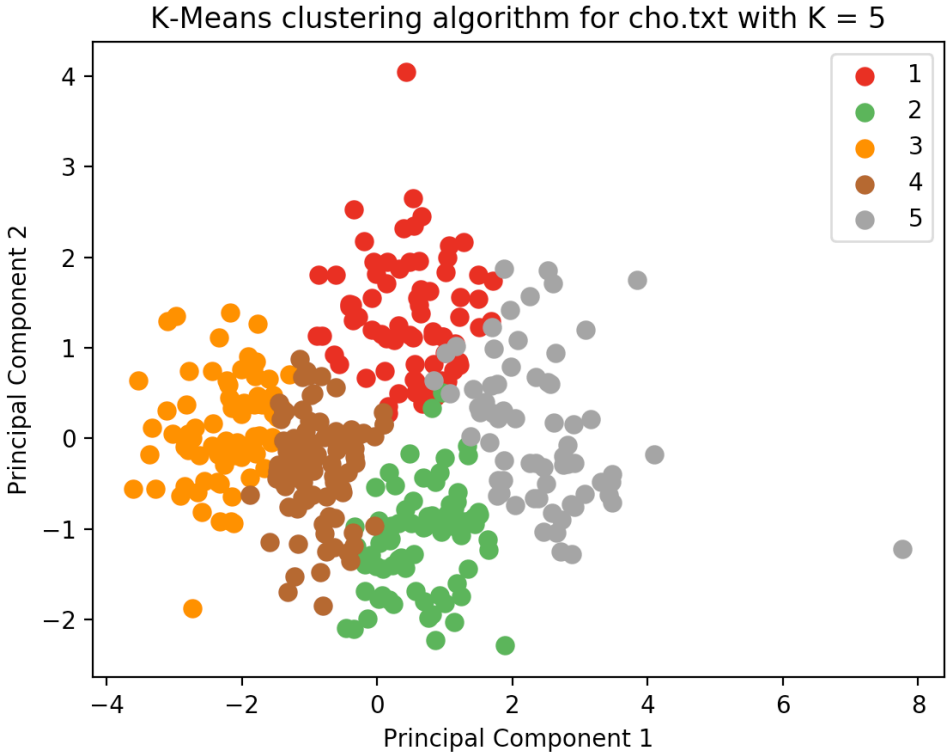
JACCARD COEFFICIENT = 0.1594



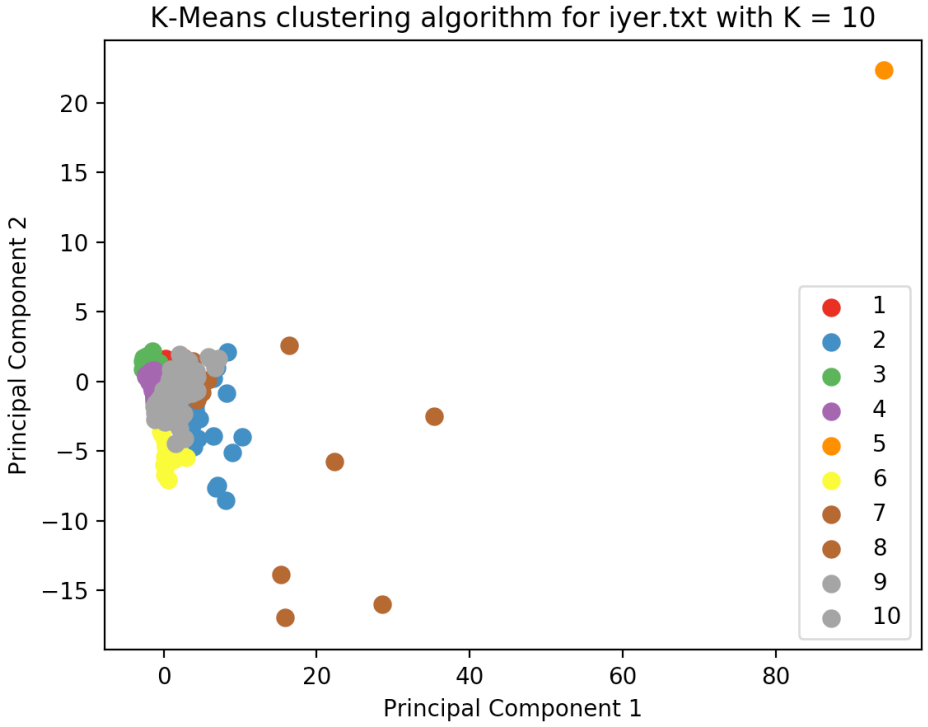
JACCARD COEFFICIENT = 0.3388



JACCARD COEFFICIENT = 0.2016



JACCARD COEFFICIENT = 0.3388



JACCARD COEFFICIENT = 0.3007

K-Means

- K-Means clustering algorithms gives us fairly distributed clusters for cho.txt and iyer.txt.
- This can be attributed to the spherical distribution of data points.
- In the case of iyer.txt specifically , what we notice is that all points around a center form a cluster, thus maintaining a spherical shape in general and also keeping nearby points as a cluster.
- Since the scale of the data points is more fair or moderate in terms of average distance from each other as of the case of cho.txt , we can say that the clustering look is seemingly better for cho.txt.
- We notice that the Jaccard coefficient is high for both the input data sets.

Hierarchical

- The points closest to each other join first to form a cluster. What we notice in the graph is that since we cut the dendrogram at a very high point to create a limited 5 clusters.
- Towards the end, only the points that are farther apart appear as a separate cluster and hence we get one dominant cluster consisting of most of the data points and other small clusters.
- The general shape of the cluster is noted to be arbitrary as the clusters when joining with each other do so based on the minimum distance from each other.
- Since we have many outliers in hierarchical clustering , the Jaccard coefficient is not the best when compared to the other clustering algorithms.

Density based scan:

- Density based scan looks for a deep difference between the density of clusters. For cho.txt we note that the differentiation between clusters is good. This has generated well-spaced clusters.
- In the case of iyer.txt , the points are closer to each other along the principal axis and are thus seemingly closer. However, they might be farther away from each other because they have been clustered independently.

Map-Reduce Hadoop:

- The observations were similar to that of K-Means.

PROS AND CONS:

K-Means clustering algorithm:

PROS:

- Time complexity: K means is linear in terms of data points in the number of data objects. Thus, time complexity is O(n). K-Means performs better than hierarchical clustering algorithm.
- Data points: It is proportional and can work well with larger data sets .

CONS:

- Partitioning: Produces a single partitioning.
- Consistency: K-means starts off by randomly choosing centroids. Due to this we cannot repeat the same output for every independent run and lack consistency.
- Number of clusters: One must manually chose how many clusters to generate at the beginning.
- Shape of the cluster: K-Means works well when the shape of the clusters is hyper-spherical.
- K–Means cannot differentiate in a scenario where clusters are not formed due to uniform distribution.
- Scale of the data points: The clustering algorithm generates very different results based on the scale of the data set. This means that we need to have the same distribution and the same variance across each axis.
- Limited in flexibility. Works only on limited data.

Agglomerative Hierarchical clustering algorithm:

PROS:

- Number of clusters: Not necessary to mention number of clusters at the beginning each time.
- Partitioning: Produces any number of partitioning based on when we cut the dendrogram.
- Flexibility: Can run on any data set. Can work with any measure.
- Dendrogram is useful for visualization which is an additional plus. It can show the relation or ordering between the points within a cluster.
- Can capture concentric clusters.

CONS:

- Time Complexity: Hierarchical clustering algorithm is quadratic to the number of data points. The time complexity is O($n^2 \log(n)$) where n is the number of data points.
- Scale of data set: Hierarchical performance is inversely proportional to data points.
- Deterministic: Algorithm progresses without the ability to undo any cluster merges.
- Cannot handle convex shapes or different sized clustering.

Density Based Scan Algorithm:

PROS:

- Time complexity: The time complexity is O(n^2) where n is the number of data points when the dimensions is low.
- DBSCAN can also find non-linearly separable clusters as a sharp difference in the density between clusters will separate them into different clusters.
- Since we only need a difference in the density between clusters to be significant we are able to differentiate clusters that do not have any defined shape.
- DBSCAN is sensitive to clustering parameters such as threshold minimum points and epsilon value.
- The algorithm cannot be partitioned when implementing with multiple processors.
- Works fast with lower dimensions of data.
- Doesn't need any number of clusters to be chosen.
- Not entirely deterministic. This is because a border point can belong to two different clusters if it appears between two core points at epsilon distance.

CONS:

- DBSCAN cannot identify clusters if density varies or if density of the data set is too sparse.
- Difficult to choose a distance measure if we do not know what the data points look in terms of distribution.
- Struggles with data with high dimensions.
- DBSCAN depends on the distance measures used. This is because we need to choose an appropriate distance value based on the type of distance measure used.

Map-Reduce Hadoop:

PROS:

- Hadoop is highly scalable. It can store and distribute large data sets across multiple servers. With the design of the map reduce processing we can operate in parallel thus improving efficiency.
- Hadoop MapReduce permits access to various new sources of data and operate on different types of data.

CONS:

- For small data sets the overhead being generated to handle the MapReduce style programming makes it slower. It is designed to work only on high capacity scenarios.

Map-Reduce Hadoop vs K-Means:

- In non-parallel K-Means , it is necessary to generate the Euclidean distance between all the points for each iteration. Thus, we can save this time if we were to implement the execution in parallel.
- Driver facilitates multiple iterations of K-Means and we store the file containing the centroids generated in the current iteration and the input matrix in a Hadoop file system. This helps in quicker execution and reduces computational intensity. We also have multiple mappers and reducers, thus facilitating parallelism.
- The speed of K-Means using MapReduce is slower than non-parallel K-Means because we run the same using a single node. In the case of execution with multiple nodes in a parallel manner, we will be able to reduce execution time.