CSE 601 Data Mining and Bioinformatics

Project 2: Clustering Algorithms

Submitted By
Mayur Tale
UB IT Name: mayurvin
Person # 50169256

K-Means clustering

It is a partitioned method for clustering ***n*** data points into ***k*** (predetermined) number of sets.

Our understanding of working of K-means algorithm:

Load data from source files (iyer.txt, cho.txt)

- Seek the number of clusters (k) from the user.
- Randomly select k genes from the entire dataset and mark them as centroids for k clusters.
- Compute distance matrix for all ***n*** genes
- While (*non-visited* genes still remain)
    - assign point to nearest cluster (min distance from k centroids)
    - *is_visited* = true
    - update centroid for k clusters
- Repeat until the centroids stop changing

Characteristics of K-means:

> **Center Based clustering with single level partitioning** unlike hierarchical agglomerative clustering where a cluster can be broken down into constituent clusters or smaller size.

> **Biggest disadvantage** of this algorithm is – **predetermination of *k***. We need to be aware of how many clusters we are to divide data into. Need to carry out repeated trials and visualization to get best quality clustering. Ground truth in the given data helps us pin point to k = 5. Although, magnitudes of Jaccard coefficient and Silhouette indicate the qualitative purity of out clustering results achieved by this implementation, it is very difficult to find the optimal solution for k-means problem even for 2 clusters.

> **Very sensitive of outliers**: Addition of even one single distant point (outlier) moves the nearest populated cluster's centroid by some extent, thereby impacting the correctness of clustering.

This can be overcome by:

o Eliminating outliers in the preprocessing step

o Eliminating small clusters (like clusters of size 1) and merging *similar* clusters

o Variants k-medoids & k-modes for continuous and categorical data respectively

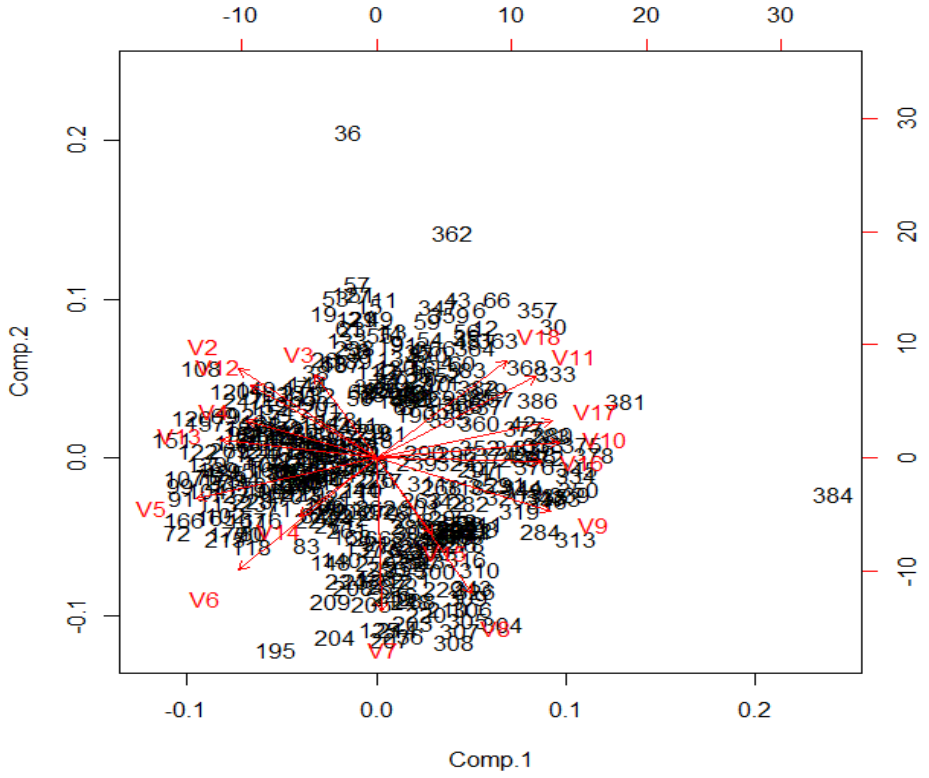
> **May yield different end results every time:** Since the selection of initial K centroids is performed on a random basis, the end result of the algorithm is highly unlikely to be same every time yielding different results for the same clustering parameters each time we run the algorithm. To get a standard result, we might need to run the algorithm with same parameters multiple times and take aggregate.

1: Initialize the list of clusters to contain the cluster containing all points.
2: **repeat**
3: Select a cluster from the list of clusters
4: **for** $i = 1$ to *number_of_iterations* **do**
5: Bisect the selected cluster using basic K-means
6: **end for**
7: Add the two clusters from the bisection with the lowest SSE to the list of clusters.
8: **until** Until the list of clusters contains K clusters

> K-means algorithm **predominantly yields spherical clusters** – not suitable for clusters of irregular shapes

> **Hard clustering** – every gene can be a part of one and only one cluster – no gene is left "un-clustered" as noise and no gene can be a part of multiple clusters.

> At any stage in the algorithm, the cluster centroids can be represented by either genes themselves or just a mathematical mean of all the genes in the cluster. In other words, the centroids may or may not be real gene data.

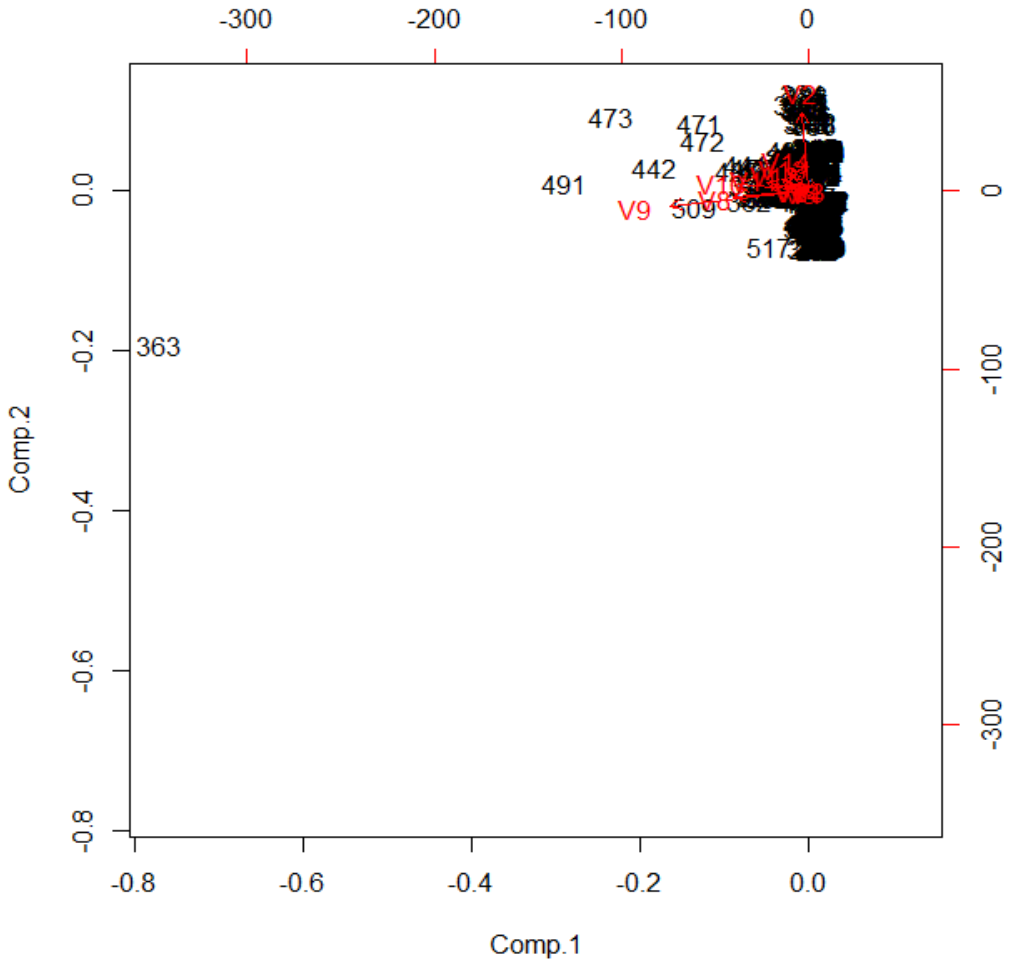


Cho.txt

Settings and Results:

K = 5

```
Jaccard value is: 0.21322962416602406
Silhoutte Index is: 0.23395311709279962
```



Iyer.txt

Settings and Results:

K = 10

```
Jaccard value is: 0.225888071378633
Silhoutte Index is: 0.023464433581429026
```

2. Hierarchical Clustering – Agglomerative approach

Description:

In the agglomerative approach, we start with each gene as a distinct cluster, and merge the nearest two clusters as we go. We repeat the merging until we are left with ***k*** clusters in the system, k is configurable by the user.

Pseudocode:

```
compute the distance matrix
let each data point be a cluster
repeat
        Merge the two closest clusters
        Update the distance matrix
until only a single cluster remains
```

I decide to terminate the algorithm when there are ***k*** clusters remaining in the system. The merging chronology is depicted by implementing HashMap of ArrayList of ArrayList of GeneIds.

Time complexity:

Time complexity of single-link clustering is O (n^2). If dn is the distance of the two clusters merged in step n, and G (n) is the graph that links all data points with a distance of at most dn, then the clusters after step n are the connected components of G (n).

o We first compute all distances in O(n^2). While doing this we find the smallest distance for each data point and keep them in a next-best-merge array. In each of the n-1 merging steps we find the smallest distance in the next-best-merge array. We merge the two identified clusters, and update the distance matrix in O(n).

o Finally, we update the next-best-merge array in O (n) in each step. We can do the latter in O (n) because if the best merge partner for ***k*** before merging ***i*** and ***j*** was either ***i*** or ***j***, then after merging ***i*** and ***j*** the best merge partner for ***k*** is the merger of ***i*** and ***j***.

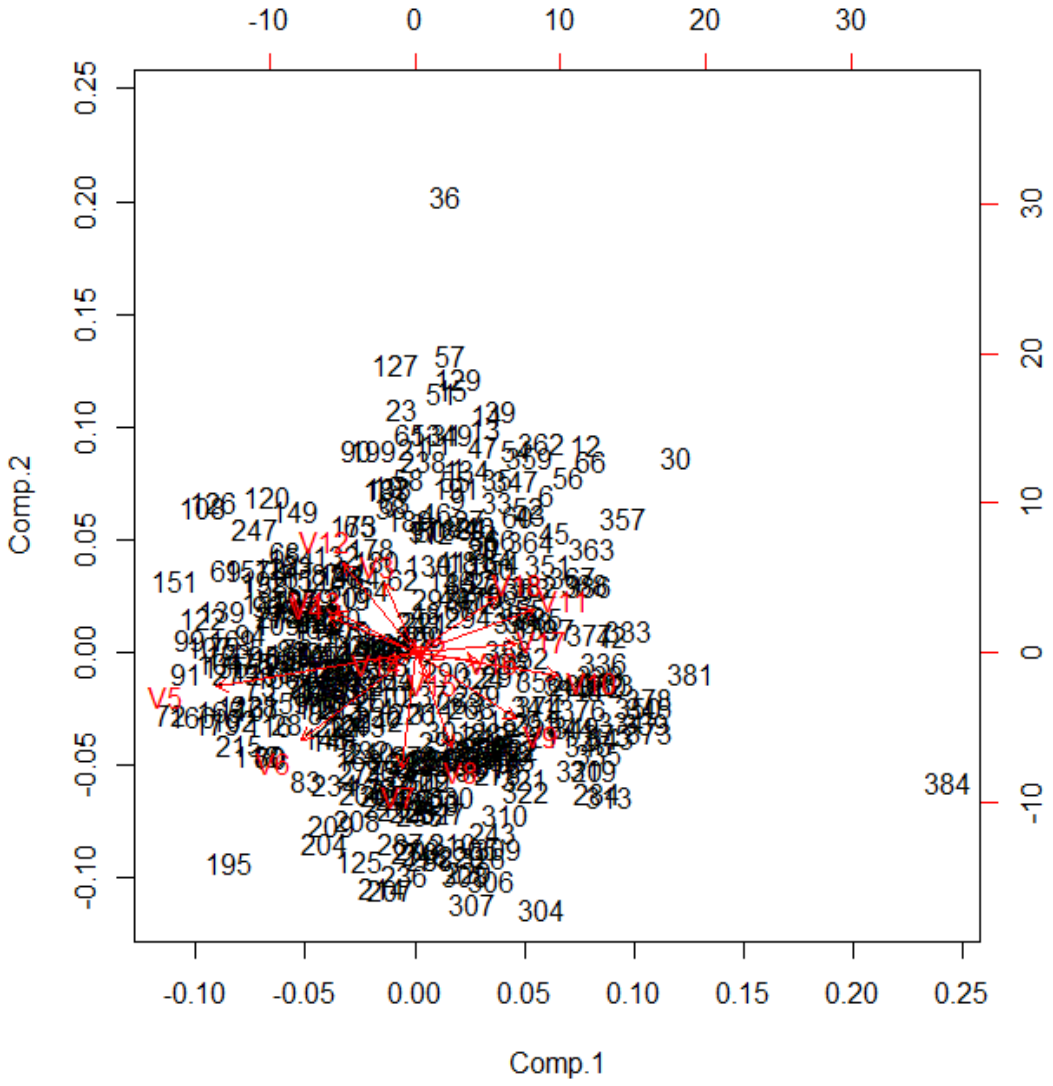
Advantages:

> Can handle non global shapes

Disadvantages:

> Sensitive to noise and outliers

> Suffers from chaining effects – It tends to produce long thin clusters in which nearby elements of the same cluster have small distances, but elements at opposite ends of a cluster may be much farther from each other than to elements of other clusters

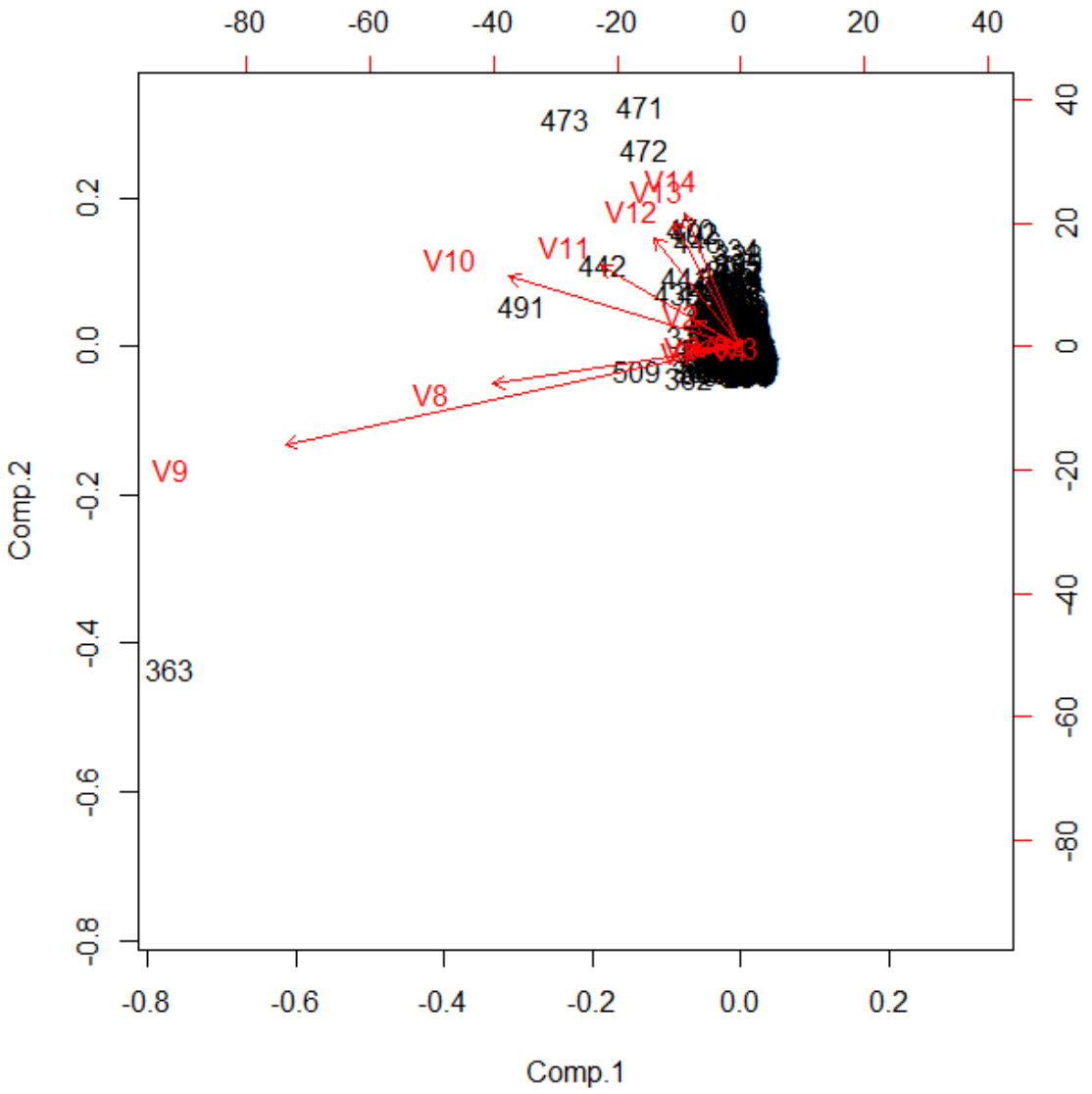


Cho.txt

Settings and Results:

K = 5

```
Jaccard value is: 0.2263593865332568
Silhoutte Index is: 0.10672843576920354
```



Iyer.txt

Settings and Results:

K = 10

```
Jaccard value is: 0.1565512844436151
Silhoutte Index is: 0.7371711570698332
```

3. DBSCAN – density based

I selected a random gene G from the dataset and find its neighbors around *eps* distance. If neighbors exceed *minPts,* we consider G as the core point, add G and its neighbors to a cluster. I then *process* each gene in the cluster and keep on expanding the cluster. when all the genes are processed, we move to other unprocessed points forming new clusters until all genes have been visited once.

Pseudocode:

```
DBSCAN(D, eps, MinPts)
C = 0
for each unvisited point P in dataset D
        mark P as visited
        NeighborPts = regionQuery(P, eps)
        if sizeof(NeighborPts) < MinPts
                mark P as NOISE
        else
                C = next cluster
        expandCluster(P, NeighborPts, C, eps, MinPts)
expandCluster(P, NeighborPts, C, eps, MinPts)
add P to cluster C
for each point P' in NeighborPts
        if P' is not visited
                mark P' as visited
                NeighborPts' = regionQuery(P', eps)
                if sizeof(NeighborPts') >= MinPts
                        NeighborPts = NeighborPts joined with NeighborPts'
        if P' is not yet member of any cluster
                add P' to cluster C
regionQuery(P, eps)
return all points within P's eps-neighborhood (including P)
```

Complexity:

O (n^2). Although this could be improved marginally be implementing spatial index. Time complexity in that case would be O (nlogn).

Advantages:

> Clusters can have arbitrary shape and size based on the density of genes in the data space

> Number of clusters is determined automatically – no need to know ***k*** in advance

> Clusters can be separated from surrounding noise in the ***post-processing*** step based on the cluster density

> Can be supported by spatial index structures

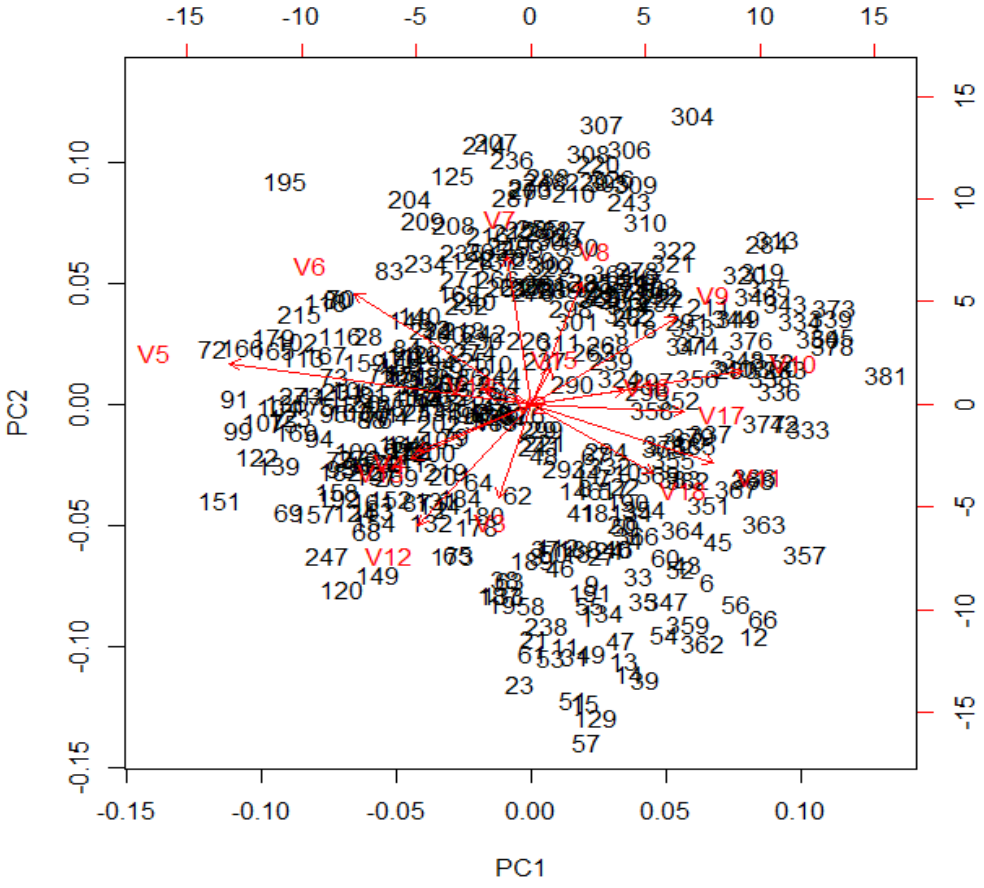
> Resistive to noise

Disadvantages:

> Results may be very sensitive to eps, minPts parameters

> eps, minPts parameters are difficult to determine without prior information

> varying densities may be mishandled and give incorrect results



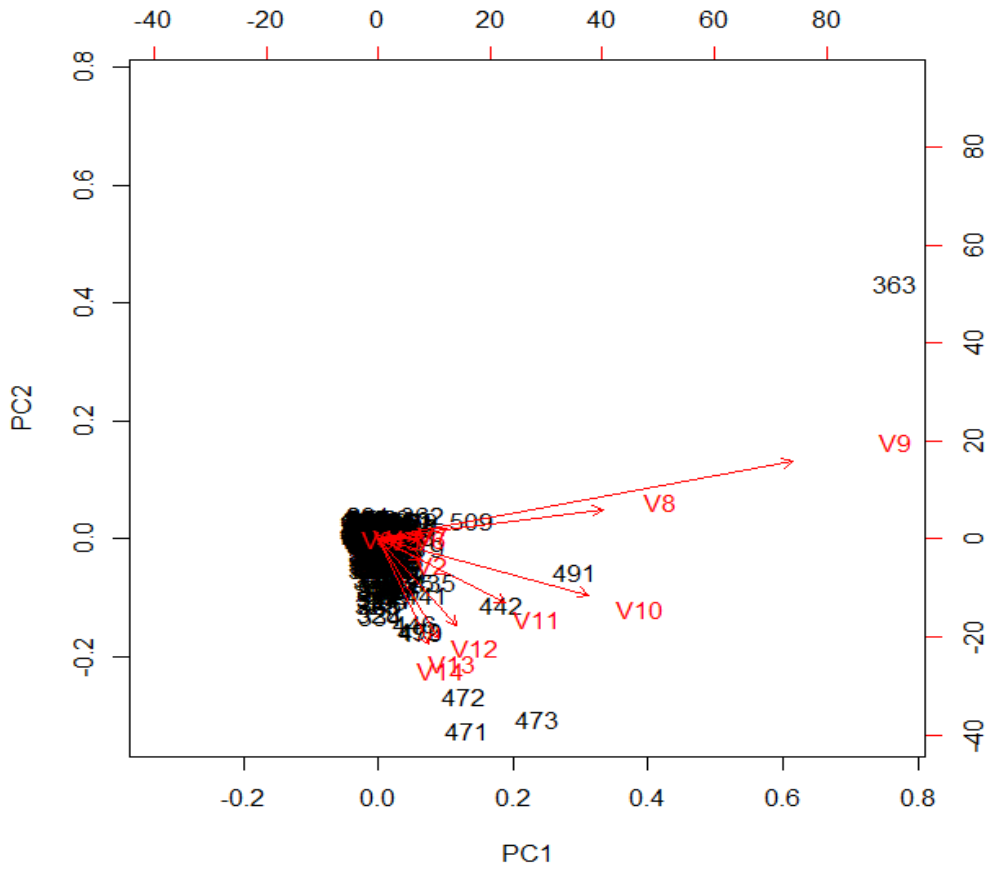
Cho.txt

Settings and Results:

Epsilon = 2

Minimum Points = 2

```
Jaccard value is: 0.22293777953914964
Silhoutte Index is: 0.0
```



Iyer.txt

Settings and Results:

Epsilon = 2

Minimum Points = 2

```
Jaccard value is: 0.17422016684802322
Silhoutte Index is: 0.7637795352582264
```

4. Comparative analysis

For each dataset, compare the performance among all three (or more) algorithms. Which works best? What are the possible reasons? Do you have any other findings?

K-Means

Pros

1. Simple
2. Fast for low dimensional data
3. It can find pure sub clusters if large number of clusters is specified

Cons

1. K-Means cannot handle non-globular data of different sizes and densities
2. K-Means will not identify outliers
3. K-Means is restricted to data which has the notion of a center (centroid)

Agglomerative Hierarchical Clustering

Starting with one point (singleton) clusters and recursively merging two or more most similar clusters to one "parent" cluster until the termination criterion is reached

DBSCAN

Density based Clustering (Maximal Set of Density Connected Points)

Pros

1. Fast for low dimensional data
2. Can discover clusters of arbitrary shapes
3. Robust towards Outlier Detection (Noise)

DBSCAN is very sensitive to clustering parameters MinPoints (Min Neighborhood Points) and EPS (Images Next)

The Algorithm is not partitionable for multi- processor systems.

DBSCAN fails to identify clusters if density varies and if the data set is too sparse. (Images Next)

K-Means Clustering on Hadoop using Map-Reduce

On Hadoop, the clustering process is carried out parallel and so it is called parallel clustering also. It is much better than the non-parallel clustering in the sense that it takes much lesser time for the parallel clustering because parallel processing using mappers and reducers computes and distributes the operations parallel.

Implementation details:
To implement the K-Means clustering, I stored the input, cho.txt file at input path which is provided for first time clustering to the mapper. The output of reducer is then inserted to a file and it is stored at the output file path. Then for clustering repeatedly, the input file is taken as the output of previous step and process is repeated continuously. The last file written is the final result of the clustering which must have a name which includes the number of rotation as ten. This is because I have used maximum ten rotations to compute the results.

Algorithm for K-Means Clustering

In the driver step

Set input/output file paths
Set Mapper class and reducer class files
Run Hadoop job for input file cho.txt and store results at output path for later use

In the map step

Read the file at input file path and create a hashmap of gene information with key and value as list of gene expression

Iterate over each gene to do the same operations

Calculate the index of gene with minimum Euclidian distance

write this index along with gene expression list to mapper output

In the reduce step

Iterate over each value received and

calculate the list of average set (Sum each vector and divide each part by the number of vectors we received).

This is the new centroid, save it into a centroidFile which will be input for next stage.

Increment an update counter for each rotation and use it for the name of output file so that the number of rotation for each file can be identified

Repeat

Run this whole thing until nothing was updated anymore. For this part I hardcoded the number of rotations to be 10 as that is what mostly it will take for the results to stop changing.