CSE 601- DataMining and BioInformatics

Project 2- Clustering Algorithms

Uttara Asthana:

Contents

I.Introduction:

In this project we have implemented clustering algorithms on the Gene-expression data set for (cho and Iyer Database). The Data points in our project are high dimensional and we have implemented K-Means, Agglomerative hierarchical clustering algorithm and Db –Scan clustering algorithms on these given data set. We have also implemented K-Means in Map reduce framework on a single node Hadoop- set up. We have also carried out PCA visualization for the results obtained so as to visualize the clusters the algorithms give. Also in order to check the performance of that algorithm for that particular data set we have computed the Jaccard and Rand similarity score. This gives us a measure of the correctness of the clusters formed by the respective algorithms.

II.Data set

We were provided with two datasets cho.txt and iyer.txt to run our clustering algorithms on. Both these datasets contained gene-based data. It consists of gene ID, ground truth values and the various attribute values corresponding to the gene.

Cho.txt:

It consists of 18 columns. The first column corresponds to the Gene ID. The second column corresponds to the ground truth. Ground truth values give a benchmark value to compare the results of the clustering algorithm. In the results, the points may/ may not belong to the same cluster.

Cho.txt has 386 gene id's and 16 attributes corresponding to every gene ID. Each gene ID is considered as a point in the clustering algorithm.

Iyer.txt:

Iyer.txt also contains gene data. It has 14 columns. The first and second columns are Gene ID and ground truth respectively. Iyer.txt has 517 gene ID's and 12 attribute values. Each point is in a high dimensional space and is thought of as a point in our clustering algorithms.

III.Tools and Technologies Used:

For Section I: Implementation of Linear K Means, Agglomerative Hierarchical single link clustering and Db-scan algorithm we have used python canopy to implement our python scripts. We can use canopy version 2.7 onwards for implementing the scripts.

For Section II: Implementation of K-Means in Hadoop Map Reduce on single-node Hadoop set up.

For PCA visualization for both the sections we have used the scikit-learn (open source software for machine learning) for performing decomposition (dimensionality reduction, reduced to 2D). Besides we have also used matplotlib which is a plotting library for python programming language to make scatter plots.

SECTION I:

I. ALGORITHMIC DESCRIPTION AND RESULT VISUALIZATION USING PCA:

A. K Means Clustering Method:

K-Means clustering method basically aims to partition N observations into K clusters in which each observation belongs to the cluster with the nearest mean. The similarity is decided on the basis of the distance between two points

In our implementation we have used the Euclidian distance as the similarity measure. Lesser the distance between two Data points more the similar they are and thus higher the probability of them falling in the same cluster.

Following is the explanation of the code:

The k means implementation consist of mainly 2 steps:

1. ***Assign the data points to their respective cluster (based on Euclidian distance in our case).***
2. ***Compute the new set of centroids for these newly formed clusters.***

Repeat the above steps till the values for newly computed centroids is same as the older one.

The pseudo code is as follows,

K← Number of clusters

Old_centroids← Initial set of centroids (based on your assumption or choose randomly)

Set Condition<-true

While Condition is True

For each point P in given Dataset

1.) Cl_id<-getwhichcluster(point P, old_centroids) ---- Get the cluster id with which the data point P should be associated with. , We start annotating the cl_id from 1…..to…k.

Old_centroids contain is a list containing the centroid for Cluster1, Cluster 2…..Cluster K respectively.

2.) Put the point P in the Cl_id associated with it

New_centroids←getnewcentroids(Clusters 1…k) --- Compute the new set of centroids for the clusters formed in the earlier iteration.

If New_centroids=Old_centroids

STOP processing further

Else

New_centroids=Old_centroid

Max_Iterations=Max_Iterations+1

PCA Visualization for K Means:

Following is the result obtained for Cho.txt:

Initial set of Centroids: Tuple number 2, 3, 4, 5, 6 from cho.txt file

We have populated our database from the value 0 and hence in order to retrieve any tuple N from the CSV file the user is expected to enter the number N-1. However we have followed the same naming convention and the labels of the tuples start from value 1 in order to avoid confusion.

Say for example the 50th tuple in the cho.csv file would be saved at the 49th position in our database, however it would have a label of 50.

[50 0.21 -0.34……..] all 16 attributes ←-found at index 49 in our database , 50 is the tuple label the same was given in the original data set.

```
In [31]: %run "C:\Users\Twinkle\Desktop\Oshin\k means with PCA try.py"

Which file you want to read: cho (Enter 1)  iyer(Enter 2) new_dataset_1(Enter 3)1

Enter the value of k (number of clusters):5
Tuple numbers should be selected in the range 0-385 (corresponds to 1-386 in csv file)

select the tuple number for centroid 1: 1

select the tuple number for centroid 2: 2

select the tuple number for centroid 3: 3

select the tuple number for centroid 4: 4

select the tuple number for centroid 5: 5
---------------------------------Iteration 0
New Centroids computed are:
[[1.0, -0.343, 0.048, 0.998, 0.596, 0.097, -0.102, -0.398, -0.619, -0.545, 0.53, 0.503, 0.252, -0.056, -0.321, -0.519, -0.41], [2.0, -0.399,
-0.586, -0.408, -0.16, -0.033, -0.028, -0.293, -0.454, -0.094, 0.538, 0.497, 0.327, 0.241, 0.165, 0.14, 0.254], [3.0, -0.159, -0.156, -0.512,
-0.329, -0.052, 0.087, 0.291, 0.416, 0.338, -0.14, -0.202, -0.078, 0.029, 0.109, 0.197, 0.135], [4.0, -1.088, -0.332, -0.026, -0.116, 0.07,
0.37, 0.496, 0.672, 0.482, -0.048, -0.22, 0.272, -0.255, 0.132, -0.042, 0.022], [5.0, -1.148, -1.175, -1.112, -0.924, -0.488, 0.082, 0.75,
1.075, 0.971, 0.073, -0.24, -0.138, 0.153, 0.485, 0.833, 0.718]]
---------------------------------Iteration 1



---------------------------------Iteration 7
New Centroids computed are:
[[1.0, -0.369, 0.061, 1.074, 0.576, -0.01, -0.197, -0.482, -0.722, -0.565, 0.688, 0.595, 0.249, -0.043, -0.325, -0.545, -0.404], [2.0, 0.104,
-0.271, -0.628, -0.652, -0.577, -0.427, -0.503, -0.382, 0.176, 0.863, 0.546, 0.262, 0.026, 0.06, 0.107, 0.441], [3.0, 0.006, 0.103, -0.264,
-0.296, -0.137, -0.051, 0.242, 0.44, 0.358, -0.21, -0.265, -0.254, -0.131, 0.028, 0.176, 0.187], [4.0, -0.688, -0.582, -0.113, 0.389, 0.678,
0.554, 0.238, -0.031, -0.254, -0.272, 0.047, 0.289, 0.3, 0.126, 0.067, -0.177], [5.0, -1.042, -1.075, -1.163, -0.791, -0.308, 0.221, 0.79,
1.033, 0.909, -0.037, -0.326, -0.061, 0.144, 0.454, 0.667, 0.552]]


Time taken for k means completition:  2.40642310632

['cluster1: 145', 'cluster2: 63', 'cluster3: 57', 'cluster4: 78', 'cluster5: 43']
Jaccard Similarity: 0.409965979588
Rand similarity : 0.802115493033
4.43485012224 seconds taken for execution of complete program
```

The program converged after 7 iterations for the given set of centroids. And the value entered for k was 5, five clusters were formed each having 145, 63, 57, 78 and 43 data points respectively. The Rand index Similarity and Jaccard co-efficient comes out to be 0.8021 and 0.4099 respectively.

SCATTER PLOT OBTAINED:

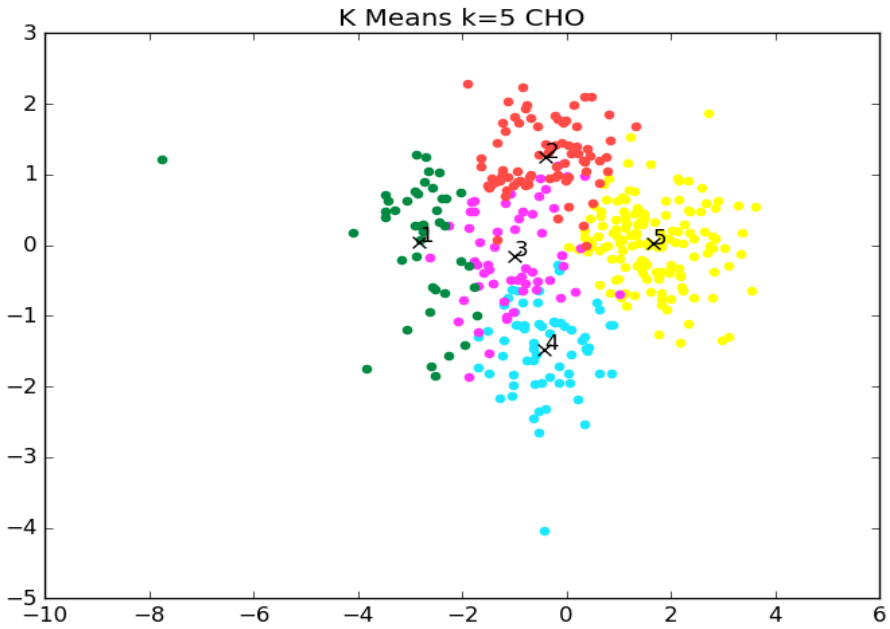


FIG: 1

In the above scatter plot we have assigned different color to different clusters.

Following is the result obtained in iyer.txt

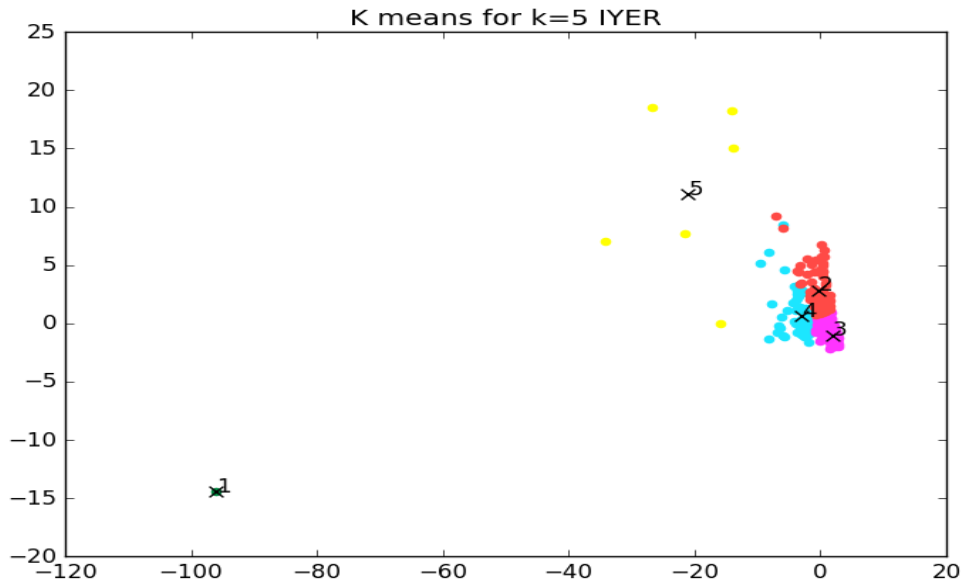
```
In [33]: %run "C:\Users\Twinkle\Desktop\Oshin\k means with PCA try.py"

Which file you want to read: cho (Enter 1)  iyer(Enter 2) new_dataset_1(Enter 3)2

Enter the value of k (number of clusters):5
Tuple numbers should be selected in the range 0-516 (corresponds to 1-517 in csv file)

select the tuple number for centroid 1: 1

select the tuple number for centroid 2: 2

select the tuple number for centroid 3: 3

select the tuple number for centroid 4: 4

select the tuple number for centroid 5: 5
----------------------------------Iteration 0
New Centroids computed are:
[[1.0, 1.0, 1.058, 1.032, 1.141, 1.23, 1.396, 1.824, 2.25, 2.25, 2.524, 2.756, 3.095], [2.0, 1.0, 0.844, 0.851, 0.847, 0.922, 1.465, 2.024,
2.343, 1.625, 1.106, 1.114, 0.956], [3.0, 1.0, 0.903, 0.833, 0.839, 0.833, 0.643, 0.512, 0.445, 0.446, 0.711, 0.705, 0.767], [4.0, 1.0, 0.957,
1.347, 1.23, 0.886, 0.96, 0.8, 0.724, 0.744, 0.784, 0.931, 0.938], [5.0, 1.0, 1.202, 1.862, 2.478, 2.148, 3.451, 3.631, 2.828, 1.888, 1.298,
1.21, 1.151]]
----------------------------------Iteration 1
```

```
Python                                                                    C:\Users\Twinkle
1.01, 1.05, 1.061, 1.13, 1.353, 1.581, 1.9, 2.108, 2.544, 3.035, 3.646], [5.0, 1.0, 2.67, 2.41, 3.75, 6.18, 38.8, 86.92, 25.58, 14.81, 6.73,
2.72, 2.42]]
---------------------------------Iteration 18
New Centroids computed are:
[[1.0, 1.0, 1.123, 2.197, 3.778, 3.393, 10.598, 13.695, 14.397, 8.973, 8.405, 7.622, 6.563], [2.0, 1.0, 1.163, 1.704, 2.417, 2.198, 3.211,
3.306, 3.376, 2.375, 1.666, 1.55, 1.423], [3.0, 1.0, 0.958, 0.976, 0.986, 0.926, 0.769, 0.644, 0.6, 0.604, 0.785, 0.789, 0.834], [4.0, 1.0,
1.011, 1.05, 1.063, 1.13, 1.338, 1.555, 1.87, 2.084, 2.548, 3.055, 3.686], [5.0, 1.0, 2.67, 2.41, 3.75, 6.18, 38.8, 86.92, 25.58, 14.81, 6.73,
2.72, 2.42]]
---------------------------------Iteration 19
New Centroids computed are:
[[1.0, 1.0, 1.123, 2.197, 3.778, 3.393, 10.598, 13.695, 14.397, 8.973, 8.405, 7.622, 6.563], [2.0, 1.0, 1.163, 1.704, 2.417, 2.198, 3.211,
3.306, 3.376, 2.375, 1.666, 1.55, 1.423], [3.0, 1.0, 0.958, 0.976, 0.986, 0.926, 0.769, 0.644, 0.6, 0.604, 0.785, 0.789, 0.834], [4.0, 1.0,
1.011, 1.05, 1.063, 1.13, 1.338, 1.555, 1.87, 2.084, 2.548, 3.055, 3.686], [5.0, 1.0, 2.67, 2.41, 3.75, 6.18, 38.8, 86.92, 25.58, 14.81, 6.73,
2.72, 2.42]]


Time taken for k means completition:  1.13413239056

['cluster1: 6', 'cluster2: 86', 'cluster3: 340', 'cluster4: 84', 'cluster5: 1']
Jaccard Similarity: 0.275597225217
Rand similarity : 0.635488179461
1.54945069983 seconds taken for execution of complete program

In [13]:
```

The program converged after 19 iterations for the given set of centroids. And the value entered for k was 5, five clusters were formed each having 6, 86, 340, 84 and 1 data points respectively. The Rand index Similarity and Jaccard co-efficient comes out to be 0.8021 and 0.4099 respectively. Note that for jaccard value improves for k=10.

```
['cluster1: 17', 'cluster2: 5', 'cluster3: 84', 'cluster4: 69', 'cluster5: 13', 'cluster6: 22', 'cluster7: 1', 'cluster8: 44', 'cluster9:
230', 'cluster10: 32']
Jaccard Similarity: 0.302008820365
Rand similarity : 0.77913793684
```

SCATTER PLOT OBTAINED:



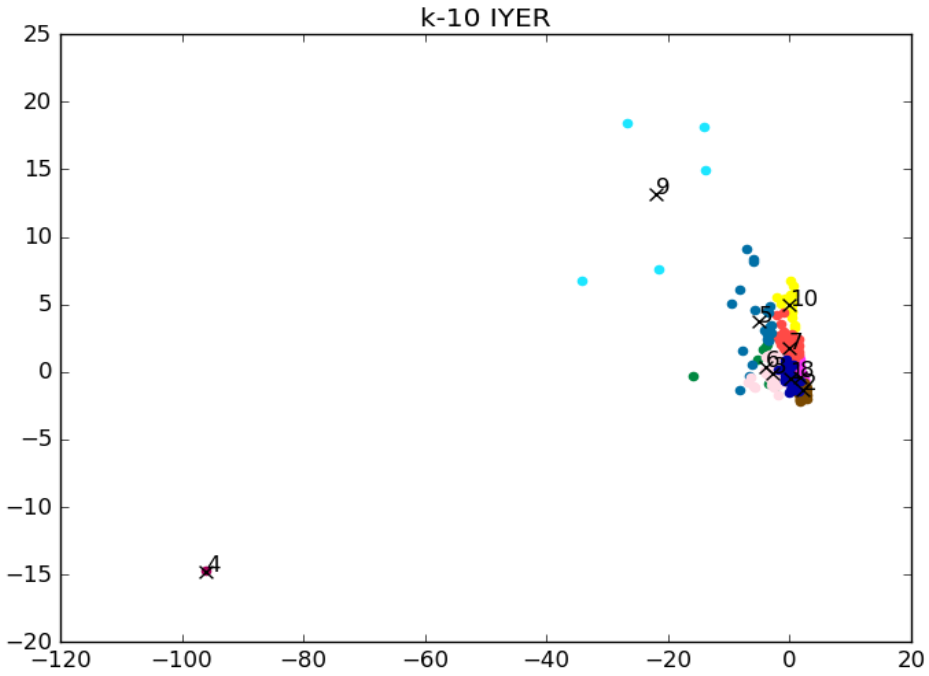


FIG: 2

Time Complexity: 0(tkn) where

n← number of objects

k← number of clusters

t←number of iterations

Hence the complexity of our K means implementation algorithm for the given set of parameters for cho.csv file is O(7*5*386)

Hence the complexity of our K means implementation algorithm for the given set of parameters for iyer.csv file is O(19*5*386)

B. <u>Agglomerative Hierarchical clustering (single link):</u>

Hierarchical clustering (also called hierarchical cluster analysis or HCA) is a method of cluster analysis which seeks to build a hierarchy of clusters. Agglomerative is an bottom up approach where each observation starts with its own cluster.

Following is the pseudo code of the algorithm implementation,

function find_pairwise_distance(input dataset)

```
    pairwise_distance_matrix=compute pairwise distance
    return pairwise_distance_matrix

function find_similar_pair(input matrix)
    minimum_value=compute minimum (input matrix)
    row_index, column_index=input matrix at minimum_value
    return pair(row_index, column_index)

function mergecluster(pairwise_distance_matrix, pair x, y)
    for item=0 and item <size of pairwise_distance_matrix
        update pairwise_distance_matrix= insert minimum(pairwise_distance_matrix[x][i] or
pairwise_distance_matrix[y][i])
    delete element at column_index y in pairwise_distance_matrix
    delete element at column_index x in pairwise_distance_matrix
    delete element at row_index y in pairwise_distance_matrix
    delete element at row_index x in pairwise_distance_matrix

    return updated_pairwise_distance_matrix

N=size of dataset-1
for iteration until N
  find_pairwise_distance_output=find_pairwise_distance(Input)
  pair x,y =find_similar_pair(find_pairwise_distance_output)
  mergecluster_output = mergecluster(pairwise_distance_matrix, pair x,y)
  Output obtained for cho.csv
```

Here the parameter set (where should the algorithm terminate) is 5 i.e. the algorithm terminates when in all only 5 clusters are left after merging the rest.

```
Clustering:
Cluster 1: Size: 1
Cluster 2: Size: 1
Cluster 3: Size: 1
Cluster 4: Size: 1
Cluster 5: Size: 382

Jaccard coefficient: 0.23
Rand Index: 0.24
Total execution time: 2.96764977294s

In [117]:
```

And the value entered for number of clusters to be formed was 5, five clusters were formed each having 1, 1, 1, 1 and 382 data points respectively. The Rand index Similarity and Jaccard co-efficient comes out to be 0.24 and 0.23 respectively.

SCATTER PLOT OBTAINED: for number of clusters=5

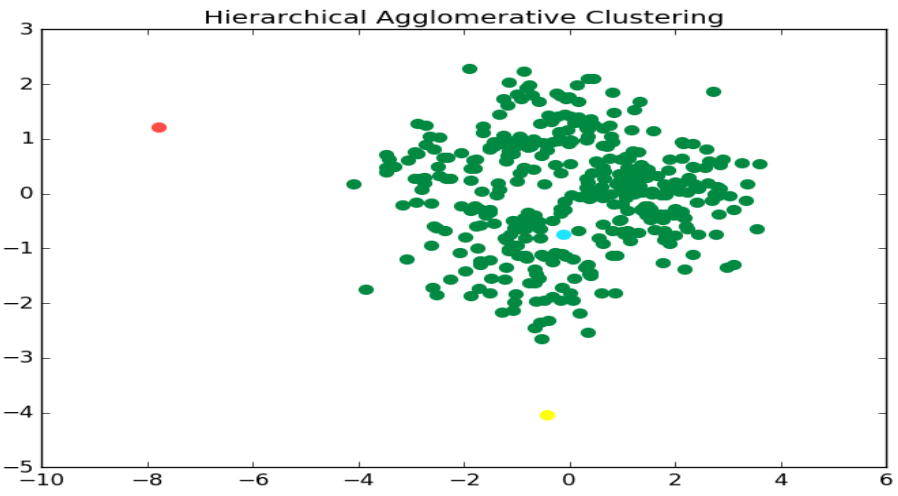


FIG-3

Output obtained for Iyer.csv for number of clusters =10

```
In [13]: %run "C:\Users\UT\project2dm\agglo.py"

Choose Input File: 1. new_data_set  2. cho  3. iyer  Answer:3

Enter number of clusters required:10

Clustering:
Cluster 1: Size: 1
Cluster 2: Size: 1
Cluster 3: Size: 1
Cluster 4: Size: 1
Cluster 5: Size: 1
Cluster 6: Size: 1
Cluster 7: Size: 1
Cluster 8: Size: 1
Cluster 9: Size: 2
Cluster 10: Size: 507

Jaccard coefficient: 0.16
Rand Index: 0.19
Total execution time: 5.11302581089s

In [14]:
```

And the value entered for number of clusters was 10, five clusters were formed each having 1, 1, 1,1,1,1,1,1,2 and 507 data points respectively. The Rand index Similarity and Jaccard co-efficient comes out to be 0.19 and 0.16 respectively.

SCATTER PLOT OBTAINED:

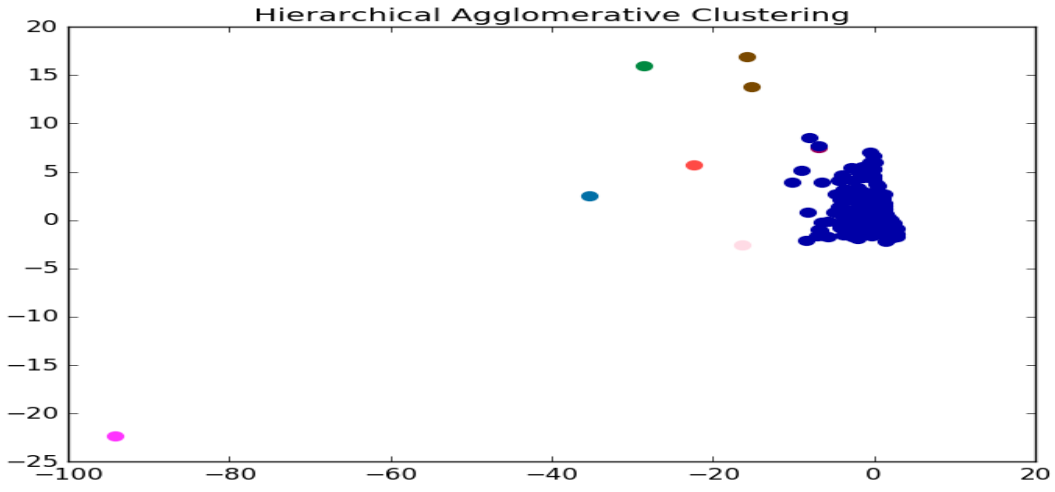


FIG -4

C.*DB-scan Clustering Method*:

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 i.e. whose nearest neighbors are too far away.

For this implementation by the meaning of low density we mean, that particular point is not within the epsilon radius of the object.

We have grouped the points together based on the measure of ε-Neighborhood. - ε-Neighborhood of an object contains at least *MinPts* of objects. MinPts is the minimum number of points that should be present in any cluster.

MinPts and Epsilon are the possible parameters to be set here.

Following is the pseudo code for our implementation,

```
For each unvisited point P in Dataset D
      Mark P as visited
      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)

Following is the result obtained for Cho.txt for epsilon value-1 and MinPts-2:

```
Time taken for completion for DB_Scan:  3.70643905781 seconds.

['cluster1: 2', 'cluster2: 4', 'cluster3: 2', 'cluster4: 11', 'cluster5: 2', 'cluster6: 138', 'cluster7: 3', 'cluster8: 2', 'cluster9: 2',
'cluster10: 2', 'cluster11: 2', 'cluster12: 4', 'cluster13: 2', 'cluster14: 4']
Outliers:
[0, 1, 4, 5, 7, 8, 9, 10, 11, 12, 13, 18, 20, 22, 23, 25, 26, 27, 29, 30, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 46, 49, 50, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 68, 70, 72, 76, 78, 82, 83, 86, 89, 90, 98, 106, 107, 108, 111, 115, 117, 119, 120, 121, 122, 123,
124, 125, 126, 128, 129, 131, 132, 138, 139, 141, 146, 147, 148, 150, 151, 152, 156, 157, 158, 160, 162, 164, 168, 169, 170, 172, 175, 176,
177, 178, 179, 180, 183, 186, 188, 189, 190, 191, 194, 196, 198, 203, 204, 205, 206, 210, 213, 214, 215, 216, 217, 218, 219, 220, 227, 228,
235, 238, 241, 242, 244, 246, 248, 249, 250, 252, 259, 260, 261, 262, 268, 271, 274, 275, 276, 282, 283, 286, 289, 292, 293, 294, 295, 296,
301, 302, 303, 304, 305, 306, 307, 310, 311, 312, 315, 318, 319, 325, 331, 332, 333, 334, 335, 338, 339, 342, 344, 347, 349, 350, 353, 354,
355, 356, 358, 359, 360, 361, 362, 363, 366, 367, 369, 370, 372, 373, 374, 375, 376, 377, 378, 380, 382, 383, 384, 385]
There are 206 outliers.
Jaccard coefficient Similarity: 0.179577783818
Rand index similarity : 0.441049382716
3.87246844876 seconds taken for execution of complete program
```

The program terminated by forming 14 clusters each of size 2, 4, 2, 11, 2, 138, 3, 2, 2,2, 2, 4,2, 4 data points respectively. The jaccard and rand similarity came out to be 0.1795 and 0.4410 respectively.

SCATTER PLOT

The cross signs denote the outliers.

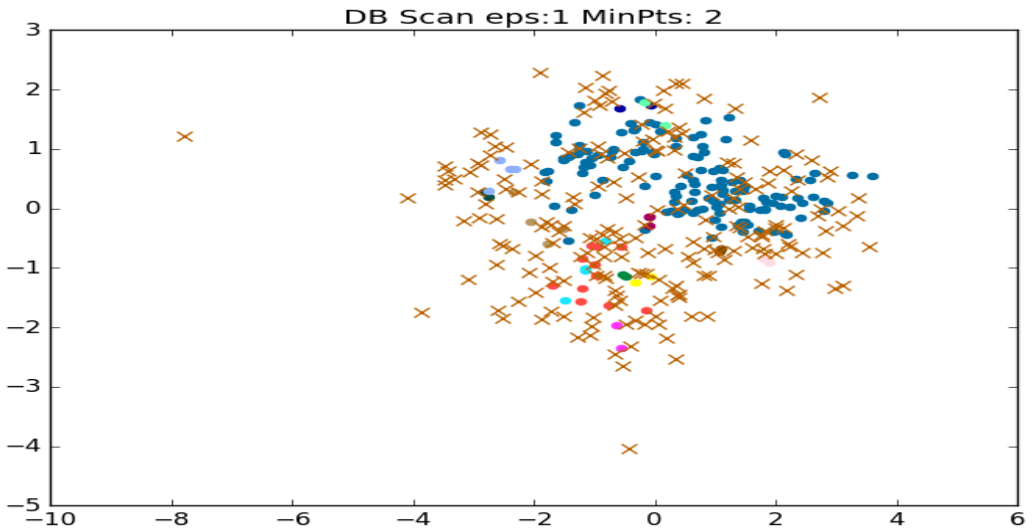


FIG -5

Below is the scatter plot removing the outliers for clarity purposes,

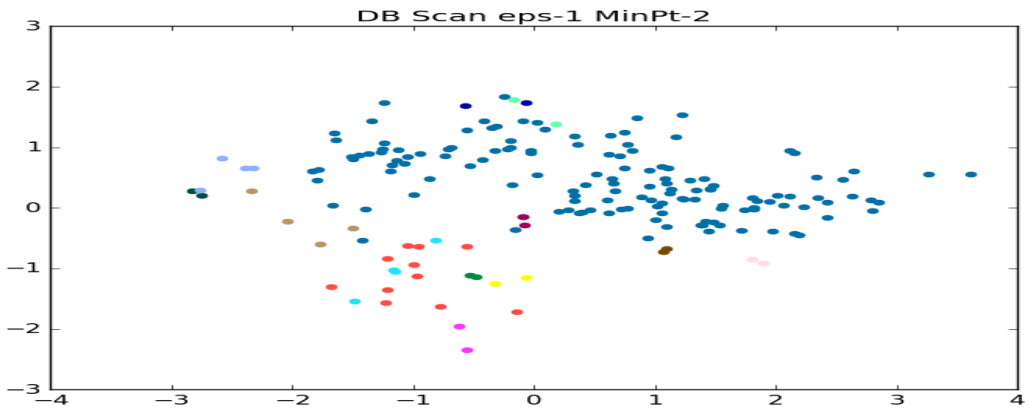


FIG-6

Following is the output for Iyer Data set for MinPt-2 and epsilon-1,

```
Time taken for completion for DB_Scan:  6.99340001599 seconds.

['cluster1: 314', 'cluster2: 2', 'cluster3: 5', 'cluster4: 2', 'cluster5: 2', 'cluster6: 3', 'cluster7: 6', 'cluster8: 2', 'cluster9: 2',
'cluster10: 30', 'cluster11: 2', 'cluster12: 2', 'cluster13: 2', 'cluster14: 2', 'cluster15: 2']
Outliers:
[101, 262, 263, 290, 292, 296, 299, 310, 319, 320, 324, 325, 326, 327, 333, 334, 338, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350,
351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378,
379, 380, 381, 382, 383, 386, 387, 388, 389, 390, 391, 392, 395, 397, 398, 407, 410, 411, 415, 419, 421, 422, 424, 425, 426, 428, 432, 434,
435, 436, 437, 440, 441, 442, 444, 445, 446, 448, 449, 455, 456, 457, 463, 464, 466, 467, 469, 470, 471, 472, 475, 480, 481, 482, 483, 484,
485, 486, 489, 490, 491, 492, 493, 494, 497, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516]
There are 139 outliers.
Jaccard coefficient Similarity: 0.219351373377
Rand index similarity : 0.410038912684
7.36012100897 seconds taken for execution of complete program
```

The program terminated by forming 15 clusters each of size 314, 2, 5, 2, 2, 3, 6, 2, 2,30, 2, 2,2 and 2 data points respectively. The jaccard and rand similarity came out to be 0.219 and 0.410 respectively.

SCATTER PLOT:

Below is the scatter plot where outliers are not marked for clarity purposes,

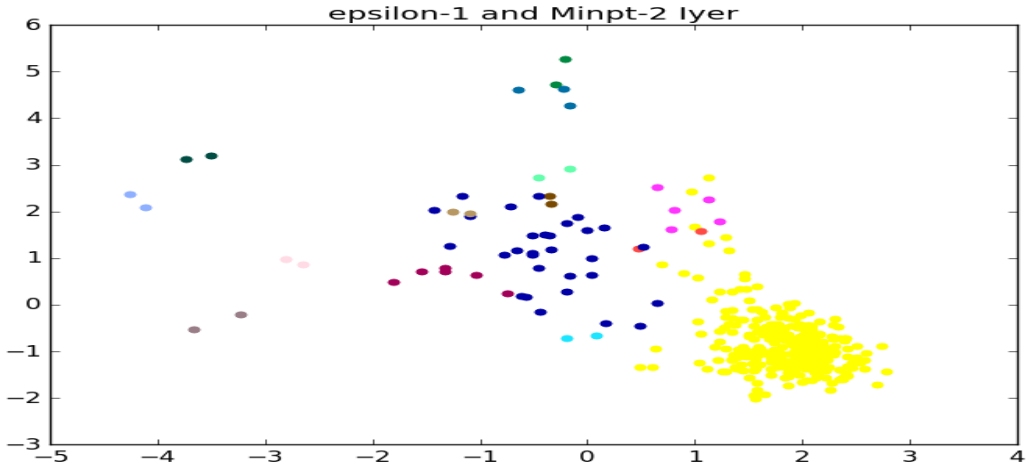


FIG-7

Scatter plot marked with outliers:

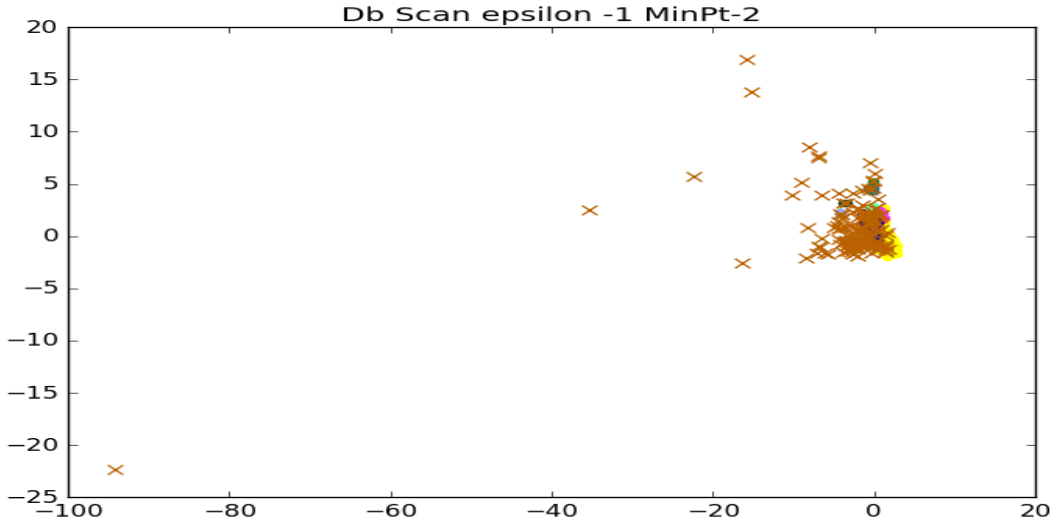


FIG-8

Time Complexity:

O(n^2) where n is the number of data tuples.

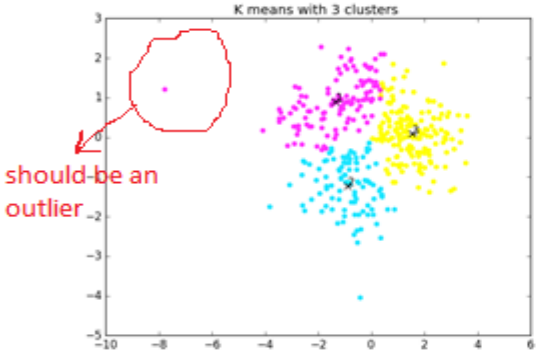
II. ATTEMPT TO REDUCE COMPUTATIONAL COMPLEXITY:

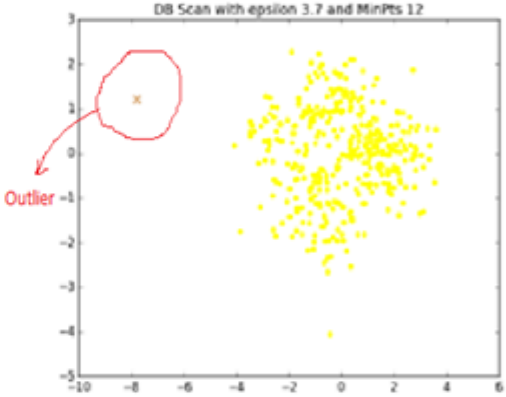
For the implementation of all three of the above algorithms we have used dictionary data structure in python for the purpose of maintaining clusters. As the dictionary structures are

equivalent to hash map in java they facilitate efficient search methods and are computationally less complex. In case of hierarchical clustering we have used the matrices data structure in python that processes data in it faster as compared to that of list of lists.

III. PROS AND CONS OF THE ABOVE IMPLEMENTED ALGORITHMS AND AN ANALYSIS OF THE RESULTS FOR CHO AND IYER DATA SET FOR BOTH

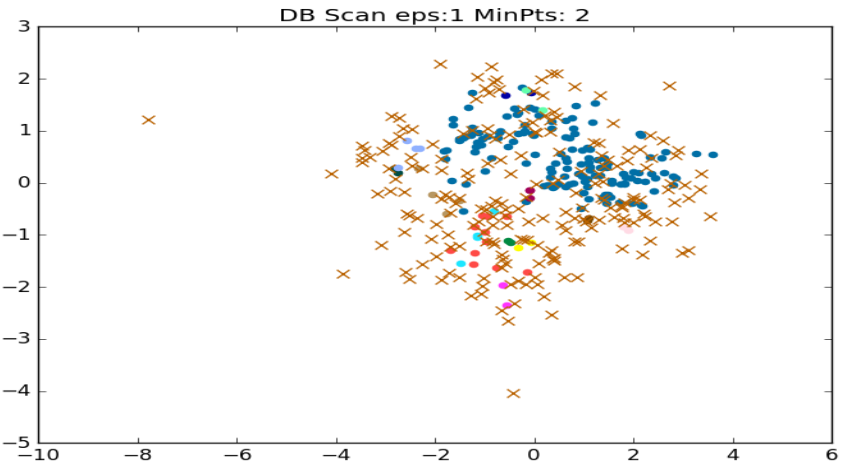
- K means yields good clustering results on uniform data set however, it is not sensitive to outliers, it does not tell when the data should not be clustered unlike DB scan which is comparatively robust towards outlier detection.
- As you can see below the leftmost figure, the K means method considers the point which is marked inside a big red circle as a part of the first cluster, ideally it should be detected as an outlier, as correctly detected by Db Scan. However DB scan cannot sometimes cluster the Data sets well as seen below.



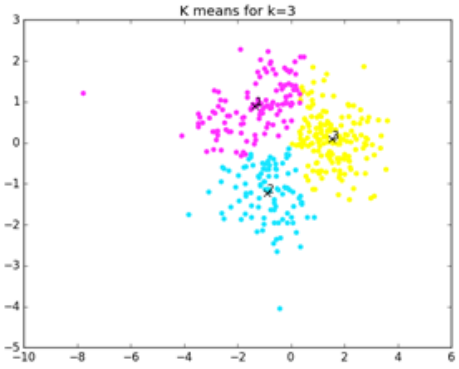


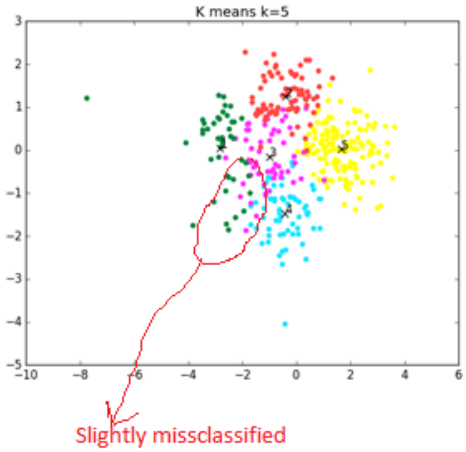
- Deciding the initial set of centroids for K means is crucial. Say for example the Jaccard Similarity score for cho.txt for set of centroids [77,120,220,300,350] here the list consists of the indices to be entered when asked to enter the initial set of centroids by our program. We get the Jaccard value as 0.038 which improves to 0.409 on selecting tuple number 2,3,4,5 and 6 i.e. [2,3,4,5,6] as centroids.
- The quality of DBSCAN depends on the distance measure used in the function regionQuery. The most common distance metric used is Euclidean distance which is also used in our implementation. However for high-dimensional data, the Euclidean distance has its own disadvantages as this method is not that effective for high dimensional data making it difficult to find an appropriate value.
- DBSCAN does not require one to specify the number of clusters in the data a priori, as opposed to k-means. DBSCAN can find arbitrarily shaped clusters. It can even find a cluster completely surrounded by (but not connected to) a different cluster. Due to the MinPts parameter, the so-called single-link effect (different clusters being connected by a thin line of points) is reduced. DBSCAN has a notion of noise. DBSCAN requires just two parameters

and is mostly insensitive to the ordering of the points in the database. (However, points sitting on the edge of two different clusters might swap cluster membership if the ordering of the points is changed. Sometimes this can result to wrong results or many points would get unnecessarily marked as outliers as seen in figure below.



- The choice of K parameter for K means affects the clustering done. For some values of k we get good clustering results however for some values of k we do not get satisfactory results. As you can see for K value of 3 we get good clustering results as compared to what we get for value k=5 for cho.csv Database.

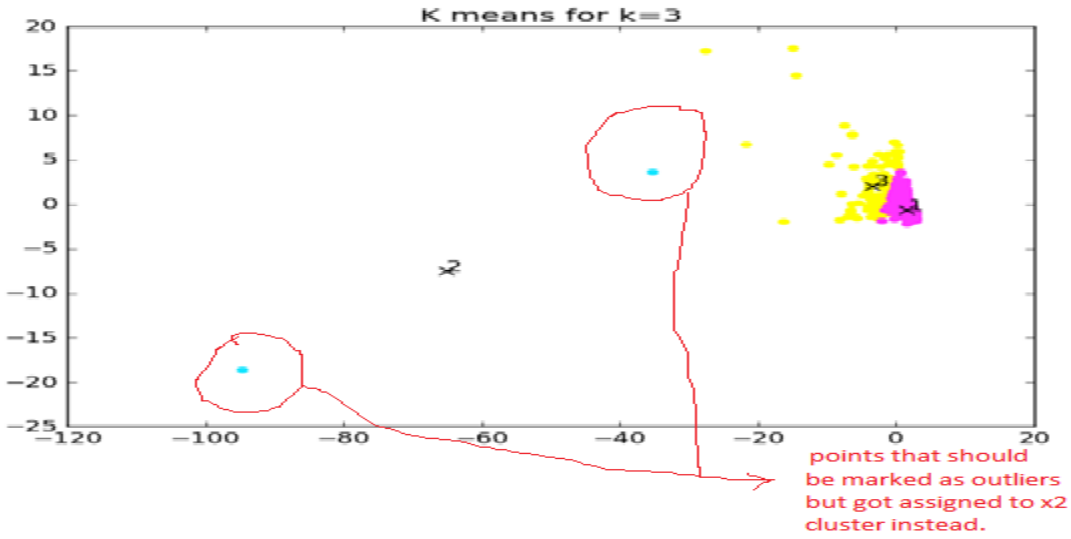




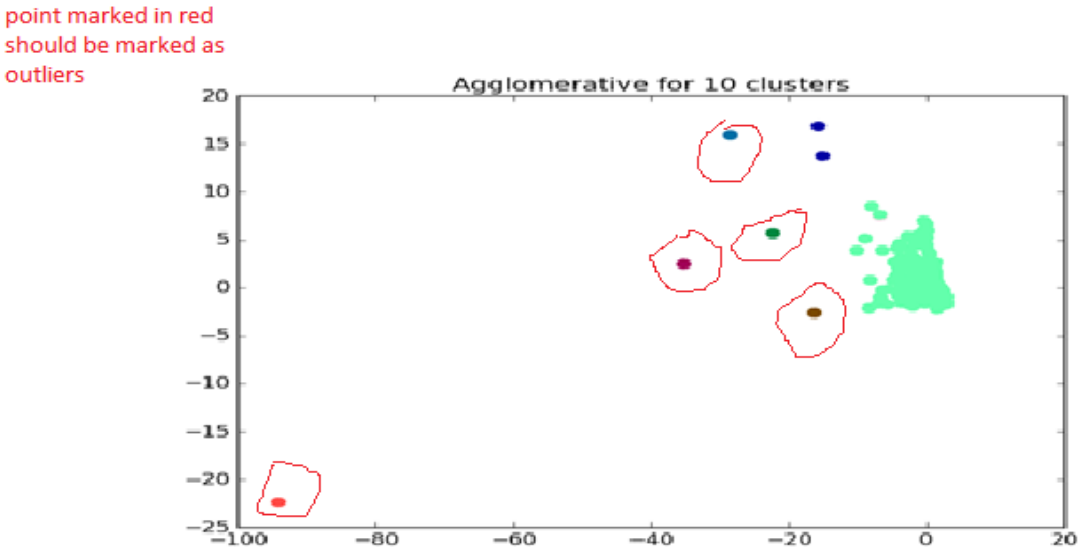
- Very often when you get two samples from the same cluster, it will get stuck in a minimum where this cluster remains split, and two other clusters merged instead. Not always, but very

often. K means is restricted to data that has the notion of centers, requires handling empty clusters but works well for low dimensional data.

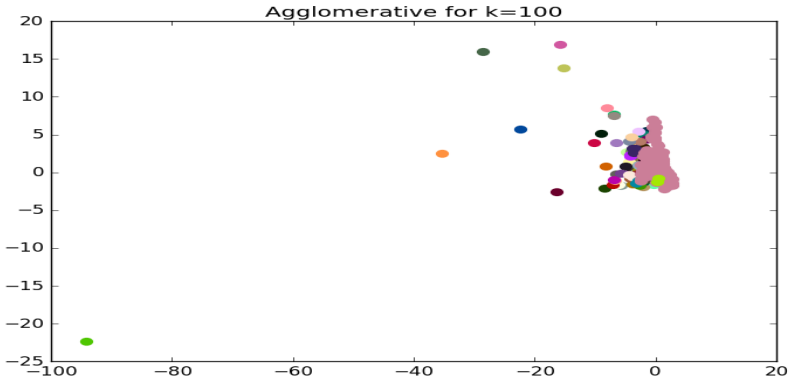
- The following example for Iyer Data set is a classic example of how k means works well for clustering but is ignorant to outliers and clusters them anyway.



- Hierarchical agglomerative clustering on the other hand is also not sensitive to noise and outliers. Handling different sized clusters and convex shapes. It is computationally complex in terms of time and space. Once a decision is made to combine two clusters, it cannot be undone.
- Hierarchical algorithm shows more quality as compared to k-mean algorithm. However it ignores outliers as shown below.



- As a general conclusion, k-mean algorithm is good for large dataset and hierarchical is good for small datasets. Comparison between these algorithms can be implemented on the basis of normalization, by taking normalized and un-normalized data will give different results.
- Hierarchical Clustering can give different partitioning depending on the level-of-resolution to be considered as for k=100 the result is not that apt as obtained for k=10 for the given fig.



- All in all for the given data set cho and Iyer linear K-Means clustering seems to work more efficiently as it is giving the most optimum jaccard values and is most time efficient however the choice of which clustering algorithm to use is strongly driven by the characteristics of the data set given.

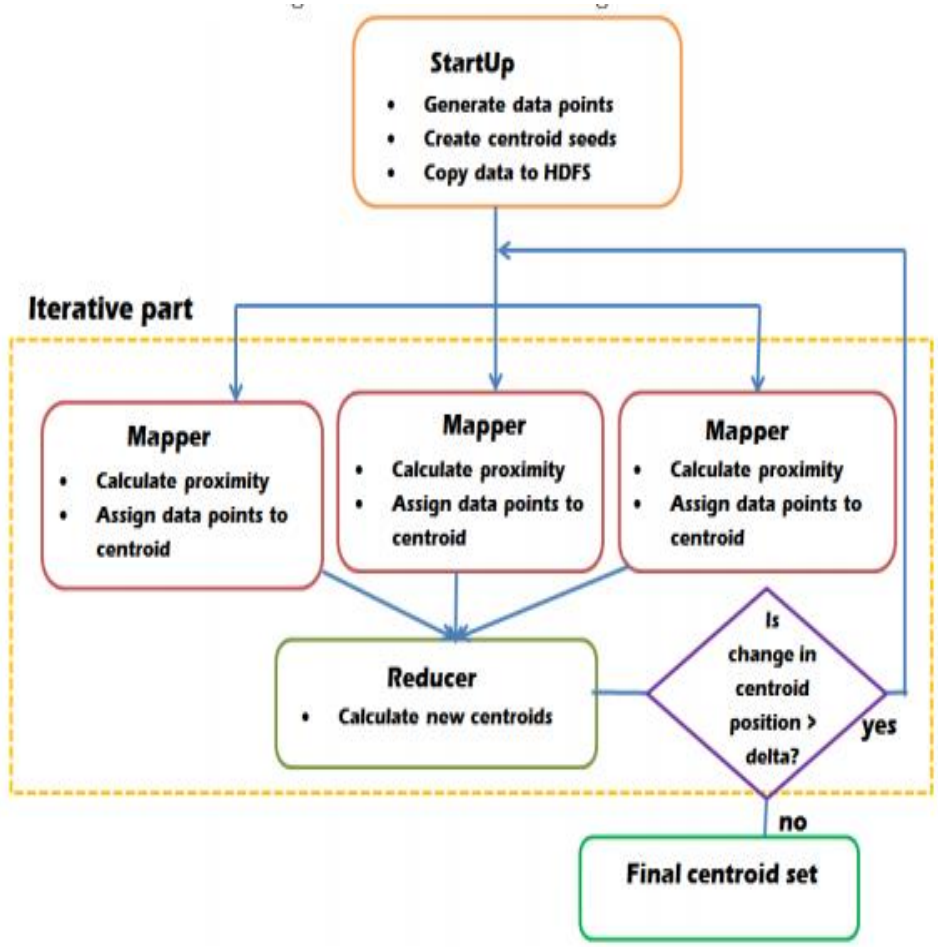
SECTION II: K Means Hadoop Implementation:

We have implemented KMeans using MapReduce in Hadoop. The output of this algorithm is saved to 'centroids_cluster_mapping.txt' text file which is later processed to visualize the clustering results using PCA in Python

Pseudo code:

```
class mapper
  function mapper(Key, Value)
      Hashmap<> centroids = array()
          for each datapoint in clusters
              minimum_distance= compute minimum distance (datapoint, centroids)
                  Emit <closest centroid, datapoint>

class reducer
```

```
    global ArrayList datapoint_list
    function reducer(Key, Value)
        for each tuple in Value
            append tuple in datapoint_list
        sum=0
        key=Centroid id
        ArrayList new_centroid_distances
        for index 2 to columnsize
            for each sublist in datapoint_list
                sum=sum + sublist at (index)
                append sum/(datapoint_list size) to new_centroid_distances

        Emit <Centroid id, new_distances>

function main()
   input path=path to input dataset file
   input k= 'Number of Clusters'
   input arraylist centroids= 'select which centroids'
   for iteration 0 to datasize
      call mapper class
      call combiner class
      call reducer class
      wait for mapper job completion
      compare (List new_centroid_distances, previous List centroids)
          if total match
              break out of for loop  (no new centroids could be calculated in this iteration)
          else
              copy (new_centroid_distances into centroids)
              continue
```



STEPS TO IMPROVE MAP/REDUCE PERFORMANCE OF KMEANS

1. As per our research, the overheads required for MapReduce calculation and the intermediate read and write between the mapper and reducer jobs makes it inadmissible for smaller datasets. Nonetheless, including a **combiner** between Map and Reduce jobs enhances execution by diminishing the measure of intermediate read/write.

This has extensively lessened the excess MapReduce calls that have brought about a critical reduction in the time required for clustering as it has **diminished the read/write operations** to an expansive degree.

2. We have used an arraylist instead of an array to compute and store the distances of the data points to the centroids on each iteration. **Arraylist** facilitates retrieval, addition, deletion and update of data in constant time as opposed to an array.

3. We have used **hashmap** to maintain the initial centroids and the new set of centroids after each iteration. By using hashmap, we have ensured that data is stored and retrieved in a much faster fashion O(1) compared to any other data structure.

4. In the traditional k-means algorithm, before the algorithm converges, the centroids are ascertained ordinarily and the data points are assigned to their closest centroids. Since, complete redistribution of the datapoints happens concurring to the new centroids, this takes O(nkl), where n is the number of data points, k is the number of clusters and l is the number of iterations. To acquire the underlying clusters, our requires O(nk). Here, a few datapoints stay in its cluster while the others move to different clusters relying upon their relative euclidean distance from the new centroid and the old centroid.

This requires O(1) if a datapoint remains in its group, and O(k) something else. As the algorithm converges, the number of datapoints moving far from their cluster diminishes with each iteration. Expecting that a large portion of the datapoints move from their groups, this requires O(nk/2). Thus the aggregate cost of this period of the calculation is O(nk), not O(nkl)

COMPARISON OF LINEAR AND MAP REDUCE IMPLEMENTATION OF K Means

- In this section, we have implemented K Means into the Map reduce framework. This framework seems to be effective for clustering comparatively large data sets. However the iterative nature of K-means makes the map reduce task more extensive or complicated.
- For each iteration of K mean and Map Reduce job is executed which results in high I/O overhead because in each iteration the whole data set is read and written to the slow disks.
- However this framework supports parallel computation and would be highly effective for large data sets and would do the computation in reasonable time which otherwise would be difficult for simple linear implementation.
- Also in the case where data is too big to store on single machine map reduce framework would be preferred.
- In case of data sets like cho and and Iyer (given to us for implementation) we would prefer linear K Means over map reduce framework owing to iterative nature of K-Means algorithm causing repeated times of restarting jobs, data reading and shuffling.
- The nature of K means algorithm is such that it has dependency on the earlier iteration and hence not suited for a parallel processing framework like map reduce specially where data size is not a matter of concern (for the given data sets cho and Iyer).

K Means in Map Reduce framework:

1) For cho.csv Initial set of centroids: [1, 2,3 ,4 ,5]→ i.e. tuple numbers 2,3,4,5 and 6 from the csv file.

```
hadoop@hadoop-VirtualBox:~/hadoop$
hadoop@hadoop-VirtualBox:~/hadoop$
hadoop@hadoop-VirtualBox:~/hadoop$ bin/hadoop jar km1.jar KMeans cho kmmm13
Enter value of K:
5
Select your centroids:
1
You Selected: 1 1       -0.69   -0.96   -1.16   -0.66   -0.55   0.12    -1.07   -1.22   0.82    1.4     0.71    0.68    0.11    -0.04   0.19    0.82
2
You Selected: 2 1       -0.21   0.19    0.86    0.04    -0.35   -0.39   -0.51   -0.2    0.0     0.77    0.41    0.14    -0.45   -1.23   -0.325  0.0
3
You Selected: 3 1       -0.3    -0.56   -0.29   -0.5    -0.27   -0.29   -0.56   -1.04   0.32    0.9     0.45    0.17    0.164   -0.12   -0.16   0.67
45
You Selected: 4 1       0.07    0.26    -0.47   -0.68   -0.63   -0.39   0.07    0.79    0.58    0.31    -0.14   -0.29   -0.103  -0.2    -0.06   0.36
5
You Selected: 5 1       -1.04   0.13    0.51    -0.44   -0.88   -0.32   0.21    0.95    1.07    0.38    0.01    -0.13   -0.78   -0.13   0.092   0.0
16/11/05 01:12:08 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
16/11/05 01:12:11 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
16/11/05 01:12:11 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
16/11/05 01:12:12 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with
.
16/11/05 01:12:13 INFO input.FileInputFormat: Total input paths to process : 1
```

Final output:

```
Number of Executions: 22
Total Execution Time: 36 seconds
hadoop@hadoop-VirtualBox:~/hadoop$
hadoop@hadoop-VirtualBox:~/hadoop$
```

As we can see in the above figure the total execution time is 36 seconds and it takes 22 iterations to converge. On the other hand as shown in linear implementation of K means takes less than 3 seconds (almost 2.04 seconds) and converges in 7 iterations.

External Index Output:

```
Python                                                                                          C:\Users\UT
135.0, 132.0, 131.0, 126.0, 124.0, 123.0, 122.0, 121.0, 120.0, 119.0, 118.0, 117.0, 116.0, 115.0, 114.0, 113.0, 111.0, 109.0,
108.0, 107.0, 106.0, 105.0, 104.0, 103.0, 102.0, 101.0, 100.0, 99.0, 98.0, 97.0, 95.0, 94.0, 93.0, 92.0, 91.0, 89.0, 88.0, 87.0,
86.0, 85.0, 84.0, 83.0, 81.0, 80.0, 79.0, 78.0, 77.0, 76.0, 75.0, 74.0, 73.0, 72.0, 71.0, 70.0, 69.0, 68.0, 64.0, 32.0, 28.0, 2.0]
Cluster 3.0: DataPoints: [214.0, 296.0, 295.0, 82.0, 216.0, 217.0, 361.0, 325.0, 289.0, 288.0, 287.0, 286.0, 285.0, 248.0, 249.0,
218.0, 281.0, 250.0, 279.0, 278.0, 128.0, 276.0, 275.0, 251.0, 254.0, 125.0, 271.0, 270.0, 255.0, 220.0, 323.0, 266.0, 256.0,
264.0, 222.0, 183.0, 96.0, 260.0, 259.0, 258.0, 257.0, 226.0, 331.0, 326.0, 227.0, 205.0, 147.0, 229.0, 145.0, 207.0, 230.0, 327.0,
176.0, 232.0, 233.0, 138.0, 137.0, 234.0, 235.0, 236.0, 237.0, 208.0, 328.0, 209.0, 240.0, 329.0, 242.0, 243.0, 330.0, 317.0,
316.0, 315.0, 314.0, 244.0, 25.0, 203.0, 310.0, 309.0, 308.0, 307.0, 306.0, 305.0, 304.0, 168.0, 210.0, 212.0, 300.0, 204.0, 110.0]
Cluster 4.0: DataPoints: [290.0, 369.0, 332.0, 29.0, 370.0, 26.0, 341.0, 371.0, 374.0, 379.0, 383.0, 10.0, 8.0, 7.0, 5.0, 67.0,
342.0, 4.0, 59.0, 347.0, 130.0, 321.0, 318.0, 312.0, 311.0, 303.0, 302.0, 301.0, 299.0, 298.0, 297.0, 294.0, 293.0, 292.0, 291.0,
282.0, 268.0, 267.0, 263.0, 262.0, 261.0, 351.0, 352.0, 353.0, 354.0, 241.0, 239.0, 322.0, 355.0, 356.0, 358.0, 340.0, 44.0, 43.0,
360.0, 324.0, 362.0, 365.0, 366.0]
Cluster 5.0: DataPoints: [30.0, 337.0, 336.0, 346.0, 345.0, 344.0, 343.0, 385.0, 66.0, 335.0, 368.0, 6.0, 384.0, 357.0, 42.0, 45.0,
12.0, 382.0, 381.0, 364.0, 380.0, 378.0, 377.0, 367.0, 376.0, 375.0, 373.0, 349.0, 56.0, 348.0, 350.0, 211.0, 339.0, 280.0, 283.0,
284.0, 372.0, 334.0, 386.0, 313.0, 319.0, 320.0, 333.0, 338.0, 363.0]

Clustering:
Cluster 1.0: Size: 64
Cluster 2.0: Size: 129
Cluster 3.0: Size: 89
Cluster 4.0: Size: 59
Cluster 5.0: Size: 45

Jaccard coefficient: 0.38
Rand Index: 0.79

In [30]: |
```

New centroids are obtained in ‘Myfile.txt’:

```
File  Edit  View  Text  Document  Navigation  Help
1   1   0.11   -0.27  -0.61  -0.64  -0.57  -0.43  -0.5   -0.38  0.17   0.85   0.54   0.25   0.02   0.06   0.1    0.44
2   1   -0.34  0.09   1.15   0.6    -0.04  -0.21  -0.53  -0.78  -0.6   0.73   0.62   0.26   -0.05  -0.36  -0.59  -0.42
3   1   -0.7   -0.52  0.03   0.41   0.6    0.44   0.17   -0.1   -0.29  -0.14  0.13   0.28   0.26   0.09   0.01   -0.21
4   1   -0.03  0.07   -0.28  -0.26  -0.08  0.03   0.26   0.44   0.33   -0.25  -0.27  -0.23  -0.11  0.02   0.19   0.16
5   1   -0.99  -1.04  -1.18  -0.78  -0.31  0.19   0.78   1.02   0.91   -0.04  -0.34  -0.07  0.14   0.45   0.65   0.54
```

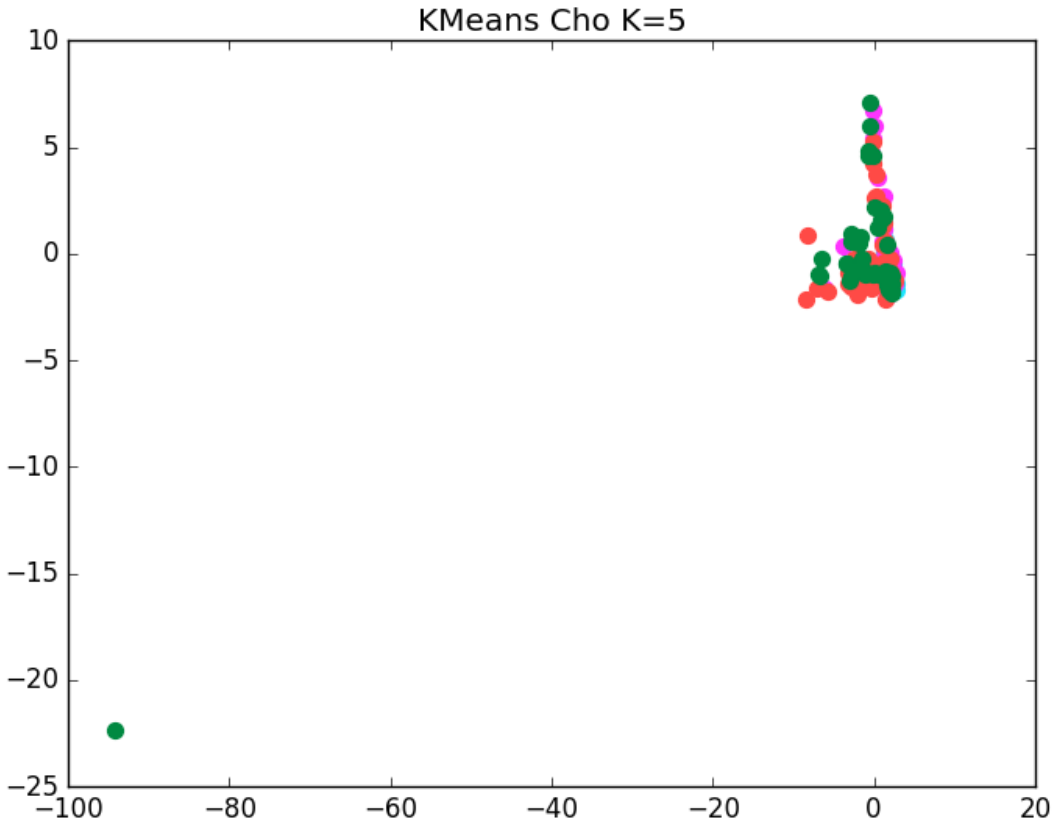
Centroid-Cluster mappings are obtained in 'centroid_cluster_mapping.txt'

centroid_cluster_mapping.txt - Mousepad

File Edit View Text Document Navigation Help

```
1	[238.0, 3.0, 9.0, 11.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0, 20.0, 21.0, 22.0, 23.0, 24.0, 27.0, 31.0, 33.0, 34.0, 35.0, 36.0, 37.0, 38.0, 39.0, 40.0, 41.0,
2	[193.0, 277.0, 274.0, 273.0, 272.0, 269.0, 265.0, 253.0, 252.0, 247.0, 246.0, 245.0, 231.0, 228.0, 225.0, 224.0, 223.0, 219.0, 215.0, 213.0, 206.0, 202.0, 201.0,
3	[214.0, 296.0, 295.0, 82.0, 216.0, 217.0, 361.0, 325.0, 289.0, 288.0, 287.0, 286.0, 285.0, 248.0, 249.0, 218.0, 281.0, 250.0, 279.0, 278.0, 128.0, 276.0, 275.0, 2
4	[290.0, 369.0, 332.0, 29.0, 370.0, 26.0, 341.0, 371.0, 374.0, 379.0, 383.0, 10.0, 8.0, 7.0, 5.0, 67.0, 342.0, 4.0, 59.0, 347.0, 130.0, 321.0, 318.0, 312.0, 311.0,
5	[30.0, 337.0, 336.0, 346.0, 345.0, 344.0, 343.0, 385.0, 66.0, 335.0, 368.0, 6.0, 384.0, 357.0, 42.0, 45.0, 12.0, 382.0, 381.0, 364.0, 380.0, 378.0, 377.0, 367.0,
```

PCA Visualization:



2.] For Iyer.csv Initial set of centroids: [1, 2, 3,4 ,5,7,8,9,10]→ i.e tuple numbers 2,3,4,5 and 6 …from the csv file. (k=10)

Output performance of linear K means implementation

```
----------------------------------Iteration 27
New Centroids computed are:
[[1.0, 1.0, 1.174, 1.303, 1.153, 1.216, 1.076, 1.152, 1.233, 1.538, 2.97, 4.514, 6.696], [2.0, 1.0, 1.018, 2.032, 3.518, 1.776, 10.202,
14.346, 15.17, 10.084, 9.856, 8.974, 7.704], [3.0, 1.0, 1.002, 0.903, 0.928, 0.92, 0.726, 0.628, 0.626, 0.737, 1.105, 1.222, 1.348], [4.0,
1.0, 0.953, 0.977, 1.053, 1.109, 1.346, 1.499, 1.823, 2.033, 2.136, 2.349, 2.608], [5.0, 1.0, 1.8, 3.832, 6.758, 5.118, 4.174, 3.512, 3.366,
1.808, 1.558, 1.46, 1.339], [6.0, 1.0, 0.989, 1.149, 1.228, 1.505, 2.853, 4.39, 5.768, 4.57, 3.212, 3.03, 2.66], [7.0, 1.0, 2.67, 2.41, 3.75,
6.18, 38.8, 86.92, 25.58, 14.81, 6.73, 2.72, 2.42], [8.0, 1.0, 0.964, 1.147, 1.434, 1.569, 3.667, 3.385, 2.955, 1.936, 1.453, 1.361, 1.274],
[9.0, 1.0, 0.935, 0.94, 0.911, 0.875, 0.691, 0.556, 0.475, 0.452, 0.646, 0.617, 0.648], [10.0, 1.0, 1.237, 1.945, 2.492, 2.054, 1.654, 1.549,
1.642, 1.45, 0.972, 0.939, 0.89]]


Time taken for k means completition:  3.38691353136

['cluster1: 17', 'cluster2: 5', 'cluster3: 84', 'cluster4: 69', 'cluster5: 13', 'cluster6: 22', 'cluster7: 1', 'cluster8: 44', 'cluster9:
230', 'cluster10: 32']
Jaccard Similarity: 0.302008820365
Rand similarity : 0.77913793684
3.95587466146 seconds taken for execution of complete program
```

As we can see from the above figure it takes 19 seconds roughly and 27 iterations.

Output performance for Map reduce Implementation:

```
hadoop@hadoop-VirtualBox:~/hadoop$
hadoop@hadoop-VirtualBox:~/hadoop$ bin/hadoop jar km1.jar KMeans iyer kmoutput 517
Enter value of K:
10
Select your centroids:
1
You Selected: 1 -1      1.0     0.72    0.1     0.57    1.08    0.66    0.39    0.49    0.28    0.5     0.66    0.52
2
You Selected: 2 1       1.0     1.58    1.05    1.15    1.22    0.54    0.73    0.82    0.82    0.9     0.73    0.75
3
You Selected: 3 1       1.0     1.1     0.97    1.0     0.9     0.67    0.81    0.88    0.77    0.71    0.57    0.46
4
You Selected: 4 1       1.0     0.97    1.0     0.85    0.84    0.72    0.66    0.68    0.47    0.61    0.59    0.65
5
You Selected: 5 1       1.0     1.21    1.29    1.08    0.89    0.88    0.66    0.85    0.67    0.58    0.82    0.6
6
You Selected: 6 1       1.0     1.45    1.44    1.12    1.1     1.15    0.79    0.77    0.78    0.71    0.67    0.36
7
You Selected: 7 1       1.0     1.15    1.1     1.0     1.08    0.79    0.98    1.03    0.59    0.57    0.46    0.39
8
You Selected: 8 1       1.0     1.32    1.35    1.13    1.0     0.91    1.22    1.05    0.58    0.57    0.53    0.43
9
You Selected: 9 1       1.0     1.01    1.38    1.21    0.79    0.85    0.78    0.73    0.64    0.58    0.43    0.47
10
You Selected: 10        1       1.0     0.85    1.03    1.0     0.81    0.82    0.73    0.51    0.24    0.54    0.43    0.51
16/11/05 02:21:30 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
16/11/05 02:21:33 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
16/11/05 02:21:33 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
16/11/05 02:21:33 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your app
.
```

Final Output:

```
                Bytes Written=0
Number of Executions: 30
Total Execution Time: 49 seconds
hadoop@hadoop-VirtualBox:~/hadoop$
```

New centroids are obtained in 'Myfile.txt':

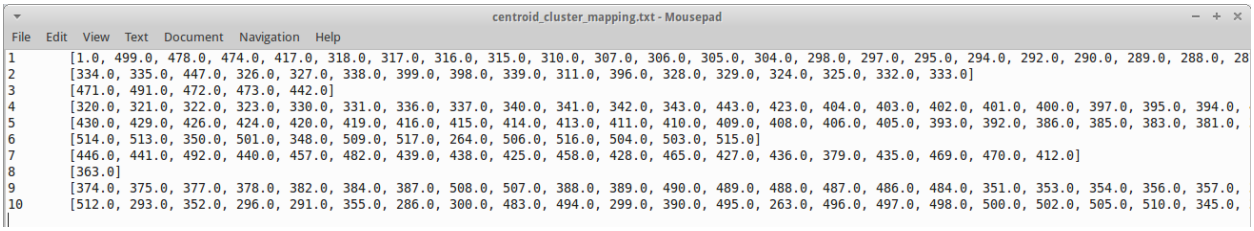
MyFile.txt - Mousepad

File Edit View Text Document Navigation Help

1	1	1.0	0.95	0.93	0.91	0.88	0.69	0.57	0.51	0.51	0.74	0.74	0.78
2	1	1.0	1.17	1.3	1.15	1.22	1.08	1.15	1.23	1.54	2.97	4.51	6.7
3	1	1.0	1.02	2.03	3.52	1.78	10.2	14.35	15.17	10.08	9.86	8.97	7.7
4	1	1.0	1.09	0.91	0.96	1.0	0.94	0.93	0.99	1.19	1.84	2.32	2.92
5	1	1.0	0.89	1.09	1.14	1.13	1.71	1.89	2.34	2.53	2.06	2.01	1.95
6	1	1.0	1.8	3.83	6.76	5.12	4.17	3.51	3.37	1.81	1.56	1.46	1.34
7	1	1.0	1.02	1.15	1.24	1.47	2.76	3.96	5.59	4.96	3.54	3.34	2.94
8	1	1.0	2.67	2.41	3.75	6.18	38.8	86.92	25.58	14.81	6.73	2.72	2.42
9	1	1.0	0.96	1.18	1.49	1.72	3.97	3.92	3.33	1.79	1.35	1.29	1.22
10	1	1.0	1.25	1.97	2.7	2.31	1.72	1.53	1.53	1.23	0.89	0.89	0.86

Column 2 in above file can be ignored as this is not included in our computation for new centroids

Centroid-Cluster mappings are obtained in 'centroid_cluster_mapping.txt'



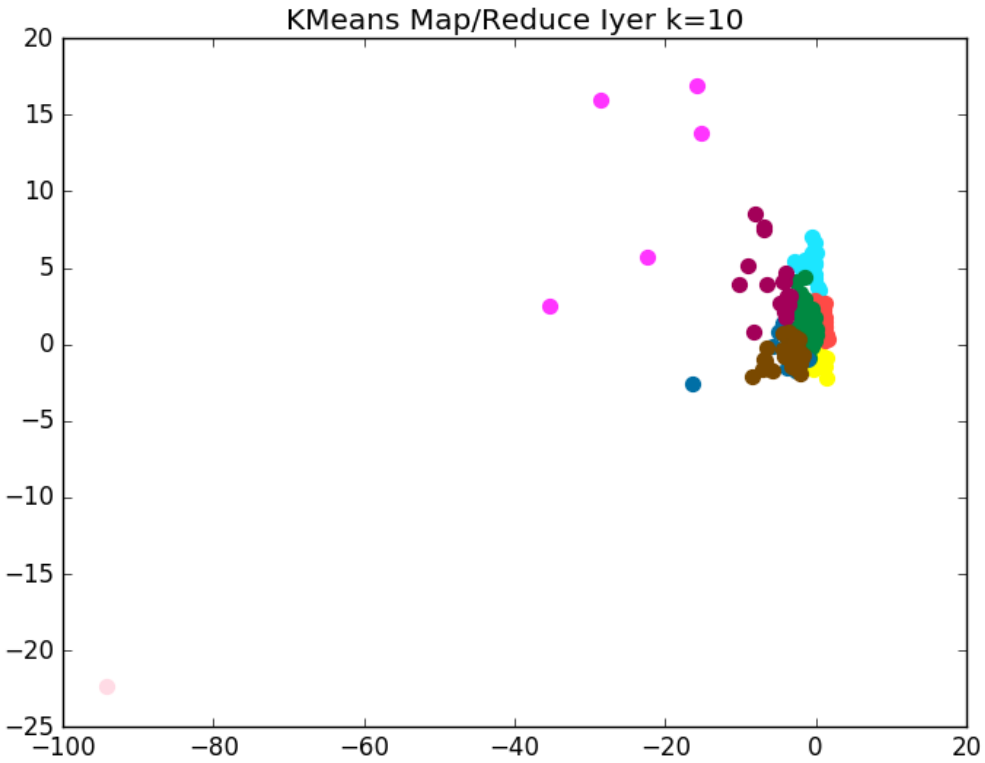
As seen the map reduce takes 46 seconds and converges after 30 iterations.

As Iyer data set has greater number of data points it took greater time (46 seconds) as compared to Map Reduce cho (36 seconds). Hence, even though Mapreduce may be better in terms of storage computation, we can clearly see that for the given dataset, linear KMeans has outperformed KMeans Map/reduce.

External Index Computation:

```
Jaccard coefficient: 0.29
Rand Index: 0.68

In [31]:
```

PCA Visualization:



IV. CONCLUSION

We have successfully implemented KMeans Clustering, DB Scan, Hierarchical Agglomerative Clustering and calculated/ inferred the time complexity of the algorithms. By using efficient data structures in python programing language, we have attempted to reduce the computational complexity of the algorithms. In addition, as a part of the project requirement, we have also implemented KMeans using MapReduce framework [Single node cluster] and compared it with the non-parallel KMeans. The results for the algorithms have been validated as per the project requirement using external index[Jaccard]. The datasets have been visualized using principal component analysis[PCA] and included in the project report.

V. REFERENCES

[1]. http://web.stanford.edu/~ashishg/papers/mapreducecomplexity.pdf

[2].https://wiki.rice.edu/confluence/download/attachments/4425835/HabaneroJava-Hadoop.pdf?version=2&modificationDate=1383968656383&api=v2

[3]. http://www.iaeng.org/publication/WCE2009/WCE2009_pp308-312.pdf