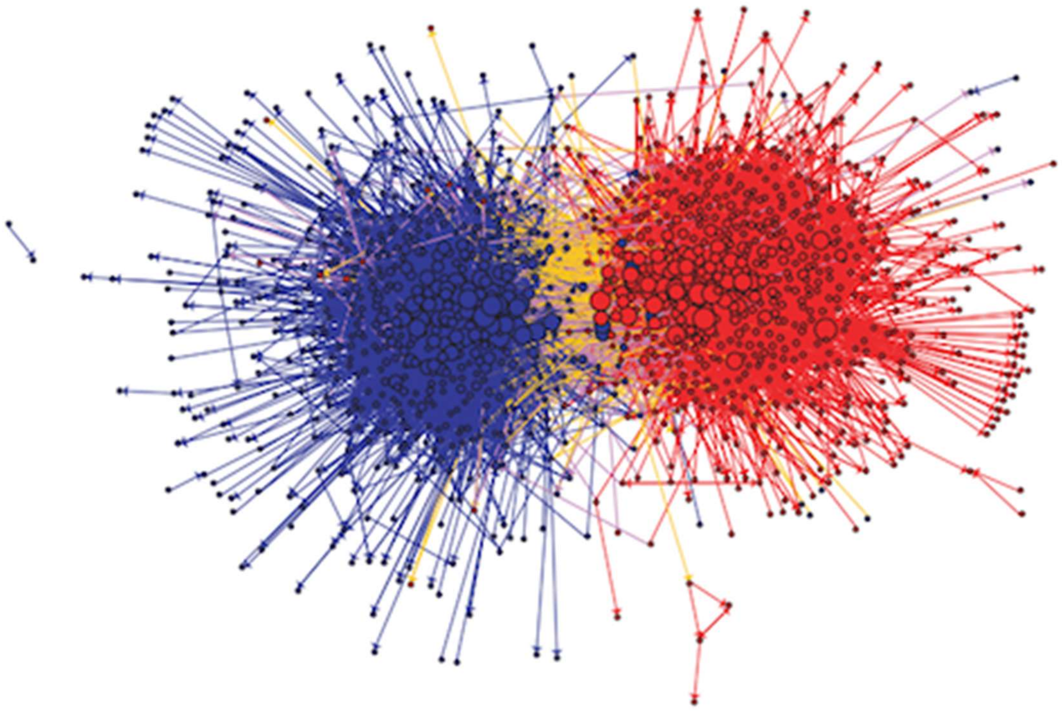
CSE 601 - Data Mining and Bioinformatics

Project 2 - Clustering

October 28, 2017



Group 34:

Amaan Akhtarali Modak	-	50206525	-	amaanakh
Arshdeep Gill	-	50139190	-	asgill3
Yaoyu Fu	-	50166363	-	yaoyufu

Introduction to Clustering Algorithms

The method of identifying similar groups of data in a data set is called clustering. Entities in each group are comparatively more similar to entities of that group than those of the other groups. In this project, we study what clustering is, different clustering algorithms and a comparison between some of the most commonly used cluster methods.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters.

Let us understand this with an example. Suppose, you are the head of a rental store and wish to understand preferences of your customers to scale up your business. Is it possible for you to look at details of each customer and devise a unique business strategy for each one of them? Definitely not. However, what you can do is to cluster all of your customers into say 10 groups based on their purchasing habits and use a separate strategy for customers in each of these 10 groups. This is what we call clustering.

Types of Clustering Algorithms

K-means

K-means is a prototype-based clustering technique. It compares the objects to the defined prototype, then cluster the objects based on the similarity with the prototype. The prototype of the cluster is call centroid.

Implementation of K-means:

- Import data
- Initialize K cluster centroids
- Repeat
- For each object, calculate the euclidean distance between each object and the initial K centroids. Assign each object to the closest centroid, and each collection of objects that assigned to the same centroid becomes a cluster.
- Calculate the new centroids based on the new clusters.
- Until centroids do not change

Implementation Details:

For k-means we first set the initial centroids (either randomly or by choice) and set the number of clusters that we have to find, that is k. Next we read the input file given to us (cho.txt and iyer.txt) and split the dataset such that column 0 is the gene_id, column 1 is the ground_truth_values and column 2 onwards will be the points_data. Now, we will perform two steps until we see that the centroids converge and no further iterations are possible. Firstly we assign each data point to the cluster of the closest centroid (of the ones chosen randomly or by the user) based on the Euclidean distance formula. Secondly, once all points have been assigned to one of the k clusters, we will now update the values of the centroids to denote the centroid of the updated clusters which include all the assigned points. This process will be iterated until the previous centroid and the new cluster centroid converge to be the same, or until the iteration have run for a pre-defined maximum value. Once the centroids have converged, we output the k clusters along with the data points that belong to these clusters.

Pros:

- It is easy to implement, and it is efficient

Cons:

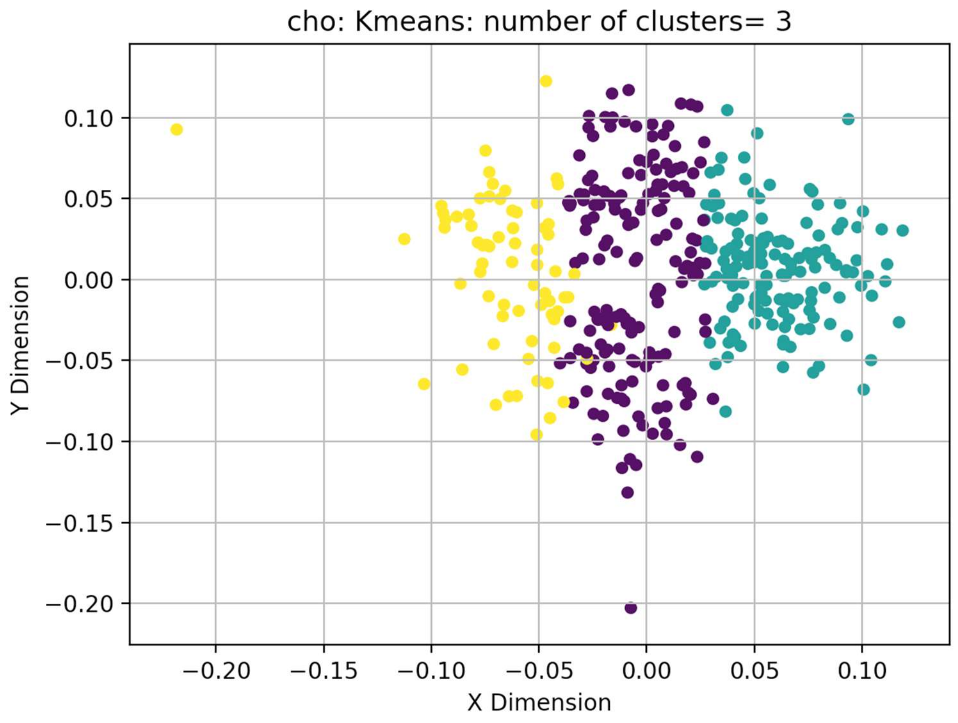
- It requires to specify the number of clusters K in advance
- Choosing the initial centroids is important to the result, randomly selected initial centroids might lead to a poor clustering result. If the initial centroids are close, it might be hard to separate them afterwards, and the clustering result will not be ideal.
- It is not suitable for all types of data. If the clusters have different sizes, density, or in irregular shapes, k-means will not be able to do the clustering very well.

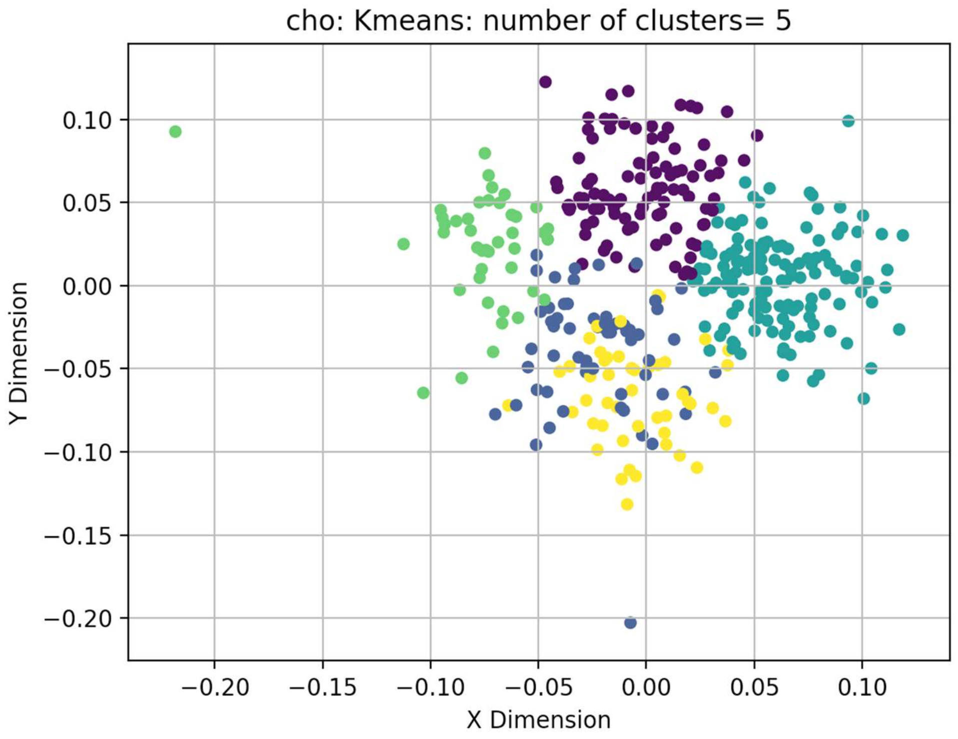
Based on our implementation of the k-means algorithm we get the following results as our output values (Jaccard Coefficient) for the *cho.txt* dataset

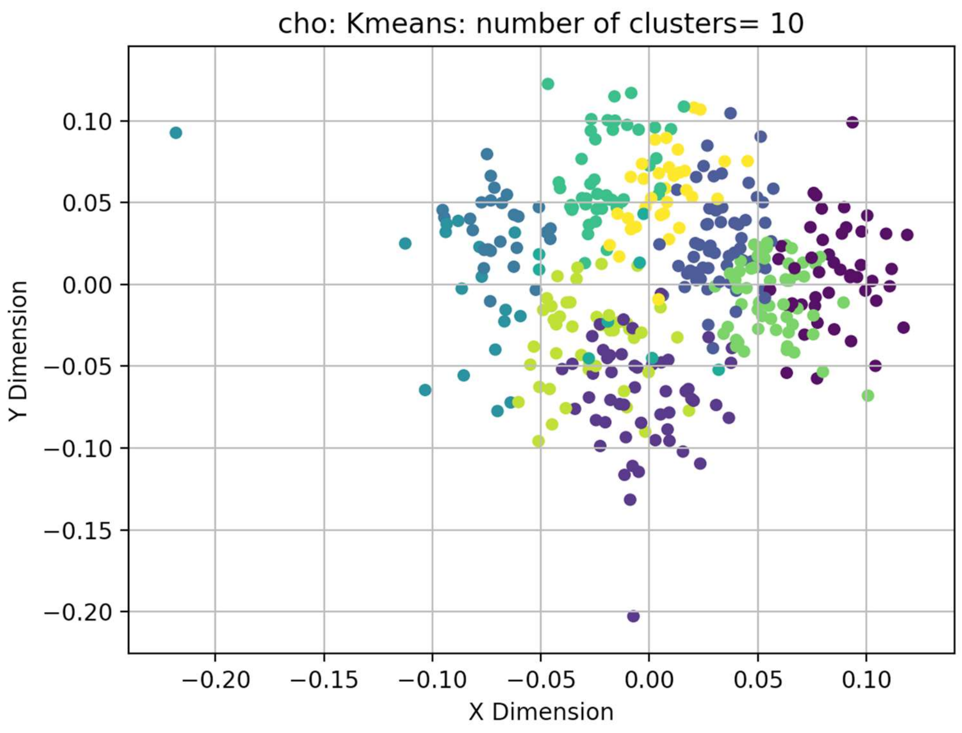
Number of Clusters	JC
3	0.4087635255
5	0.3307981613
10	0.2066982409
15	0.200082106

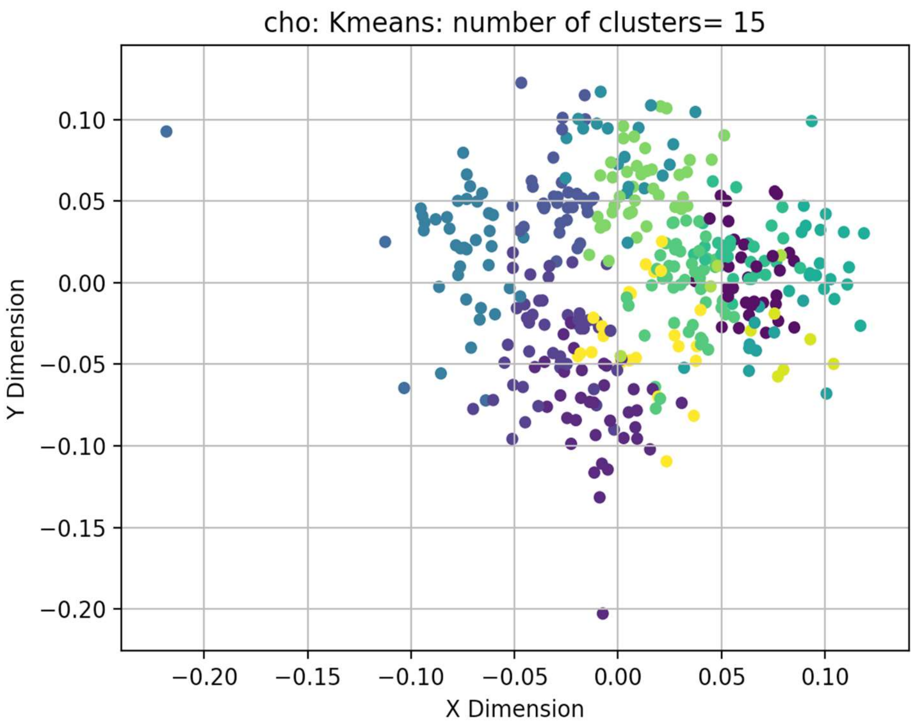
Given below are the scatter plot outputs for various k clusters as specified by the user for the two given datasets. From the visual representations we can make out that for the cho.txt dataset, when k=5 we get the cleanest and most distinct output which matches the results in the ground_truth.

Similarly for the iyer.txt dataset, we see that we get fairly distinct cluster distribution when the number of clusters is set to be 10.


cho: Kmeans: number of clusters= 3
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
Y Dimension
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10
X Dimension

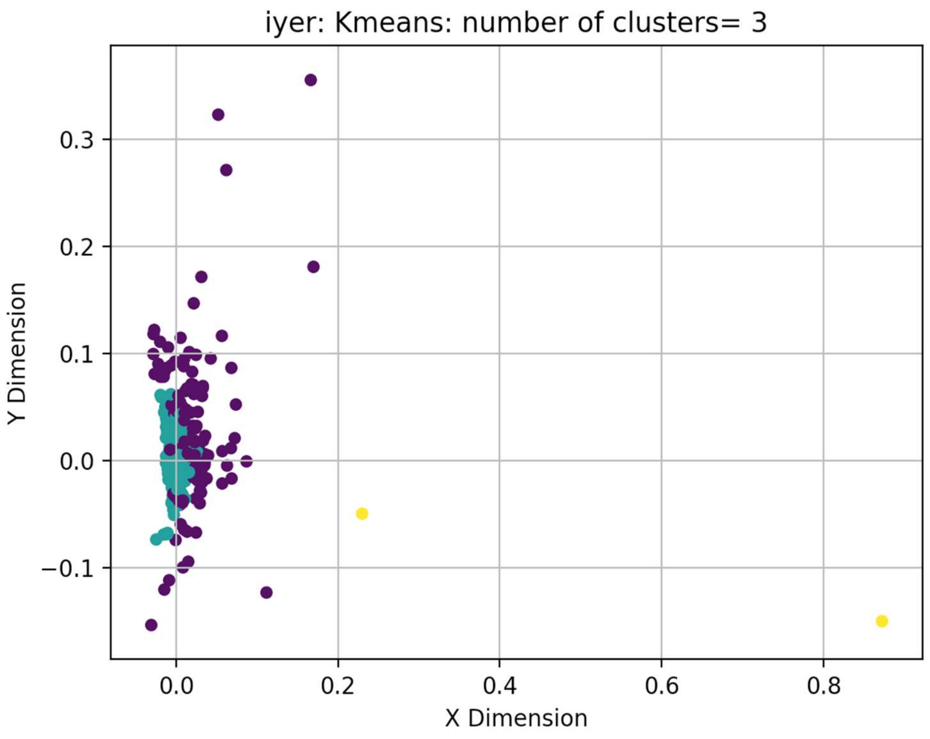

cho: Kmeans: number of clusters= 5
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
Y Dimension
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10
X Dimension

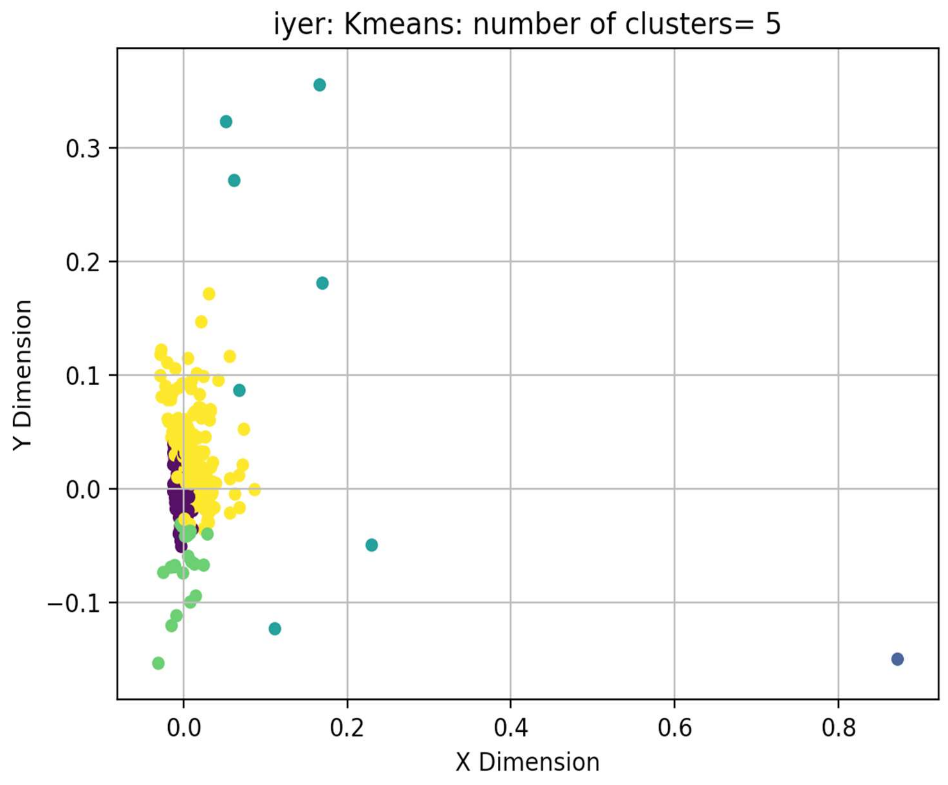

cho: Kmeans: number of clusters= 10
Y Dimension
X Dimension
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10


cho: Kmeans: number of clusters= 15
Y Dimension
X Dimension
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10

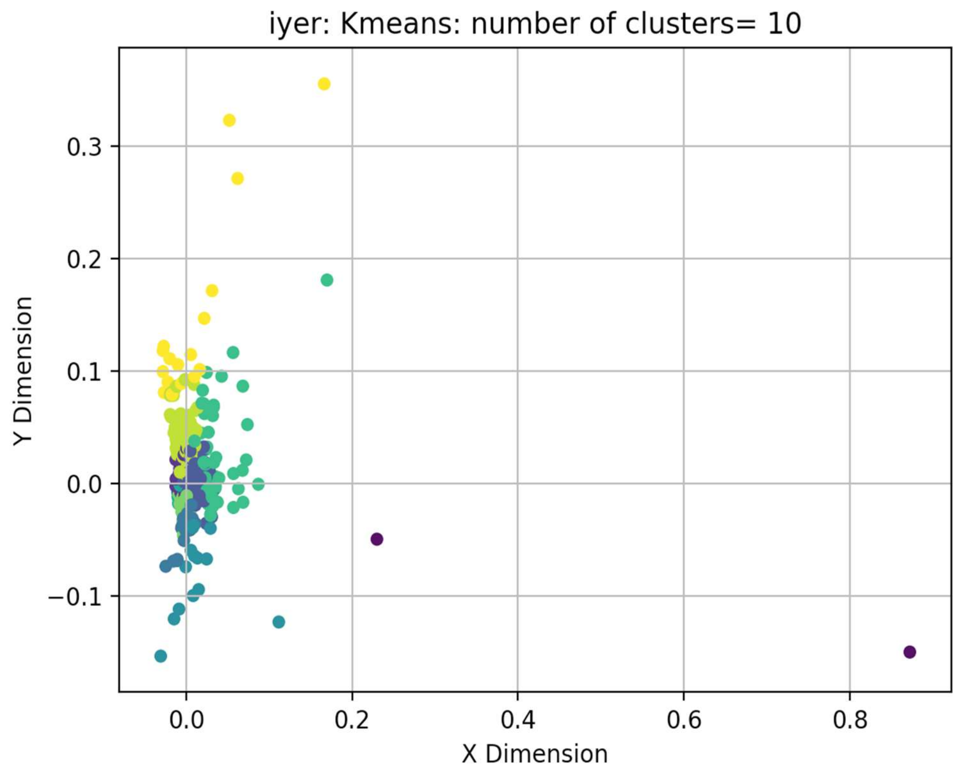
Similarly, these are the output values obtained for the *iyer.txt* dataset after implementing k-means using a range of cluster numbers:

Number of clusters	JC
3	0.2175820319
5	0.276156905
10	0.275085916
15	0.3543840629

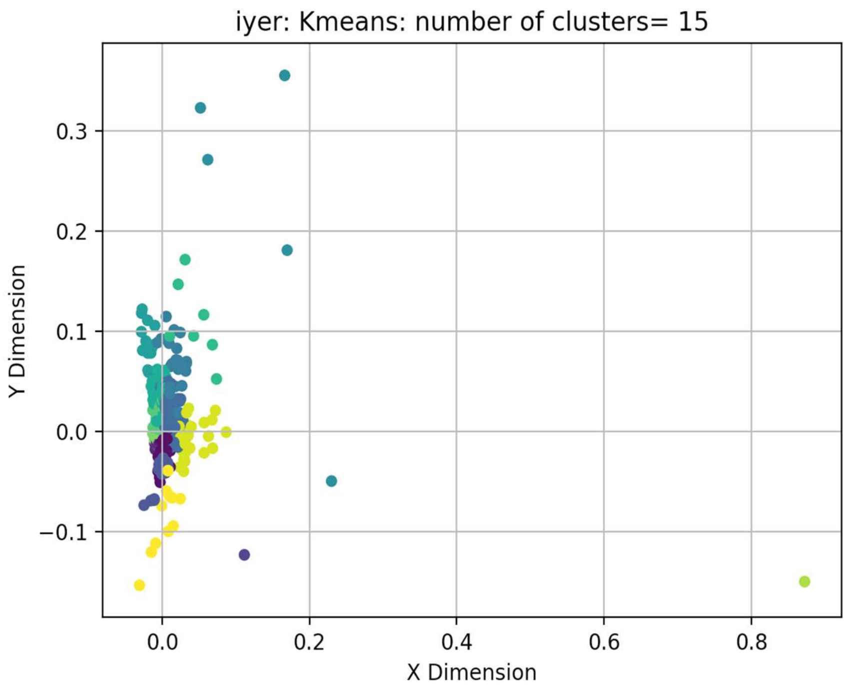




iyer: Kmeans: number of clusters= 5
Y Dimension
X Dimension
0.3
0.2
0.1
0.0
−0.1
0.0
0.2
0.4
0.6
0.8

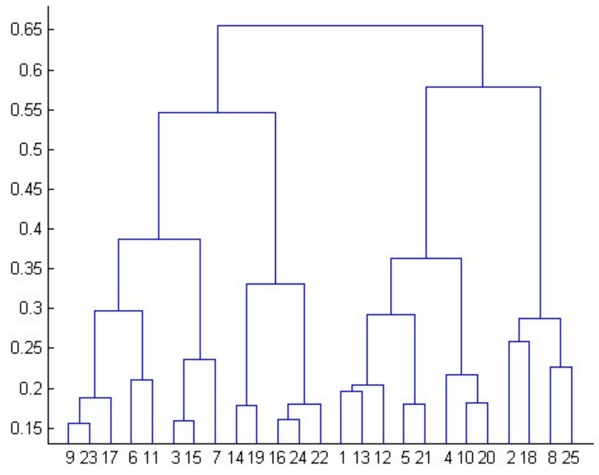


iyer: Kmeans: number of clusters= 10
Y Dimension
X Dimension
0.3
0.2
0.1
0.0
−0.1
0.0
0.2
0.4
0.6
0.8



Hierarchical Agglomerative clustering with Single Link (Min)

Hierarchical clustering, as the name suggests is an algorithm that builds hierarchy of clusters. This algorithm starts with all the data points assigned to a cluster of their own. Then two nearest clusters are merged into the same cluster. In the end, this algorithm terminates when there is only a single cluster left. The results of hierarchical clustering can be shown using dendrogram. The dendrogram can be interpreted as:

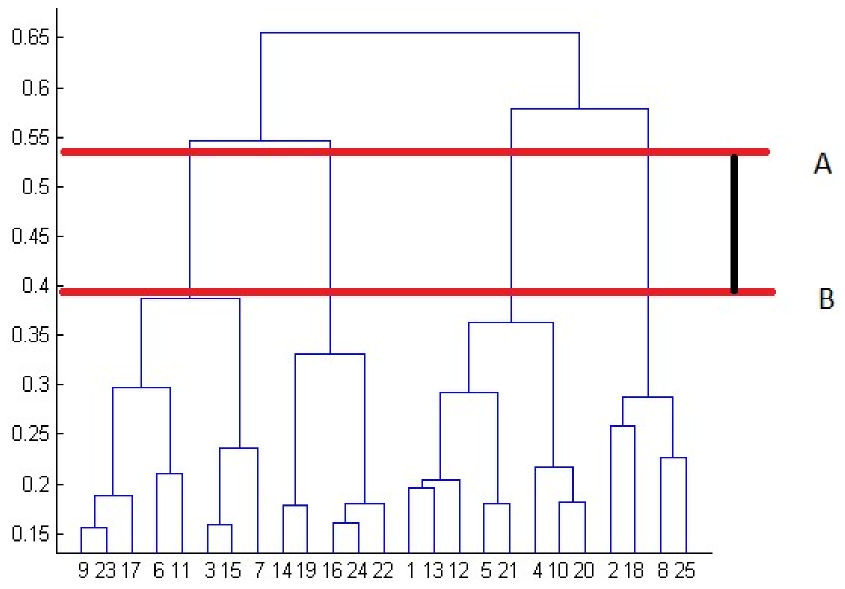


At the bottom, we start with 25 data points, each assigned to separate clusters. Two closest clusters are then merged till we have just one cluster at the top. The height in the dendrogram at which two clusters are merged represents the distance between two clusters in the data space.

The decision of the no. of clusters that can best depict different groups can be chosen by observing the dendrogram. The best choice of the no. of clusters is the no. of vertical lines in the dendrogram cut by a horizontal line that can transverse the maximum distance vertically without intersecting a cluster.

In the above example, the best choice of no. of clusters will be 4 as the red horizontal line in the dendrogram below covers maximum vertical distance AB.

Hierarchical agglomerative approach starts with consider each object as a single cluster. At each step, merge two closest clusters into a new cluster until all the objects are merged into a single cluster. In the Single Link technique, the distance between two clusters is defined as the minimum distance between any two objects belonging to different clusters.



Implementation Details:
In hierarchical agglomerative cluster. We first set each data point in the dataset provided as an individual cluster. Then we calculate the minimum distance between all the data points. Then the two centroids that are found to be closest to be each other are merged together. In the next step the distance between this merged centroid and the rest of the points are calculated along with the distance between all the points with the

dataset. Again, the points that are closest to each other are merged. This process is repeated until all the clusters are merged into one. However, this is not the ideal result. We want to find a suitable number of clusters into which the points are assigned. To do this, we specify the number of clusters required, and this is the part where the cut in the dendogram is made. We can select any point at which to make the cut and accordingly obtain the optimum result. In our implementation, we have made cuts at level 5 and 10 in both our datasets to find the optimum results.

Pros:

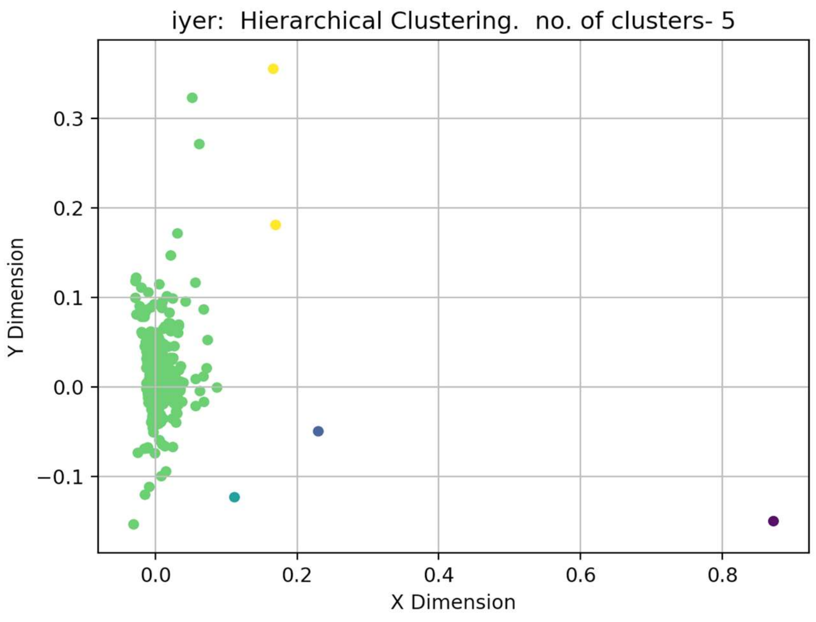
- It can handle non-elliptical shapes
- Compare to k-means, it doesn't require any assumption of number of clusters.

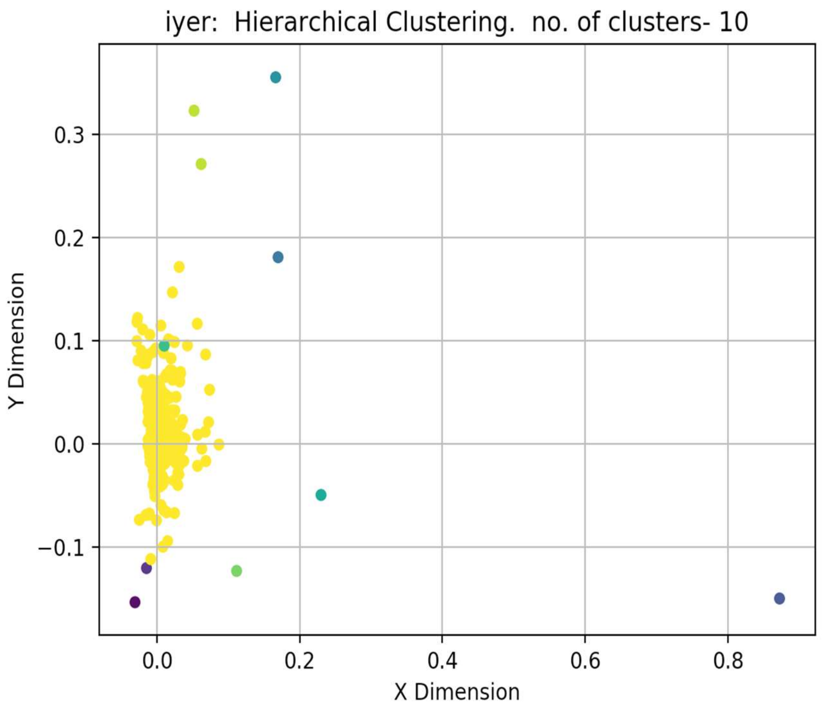
Cons:

- It is sensitive to noise and outliers. If there are outliers, Single Link technique will include the outliers while doing the distance calculation.
- Once two clusters are combined, it cannot be undone. Which means if an object is assigned to the wrong cluster, it cannot be undone.
- It is difficult to handle clusters with different sizes and irregular shapes.

Based on our implementation of the hierarchical clustering algorithm we have found the following values of the Jaccard Coefficient for the dataset *iyer.txt*

Number of Clusters	JC
5	0.173998
10	0.207371



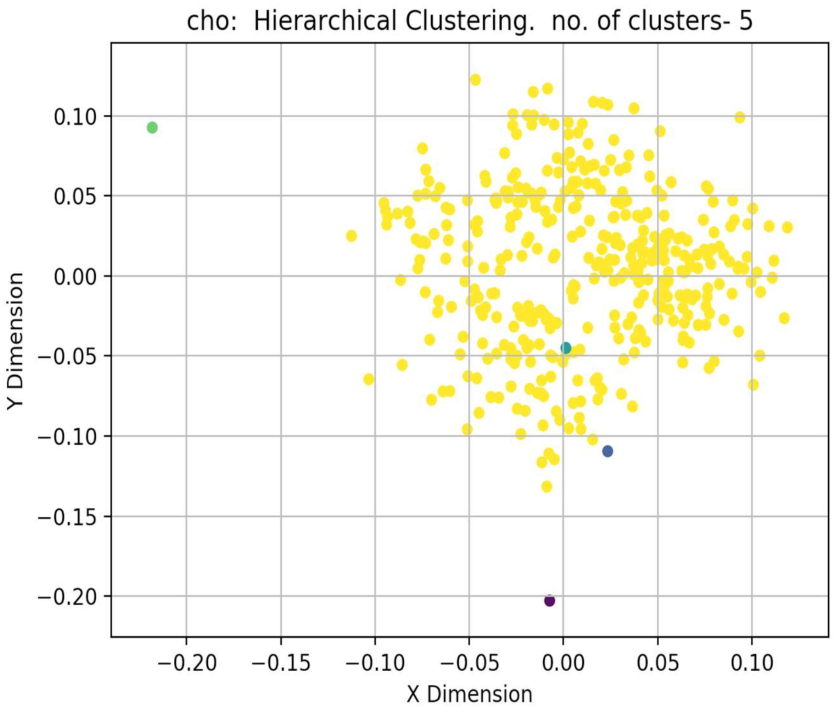


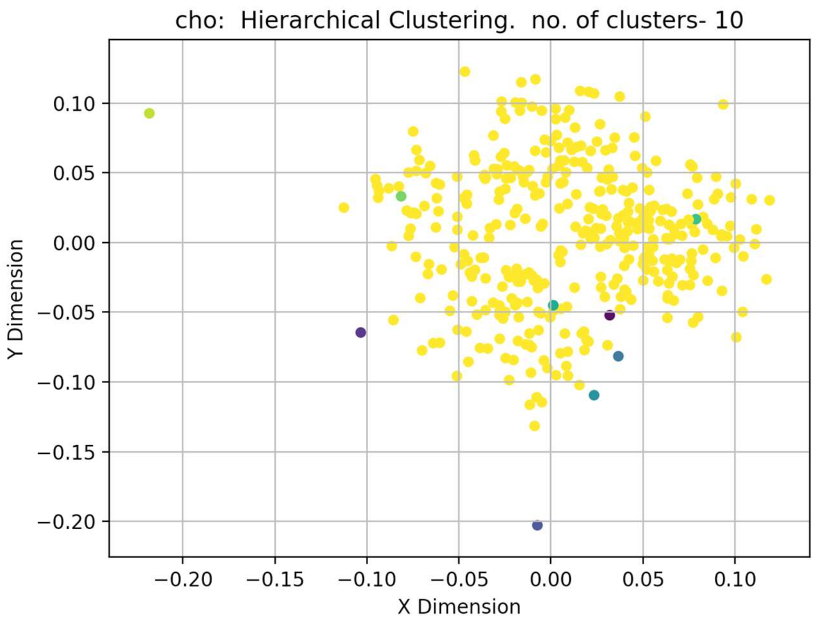
For the iyer.txt dataset, according to the scatter plots we see that we get a better clustering result when we set the number of clusters, that is the level at which to cut the dendogram as 10.

Similarly, for the cho.txt dataset, according to the scatter plots we see that we get a better clustering result when we set the number of clusters, that is the level at which to cut the dendogram as 5.

Based on our implementation of the hierarchical clustering algorithm we have found the following values of the Jaccard Coefficient for the dataset *cho.txt*

Number of Clusters	JC
5	0.227785
10	0.207371


cho: Hierarchical Clustering. no. of clusters- 5
Y Dimension
X Dimension
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10

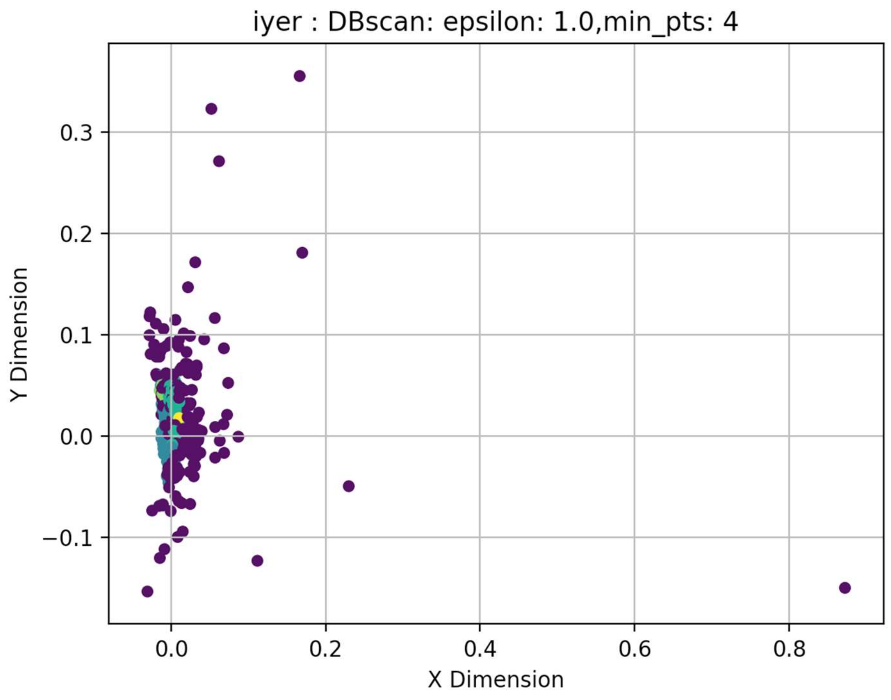

cho: Hierarchical Clustering. no. of clusters- 10
Y Dimension
X Dimension
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10

Density-based Clustering (DBSCAN)

Density-based clustering is a technique that separate regions with high density by regions of low density. It includes core points, border points and noise points. A point is a core point if the number of points within a specified radius (*Eps*) of this point, exceeds a certain threshold, *MinPts*. A border point has fewer points than *MinPts* within Eps, but falls within the neighborhood of a core point. A noise point is neither a core nor a border point. The DBSCAN algorithm put any two points that are within the distance *Eps* of on another in the same cluster.

Based on our implementation of the DBscan clustering algorithm we have found the following values of the Jaccard Coefficient for the dataset *iyer.txt*

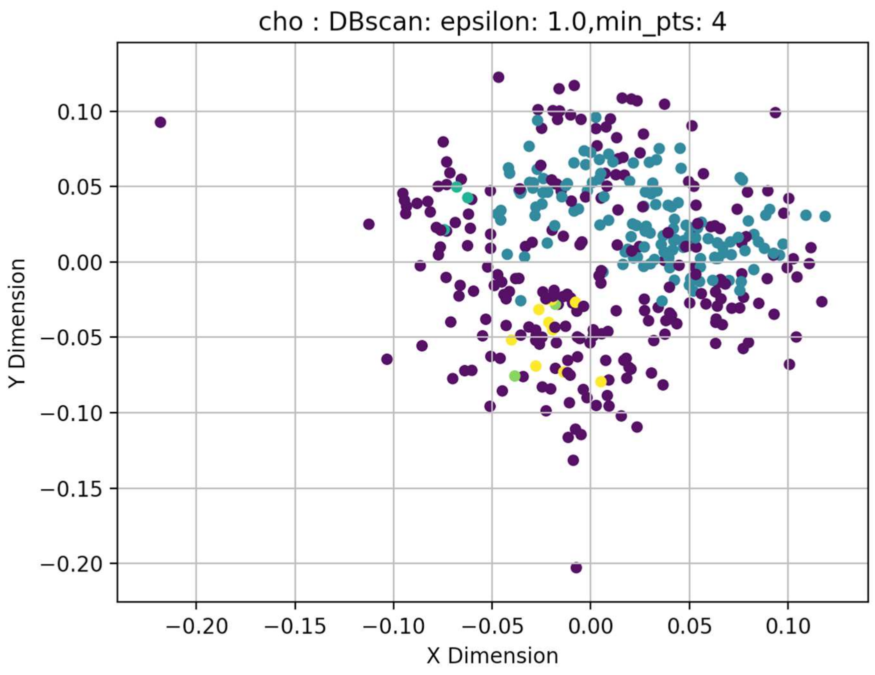
radius (*Eps*)	*MinPts*	JC
1	4	0.2788545246277205



From the scatter plot graphs, we see that when we set radius as 1 and take the minimum threshold as 4, we get the ideal plots and optimum cluster assignments.

Based on our implementation of the DBscan clustering algorithm we have found the following values of the Jaccard Coefficient for the dataset *cho.txt*

radius (Eps)	$MinPts$	JC
1	4	0.20110754881958612



Implementation Details:
DBscan is algorithm that takes 2 parameters epsilon which is the radius and minPoint which is the minimum points you want in that radius to form a cluster, any less are regarded as outliers. Initially we start by assigning radius and keep looking for neighbor points in that radius. We use Euclidean distance formula to find any distance and if it satisfies the given epsilon, it gets added to form 1 cluster. In the end of the algorithm we assign each point to cluster if it satisfies epsilon requirement otherwise is an outlier.

Pros:
- It can handle clusters with different size and irregular shapes
- It is not sensitive to noise
- It does not require assumption of number of clusters

Cons:
- It has trouble when clusters with large differences in densities, it might consider a whole cluster as noise.
- If the data are high-dimensional, it will be difficult to define density

MapReduce in Hadoop

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. A Hadoop frame-worked application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

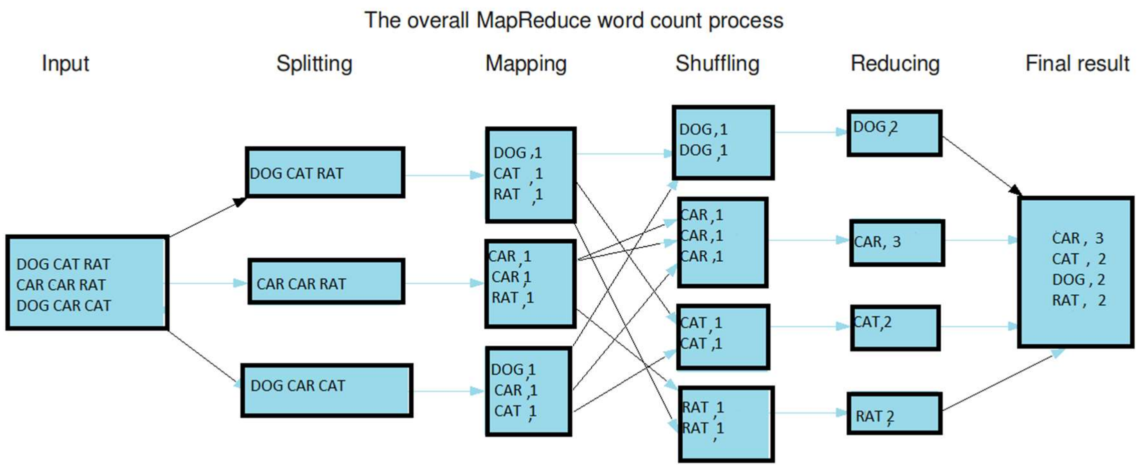
Hadoop can work directly with any mountable distributed file system such as Local FS, HFTP FS, S3 FS, and others, but the most common file system used by Hadoop is the Hadoop Distributed File System (HDFS). HDFS uses a master/slave architecture where master consists of a single **NameNode** that manages the file system metadata and one or more slave **DataNodes** that store the actual data.

A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of DataNodes. The NameNode determines the mapping of blocks to the DataNodes. The DataNodes takes care of read and write operation with the file system. They also take care of block creation, deletion and replication based on instruction given by NameNode.

MapReduce is a programming paradigm that allows for massive scalability across hundreds or thousands of servers in a Hadoop cluster.
The term MapReduce actually refers to two separate and distinct tasks that Hadoop programs perform. The first is the map job, which takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce job is always performed after the map job.

MapReduce Algorithm

- Driver:
  - Runs multiple iteration jobs using mapper+combiner+reducer.
- Mapper:
  - Configure: A single file containing cluster centers
  - Input: Input data points
  - Output: (cluster id, data)
- Reducer:
  - Input: (cluster id, data)
  - Output: (cluster id, cluster centroid)
- Combiner:
  - Input: (cluster id, data)
  - Output: (cluster id, (partial sum, number of points))



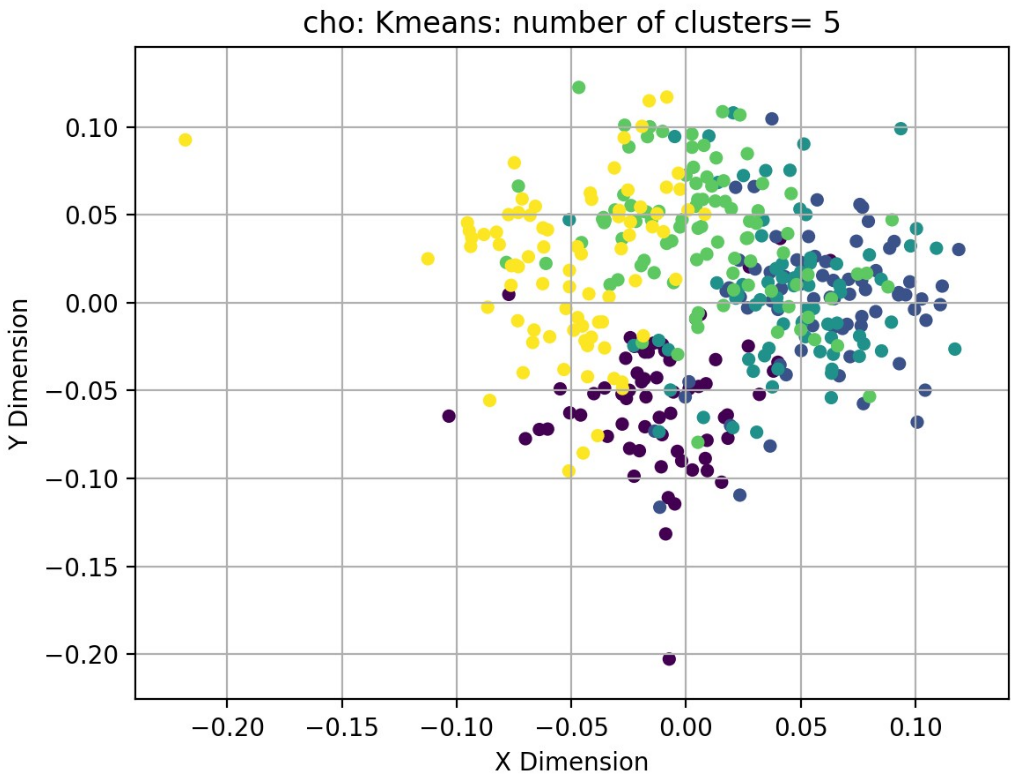
Implementation Details:

In this, we have three files that hold the code for MapReduce K-means: hadoop.py, mapper.py and reducer.py. The Hadoop.py is the wrapper code to execute the commands for Hadoop, and also to interact with the user to set number of clusters, initial centroids and where the cluster updations take place. In the mapper.py file, the feature matrix consisting of the centroid values and the data points are taken and then it is processed in to key-value pairs and given to the reducer. The reducer.py file will then take the output of the mapper as its input and then assign the points to the closest cluster centroid. This process will keep on iterating until the centroids converge similar to the normal K-means algorithm. The difference here being that the number of mappers and reducers that are set up run parallel to each other and not sequentially. This is especially invaluable in the case of large datasets, however since the datasets given to us are considerably small, this process takes longer to execute than the regular K-means algorithm.

Given below is the scatterplot representation of the cho.txt dataset with number of clusters set to 5 and centroids assigned randomly.

The Jaccard Coefficient for these parameters was found to be **0.5262.**

Based on the virtual representation we see fairly distinct scatterplots and having a high enough value for the JC compared to the normal K-means however the difference was due to the assignment of random centroids on both occasions.



Pros:

- Range of data sources
- Cost-effective for large datasets
- High speeds for big data applications
- Multiple copies can be stored

Cons:

- Not very efficient for small data sources
- Could have security concerns
- Because of the better decoupling of computation and storage in the GFS+Map-Reduce model - tolerating hot spots (resulting from MR jobs) is much easier
- The key-value stores are rarely arranged to have schemas optimized for analytics

Conclusion

The accuracy of a given clustering algorithm can be measured using an external index such as the Jaccard coefficient or a RAND index. In our implementations we have primarily focused on the Jaccard coefficient (JC) values.

We know that the higher the JC values the higher the efficiency of the clustering algorithm, since it measures the similarity between the ground truth and the clusters obtained.

Based on the findings of our implementations of the various algorithms on the given datasets *cho.txt* and *iyer.txt,* we are able to conclude that in terms of efficiency alone we can say that k-means works best for our given datasets. But before choosing any one algorithm as the best option, there are more aspects other than efficiency alone to consider, such as the input parameters that need to be given, the amount of information that we have regarding the data before implementation, and so on.

Between normal k-means algorithm and MapReduce k-means, the difference is that the task is split between the mappers and the reducers to speed up the processing time. However, since our dataset is considerably small, the process takes longer to execute in Hadoop than it does to execute serially on the local system. If we had a larger given dataset, the difference in the processing times between the k-means and MapReduce k-means would be very large and in that situation, the ideal choice of approach would be MapReduce k-means.

Citations

- Data mining and Bioinformatics Lectures(CSE-601 UB)
- https://sites.google.com/site/dataclusteringalgorithms/k-means-clustering-algorithm
- https://www.ibm.com/analytics/us/en/technology/hadoop/mapreduce/