Project 2:
Clustering Algorithms

Team Members (Person number / UBIT name):

Mitali Vijay Bhiwande (50208163 / mitalivi)

Sumedh Sadanand Ambokar (50207865 / sumedhsa)

Tejasvi Balaram Sankhe (50207071/ tejasvib)

1. K-MEANS ALGORITHM

1.1 INTRODUCTION:

K-means clustering is a type of unsupervised learning, used with unclassified data. Algorithm aims at finding groups within the data with each group specifying similar characteristics. The algorithm works iteratively to assign each data point to one of *K* groups based on the features that are provided. Finally, the data points are clustered based on feature similarity.

1.2 IMPLEMENTATION:

- **Partition {x_1,…,x_n} into *K (predefined)* clusters**
- **Initialization**
  Specify the initial cluster centers (centroids)
- **Iteration until no change**
  For each object x_i
  Calculate the distances between x_i and the *K* centroids
  (Re)assign x_i to the cluster whose centroid is the closest to x_i
  Update the cluster centroids based on current assignment

Algorithm:

Algorithm is implemented using the following modules.

1. **fetch_data**(*filename, num_of_clusters, centroid_array, num_of_iterations*):

- This function takes as input the input file name, required number of clusters in the final result, list specifying the data points with IDs as initial centers and the iteration number to be used.
- It reads the data points from the file and stores them as “original_data” which is being used further in computing k mean clusters.
- The initial centroids,”centroids” are fetched using the IDs of the data points provided as input. If the ID’s aren’t provided, an empty list need to be passed as input and the function will randomly select centroids from original data as its initial cluster centroids.
- The transformed data and computed centroids along with the number of iterations to be performed are used to compute k mean clusters.

2. **compute_kmean**(*original_data, centroids, num_of_iterations*):

- This function takes as input the original data, the current centroids that are to be used for computing k mean clusters and the number of iterations to be performed.
- It divides the original data into clusters by computing the Euclidean distance of each point in original data with the input centroids. The data points with minimum distance from one centroid belong to that particular cluster signified by the centroid. It also maintains the indices of each data point, ”cluster_indices”, in original data in clusters it corresponds to.
- The formed clusters are then used again to compute new centroids and look for convergence where the algorithm would finalize the clusters and provide same centroids after computing new clusters. The stored cluster indices are used for verifying results using jaccard co-efficient / rand index.

3. **compute_centroids**(*centroids, clusters, original_data, cluster_indices, num_of_iterations*):

- The computation of k mean clusters includes an intermediate step of computing the centroids of newly formed clusters and verifying them. The compute_centroids function performs this action by taking as input the previous centroids, formed clusters, the original data set, stored cluster indices and the remaining number of iterations.
- The mean for the formed clusters is computed and is compared with previous centroid values. If the values match or the number of iterations to be performed equals zero, then the newly computed centroid value," average", is used as final cluster centroids and the data is plotted as cluster representation of all data points in original data. If the values do not match and the number of iterations to be performed are greater than zero then compute_kmean function is called again by passing the updated values.

4. **compute_coeffs**(*original_data, cluster_indices*):

- Using the original data and cluster indices computed in compute_kmean method are used to compute the jaccard co-efficient and rand index.
- Jaccard co-efficient is computed as:

$$Jaccard\ coefficient = \frac{|M_{11}|}{|M_{11}| + |M_{10}| + |M_{01}|}$$

- Similarly, Rand Index is computed as:

$$Rand = \frac{|Agree|}{|Agree| + |Disagree|} = \frac{|M_{11}| + |M_{00}|}{|M_{11}| + |M_{00}| + |M_{10}| + |M_{01}|}$$

Where, M_{11} : (agree, same cluster) M_{00} : (agree, different clusters)

M_{10} : (disagree, different clusters) M_{01}: (disagree, different clusters)

5. **draw_plots** and **draw_scatter_plot** functions are used to visualize the k mean clustering results. The resulting data points in each cluster are represented by PCA (Principal Component Analysis). The ground truth and cluster numbers for each data point is maintained and data points belonging to same cluster are represented by same color on the plot.

External Libraries used:

- **numpy as np**: for numpy array/list related operations
- **distance from scipy.spatial** : to compute the Euclidean distance
- **PCA from from sklearn.decomposition** : to reduce dimensions of data points and visualize them while plotting
- **matplotlib.pyplot as plt :** For plotting the clustering results and PCA reduced data point

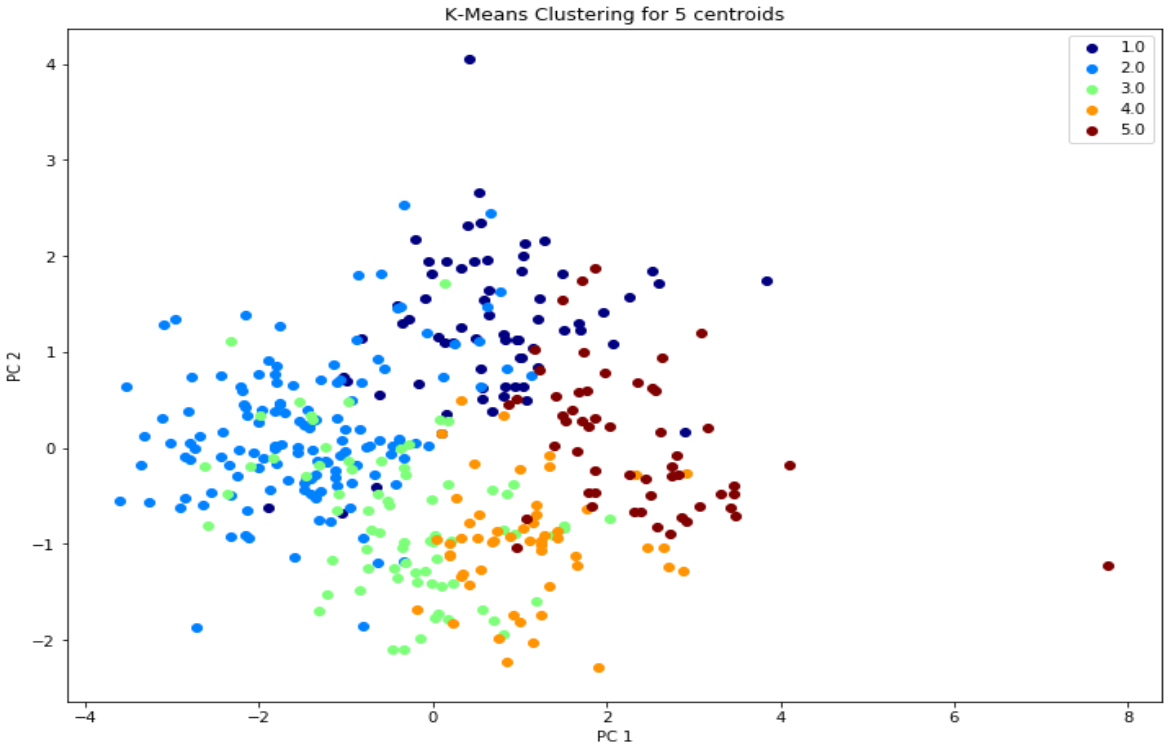
1.3 VISUALIZATION

The code outputs iteration number, number of clusters with gene id's and the Jaccard co-efficient and Rand Index

File Name: cho.txt

Original data with ground truth values is visualized as:

Here, the colors correspond to the cluster id assigned by the algorithm.



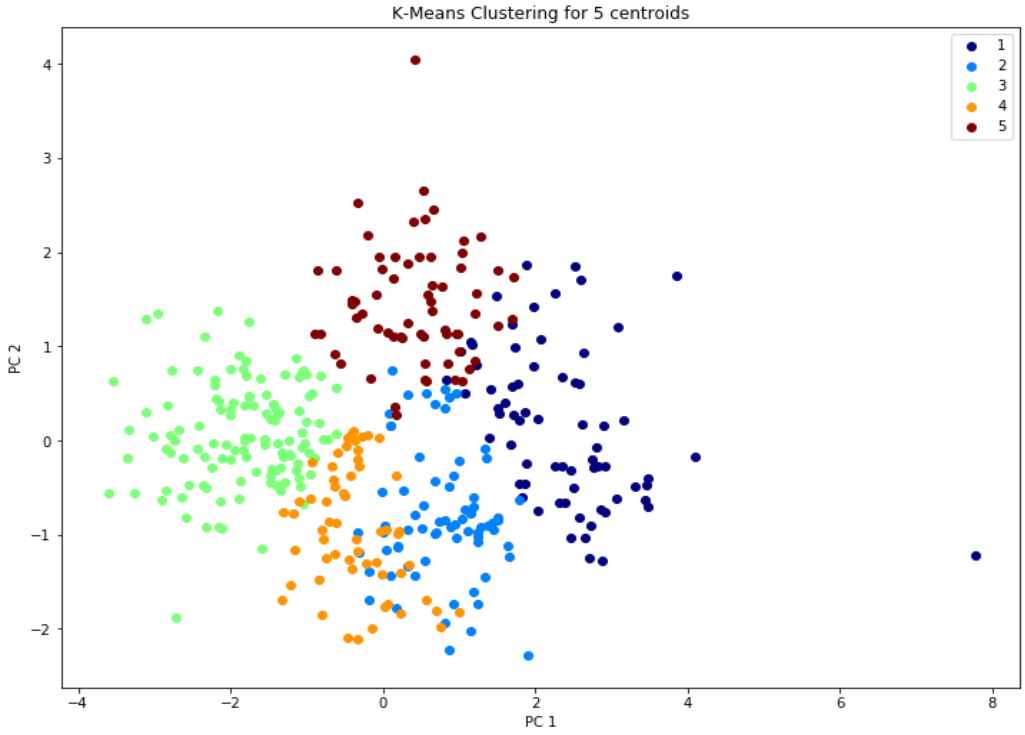
Cho.txt

1. Using the initial parameters as:
   fetch_data("cho.txt",5, [5,25,32,100,132],100)
   where, total number of clusters =5
   initial centroid data Ids: [5,25,32,100,132]
   Number of Iterations: 100

The obtained jaccard co-efficient and rand index for these fixed set of original centroids are:

jaccard_coefficient: 0.4062599065874844
rand_index: 0.8109548482605478

The plot for fixed set of centroids is:



Cho.txt

2. Using the initial parameters as:
   fetch_data("cho.txt",5,[],100)
   where, total number of clusters =5
   initial centroid data Ids: []
   Number of Iterations: 100
   As the initial centroids are not specified, the function will take as input an empty list and add values to the list by selecting random data points from the data set.

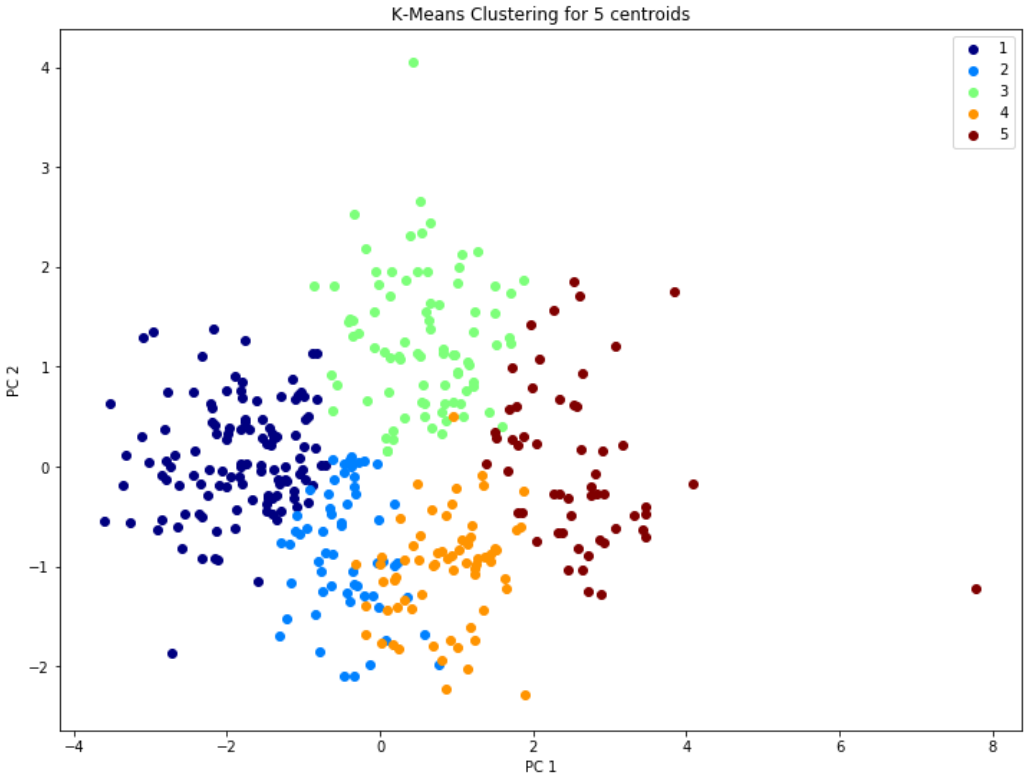
The obtained jaccard co-efficient and rand index for these fixed set of original centroids are:

jaccard_coefficient: 0.3386493117644613
rand_index: 0.7859699885606621

By selecting random values as initial centroids, we observe that the jaccard co-efficient for 5 cluster varies in the range 0.33 to 0.41

Similarly, the rand index varies in the range 0.75 to 0.81

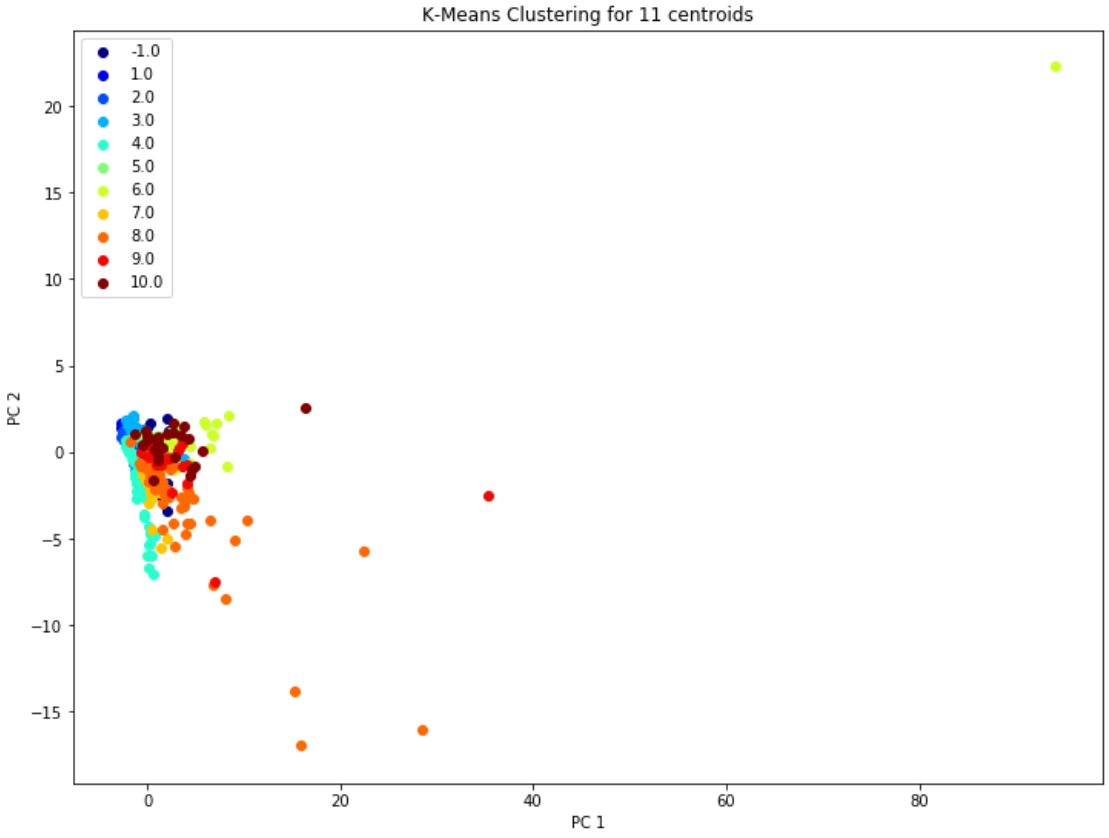
Plot for random centroid is:



Cho.txt

File Name: iyer.txt

Original data with ground truth values is visualized as:

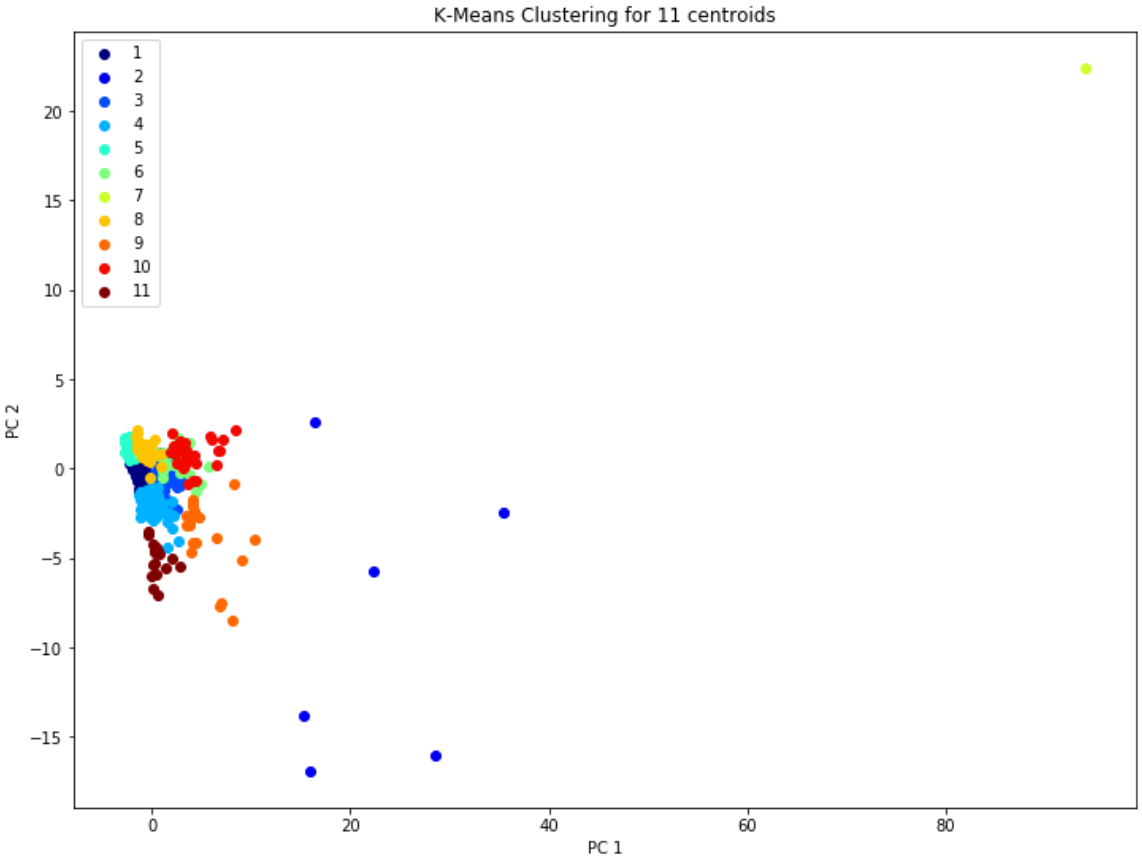


iyer.txt

3. Using the initial parameters as:
   fetch_data("iyer.txt",11,[45,2,56,23,1,67,3,11,5,89,180],100)
   where, total number of clusters =11
   initial centroid data Ids: [45,2,56,23,1,67,3,11,5,89,180]
   Number of Iterations: 100

The obtained jaccard co-efficient and rand index for these fixed set of original centroids are:
jaccard_coefficient: 0.3156443650877009
rand_index: 0.8046046811509454

The plot for fixed set of centroids is:



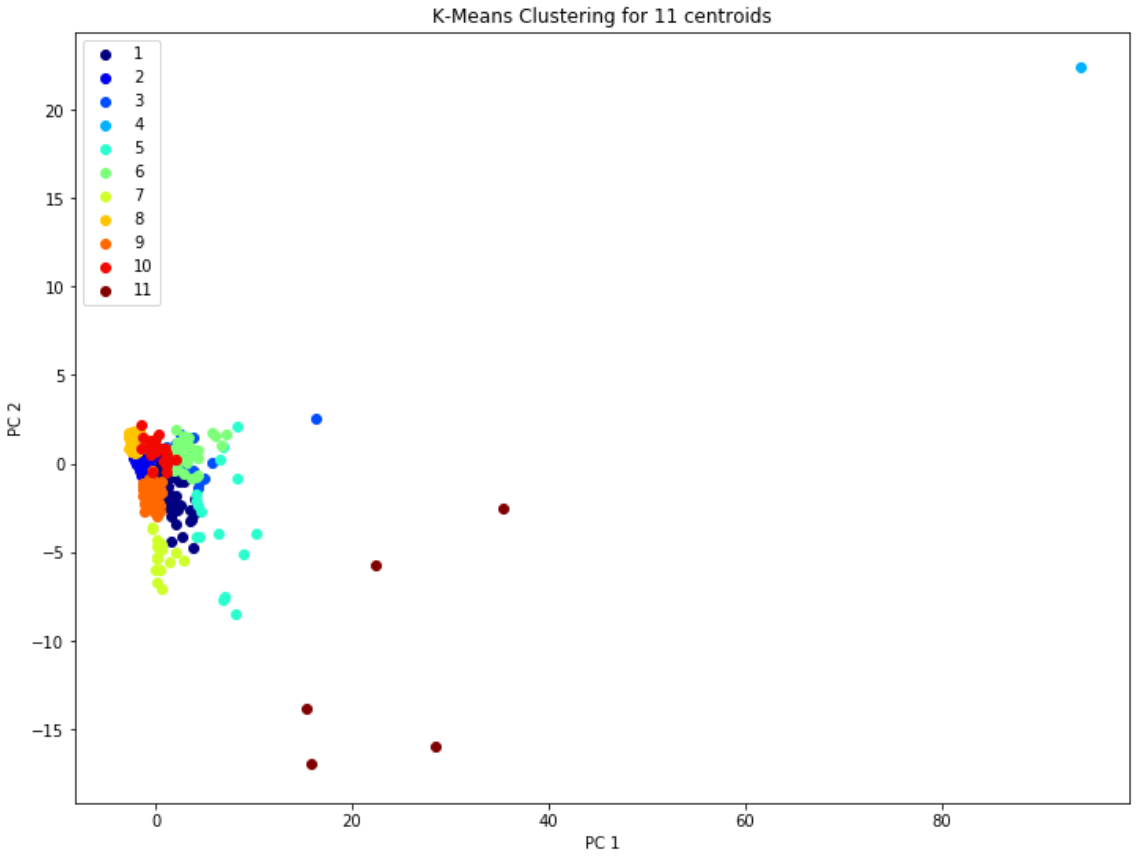
Cho.txt

4. Using the initial parameters as:
   fetch_data("iyer.txt",11,[],100)
   where, total number of clusters =11
   initial centroid data Ids: []
   Number of Iterations: 100
   As the initial centroids are not specified, the function will take as input an empty list and add values to the list by selecting random data points from the data set.

The obtained jaccard co-efficient and rand index for these fixed set of original centroids are:

jaccard_coefficient: 0.320355182177086
rand_index: 0.8083419549278035

By selecting random values as initial centroids, we observe that the jaccard co-efficient for 11 clusters varies in the range 0.30 to 0.33
Similarly, the rand index varies in the range 0.77 to 0.81



iyer.txt

1.4 RESULT EVALUATION

1. Higher jaccard similarity scores are obtained when initial centroids are chosen at random.
2. Sometimes the initial centroids will readjust themselves in 'right' way, and sometimes they don't

Advantages

- For smaller values of k, k-means provides computationally faster results.
- K means provides compact clustering results.
- K-means clustering is usually more efficient run-time wise :
  O(*tkn*), where *n* is # objects, *k* is # clusters, and *t* is # iterations. Normally, *k*, *t* << *n*
- k-mean algorithm is good for large dataset clustering

Disadvantages:

- Without initial ground truth values, it is difficult to predict the value of K.
- Selecting different initial centroids, provides different similarity results.

- Can lead to empty clusters. But most of the data points are usually clustered.
- Some pre and post processing may be required on the initial and final data for obtaining perfect results.
- Doesn't perform well when the clusters differ in sizes, densities and shapes.

Applications

1. Feature extraction from from images,videos,text, etc.
2. K Means clustering method can be used for anomaly detection in **Healthcare Fraud Detection.**
3. Can be used in cluster based techniques for plagiarism detection

2. HIERARCHICAL AGGLOMERATIVE CLUSTERING

2.1 DESCRIPTION

In hierarchical agglomerative clustering consists of a bottom up approach, we build a hierarchy of clusters represented using a dendogram. At the beginning each point is considered as a single cluster. Then each cluster is merged with their nearest neighbour (here based on minimum Euclidean distance) recursively till all points belong to a single cluster. The merging policy is greedy, each time, we only merge a pair of clusters that has the smallest inter-cluster distance. Other similarity functions can also be used in this algorithm to merge two clusters.

2.2 IMPLEMENTAION

Algorithm:

Given a set of N items to be clustered, and an N*N distance (or similarity) matrix

1. Start by assigning each item to a cluster, so that if you have N items, you now have N clusters, each containing just one item. Let the distances (similarities) between the clusters the same as the distances (similarities) between the items they contain.
2. Find the closest (most similar) pair of clusters and merge them into a single cluster, so that now you have one cluster less.
3. Compute distances (similarities) between the new cluster and each of the old clusters.
4. Repeat steps 2 and 3 until all items are clustered into a single cluster of size N. (*)

Implementation:

- We implemented the algorithm using Python 3. It consists of two functions hac() and hac_labels().
- The hac() takes as input the file path and number of clusters to be formed. It reads the points from the file and stores each row in matrix.
- The scipy.spatial package was used to compute the Euclidean distance for the points stored in the matrix.
- Dictionary data structure was used to track the clusters. At the start each point is stored as single cluster.
- The function then starts merging the clusters till only the given number of clusters remain.
- We implemented Single-linkage: the similarity of the closest pair
- For merging, first the minimum distance that is greater than 0 is identified between two points from the distance matrix and then those points are merged by updating the list of points in the dictionary and deleting the entry for the single point.
- The distance matrix is recomputed for remaining points using minimum distance between two of the merged points. For example if points 2 and 5 are merged in this step. Distance of all other points will be calculated w.r.t to cluster (2,5) using,

$$d((2,5), P) = \min(d(2,P),d(5,P)$$

- The distance matrix is recomputed every time two points or clusters are merged. The entry for one of the merged point is changed to 0, so that it will not be considered in the next iteration.
- The clusters from the dictionary are added to list finalclusters which contains the final given number of clusters formed.

- The function hac_labels(), takes as input this list of final clusters and assigns each point a label i.e. the cluster id they belong to. These labels are used for the PCA visualization.
- Since only two clusters are merged in one iteration the time complexity for this algorithm is $O(n^2)$ for all the points to be merged into one cluster.

External Index

- compute_coeffs() takes as input the original data and the gene ids with the cluster numbers they belong to.
- Ground truth labels and clustering labels are compared and the values for M11, M00, M10, M01 are updated accordingly.
- The Jaccard coefficient and Rand Index are calculated as per the formulas and displayed

Visualization

- Inbuilt PCA package is used from sklearn.decomposition to decompose the original data to two principal components for plotting purposes. The data points are colored as per the cluster they belong to.
- Cluster id are shown as legends for the plots.

External Libraries used:

- **numpy as np**: for numpy array/list related operations
- **distance from scipy.spatial** : to compute the Euclidean distance
- **PCA from from sklearn.decomposition** : to reduce dimensions of data points and visualize them while plotting
- **matplotlib.pyplot as plt :** For plotting the clustering results and PCA reduced data point
- **math:** to get the minimum Euclidean distance from the distance matrix.

2.3 VISUALIZATION

1. For cho.txt

```
jaccard_coefficient:  0.22839497757358454
rand_index:  0.24027490670890495
```

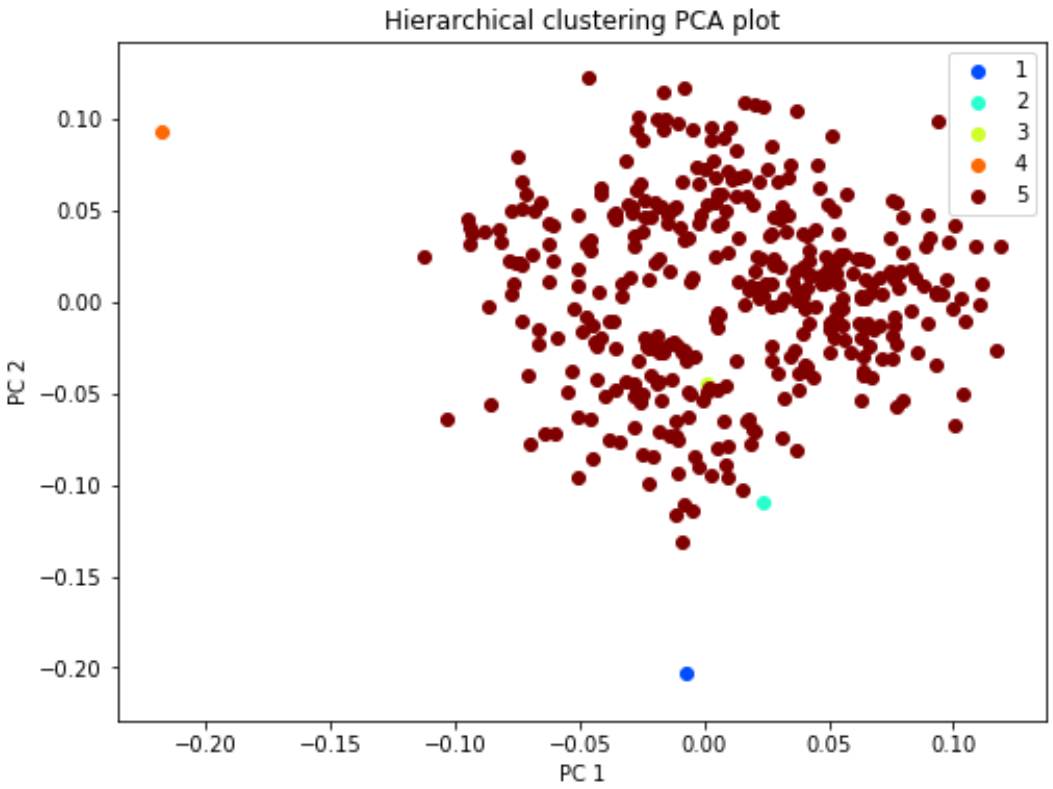


Fig 1. Result of hierarchical clustering on Cho Dataset, K=5

Different combinations of number of clusters provide a range for external index. The Jaccard co efficient varies from 0.22 to 0.24

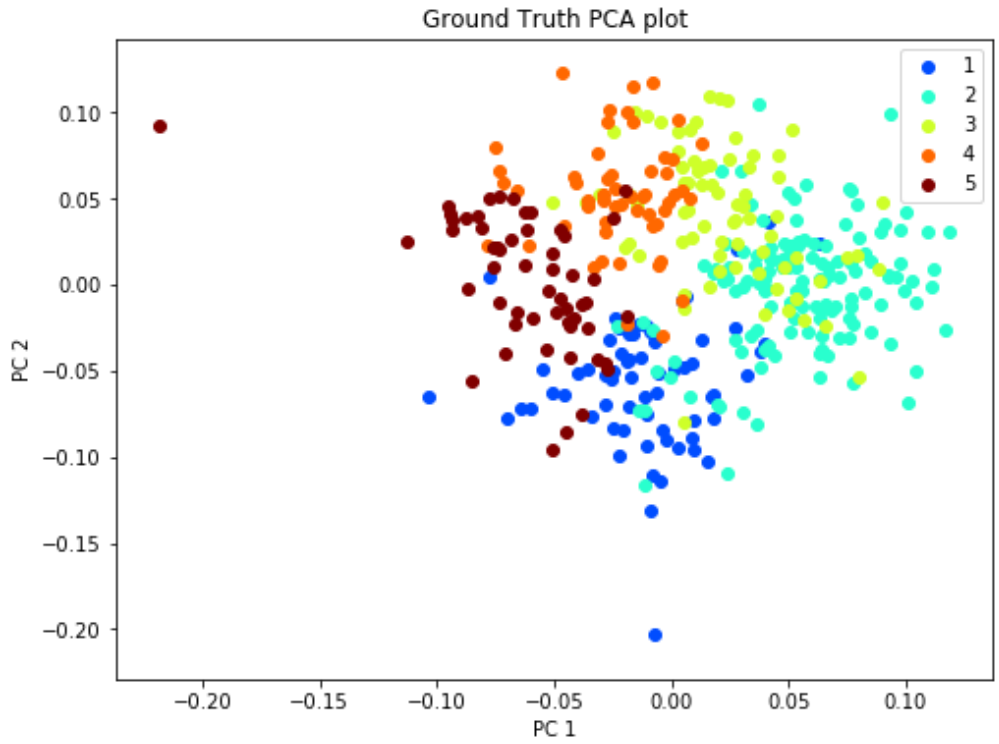


Fig 2. Ground truth PCA Cho Dataset, K=5

2. For Iyer.txt

```
jaccard_coefficient:  0.15824309696642858
rand_index:  0.1882868355974245
```

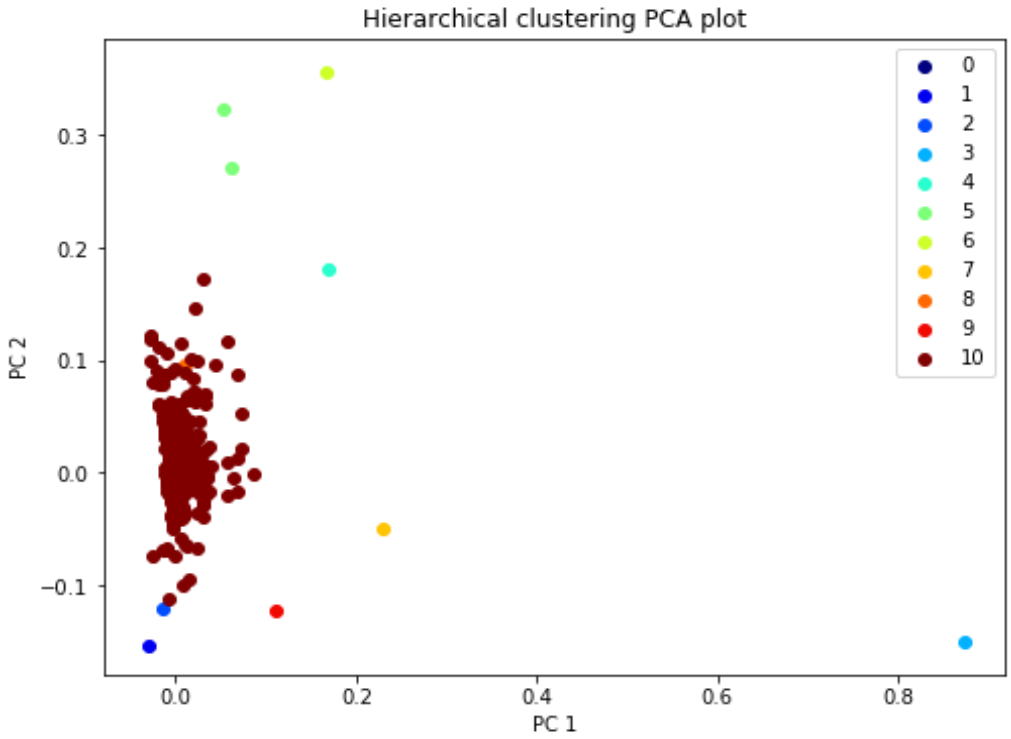


Fig 3. Result of hierarchical clustering Iyer Dataset, K=10

Different combinations of number of clusters provide a range for external index. The Jaccard co efficient varies from 0.15 to 0.18

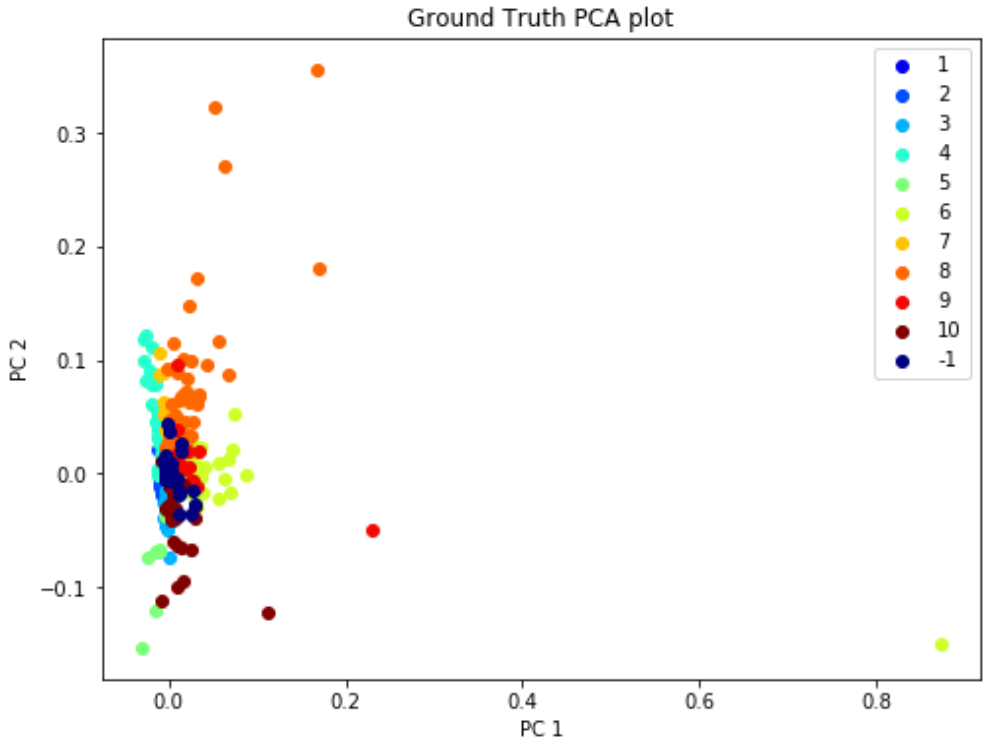


Fig 4. Ground truth PCA Iyer Dataset, K=10

2.4 RESULT EVALUATION

- We were required to implement hierarchical clustering using single linkage. In *single-linkage clustering*, the similarity of two clusters is the similarity of their *most similar* members i.e. while recalculating distance matrix the minimum distance of a point from any point belonging to that cluster is stored.
- This single-link merge criterion is *local*. We pay attention solely to the area where the two clusters come closest to each other. Distant points of the cluster and the clusters' overall structure are not taken into account.
- Hierarchical agglomerative clustering does not identify outliers in the data as it merges all the points. Also it does not provide evenly distributed clusters for user defined number of clusters. HAC used together with visualization of the dendrogram can be used to decide how many clusters exist in the data.

Advantages:

- Produces an ordering of the objects, which may be informative for data display.
- No a priori information about the number of clusters required.
- Useful summarization tool.

Disadvantages:

- Cannot identify outliers in the data
- Time complexity is higher
- Single linkage suffers from chaining. When merging only points with minimum distance are considered, irrespective of other points in the clusters.
- For datasets like cho, clusters with only a single point in it can be formed and thus the clusters are not always evenly distributed.

Applications:

- Hierarchical agglomerative clustering can be used in designing recommendation systems for music.
- Similarity groupings of images or documents
- Pattern recognition

3. DENSITY-BASED SPATIAL CLUSTERING AND APPLICATION WITH NOISE (DBSCAN)

3.1 DESCRIPTION

DBSCAN (**Density-based spatial clustering of applications with noise**) is a data clustering algorithm. It considers the basic idea that clusters are dense regions in the data space, separated by regions of lower object density. It groups together points that are closely packed together and marking as outlier's points that lie alone in low-density regions. It takes as input two parameters:

- ε-Neighborhood – Objects within a radius of ε from an object.
- "High density" - ε-Neighborhood of an object contains at least *MinPts* of objects.

3.2 IMPLENTATION

Algorithm:

```
DBSCAN(D, eps, MinPts)
    C = 0
    for each unvisited point P in dataset D
     mark P as visited
     NeighborPts = regionQuery(P, eps)
    if sizeof(NeighborPts) < MinPts
    mark P as NOISE
    else
    C = next cluster
    expandCluster(P, NeighborPts, C, eps, MinPts)
```

```
expandCluster(P, NeighborPts, C, eps, MinPts)
        add P to cluster C
        for each point P' in NeighborPts
            if P' is not visited
                mark P' as visited
                NeighborPts' = regionQuery(P', eps)
              if sizeof(NeighborPts') >= MinPts
              NeighborPts = NeighborPts joined with NeighborPts'
          if P' is not yet member of any cluster
              add P' to cluster C
```

```
regionQuery(P, eps)
        return all points within P's eps-neighborhood (including P)
```

Implementation

Step 1:

First we obtain the data from the input text file from the module fetch_data(filename,eps,minPts) The ε-Neighborhood and minimum number of points to be considered are specified while running the file. We have the examined the results for varying ε and MinPts. By setting at least 3 minPts and ε is selected such that only a small fraction of points should be within this distance of each other.

Step 2:

After fetching data for processing, compute_dbscan function is used to assign label to each data point. While labelling the cluster numbers to the data points, it also considers the neighbors that are density reachable and clusters them accordingly.

Step 3:

If the neighboring points are density reachable, then expandCluster module is called and it adds the neighbors of these data points to the same cluster. This process is being processed recursively by calling the regionQuery function which provides as output the neighbors of a data point that are at ε distance from the point and satisfy the minPts condition.

Step 4:

Finally, the entire data set is labelled by the cluster number it corresponds to and is represented by -1 if it's noise. This labelled list is then used to compute the external index and visualize the clustering result. We have used PCA representation of data points for visualization.

External Libraries used:

- **numpy as np**: for numpy array/list related operations
- **distance from scipy.spatial** : to compute the Euclidean distance
- **PCA from from sklearn.decomposition** : to reduce dimensions of data points and visualize them while plotting
- **matplotlib.pyplot as plt :** For plotting the clustering results and PCA reduced data points.

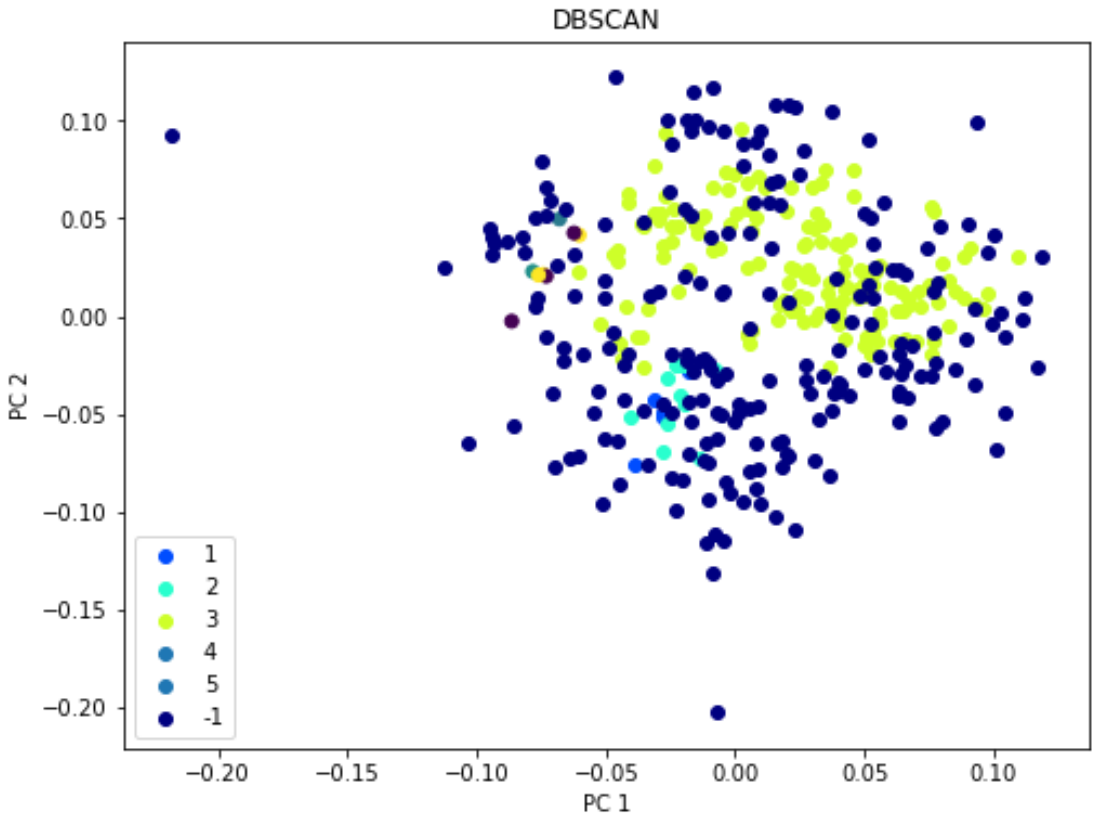
3.3 VISUALIZATION

File Name: cho.txt

Here, the colors correspond to the Cluster ID assigned by the algorithm.

After experimenting with varied values of ε and MinPts, we have clustered the data by fixing the value of ε 1.03 and the value of Minpts to 3.

Resulting clusters are as shown:



cho.txt

As we can see, the algorithm has efficiently identified some of the data points as noise by clustering them with a cluster ID as -1 and the remaining data points are represented by their cluster number. The dimensions of the data points have been reduced and we have used PCA for effective visualization.

The value of jaccard co-efficient and rand index is :

jaccard_coefficient: 0.20310073844038545
rand_index: 0.5475943745373797

Different combinations of ε and MinPts provide a range for external index. The data is clustered into noise or with cluster IDs accordingly depending upon these input parameters. The Jaccard co efficient varies from 0.20 to 0.23

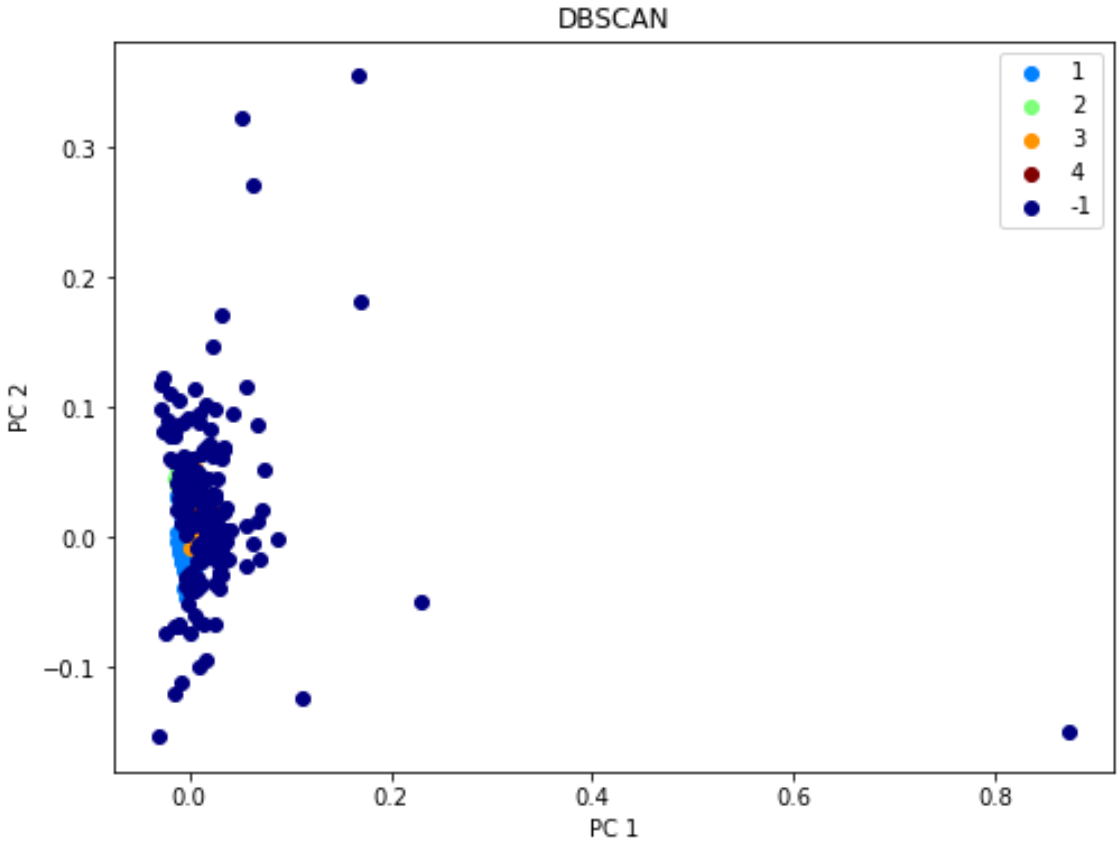
Similarly, the rand index is noted to vary from 0.23 to 0.55

File Name: iyer.txt

Here, the colors correspond to the Cluster ID assigned by the algorithm.

After experimenting with varied values of ε and MinPts, we have clustered the data by fixing the value of ε 0.8 and the value of Minpts to 3.

Resulting clusters are as shown:



iyer.txt

With the specified values of ε and MinPts, after the implementation of DBCSN algorithm, the data set is clustered.

The value of jaccard co-efficient and rand index for the current input parameters is:

jaccard_coefficient: 0.28512625362917593
rand_index: 0.6464996326451052

The algorithm effectively clusters all the data points and identifies the non density reachable data points as noise, which is represented by -1.

Different combinations of ε and MinPts provide a range for external index. The data is clustered into noise or with cluster IDs accordingly depending upon these input parameters.

The Jaccard co efficient varies from 0.17 to 0.28

Similarly, the rand index is noted to vary from 0.31 to 0.59

It can thus be observed that the similarity measure is highly affected on the neighborhood distance and minimum number of points in neighborhood. The data points that are more closely placed are clustered together and thus density along with the input parameter setting accounts as an important measure for detecting similarities using DBSCAN algorithm

3.4 RESULT EVALUATION

- DBSCAN algorithm assigns labels to each point based on the number of neighbours (n) that are reachable within the epsilon distance.
- Thus user can control density of the clusters by determining the number of minimum reachable neighbour points for a point to belong to that cluster.
- The algorithm has time complexity of $O(n^2)$, since it visits all the points to label them.
- Border points could reachable from two core points, in that case it would be labelled same as the core point considered last.
- This algorithm is capable of identifying noise/outliers in the data.
- If n is too large=> small clusters (of size less than n) are likely to be labeled as noise
- If n is too small=> Even a small number of closely spaced that are noise or outliers will be incorrectly labeled as clusters

Advantages:

- DBSCAN does not require one to specify the number of clusters in the data a priori, as opposed to k-means.
- It can find arbitrarily shaped clusters. It can even find a cluster completely surrounded by (but not connected to) a different cluster.
- It has a notion of noise, and is robust to outliers.

Disadvantages:

- It is not entirely deterministic: border points that are reachable from more than one cluster can be part of either cluster, depending on the order the data is processed.
- The quality depends on the distance measure used in the function regionQuery, used to identify the neighbours.
- If the data and scale are not well understood, choosing a meaningful distance threshold ε can be difficult.

Applications:

- Anomaly detection in data.
- Recommendation system for products
- Identifying objects in satellite images.

4. MAP REDUCE K-MEANS CLUSTERING

4.1 DESCRIPTION

This is implementation of standard Kmeans algorithm using a distributed processing system. For this implementation a Hadoop MapReduce model has been used. The task of deriving the clusters and calculating the new cluster centroids are wisely divided into mapper and reducer functions. This work distribution along with the help of parallel processing provided by Hadoop infrastructure is an efficient way to handle large inputs of data file while computing Kmeans algorithm.

4.2 IMPLENTATION

Algorithm

- Data records are split and are assigned to independent mappers objects.
- Map functions identify the nearest centroid (which is predefined or calculated) for a specific input record and emit the centroid id along with the record to the combiner.
- Combiners combines all the records mapped to one centroid value and passes it to the Reducer. Thus, combiner creates a cluster of records against every centroid passed by the mappers.
- Reducer is used to calculate new values of centroid from the passed cluster of records.
- This process continues until the new values of centroids, calculated by reducer match with the values of previous centroids.

Implementation

To provide parallel processing for implementing Kmeans on given dataset a Hadoop infrastructure and its Mapreduce model is used. Below given is the implementation details using Hadoop MapReduce.

Initiation:

- A predefined set of centroids need to be specified in centroids.txt file to initiate the clustering and centroid calculation iterative process.
- An iterator is used to maintain the number of iterations. This iterator is also used to uniquely identify output from individual iterations of Mapreduce executions, as Hadoop does not support overriding of any directory or file in hdfs.
- Mapreduce Job is initiated by specifying the paths of data file and the centroid file.
- From the second iteration output from the previous iteration acts provides centroid values for current iteration.

Mapper:

- In Hadoop every file is assigned to a single mapper unless it surpasses the size limit. Thus, a large file is split by Hadoop and can be assigned to multiple mappers.
- The mapper calls map function for every single record present in the data set.
- The centroids are accessed from the file stored on hdfs. The path to the file is passed to the mapper using configuration variables.
- A setup function has been added to the mapper which fetches the centroids and provides a global access to the individual map functions.

- Using the centroid values, Euclidean distance is calculated for every data record and the centroid id of the set of centroids with least Euclidean distance is emitted to the combiner along with the complete record (which is in form of text). [IntWritable -> Text]

Combiner:

- A custom combiner was implemented but has not been added in final submission as it does not help execution for small datasets.
- The inbuilt combiners combine the values emitted by mapper with help of their corresponding centroid id.
- A combiner forms a cluster of data records against a given centroid value and passes it to a Reducer. [IntWritable -> Iterable<Text>]

Reducer:

- A reducer is provided with a cluster of data records containing all their attributes and the centroid id.
- Reducers calculate the new value of the centroid for that specific cluster of data records by taking mean of every attribute and then print it out to the output file along with the centroid id.

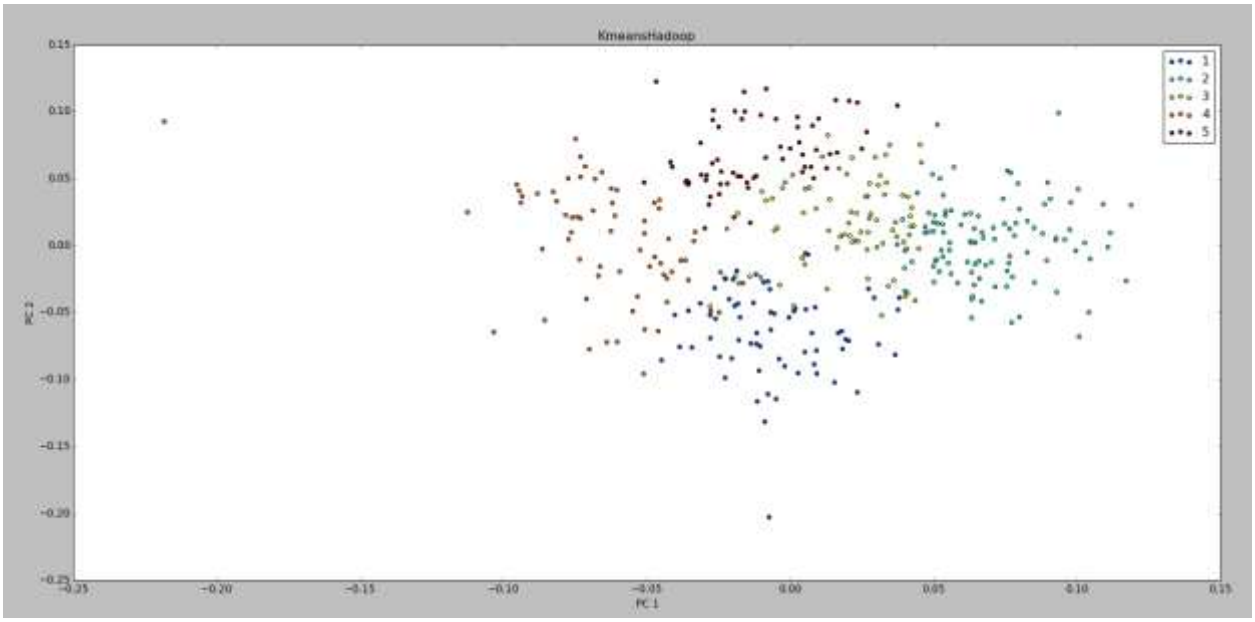
Termination:

- Output from the reducer is checked with the previous set of centroids.
- If both the sets of centroids match, then the algorithm terminates as the final set of clusters and centroids have been identified. An additional iteration is added to the mapreduce process to print out the final clusters in the format similar to the input data file.
- If the values do not match, then the process is carried all over again until a matching value of centroids is generated by the algorithm.
- For the new iteration output from the previous mapreduce execution is used to identify the centroids and then calculate their clusters. This is done using a configuration variable with is passed along with the mapreduce job.

4.3 VISUALIZATION

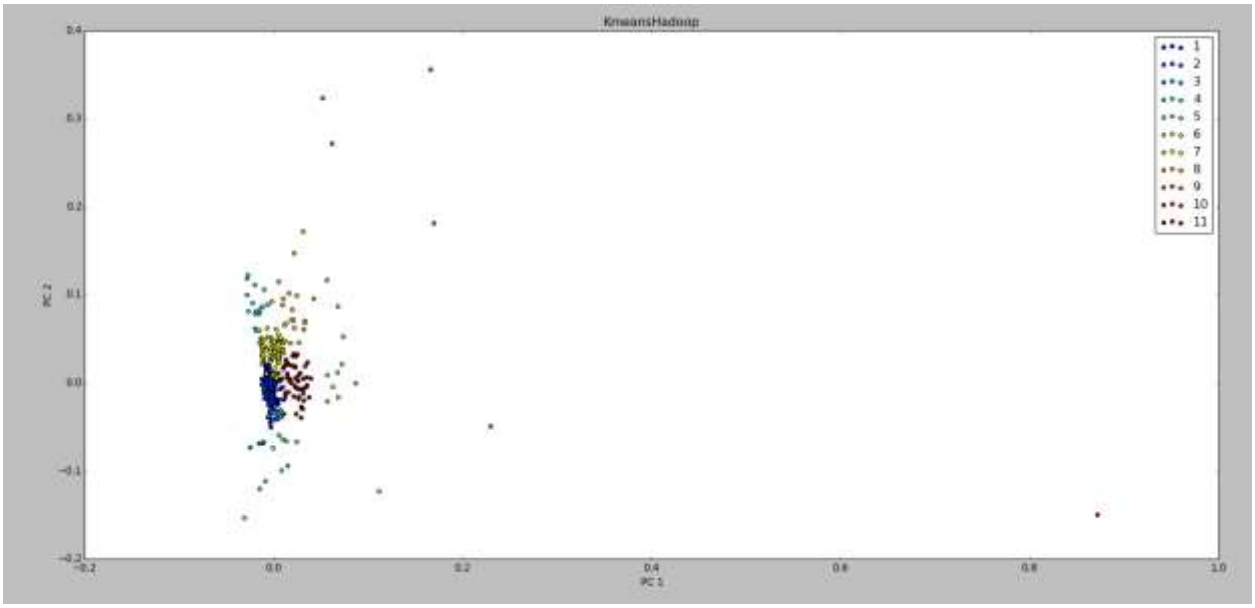
Below is the final cluster plots derived from the Hadoop MapReduce execution of Kmeans performed on given datasets with a random sets of initial centroids.

Input File: cho.txt Number of Clusters[K]: 5



The Jacard coefficient and Rand Index values for this data set varied when the initial centroids were altered. Jacard coefficient varied between values 0.3 to 0.4 and Rand Index varied between 0.7 to 0.8. The values obtained were similar to the non-parallel Kmeans execution.

Input File: iyer.txt Number of Clusters[K]: 11



The Jacard coefficient and Rand Index values for this data set varied when the initial centroids were altered. Jacard coefficient varied between values 0.3 to 0.33 and Rand Index varied between 0.73 to 0.8. The values obtained were similar to the non-parallel Kmeans execution.

4.4 RESULT EVALUATION

Comparison with non-parallel Kmeans

- The map reduce kmeans provides similar results to a non-parallel implementation of kmeans. The final cluster outputs of both the implementations were similar.
- However, mapreduce implementation of kmeans was observed to run slowly than the non-parallel kmeans. This is because of the overhead cost of using hadoop and setting up node structure and job trackers.
- The mapreduce implementation of kmeans would likely outperform the non-parallel kmeans for large datasets, but the datasets provided for this assignment were relatively small.
- Parallel calculation of centroids and clusters helps in reducing the time spent on sequential calculations, as these executions are independent of each other.

Methods to Improve running time

1. **Setup function Implementation:**

- As a part of the implementation we had to read from the centroid file to get the current values of centroids.
- This was a repetitive process for every map function adding to the execution time.
- To avoid reprocessing of already processed data we added a setup function and created a global variable to store centroids in Mapper class.
- The setup function on initiation of a Mapper class retrieves the centroids from the specific file and then assigns it to the global variable which can be accessed by all the map functions initiated by the mapper.

2. **Combiner Implementation:**

- To improve the processing time of the Mapreduce implementation of Kmeans we tried adding a custom combiner class.
- But we observed that a custom combiner on a small dataset is more or equal time consuming than an implementation with it.
- As the datasets were small, by adding a custom combiner, it performed the same tasks as the built-in Hadoop combiner.
- On a large dataset where there are multiple mappers with each processing large amount of data, a custom combiner combining the data records would help in reducing the effort of built-in combiner to combine the data records against a specific centroid.
- Below is the code snippet of the combiner class which was implemented but is not part of the final code submission.
- This combiner class helps in adding up all the attributes of the clustered data records emitted by a single mapper and then passing its average to the reducer.

```
public class KmeansCombiner extends Reducer<IntWritable,Text,IntWritable,Text>{

    @Override
    protected void reduce(IntWritable key, Iterable<Text> values,
                          Context context)
            throws IOException, InterruptedException {
        Iterator<Text> valuesIt = values.iterator();
        ArrayList<ArrayList<Double>> cluster = new ArrayList<ArrayList<Double>>();
        ArrayList<Double> gene = new ArrayList<Double>();
        String[] geneAttr = null;
        while(valuesIt.hasNext()){
            String strGene =  valuesIt.next().toString();
            gene = new ArrayList<Double>();
            geneAttr = strGene.toString().split("\t| ");
            for(int i = 2; i < geneAttr.length; i++) {
                gene.add(Double.parseDouble(geneAttr[i]));
            }
            cluster.add(gene);
        }
        Integer attrLen = cluster.get(0).size();
        ArrayList<Double> centroids = new ArrayList<Double>(Arrays.asList(new Double[attrLen]));
        Double attrSum = null;
        for(int i = 0; i < attrLen; i++) {
            attrSum = 0.0;
            for(ArrayList<Double> ele : cluster) {
                attrSum += ele.get(i);
            }
            centroids.set(i, attrSum/cluster.size());
        }
        context.write(key, new Text(centroids.toString()));
    }
}
```

Advantages:

- A parallel implementation of Kmeans provides all the advantages provided by standard implementation of Kmeans.
- Additionally, parallel Kmeans can be used on large dataset and give time efficient results.
- Parallel Kmeans provides a better time complexity than the non-parallel Kmeans. The time complexity is assumed to be O(log k^2) as compared to O(tkn) in case of non-parallel Kmeans.

Disadvantages:

- A parallel implementation of Kmeans has similar disadvantages as non-parallel Kmeans.
- For large dataset computing the value of K is not feasible.
- When used on a smaller dataset, a parallel Kmeans needs more time for execution as the parallel setup is time expensive.

Packages Used:

- Hadoop-common-2.6.4
- Hadoop-mapreduce-client-core-2.6.4

5. CONCLUSION

- K-means algorithm requires to provide the number of clusters to be formed. For large datasets with no insights the value of k would be difficult to predict.
- On the other hand, hierarchical algorithm does not need a value for number of clusters and can provides useful summary of data on visualising the dendrogram.
- These two algorithms if used together could prove beneficial. First hierarchical technique can be used to gain an insight of the data and then value of k can be decided to form the clusters.
- Both these algorithms cannot identify outliers in the data.
- DBSCAN algorithm can help in identifying noise or outliers in data and also does not require predetermining the number of clusters.
- But DBSCAN is sensitive to the eps and the minimum points parameters and requires expertise to correctly determine these values.
- Map-reduce k-means would run faster as compared to non-parallel k-means for larger datasets, as it would compute centroids in parallel and thus provide faster execution.
- However the dataset provided to us was relatively small and thus parallel k means did not produce faster results.
- Parallel and non-parallel k-means produce same results for the clusters.

6. REFERENCES

- http://www.cs.princeton.edu/courses/archive/spr08/cos424/slides
- http://www.stat.cmu.edu/~ryantibs/datamining/lectures
- https://sites.google.com/site/dataclusteringalgorithms
- https://cs.wmich.edu/alfuqaha/summer14/cs6530/lectures/ClusteringAnalysis.pdf