# DATA MINING & BIOINFORMATICS

**PROJECT 2**

CLUSTERING ALGORITHMS

Himal Dwarakanath | himaldwa | 50207594

MuthuPalaniappan Karuppayya | muthupal | 50208484

Neeharika Nelaturu | nnelatur | 50207062

## K-means clustering:

K-means algorithm clusters the set of points based on their distance from the k cluster centroids. The algorithm runs in 2 phases:

*Phase 1*: Find the closest centroid to each point in dataset and add them to the centroid's cluster.

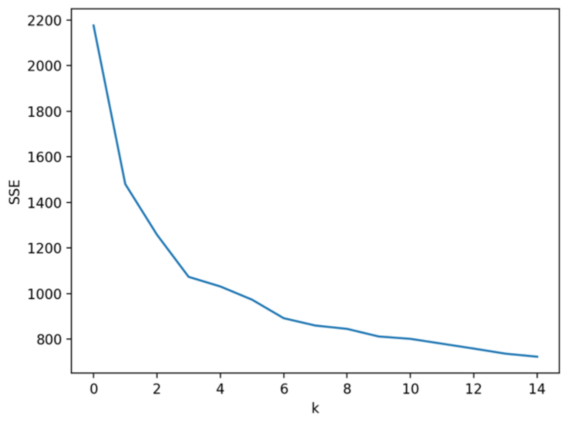
*Phase 2*: Find new centroid of each cluster as mean of all the points assigned to the cluster.

The 2 phases are repeated until convergence, i.e. when there is no change in position of centroids.

K-means algorithm is implemented as follows:

1. The feature matrix is extracted from the input file and stored in *adjusted_matrix*.
2. If centroids are given as initial input then they are read into *initial_centroids*. Otherwise we initialize with *k* random centroids.
3. The kMeans implementation begins with a loop with convergence condition (*initial_centroids == centroids_old*) where *centroids_old* is initialized to a matrix with all -1s.
4. Each point is assigned to the cluster whose centroid is at minimum Euclidean distance from the point.
5. Prior to finding the new centroid, *centroids_old* takes the value of *centroids* in order to check for convergence before the next iteration.
6. The points in each cluster are averaged to find the new centroid of the cluster.
7. 4 to 6 are repeated until convergence is achieved or until the iterations exceed the set threshold which is stored in the variable *break_count*.

To deal with a condition when initial number of clusters are unknown, we have calculated the Sum Squared Error (SSE) between the cluster centroid and points in each cluster while ranging the number of clusters between 1 to 14. The plots are seen as below for datasets cho.txt(left) and iyer.txt(right) while having a *break_count* = 20.



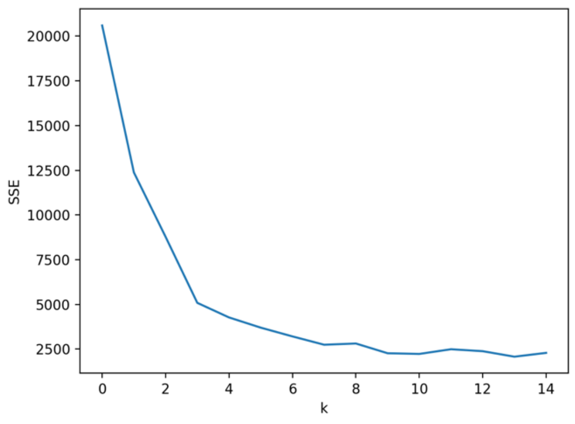


**Fig1: SSE plots for cho.txt(left) and iyer.txt(right)**

The plots indicate that k >= 5 gives good clustering results for cho.txt while k >=9 gives good clustering results for iyer.txt. For our visualizations and external index computation, we choose the following values of k:

cho.txt: k = 5

iyer.txt: k = 11

**Advantages of K-Means:**

1. Ease of implementation
2. Runs with time complexity O(t * k * n), where k is the number of clusters, t is the number of iterations and n is the number of data points. Thus the algorithm works very efficiently for low values of k.
3. Closely packed clusters are formed if the clusters are globular.

**Disadvantages of K-Means:**

1. Difficulty in predicting optimal value of k.
2. The algorithm is dependent on the random points chosen initially, different initial points may result in different cluster assignment.
3. Works poorly when clusters aren't similarly sized and are of varying densities.
4. Clustering results may not be good when the original clusters aren't globular in shape.

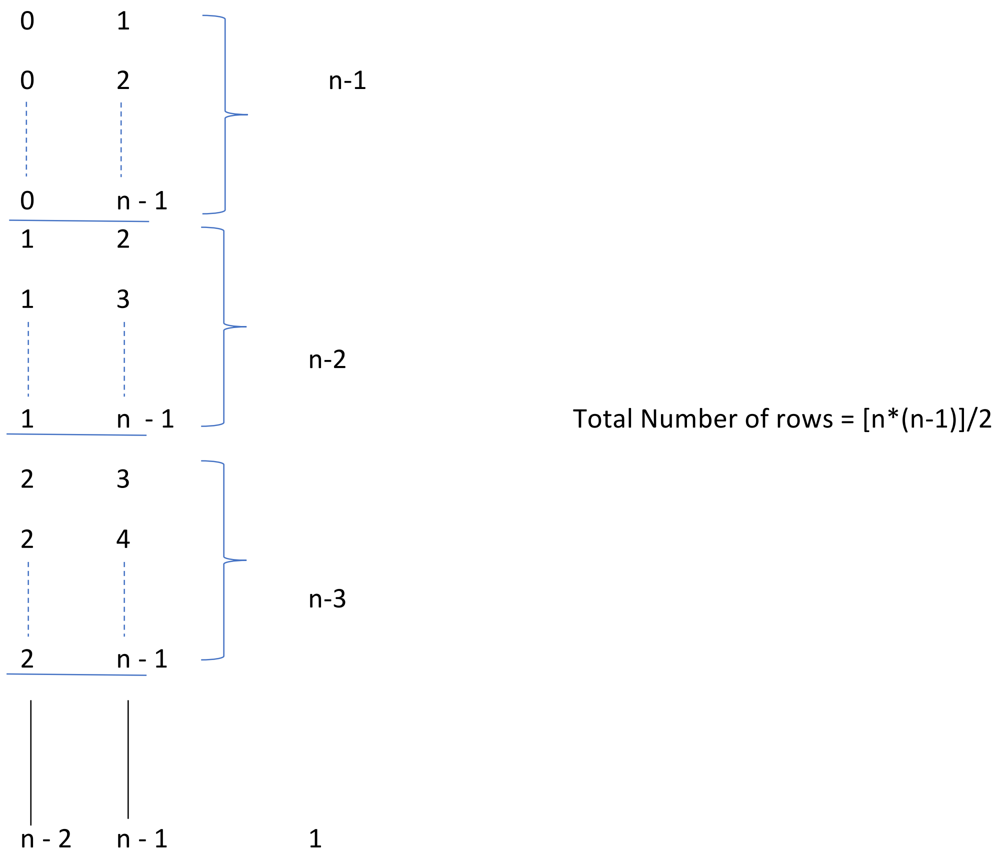
## Agglomerative Hierarchical Clustering:

- Agglomerative Clustering is a bottom-up approach starting with each point in a separate cluster, repeatedly joining the most similar pair of clusters and updating the similarity of the new cluster to other clusters until there is only one cluster
- We use single linkage i.e. we consider the distance of the *closest pair of data objects* belonging to different clusters
- In single-link clustering or single-linkage clustering, the similarity of two clusters is the similarity of their most similar members
- This single-link merge criterion is local

**Algorithm:**

- The input file is read and loaded into 'data'
- The columns with no variation (i.e. Standard Deviation/Variance = 0) are removed as they are not significant for clustering
- Further, the data is normalized and stored in 'adjusted_matrix' (Normalization is done to ensure uniformity in the data)

- A function 'dist_mat' is defined to calculate the distance between each data point to every other data point as follows:
    1. The function dist_mat takes a matrix as the input parameter
    2. The distance matrix is initialized with [n*(n-1)]/2 rows where n corresponds to number of points and 3 columns (point1, point2, distance) with zeros



    3. Distance matrix is populated using 2 loops where first loop is for the point1 and second loop is for the point2
    4. Euclidean distance is calculated using numpy's linalg.norm function and the third column is populated
    5. The dist_mat function returns the populated 'distance_matrix'

- A function hc performs hierarchical clustering on the input matrix in the following way:
    1. The function dist_mat takes two input parameters – input matrix, number of clusters
    2. The function dist_mat is called and the input matrix is passed as the parameter
    3. The obtained distance_matrix is converted to a data frame
    4. Further, it is sorted in ascending order
    5. A np array 'label' is initialized in the range of 0 to total number of points
    6. As we have sorted, the points in the first row correspond to the minimum distance. Hence, we replace the cluster ID of both point1 and point2 in first row with minimum(point1, point2) and replace both the points in distance_matrix with minimum(point1, point2)

7. Step 6 for all the other points is repeated
8. The hc function returns the label array

**Advantages:**

- No prior information about the number of clusters required
- It can produce an ordering of the objects, which is informative for data display (visualized mostly as dendograms)
- It is shown to capture concentric clusters
- It provides hierarchical relations between clusters
- We can create smaller clusters depending on where we cut the dendrograms which is helpful for discovery
- Meaningful taxonomies might be produced
- Allows anisotropic and non-convex shapes

**Disadvantages:**

- Greedy – less accurate and computationally expensive
- A single-link clustering can produce a chaining effect i.e. it produces straggling clusters. As the merge criterion is local, a chain of points can be extended for long distances without regard to the overall shape of the emerging cluster
- Sensitivity to noise and outliers
- hierarchical clustering is high in time complexity
- No objective function is directly minimized
- Algorithm can never undo what was done previously
- Sensitive to outliers
- Difficulty handling different sized clusters and convex shapes

## DBSCAN:

- DBSCAN is a density-based algorithm.
- DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise.
- It locates regions of high density that are separated from other regions of low density, where density = number of points within a specified radius.
- Points are classified into the following:
    1. Core points: Points having more than minimum number of points in its neighborhood. These points are present at the interior of a cluster. Any core points that are close enough – within a distance of given radius – are present in the same cluster.
    2. Border points: Points having less than minimum number of points in its neighborhood, but still present in neighborhood of other core points. Any border point that is close enough to a core point is present in the same cluster as the core point.
    3. Noise: Any point that is not a core or border point and also having less than minimum number of points in its neighborhood. Noise are not present in any cluster.

- Density-reachable: An object q is directly density-reachable from object p if q is within the ε-neighborhood of p and p is a core object.
- Density-connectivity: An object p is density-connected to object q with respect to ε and minimum number of points if there is an object o such that both p and q are density-reachable from o with respect to ε and minimum number of points.

**Algorithm:**

1. The data is loaded as *input_file* and then cast as a numpy array in to *data*. From this *data*, the first 2 columns are removed to get only the feature matrix and assign it to *data_feature_matrix*.
2. The epsilon (radius) value and the minimum no. of points are taken as user input and passed as arguments to the *dbscan* function.
3. In the *dbscan* function, *clusterid* is initially set to 0 and a numpy zeros matrix of shape no. of rows of *data_feature_matrix* x 1.
4. For each unvisited point *pt* in the matrix, find the neighbors of the point using '*regionQuery*', which returns the points within the distance *eps* of the point using Euclidean distance.
5. If the number of neighbors is less than the minimum number of points, classify the label of that point *pt* to be -1 and the next point is checked.
6. If the number of neighbors is more than the minimum number of points, classify the label of that point *pt* to the corresponding *clusterid*, which is incremented every time when a new cluster is formed.
7. The point is then passed through the '*expandCluster'* function, in which for every unvisited point '*point'* in the neighborhood of point *pt*, it is assigned to the same cluster id as *pt*'s cluster id. The point's neighbors are calculated by '*regionQuery'* and then added on to the existing neighboring points.
8. For every point in the neighborhood of point *pt* which has been assigned as noise earlier are classified as border points to the current cluster and thus, labeled as same cluster id.
9. At the end of the loop, after every point *pt* in the matrix are visited, *label* gives *clusterids* of every point in which points are identified using the index of the *clusterids*.

**Advantages:**

- DBSCAN has a time complexity of O($n^2$) and space complexity of O(n), where n is the number of data points.
- DBSCAN doesn't require to specify the number of clusters, 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.
- DBSCAN is resistant to noise.
- DBSCAN can handle clusters of different shapes and sizes.
- DBSCAN requires just 2 parameters and is mostly insensitive to the ordering of the points in the database.

**Disadvantages:**

- DBSCAN cannot handle varying densities because then choosing a meaningful distance measure (ε) can be difficult.

- DBSCAN cannot handle cluster data sets well with large differences in densities, since the minimum points – ε combination cannot then be chosen appropriately for all clusters.
- Due to the above 2 points, it is hard to determine the correct set of parameters, and thus it is highly sensitive to the parameters.
- DBSCAN 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 of DBSCAN depends on the distance measure used in the regionQuery function. Mostly Euclidean distance is used. But for high dimensional data, this measure is rendered almost useless due to curse of dimensionality.

## K-Means MapReduce:

The MapReduce model can be used to implement K-means in a parallelized manner. The benefits of this approach may be observed when executed on multiple nodes. The MapReduce model works as follows:

*Map Phase*: The input data is split into key value pairs, in case of K-means the mapper generates pairs where the key is label of the cluster and value is index of a point assigned to the cluster. The key value are tab separated in the output. An example is shown below:

| Cluster_id | Point_indices |
|---|---|
| **3** | 0 |
| **1** | 1 |
| **4** | 2 |
| **2** | 3 |
| **5** | 4 |
| **5** | 5 |
| **2** | 6 |
| **1** | 7 |
| **1** | 8 |

*Reduce Phase*: The reduce phase aggregates the point indices in each cluster and prints each cluster_id along with its list of point_indices. An example is shown below:

| Cluster_id | Point_indices |
|---|---|
| **1** | 1, 7, 8 |
| **2** | 3, 6, 9 |
| **3** | 0 |
| **4** | 2 |
| **5** | 4, 5 |

Our implementation of the MapReduce algorithm is explained in the below:

1. The main controller file “KMeans_hadoop_script.py” generates the input file “centroids_and_features.txt” which is a concatenation of the matrices: *initial_centroids* and *adjusted_matrix* (feature matrix).
2. The “centroids_and_features.txt” file is the input to Mapper.py which generates key-value pairs, where key is the centroid’s index and value is the index of the point which selects the centroid as the minimum distance centroid, along with features in the point. The output template is as follows:

   ```
   Centroid_index    \t    Point_index    \t    features<f1,f2,…>
   ```

3. This input is then fed into the reducer (Reducer.py) after sorting is done based on the centroid_index by Hadoop.
4. The reducer reads each line and continues to aggregate the list of point indices along with sum of point features (used along with an incrementing count for computing new centroid) until a new centroid_index is seen.
5. As soon as a new centroid_index is seen, the reducer prints the previous centroid_index as the key along with the list of point_indices and the new centroid as value. The output template is as follows:

   ```
   Centroid_index    \t    List_of_point_indices    \t    New_centroid
   ```

6. The new centroid is read by the controller file and it uses it to update the centroids stored in “centroids_and_features.txt”.
7. Step 2 to 6 are repeated until convergence is reached when the old and new centroids are equal.

The running time of K-Means in MapReduce model is 182 seconds. The duration can be attributed to execution of the program on single-node. The efficiency would improve multiple times based on the number of nodes on which the MapReduce runs.

The sorted input provided by Hadoop to the reducer is leveraged to achieve improved computational performance and optimized space complexity as the Reducer reads the input line by line and computes the aggregated list of point indices as well as the sum for the new centroid on the fly. In contrast to this, the non-parallel approach of K-Means required storing the points and clusters in matrices in order to compute the aggregates.

Also, access to the input file “centroids_and_features.txt” is speeded up by placing the file in the Hadoop distributed files system. This also makes it available for usage by mappers and reducers running across several nodes.

We assign the number of mappers and reducers to be equal to the number of required clusters in order to ensure a highly parallelized computation of the new centroids in each iteration my the mapper and reducer jobs.

## Visualizations:

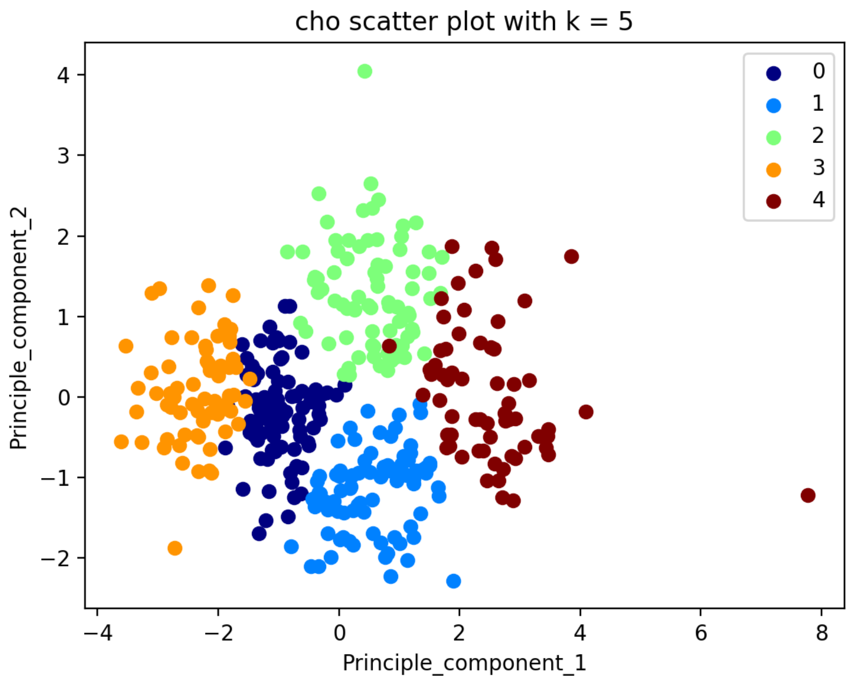


**Fig 2: K-Means: cho.txt scatter plot**

**Jaccard Coefficient: 0. 4305 (Best among 10 runs)**

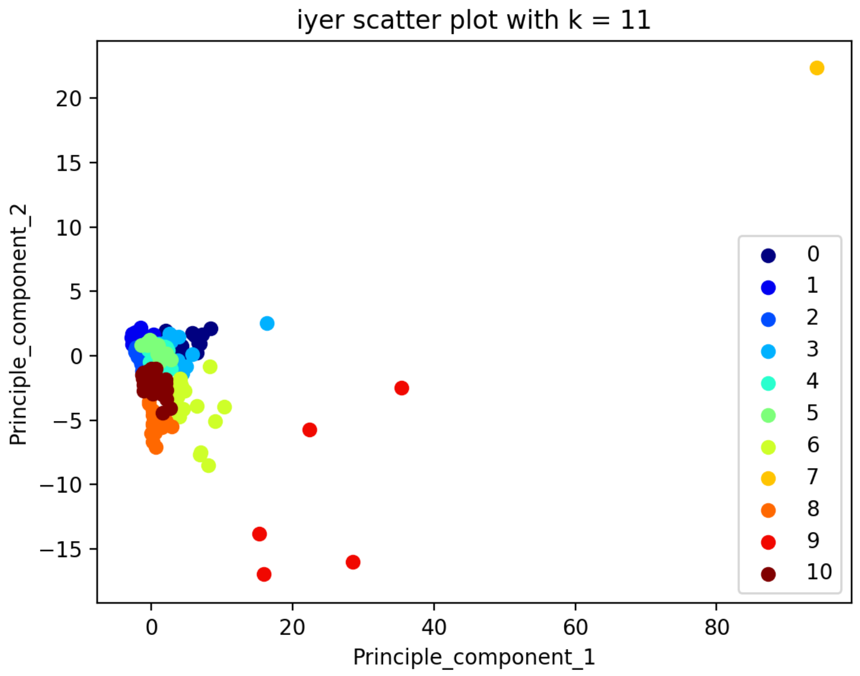


**Fig 3: K-Means: iyer.txt scatter plot**

**Jaccard Coefficient: 0.3652 (Best among 10 runs)**

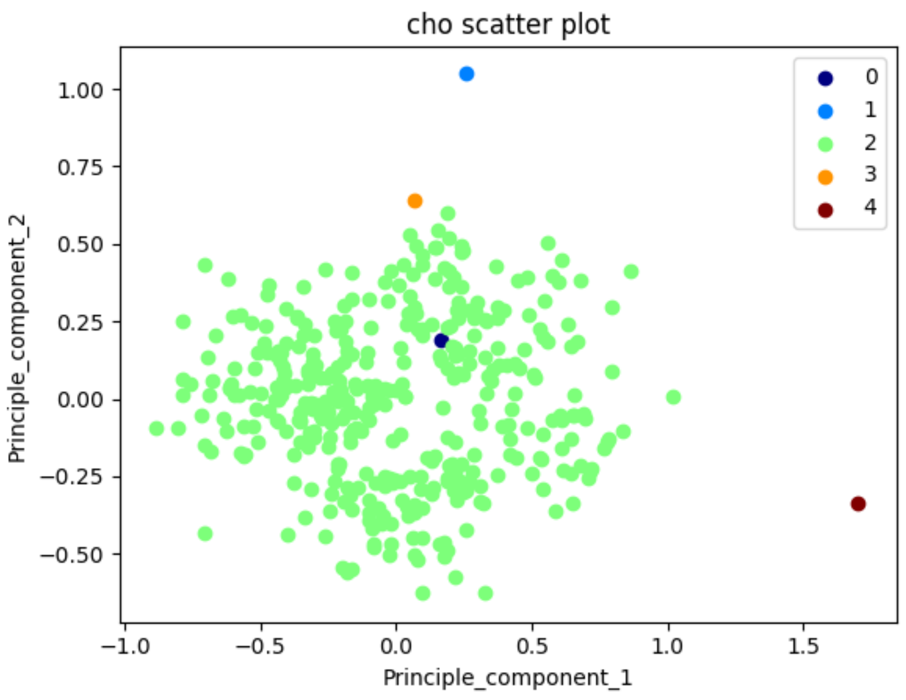


**Fig 4: Hierarchical Agglomerative Clustering: cho.txt scatter plot**

**Jaccard Coefficient: 0.2283**

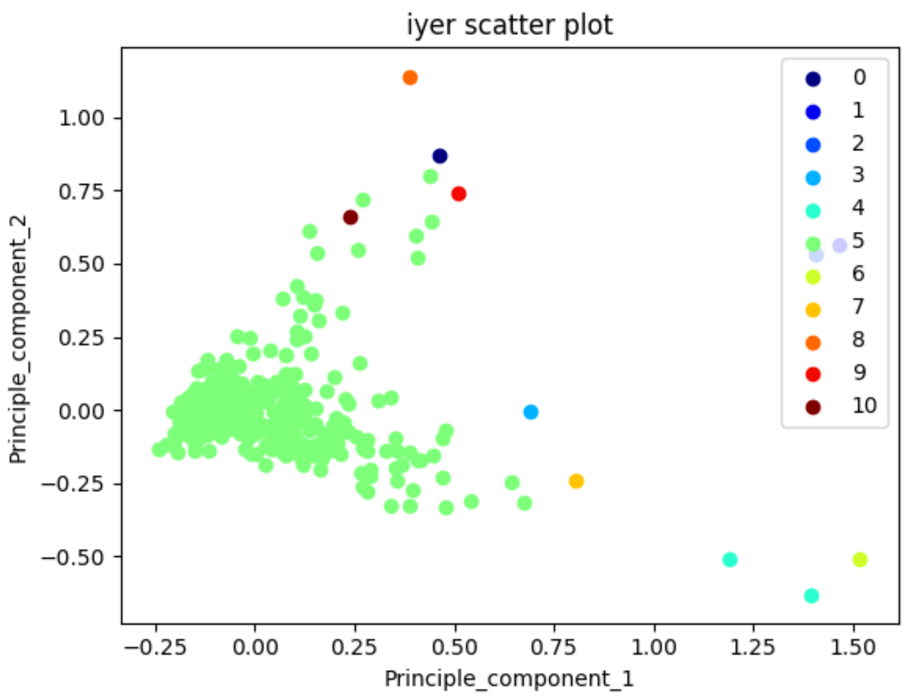


**Fig 5: Hierarchical Agglomerative Clustering: iyer.txt scatter plot**

**Jaccard Coefficient: 0.1544**

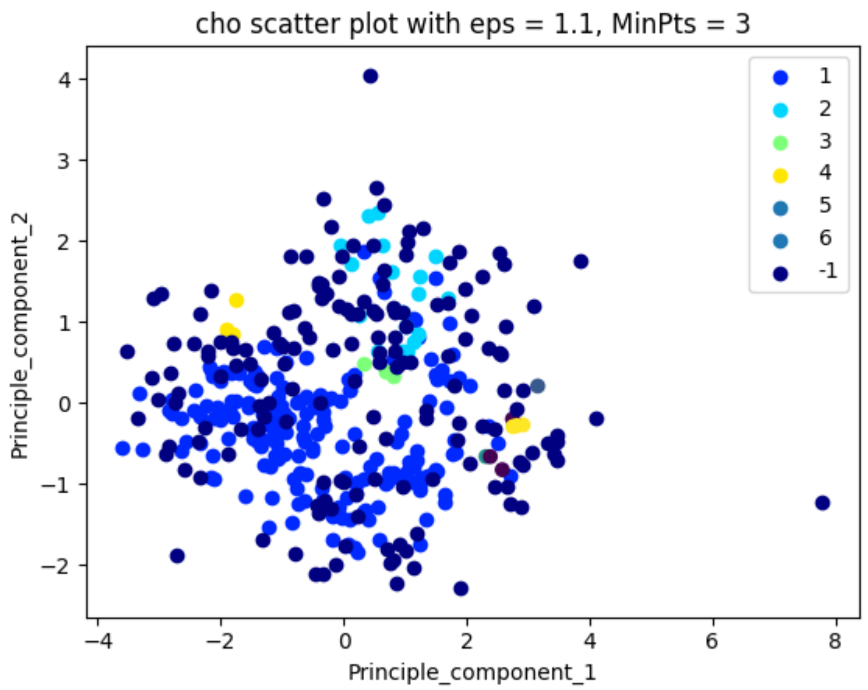


**Fig 6: DBSCAN: cho.txt scatter plot**

**Jaccard coefficient = 0.2037**

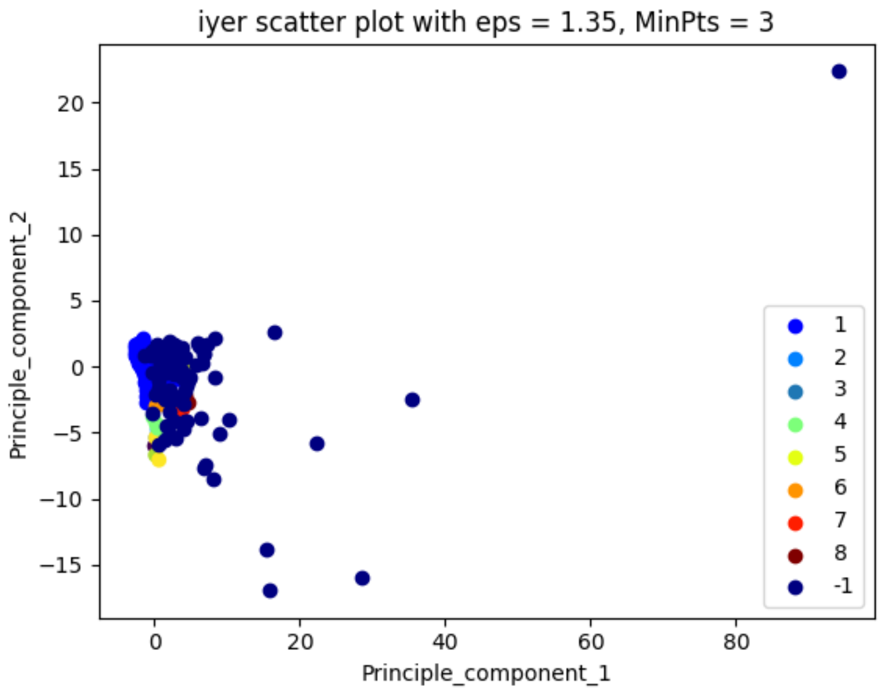


**Fig 7: DBSCAN: iyer.txt scatter plot**

**Jaccard coefficient: 0.2073**

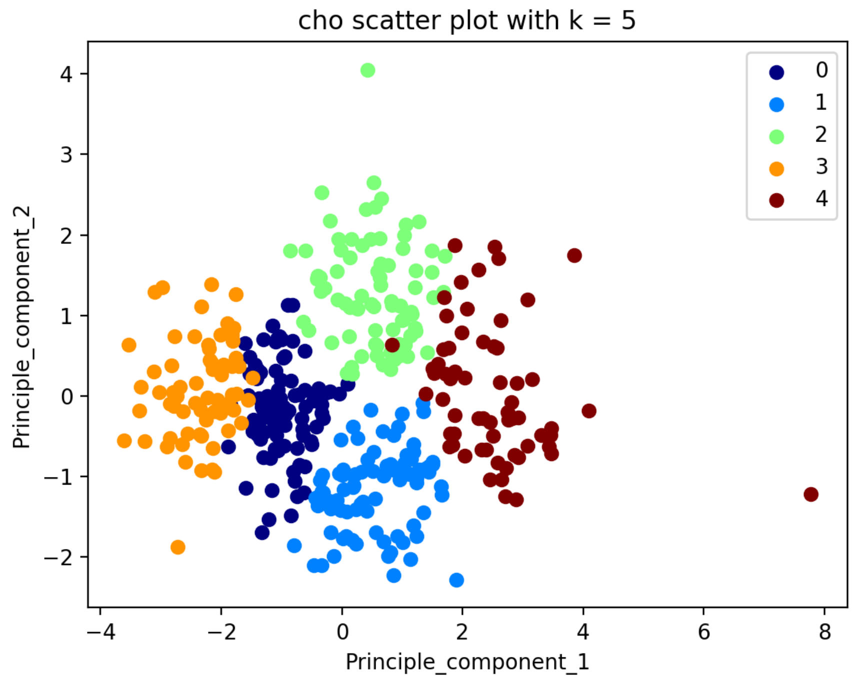


**Fig 8: K-Means MapReduce: cho.txt scatter plot**

**Jaccard Coefficient: 0. 4305 (Best among 10 runs)**

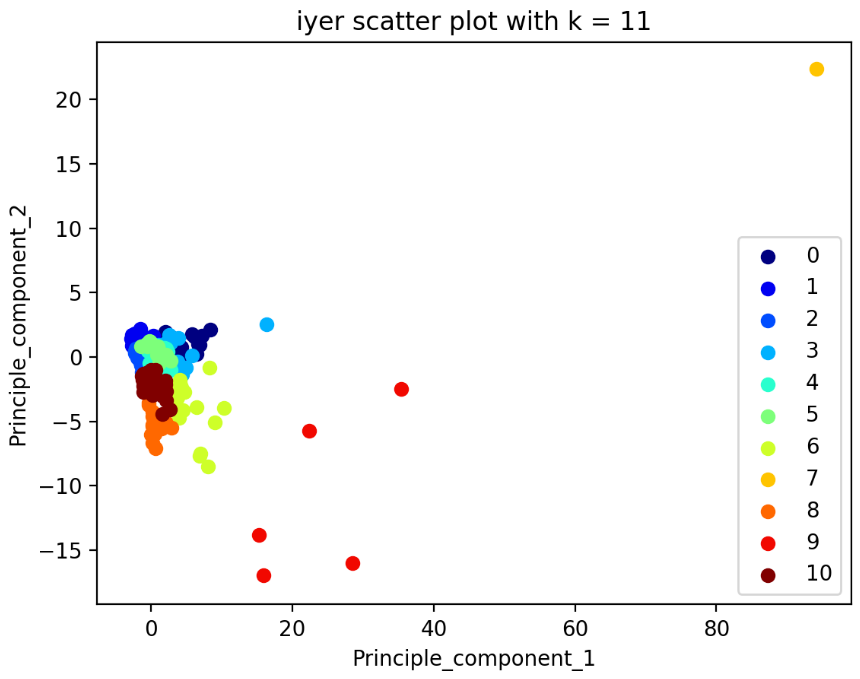


**Fig 9: K-Means MapReduce: iyer.txt scatter plot**

**Jaccard Coefficient: 0.3652 (Best among 10 runs)**

## Analysis of Visualizations and External Index:

To obtain the visualizations, we applied principal component analysis to the feature matrix. PCA reduces the feature matrix of ‘d’ dimensions to a matrix of 2 dimensions. These form the principle components which are then mapped to their corresponding cluster labels.

Some of the pros and cons of each clustering algorithms can be observed from the visualizations shown above. They are discussed as follows:

**K-Means plots:**

1. The dataset in “cho.txt” is very well clustered by K-Means clustering as seen from the high Jaccard value. A possible reason for this could be the globular structure of original clusters which are shown as circular clusters in the 2 dimensional plot (Fig 1).
2. Even in the reduced dimensional space, it is evident that the points are mapped to the cluster whose centroid is at minimum distance from them.
3. A closer look at the data shows that even the outliers are mapped into their closest clusters, which adheres to the fact that K-Means clustering is sensitive to outliers.
4. Another point to note is that K-Means attempts to form globular shaped clusters in the “iyer.txt” dataset although the distribution of points in the dataset hints that the points may not originally belong to globular clusters.

**Hierarchical Agglomerative Clustering plots:**

1. A vivid observation from the plot for dataset “iyer.txt” by HAC is that it is capable of clustering together points belonging to non-elliptical shaped distributions. This is a consequence of using MIN to determine inter-cluster distance.
2. It is also noted that a few points in the central region of the large cluster do not actually belong to the large cluster. This may be a consequence of dimensionality reduction as these points may in reality be at a large distance from the large cluster in dimensions which aren’t primarily represented after principle component analysis.
3. We also note that the plots are adversely influenced by outliers which cause the Jaccard of HAC to be lower than that of K-Means.

**DBSCAN plots:**

1. Among the three clustering algorithms, DBSCAN is  the only one which explicitly labels a set of points as noise based on their distance from the data distribution.
2. It is observed that the clusters formed are of various shapes and sizes, which is advantageous in many cases where the data doesn’t belong to any regularly shaped distributions.
3. In spite of these advantages, DBSCAN’s Jaccard index is much lower than that obtained from K-Means for both datasets. DBSCAN performs better than agglomerative clustering in the dataset “iyer.txt”, which is intuitive as the distribution of the data points is irregular for “iyer.txt”

**K-Means MapReduce plots:**

1. The plots obtained from the MapReduce model of K-Means exactly match those obtained from the non-parallel K-Means implementation and they also give the same value of Jaccard index.
2. The reason for this is that the methodology of clustering by K-Means remains unchanged even when the clustering is done in parallel.

**Jaccard Coefficients summarized:**

| | **cho.txt** | **iyer.txt** |
|---|---|---|
| **K-Means non-parallel** | 0. 4305 | 0.3652 |
| **Agglomerative** | 0.2283 | 0.1544 |
| **DBSCAN** | 0.2037 | 0.2073 |
| **K-Means MapReduce** | 0. 4305 | 0.3652 |