**CSE 601**

**DATA MINING AND BIO INFORMATICS**

**PROJECT 3 – REPORT**

**IMPLEMENTATION OF Classiferss**

**SUBMITTED BY:**

**ARTI GUPTA – 50170010**

**artigupt@buffalo.edu**

**SUDHARCHITH SONTY - 50169912**

[**sudharch@buffalo.edu**](mailto:sudharch@buffalo.edu)

**K-NEAREST NEIGHBOURS CLUSTERING ALGORITHM**

**Implementation Details:**

* We have used JAVA for the implementation of KNN Algorithm.
* The following methods have been used and called as a part of the implementation.

readFile (String input)

*Parse the input data from .txt file and store into ArrayList of objects which map to a class with two attributes the dimensions and the classname.*

splitdata(input)

*This function is used to perform the 10-fold split for cross validation.*

*The split is performed by dividing the data into 10 parts, and then selecting each part randomly.*

Find majority element()

*This method is used for finding out the majority class by consensus. Ties are broken by choosing an element at random.*

calculate\_distance()

*function to get Euclidean Distance*

calculatestatistics(list)

*Function to get the average of the cumulative performance measures.*

**CLASSES:**

*Class points: attributes are list of dimesions and classname*

*Class result: attributes are distance and classname.*

**STEPS:**

1. The algorithm first calls the parse function, to generate a list of objects which are instances of the points class described above from the input file.
2. Then, we call the split method to split the data as per our k-fold validation requirement.
3. We, take a random variable and assign the corresponding set of values i.e.; list(get) to test and the other 9 parts as training.
4. While, we are looping through the data, we make sure that the condition is that the test data is not equal to the training data to avoid confusion
5. For each, n-dimensional point in the test, we compute the Euclidean distance with respect to each point in the training and put the distance along with its corresponding class in the result list
6. Then, we get sort the distance list and get the class names of the top k distances and transfer them into an array.
7. Next, we call the Majority element method and get the top repeated terms from the array and ties are broken by choosing an element at random.
8. The point is assigned its new class and then we check for equality between the new class and the old class.
9. Based on the values we update the TP, TN, FP, FN values and get the values
10. At the end of the final run, (number of folds) we calculate the cumulative accuracy and other performance measures
11. The algorithm automatically stops, when the number of iterations reaches the maximum number of iterations.

**Datasets Used and Results Obtained:**

1. **Cho.txt –** 
   1. *Run1, k =5*

*Cluster 1 Size is 39*

*Cluster 2 Size is 97*

*Cluster 3 Size is 125*

*Cluster 4 Size is 62*

*Cluster 5 Size is 63*

*Jaccard Coefficient is 0.5*

* 1. *Run2, k = 5*

*Cluster 1 Size is 84*

*Cluster 2 Size is 86*

*Cluster 3 Size is 43*

*Cluster 4 Size is 93*

*Cluster 5 Size is 80*

*Jaccard Coefficient is 0.181347150259*

*Run3, k =3*

Cluster 1 Size is 194

Cluster 2 Size is 71

Cluster 3 Size is 121

Jaccard Coefficient is 0.113989637306

1. **Iyer.txt –**
   1. *Run1, k =10*

*Cluster 1 Size is 11*

*Cluster 2 Size is 49*

*Cluster 3 Size is 32*

*Cluster 4 Size is 82*

*Cluster 5 Size is 42*

*Cluster 6 Size is 176*

*Cluster 7 Size is 37*

*Cluster 8 Size is 22*

*Cluster 9 Size is 45*

*Cluster 10 Size is 21*

*Jaccard Coefficient is 0.129593810445*

* 1. Run2, k = 5

*Cluster 1 Size is 37*

*Cluster 2 Size is 181*

*Cluster 3 Size is 33*

*Cluster 4 Size is 68*

*Cluster 5 Size is 198*

*Jaccard Coefficient is 0.0812379110251*

* 1. *Run3, k =3*

Cluster 1 Size is 327

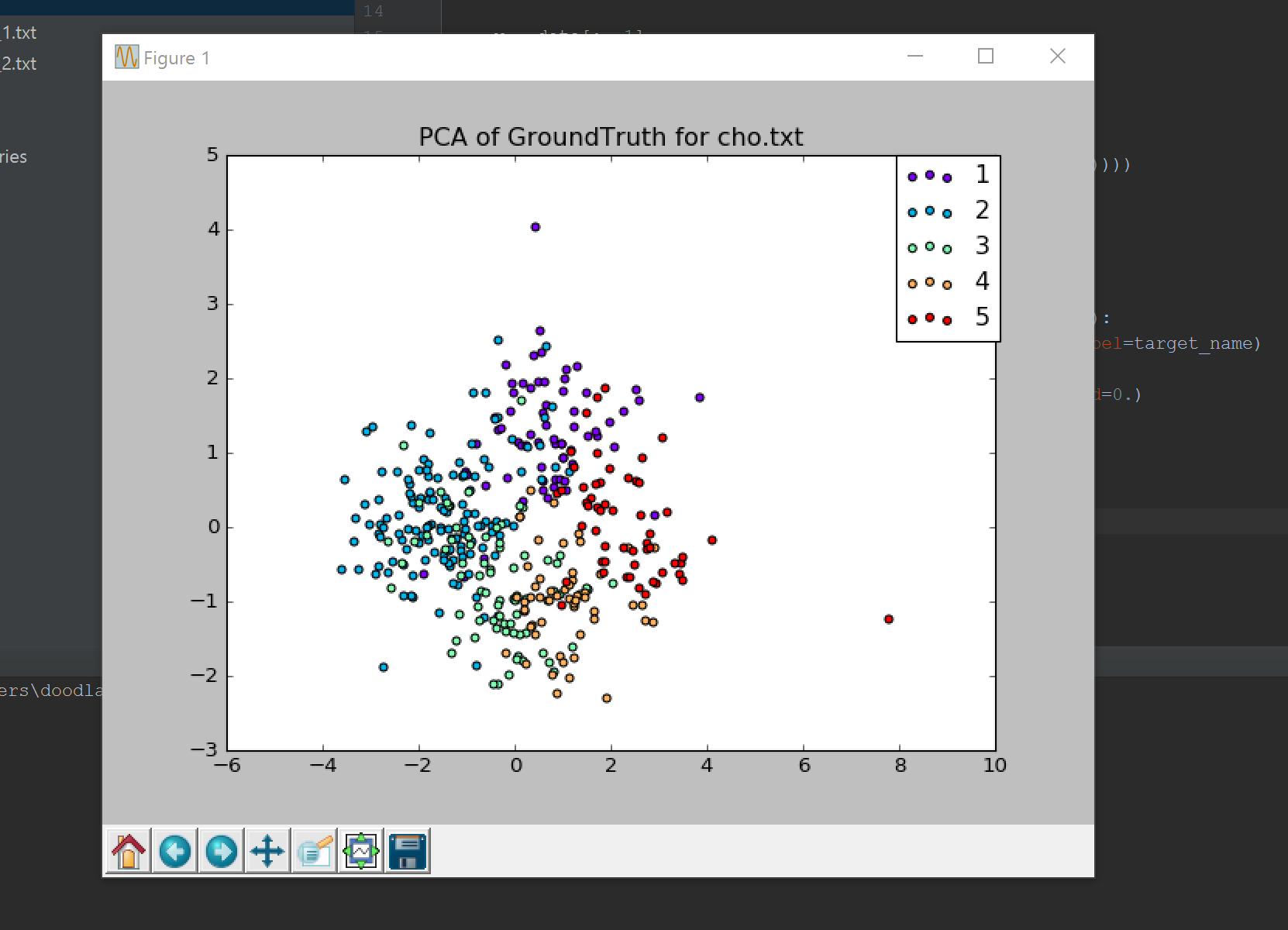
Cluster 2 Size is 83

Cluster 3 Size is 107

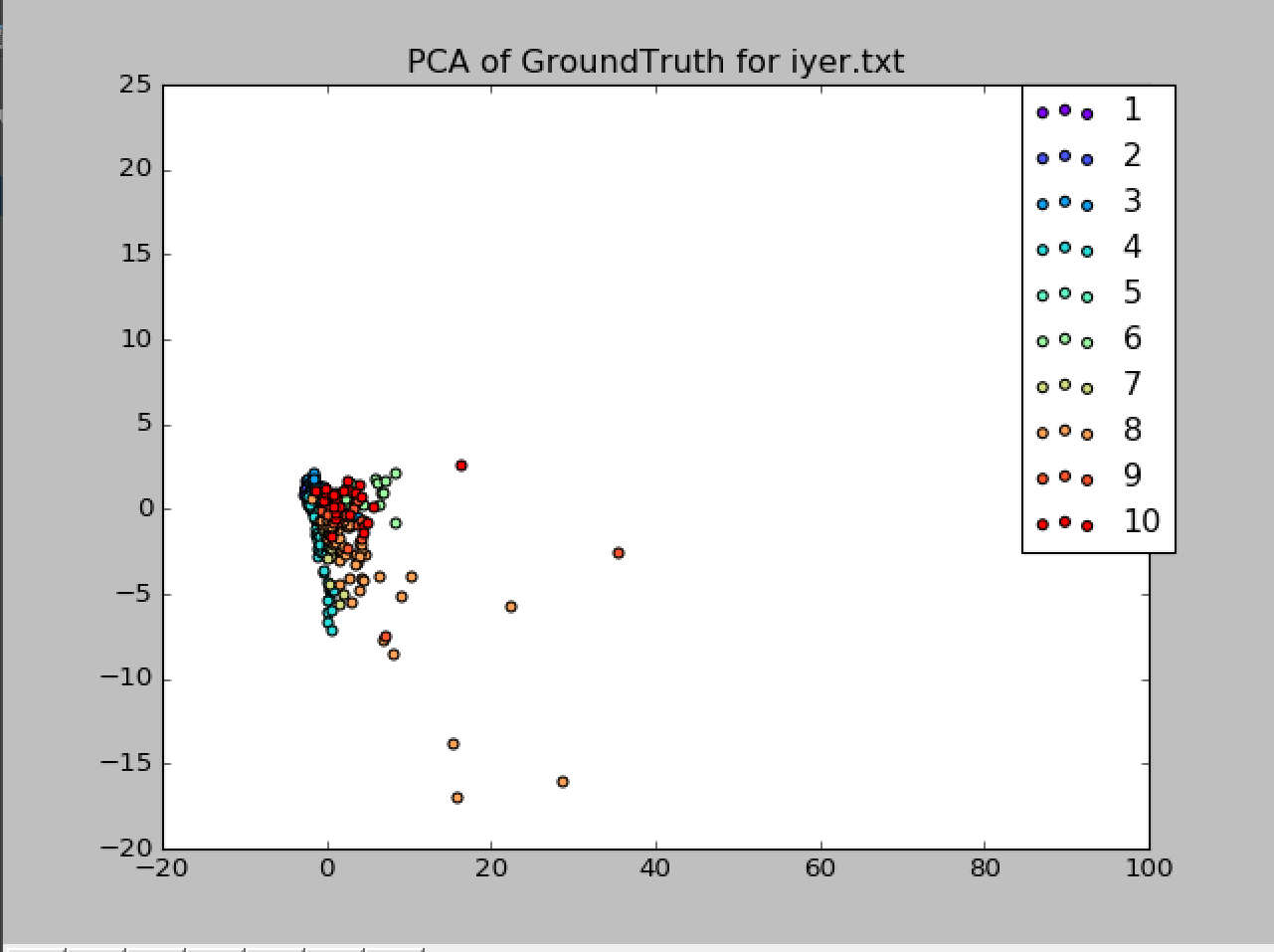
Jaccard Coefficient is 0.193423597679

**Visualization Details:**

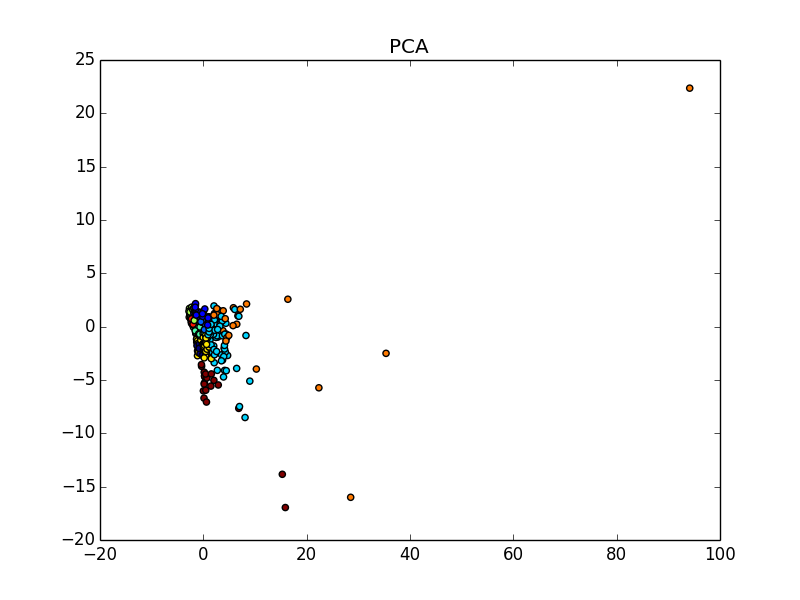
***PCA FOR GROUNDTRUTH – CHO.TXT***

****

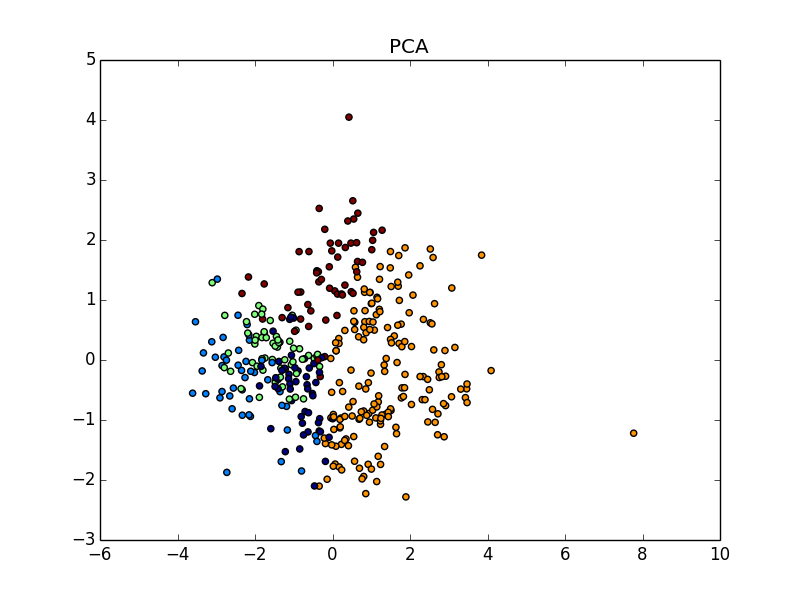
***PCA FOR GROUNDTRUTH – IYER..TXT***

****

***PCA FOR RESULT OF RUN 1 – IYER..TXT- KMEANS***

******

***PCA FOR GROUNDTRUTH – CHO.TXT-KMEANS***

******

**PROS:**

* For a large dataset, K means algorithm is preferred.
* K-means clustering is usually more efficient run time wise ( O()).
* K-means algorithm is linear in the number of data objects.

**CONS:**

* It needs the number of clusters to be pre-specified.
* It only works well when the shape of the clusters is hyper-spherical.
* Since, the initial set of centroids is random, we may get different clustering results on different runs of the algorithm. Thus, the results may not be repeatable and lack consistency.

**HIERARCHICAL AGGLOMERATIVE CLUSTERING:**

**Implementation Details:**

* **I**mplementation of Hierarchical Agglomerative clustering algorithm is done in JAVA.
* Following functions we have called here.

read\_data(fname)

to read data from file

compute\_dist\_mat(pts)

to compute distance matrix

clustering(dist\_mat)

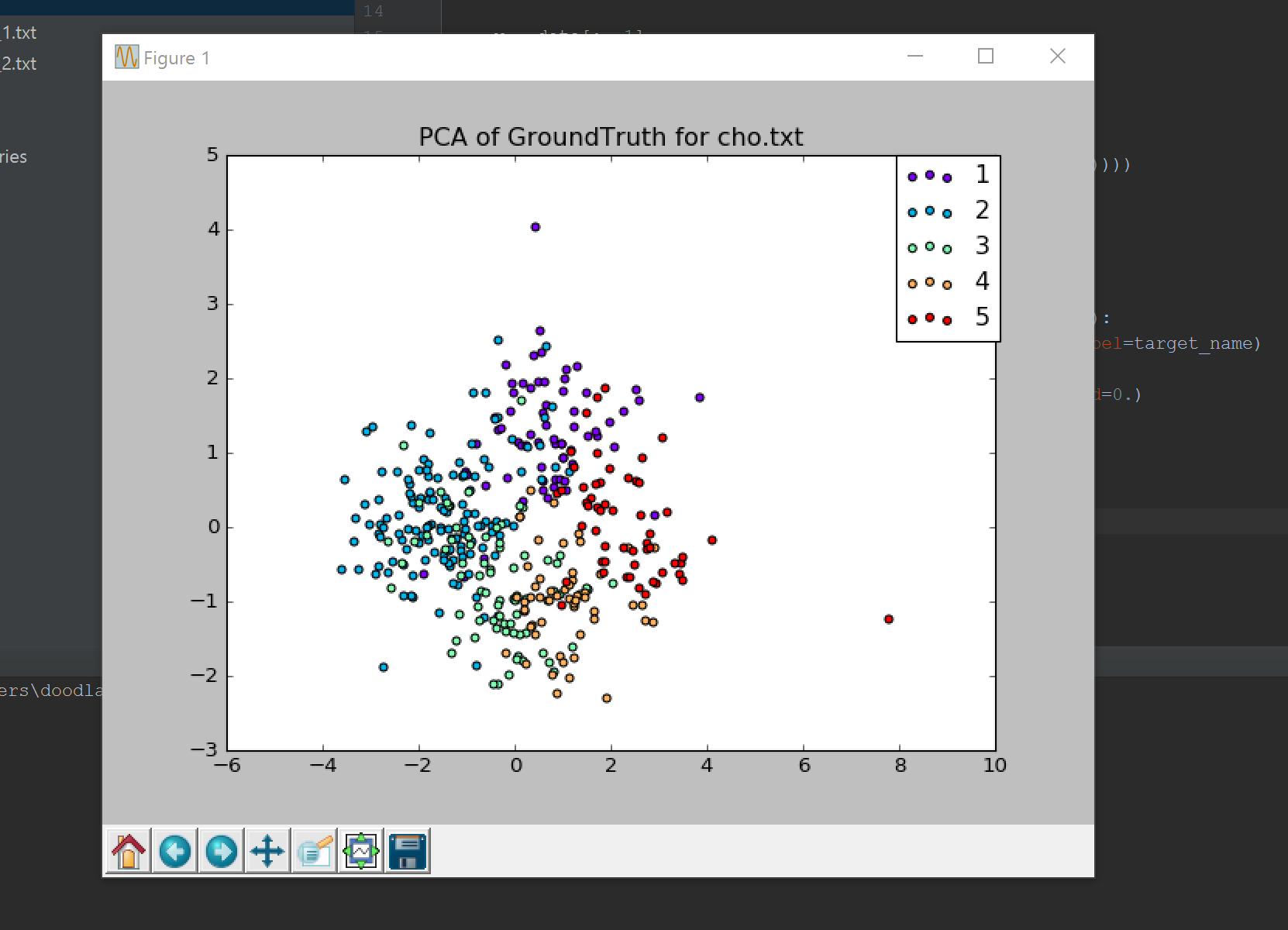
to cluster each pair and finally getting one cluster comprises of several small pair of clusters

**STEPS:**

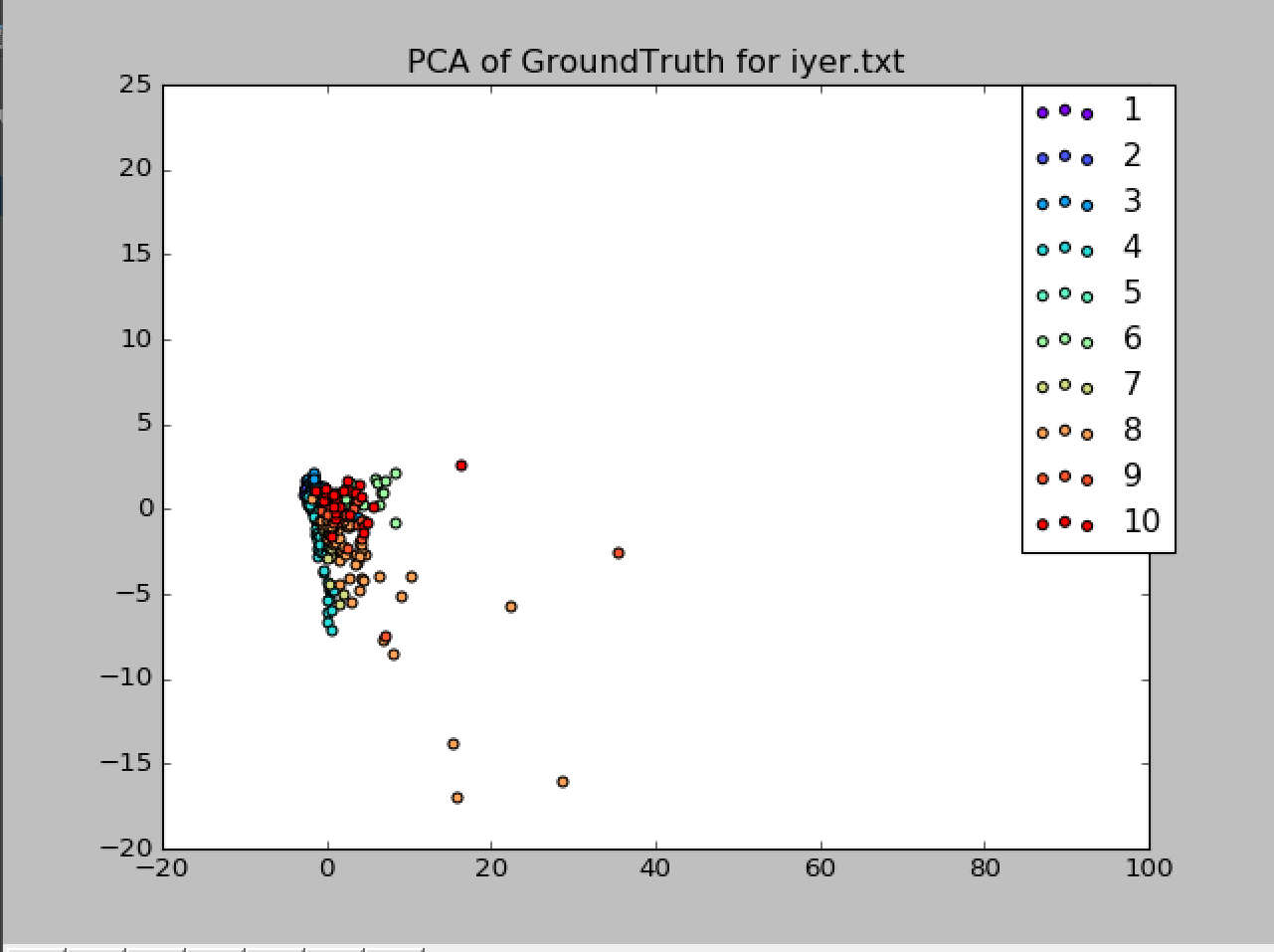
1. The algorithm first calls the read\_data function to read the input file data
2. Then, the algorithm computes the distance matrix using numpy library.
3. It makes clusters, starting with the pair of points which have minimum distance between them; (plist).
4. Then, it keeps merging clusters into altogether till the algo is asked to stop after certain points; (n-5) cluster etc.
5. Finally, we get one cluster comprises of several small clusters.

**Visualization Details:**

***PCA FOR GROUNDTRUTH – CHO.TXT***

****

***PCA FOR GROUNDTRUTH – IYER..TXT***

****

**PROS:**

* It can produce an ordering of the objects, which may be informative for data display.
* Smaller clusters are generated, which may be helpful for discovery.
* It shows more quality as opposed to K-means.
* Does not need number of clusters to be pre-specified.
* With hierarchical clustering, you will most definitely get the same clustering results.
* Hierarchical Clustering can give a diﬀerent partitioning of data depending on the level-of-resolution.

**CONS:**

* Works well only when the data set is relatively small.
* For, large datasets, the performance decreases and the execution time increases. O(n2).
* No provision can be made for a relocation of objects that may have been 'incorrectly' grouped at an early stage.
* Use of different distance metrics for measuring distances between clusters may generate different results.

**DENSITY-BASED ALGORITHM (DBSCAN):**

**Implementation Details:**

* To find Density based algorithm, we have used Python.
* Different values of parameters (epsilon, minimum\_pts) is given to check the no. of clusters formed.
* Jaccard similarity score is calculated for each parameters using inbuilt python function.
* The following method have been used and called for implementation.

\_regionQuery(point\_id,eps)

*get neighboring points w.r.t centroid*

DBSCAN(eps, min\_pts)

*density based cluster algorithm is implemented which checks for each point in the data.*

expandCluster(point\_id, neighbors, clusters, count, eps, min\_pts, visited)

*Repeat the process for other points and add values to Noise*

def get\_groundtruth()

*function to get ground truth values from file*

get\_jaccard\_score()

*function to calculate jaacard score*

**STEPS:**

1. The algorithm first read the input text file.
2. Then, it calculates its neighbour.
3. It calculates DBSCAN algorithm, where, it adds each point to visited if it has been visited in data.
4. Else, it adds that point to Noise and expand algorithm further.
5. In expandcluster function, it again proceed with points which are not visited, then add them and continue processing like that.
6. Finally, it calculates groundtruth values.
7. Jaccard score and random index has been calculated.

**Datasets Used and Results Obtained:**

1. **Cho.txt –**

DBSCAN(4,80)

eps = 4

min\_pts = 80

jaccard\_coefficient is 0.176165803109

adjusted\_rand\_index is 0.32136703885094814

DBSCAN(3,3)

eps = 3

min\_pts = 3

jaccard\_coefficient is 0.383419689119

adjusted\_rand\_index is 0.47340450713557203

DBSCAN(1.8,20)

eps = 1.8

min\_pts = 20

jaccard\_coefficient is 0.142487046632

adjusted\_rand\_index is 0.16053228077880788

DBSCAN(3, 100)

eps = 3

min\_pts = 100

jaccard\_coefficient is 0.178756476684

adjusted\_rand\_index is 0.3660110493976132

1. **Iyer.txt –**

DBSCAN(4,80)

eps = 4

min\_pts = 80

jaccard\_coefficient is 0.243713733075

adjusted\_rand\_index is 0.2321465145855259

DBSCAN(3, 3)

eps = 4

min\_pts = 80

jaccard\_coefficient is 0.193423597679

adjusted\_rand\_index is 0.3168913262717644

DBSCAN(1.8,20)

eps = 1.8

min\_pts = 20

jaccard\_coefficient is 0.2166344294

adjusted\_rand\_index is 0.390976042124015

DBSCAN(3,100)

eps = 3

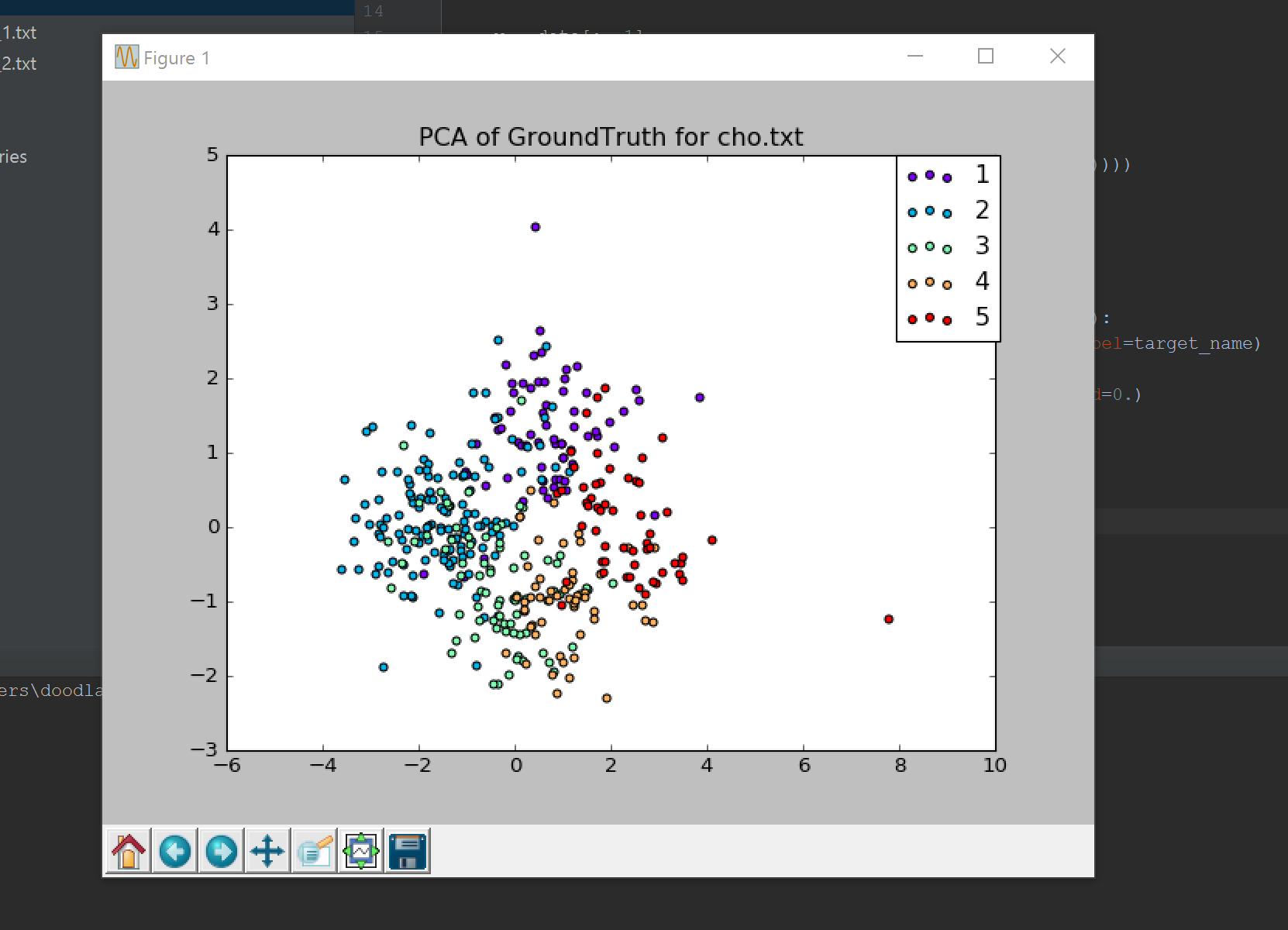
min\_pts = 100

jaccard\_coefficient is 0.203094777563

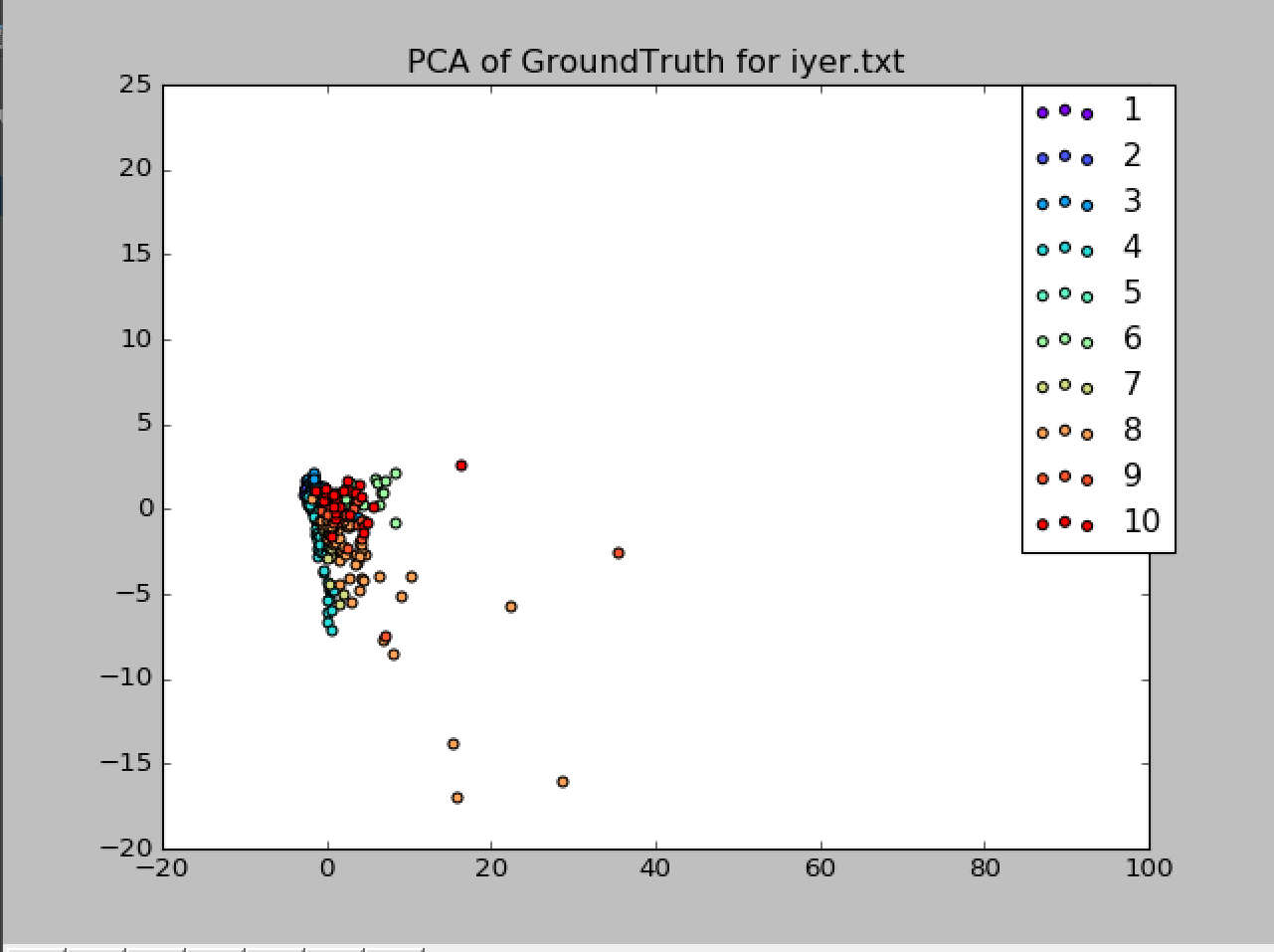
adjusted\_rand\_index is 0.31518570556874526

**Visualization Details:**

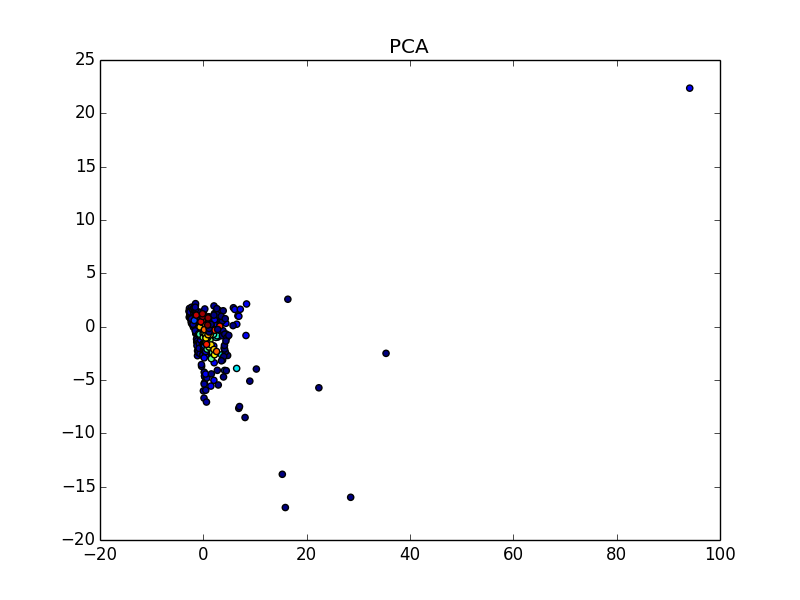
***PCA FOR GROUNDTRUTH – CHO.TXT***

****

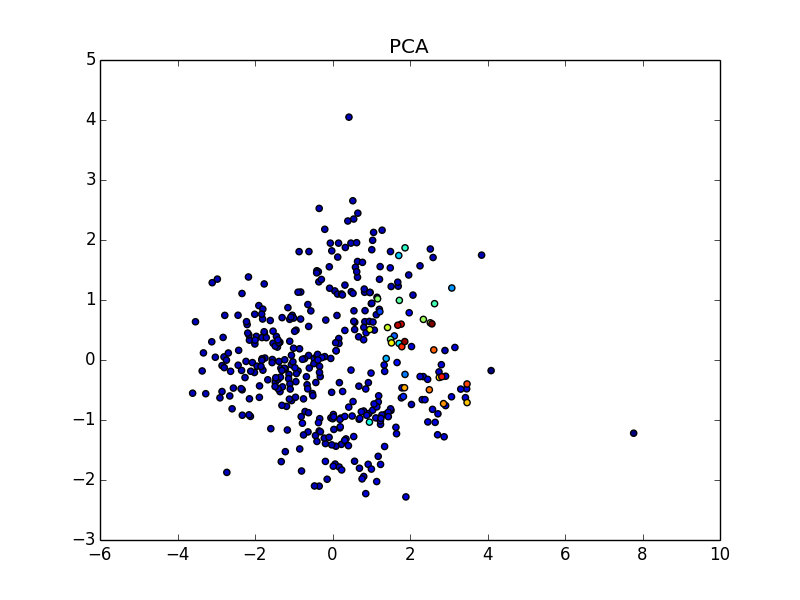
***PCA FOR GROUNDTRUTH – IYER..TXT-DBSCAN***

****

***PCA FOR RESULT– IYER..TXT-DBSCAN***

******

***PCA FOR RESULT– CHO..TXT-DBSCAN***

******

**PROS:**

* DB Scan algorithms produces clusters of arbitrary shape.
* The algorithm, to a large degree is robust to noise.
* The number of clusters does not have to be pre specified.
* DB Scan requires just two parameters and is mostly insensitive to the ordering of the points in the database.

**CONS:**

* Deterministic Data sets with varying densities are problematic.
* Requires connected regions of sufficiently high density.
* DB Scan cannot cluster data sets well with large differences in densities.

**COMPARISION BETWEEN DIFFERENT ALGORITHMS:**

|  |  |  |  |
| --- | --- | --- | --- |
| ATTRIBUTE | K-MEANS | HIERARCHIAL AGGLOMERATIVE | DB SCAN |
| RUNNING TIME | O(kn), k = number of clusters, n is number of points | O(n2), n is the number of points | O(n), n is the number of points |
| SIZE OF DATA SET | Works well on Large and small datasets | Works well on small data sets | Works well on smaller data sets |
| VALUE OF K(number of cluster) | Pre-Specified | Dynamic | Dynamic |
| SHAPE OF CLUSTERS | Hyper-spherical | Arbitrary | Arbitrary |
| RUNTIME SCALING | Maintains a linear runtime. Does scale up in large data sets but still is better than Hierarchical. | Scales up to quadratic runtime. | Worst case, it can scale up to quadratic runtime. |
| CONSISTENCY | Random centroids, leads to random results for each run | More or less the same clusters for each iteration, thus being consistent | Is insensitive to ordering and produces different results based on parameters. |
| EFFECT OF NOISE IN DATA SET | Outliers have some effect | Noisy data has effect on result. | Robust towards Noise |
| JACCARD COEFFICIENT VALUE | For new\_dataset\_1  JC = 0.56 | For new\_dataset\_2  JC = | For new\_dataset\_1  JC = |

**HADOOP MAP-REDUCE K-MEANS:**

**Implementation Details:**

**PREPROCESSING STAGE:**

* In the preprocessing stage of the Algorithm, we use two functions,

**parseInputData :** to read from the file and create an input data map with key = geneId and value = list(expression values)

Running Time: O(n)

**getrandomcentroids:** to read from the inputdata map and select 5 random values and create initial centroids that will be used for the first iteration of the Map Reduce K-Means Algorithm

Running Time: O(k), k = initial value

Number of Mappers is 1

Number of Reducers is 1

**MAPPER FUNCTION:**

* The Mapper function, takes as input each line from the input file(new\_dataset\_1) and extracts the geneId(point) and the list of expression values (expressionvalues)
* For the extracted expression values, we calculate the Euclidean distance of each n-Dimensional point with the initial centroids, computed by the Calculate Distance Method.
* The computed distance values are then compared with each other and for each point, the centroid closest to it is marked as it Cluster ID.
* We then write, the closestcluster(Cluster ID) and the point corresponding to it and send it to the Reducer.
* Running Time of Mapper Phase: O(n), n is the length of input file.

**REDUCER FUNCTION:**

* The input to the reducer function is a tuple which has the point (Cluster ID) and a list of values (points belonging to that cluster).
* We then iterate over the set of points and for each point, we get the list of values associated with that point.
* We then compute the mean of all the expression values thus obtained and store the mean in a temporary list.
* We then update the final\_initial\_centroids map created in the Pre-Processing Stage to reflect the updated centroids.
* The updated map is then fed as the input to the Mapper again to recomputed the clusters.

1. The program runs till the maximum iterations have been reached.
2. If, during the iterations, for a given iteration, the set of previous centroids is equal to the set of new centroids, then the program stops and outputs that as the result.
3. Total Running Time is O(cn), where c is the number of iterations that are needed.

**Datasets Used and Results Obtained:**

**Dataset Used: iyer.txt**

**Result Obtained:**

**JACCARD COEFFICIENT IS: 0.27**

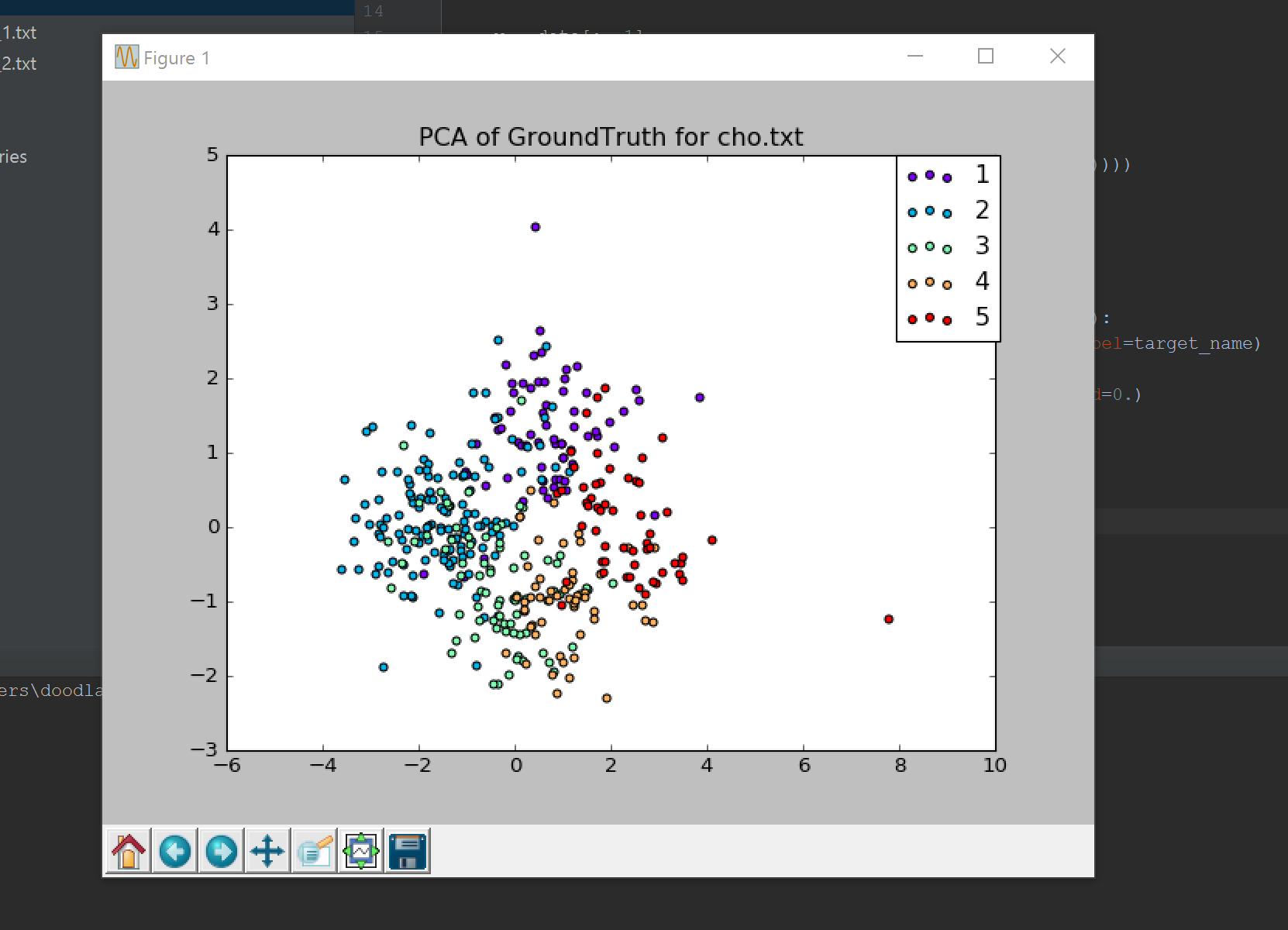
**Dataset Used: cho.txt**

**Result Obtained:**

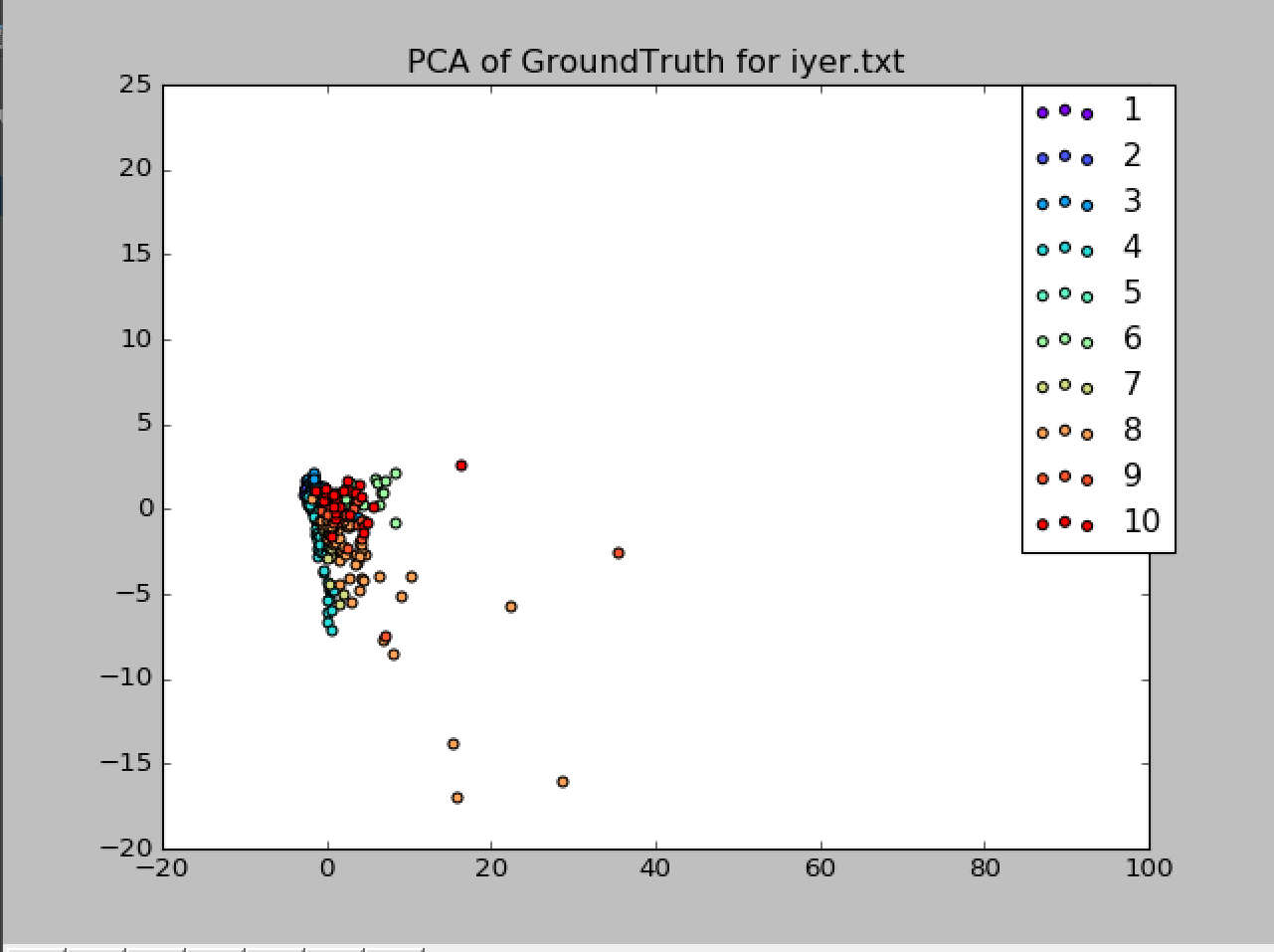
**JACCARD COEFFICIENT IS: 0.25**

**Visualization Details:**

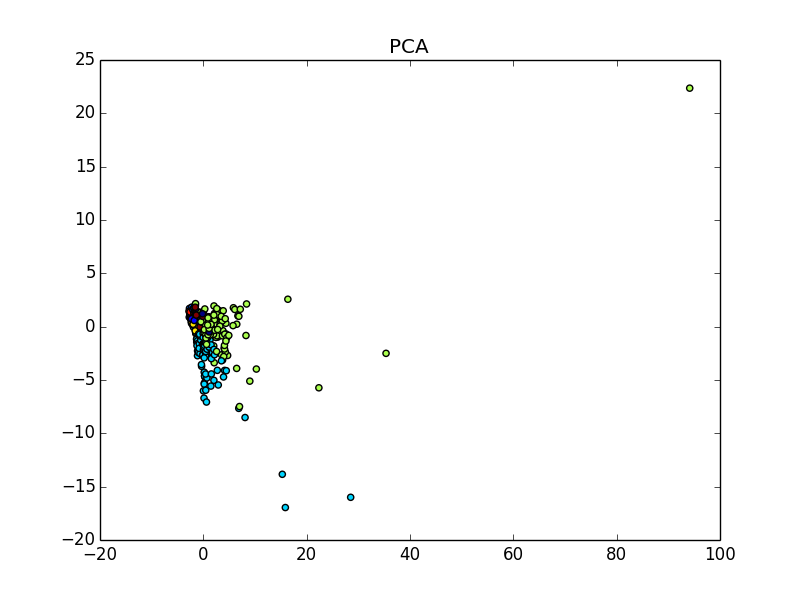
***PCA FOR GROUNDTRUTH – CHO.TXT***

****

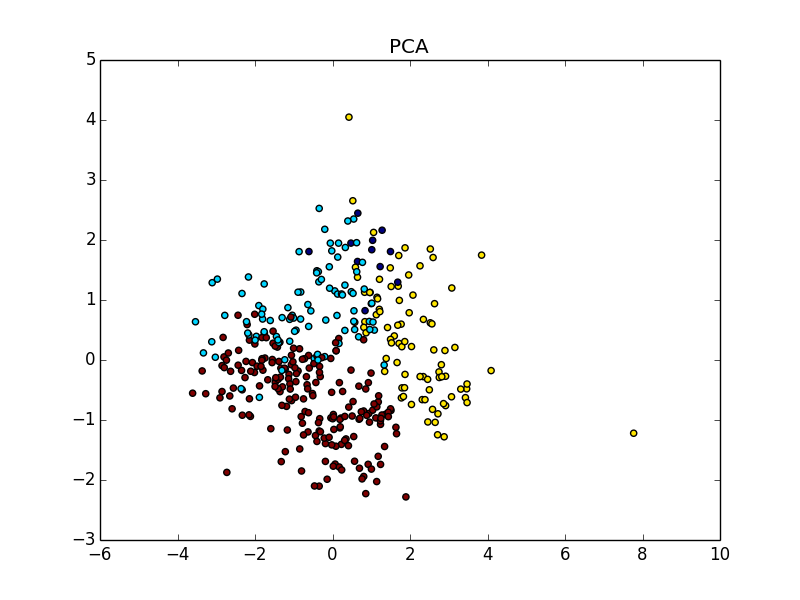
***PCA FOR GROUNDTRUTH – IYER..TXT***

****

***PCA FOR HADOOP MAP REDUCE– IYER..TXT***

******

***PCA FOR HADOOP MAP REDUCE – cho.TXT***

******

**IMPROVEMENTS THAT CAN BE MADE and OBSERVATIONS:**

* Improvements in the performance of the K means algorithm, is to use multiple mappers and reducers as opposed to the Single mapper and Single Reducer that we have implemented.
* **SPEEDUP:** To measure the speedup, we can keep the dataset constant and increase the number of computers in the system. The perfect parallel algorithm demonstrates linear speedup: a system with m times the number of computers yields a speedup of m. However, linear speedup is difficult to achieve because the communication cost increases with the number of clusters becomes large.
* **SIZEUP:** The Size up factor can be measured, if we keep the Number of computers constant and keep increasing the size of the dataset. Since, K means generally works well on larger data sets, Parallel K means also works increasingly well
* **SCALABILITY:** Scalability factor can be measured by increasing the size of the datasets, proportional to the number of the computers.