Report About

Clustering Techniques

Content

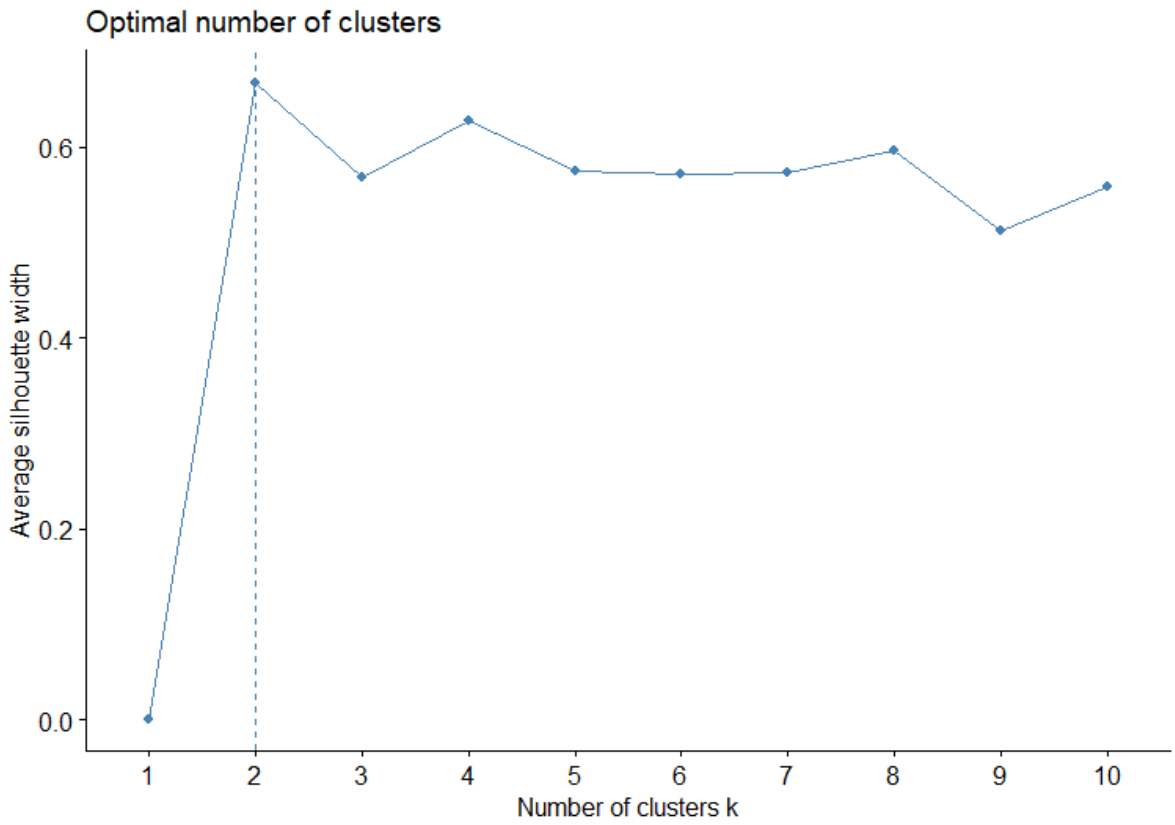
- Introduction.
- K-means clustering.
- Hierarchical clustering using Agglomerative.
- Hierarchical clustering using Divisive.
- Density Based clustering.

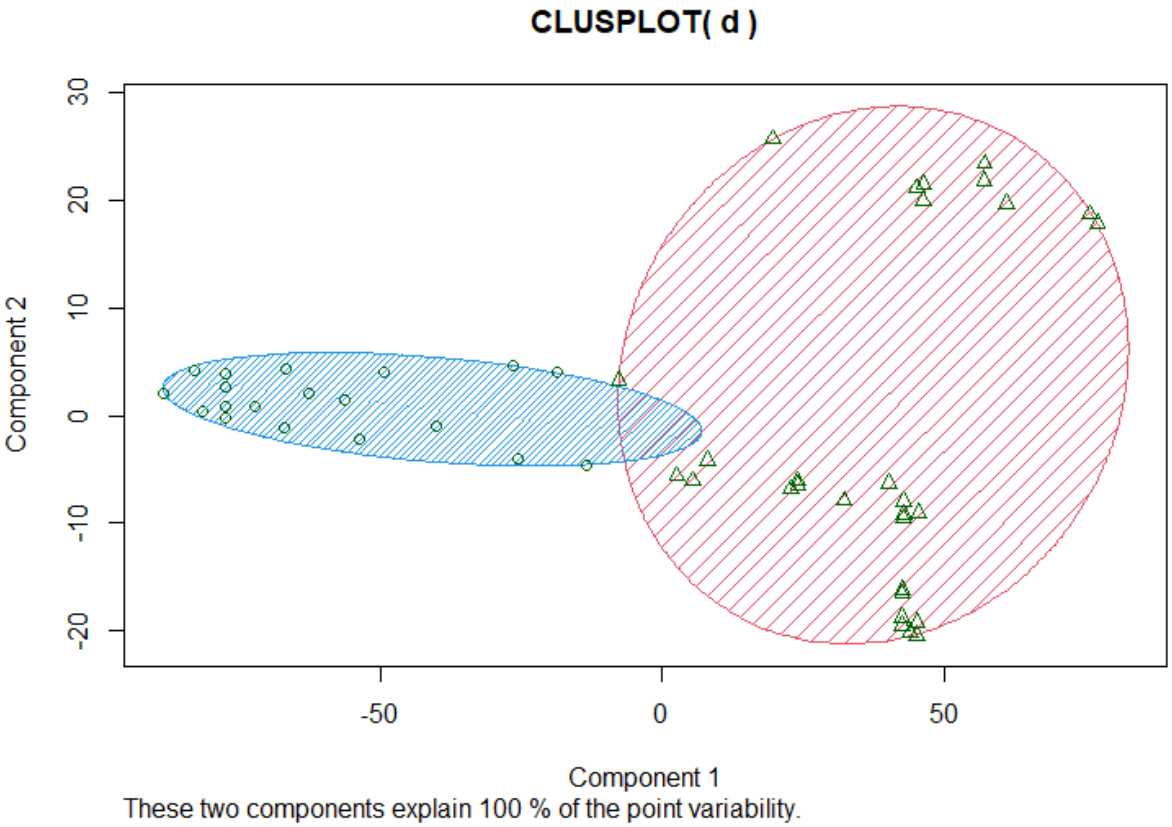
Introduction

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). It is a main task of exploratory data analysis,
and a common technique for statistical data analysis, used in many fields, including pattern recognition, image analysis, information retrieval, bioinformatics, data compression, computer graphics and machine learning.

K means clustering

```r
# Loading package
library(factoextra)
library(ggfortify)
library(ggplot2)
library(dplyr)
library(cluster)
df←read.csv("data 2.enc") #read the data
df  #View the data
d= select(df,c(2,3))
d
#function to compute average silhouette for k clusters
fviz_nbclust(d, kmeans, method = "silhouette")

# Fitting K-Means clustering Model
kms=kmeans(d,2)
kms

#the centers of clusters
kms$centers
# Cluster identification for each observation
kms$cluster

# Visualizing clusters

autoplot(kms,d,frame=T)
clusplot(d,kms$cluster ,color=T,shade=T)
#plot results of final k-means model
fviz_cluster(kms, data = d)
```


Optimal number of clusters
Average silhouette width
0.0
0.2
0.4
0.6
1
2
3
4
5
6
7
8
9
10
Number of clusters k


CLUSPLOT(d)
Component 2
30
20
10
0
-10
-20
-50
0
50
Component 1
These two components explain 100 % of the point variability.


```
K-means clustering with 2 clusters of sizes 19, 29

Cluster means:
     times       accel
1 21.06316 -104.305263
2 21.85517   -7.262069

Clustering vector:
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[37] 2 1 1 2 2 2 2 2 2 2 2 2

Within cluster sum of squares by cluster:
[1] 10114.99 18170.82
 (between_SS / total_SS =  79.3 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"
[5] "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
>
> #the centers of clusters
> kms$centers
     times       accel
1 21.06316 -104.305263
2 21.85517   -7.262069
> # Cluster identification for each observation
> kms$cluster
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[37] 2 1 1 2 2 2 2 2 2 2 2 2
```

Advantages:

- If variables are huge, then K-Means most of the times computationally faster than hierarchical clustering, if we keep k smalls.
- K-Means produce tighter clusters than hierarchical clustering, especially if the clusters are globular.
- Relatively simple to implement.
- Scales to large data sets.
- Can warm-start the positions of centroids.
- Easily adapts to new examples.

Disadvantages:

- Clustering outliers.
- Centroids can be dragged by outliers, or outliers might get their own cluster instead of being ignored. Consider removing or clipping outliers before clustering.
- Scaling with number of dimensions.
- As the number of dimensions increases, a distance-based similarity measure converges to a constant value between any given examples. Reduce dimensionality either by using PCA on the feature data, or by using “spectral clustering” to modify the clustering algorithm as explained below.
- Different initial partitions can result in different final clusters.
- It does not work well with clusters (in the original data) of Different size and Different density.

when k-means preferable to be used?

k-means becomes a great solution for pre-clustering, reducing the space into disjoint smaller sub-spaces where other clustering algorithms can be applied.

The K-means clustering algorithm is used to find groups which have not been explicitly labeled in the data. This can be used to confirm business assumptions about what types of groups exist or to identify unknown groups in complex data sets.

Can warm-start the positions of centroids. Easily adapts to new examples. Generalizes to clusters of different shapes and sizes, such as elliptical clusters.

if variables are huge, then K-Means most of the times computationally faster than hierarchical clustering.

How can k-means clustering be improved?

K-means clustering algorithm can be significantly improved by using a better initialization technique, and by repeating (re-starting) the algorithm.

When the data has well separated clusters, the performance of k-means depends completely on the goodness of the initialization.

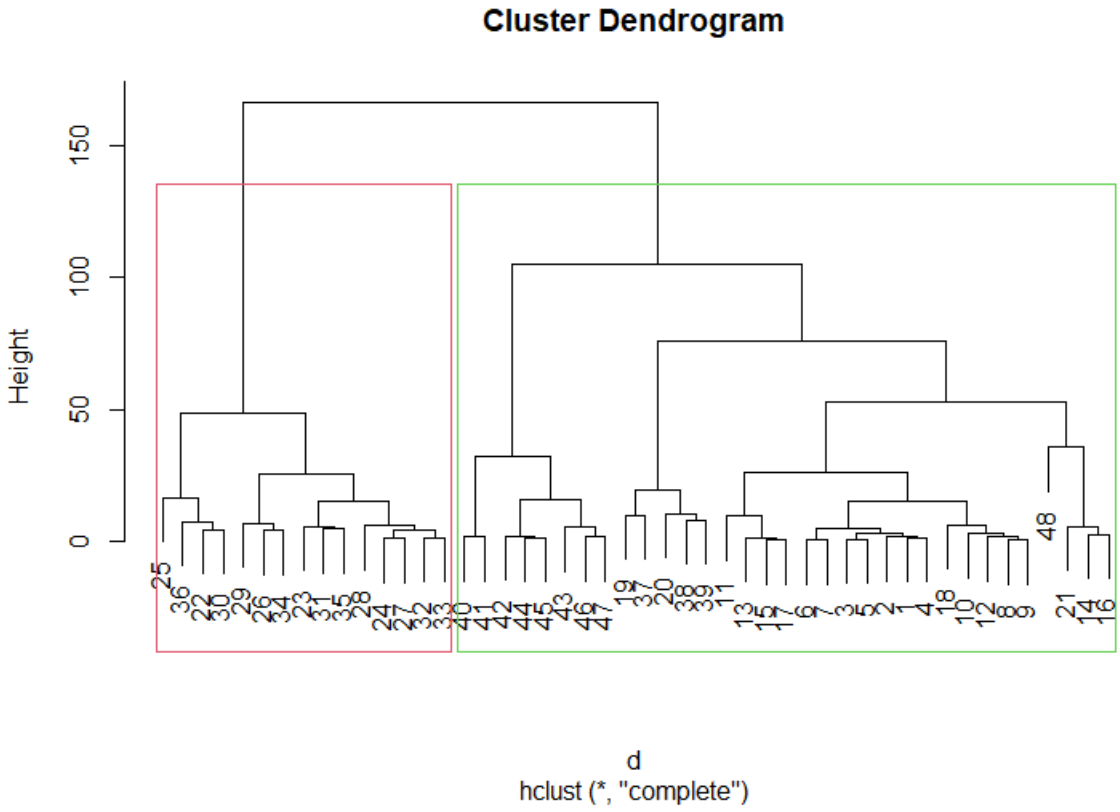
Initialization using simple furthest point heuristic (Maxmin) reduces the clustering error of k-means from 15% to 6%, on average.

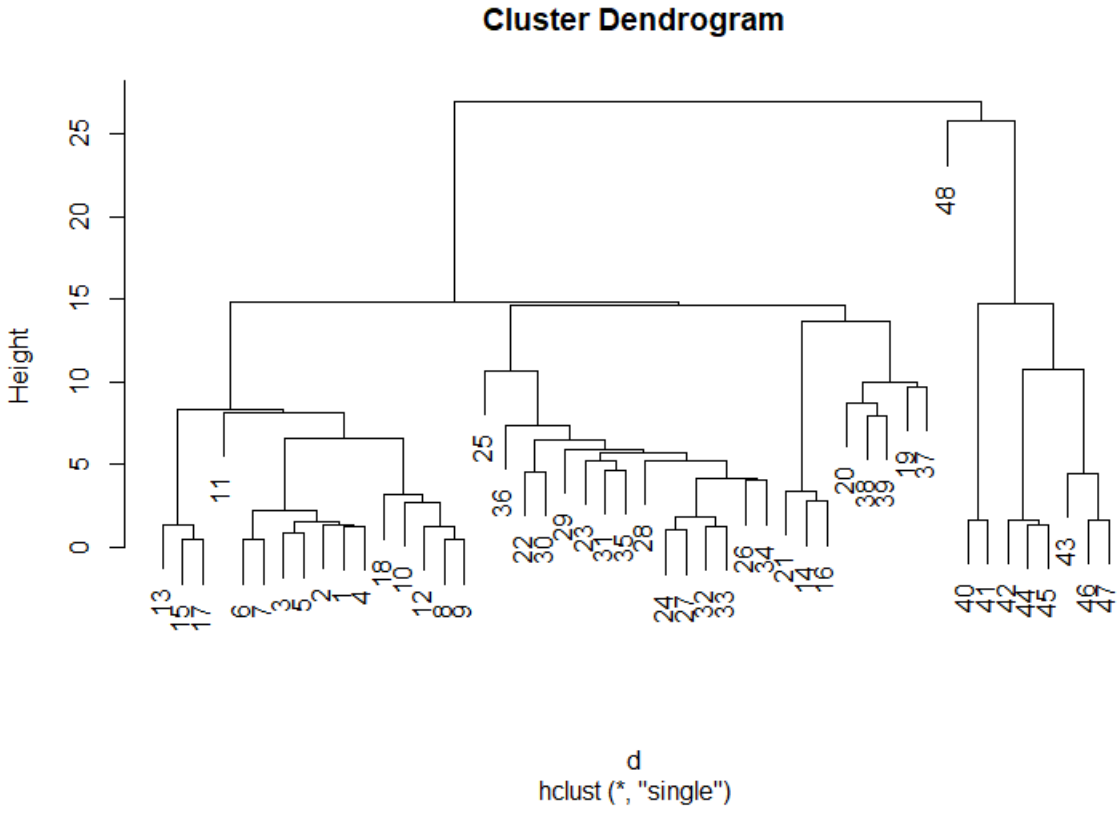
Hierarchical clustering using Agglomerative

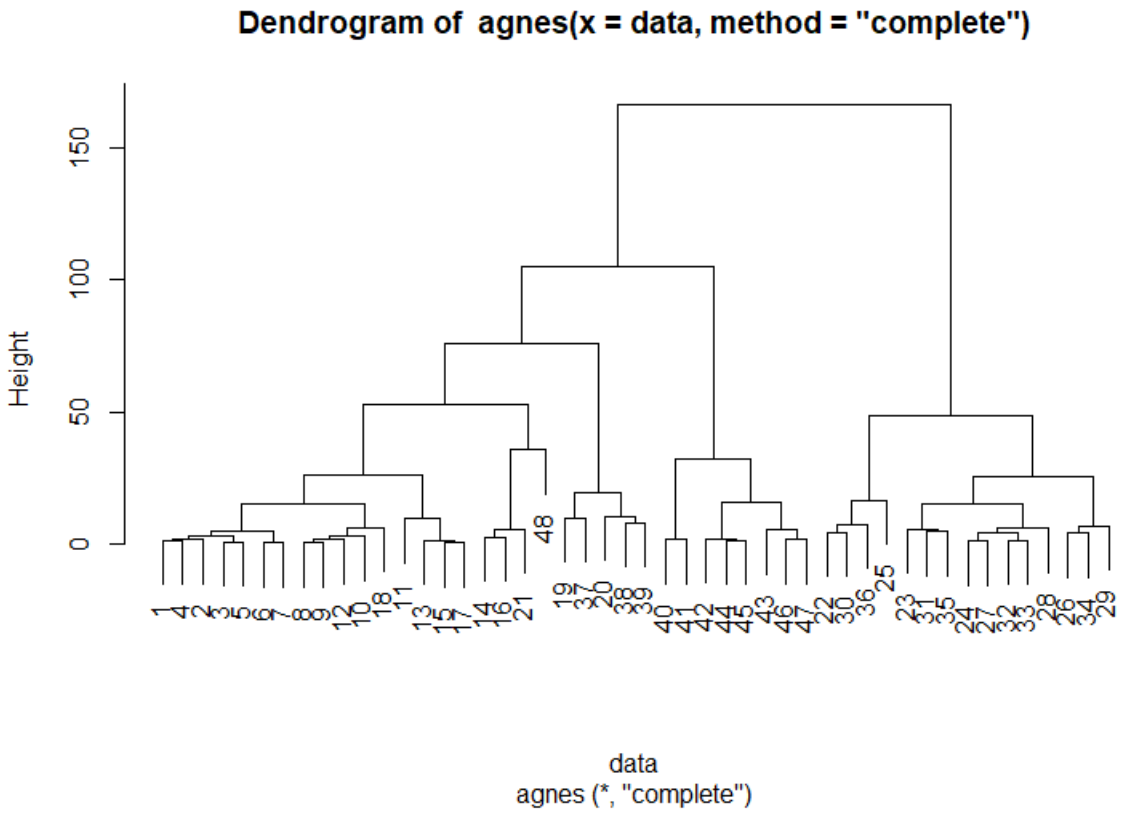
```r
# importing libraries
library(ggplot2)
library(cluster)
library(ggfortify)
library(stats)
library(dplyr)
library(factoextra)
#import data
data = read.csv("data 2.enc")
data = data[,c(2,3)]

#Agglomerative using hclust
d = dist(data, method = "euclidean")
#complete method
hcluster = hclust(d, method = "complete")
plot(hcluster)
group1 ← cutree(hcluster, k=3) # cut tree into 3 clusters
# draw dendogram with red borders around the 5 clusters
rect.hclust(hcluster, k=3, border=2:5)
table(group1)
#single method
hcluster = hclust(d, method = "single")
plot(hcluster)
#select best number of clusters(k)
fviz_nbclust(data, FUN = hcut, method = "silhouette")#3 clusters
#-------------------------------------------------------------------------
```

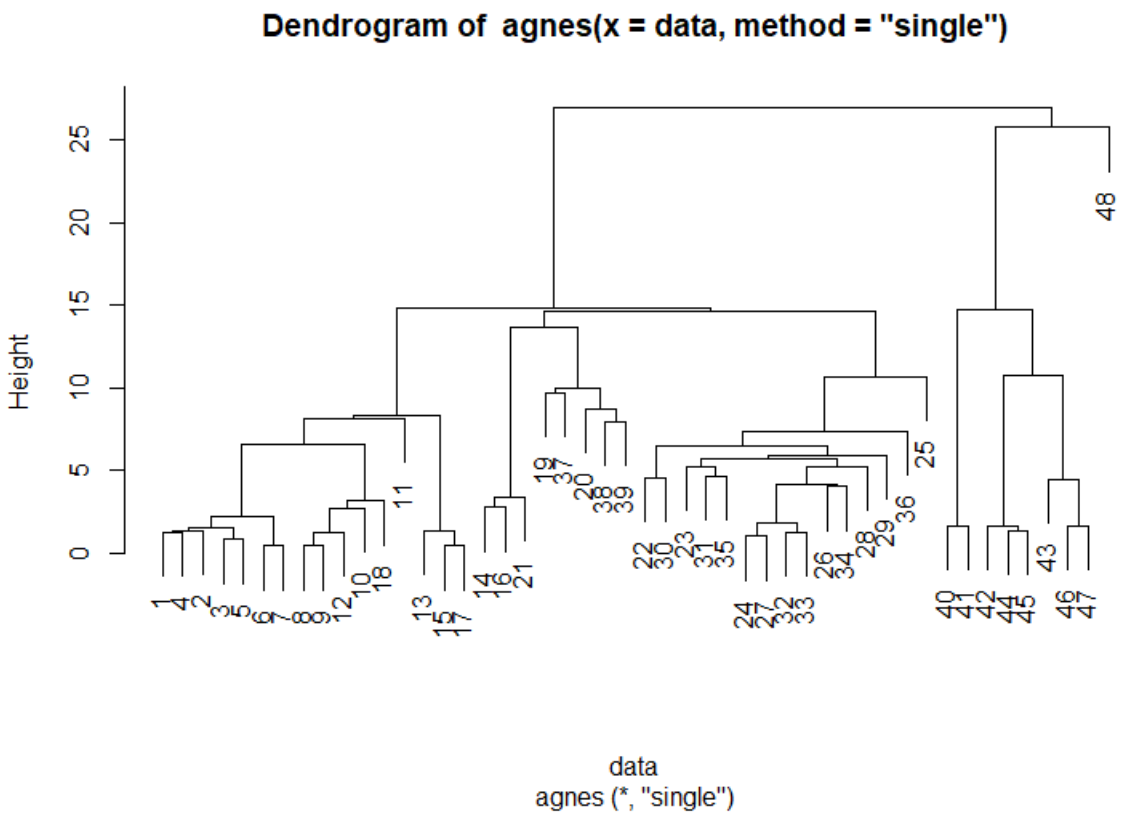
```r
#Agglomerative using agnes
agh=agnes(data,method="complete")
agh$ac #look at the agglomerative coeffecient
group2 = cutree(as.hclust(agh), k = 3)
table(group2)
pltree(agh)
#Single method
agh=agnes(data,method="single")
agh$ac
pltree(agh)
```

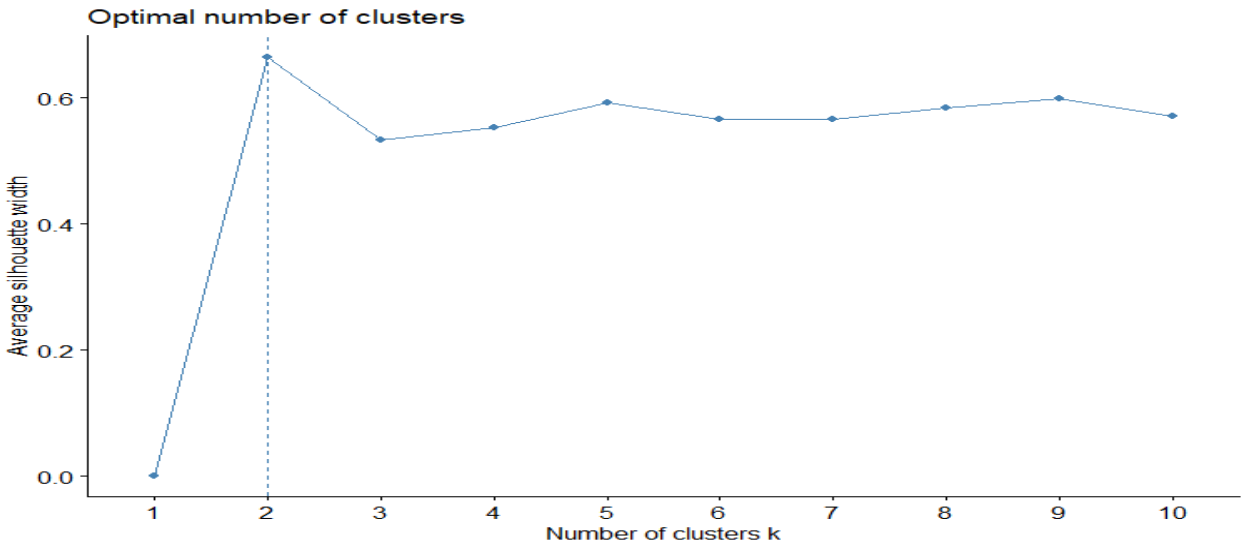
Result


Cluster Dendrogram
Height
0
50
100
150
d
hclust (*, "complete")


Cluster Dendrogram
Height
0
5
10
15
20
25
d
hclust (*, "single")


Dendrogram of agnes(x = data, method = "complete")
Height
0
50
100
150
1 4 2 3 5 6 7 8 9 12 10 18 11 13 15 17 14 16 21 48 19 37 20 38 39 40 41 42 44 45 43 46 47 22 30 36 25 23 31 35 24 27 32 33 28 26 34 29
data
agnes (*, "complete")


Dendrogram of agnes(x = data, method = "single")
Height
0
5
10
15
20
25
1 4 2 3 5 6 7 8 9 12 10 18 11 13 15 17 14 16 21 19 37 20 38 39 22 30 23 31 35 24 27 32 33 26 34 28 29 36 25 48 40 41 42 44 45 43 46 47
data
agnes (*, "single")



Hclust

```
> table(group1)
group1
 1  2
33 15
```

Agnes

```
> table(group2)
group2
 1  2
33 15
```

Comments on result

Both hclust and agnes have the same result and behave very similarly in single and complete methods As we see in group 1 and group 2and Agnes has more methods than hclust.

The proximity of two observations can be drawn based on the height where branches containing those two observations first are fused (like height of 14&16 in complete) But we can't do that with the horizontal axis as a criterion of their similarity. In complete method it tends to produce more compact clusters But in single it tends to produce long "loose" clusters.

We see that the best number of clusters here is 2 by silhouette method.

Agnes yields the agglomerative coefficient which measures the amount of clustering structure found (0.9734672 in complete and 0.8583913 in single), the closer to 1 the strongest clustering structure

Advantage of Agglomerative

- Agnes function can get the agglomerative coefficient, which measures the amount of clustering structure found (values closer to 1 suggest strong clustering structure).
- Agnes, it allows us to find certain hierarchical clustering methods that can identify stronger clustering structures.
- Agnes contains six agglomerative algorithms, some not included in hclust. Has it's own plot method.
- Hclust allow us to use rect.hclust method - which divides dendrogram into given number of groups (argument k) and draws rectangles around samples in these groups.
- Hclust allow us to use cutree method - cuts the tree (dendrogram) into given number of clusters (argument k) or according to given level of similarity
- hclust function implements both Ward's algorithms (the genuine one, named ward.D2, as well as the second one, called ward.D) unlike agnes which use the genuine one.
- agglomerative clustering is Easy to implement and good at identifying small clusters
- It doesn't require the number of clusters to be specified.

Disadvantage of Agglomerative

- Generally, has long runtimes.
- Time complexity is higher at least 0(n^2logn).
- Can never undo what was done previously, which means if the objects may have been incorrectly grouped at an earlier stage, and the same result should be close to ensure it.
- the usage of various distance metrics for measuring distances between the clusters may produce different results.
- Not suitable for large datasets.
- Initial seeds have a strong impact on the results.
- The order of the data has an impact on the results
- Very sensitive to outliers

Some cases in which Agglomerative is preferable to be used

when we want to partition our data into groups at different levels such as in a hierarchy and use an algorithm which is Easy to implement and good at identifying small clusters and when it doesn't require the number of clusters to be specified.

How can we improve Agglomerative?

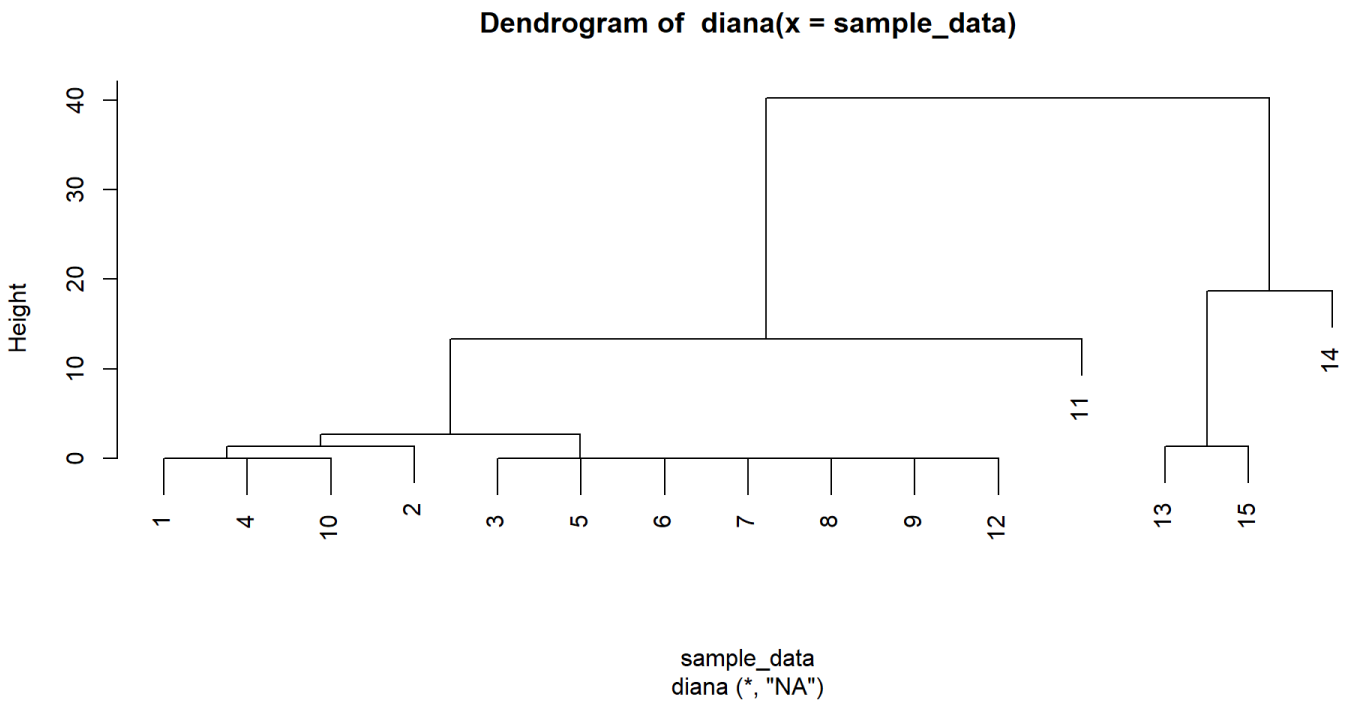
A main drawback of the Agglomerative is that it is very sensitive to noise and thus results in its lack of robustness, which limits its use in real-world applications.

The kernel method in the machine learning theory refers to increasing the computational power of linear methods by mapping the data into high-dimensional feature space and has shown its great power in several kernel-based learning machines, e.g., support vector machines (SVMs) and kernel principal component analysis (KPCA). So, a robust batch clustering algorithm has been developed for clustering incomplete data (nonstationary data) using the kernel method

Hierarchical clustering using Divisive

```r
install.packages("ggplot2")
install.packages("ggfortify")
install.packages("factoextra")
library(tidyverse)
library(cluster)
library(ggplot2)
library(factoextra)
library(mclust)
library(ggfortify)
library(stats)
library(dplyr)
View(data.2)
data = select(data.2,c(2,3))
sample_data=(data[1:15,2,3])
dh=diana(sample_data)
dh$dc
pltree(dh)

```

Result:



```
> data = select(data.2,c(2,3))
> sample_data=(data[1:15,2,3])
> dh=diana(sample_data)
> dh$dc
[1] 0.9404643
> pltree(dh)
```

Advantages of divisive:

Divisive clustering is more efficient if we do not generate a complete hierarchy all the way down to individual data leaves. Time complexity of a naive agglomerative clustering is O(n3) because we exhaustively scan the N x N matrix dist_mat for the lowest distance in each of N-1 iterations

Increased performance: Multiple machines provide greater processing power. Greater scalability: As your user base grows and report complexity increases, your resources can grow. Simplified management: Clustering simplifies the management of large or rapidly growing systems.

divisive algorithm does not require to prespecify the number of clusters

Disadvantages:

A challenge with divisive methods is how to partition a large cluster into several smaller ones. For example, there are $2^{n-1} - 1$ possible ways to partition a set of n objects into two exclusive subsets, where n is the number of objects. When n is large, it is computationally prohibitive to examine all possibilities. Consequently, a divisive method typically uses heuristics in partitioning, which can lead to inaccurate results. For the sake of efficiency, divisive methods typically do not backtrack on partitioning decisions that have been made. Once a cluster is partitioned, any alternative partitioning of this cluster will not be considered again. Due to the challenges in divisive methods, there are many more agglomerative methods than divisive methods.

Hierarchical Agglomerative vs Divisive clustering:

Divisive clustering is more

complex as compared to agglomerative clustering, as in the case of divisive clustering we need a flat clustering method as “subroutine” to split each cluster until we have each data having its own singleton cluster.

Divisive clustering is more efficient if we do not generate a complete hierarchy all the way down to individual data leaves. The time complexity of a naive agglomerative clustering is O(n3) because we exhaustively scan the N x N matrix dist_mat for the lowest distance in each of N-1 iterations. Using priority queue data structure we can reduce this complexity to O(n2logn). By using some more optimizations it can be brought down to O(n2). Whereas for divisive clustering given a fixed number of top levels, using an efficient flat algorithm like K-Means, divisive algorithms are linear in the number of patterns and clusters.

A divisive algorithm is also more accurate. Agglomerative clustering makes decisions by considering the local patterns or neighbor points without initially taking into account the global distribution of data. These early decisions cannot be undone. whereas divisive clustering takes into consideration the global distribution of data when making top-level partitioning decisions.

Advantages of Hierarchical clustering

- Hierarchical clustering outputs a hierarchy, is a structure that is more informative than the unstructured set of flat clusters returned by k-means. Therefore, it is easier to decide on the number of clusters by looking at the dendrogram.
- Easy to implement.
- Unlike, K-means clustering where we set the number of clusters and then analyze the data and again change number of clusters to be formed for better results, here the algorithm is much easy for applying and interpretation.
- even after hierarchical clustering's more time complexity, it is worth to use it over other clustering technique

Disadvantages

- the time taken to apply this clustering is more than K-means. So, hierarchical is advised not to be used for large datasets, as the time complexity is O(n^2), that is with larger dataset, time taken would take squared more time.
- It does not always provide the best solution: When you cluster multi-dimensional retail data that cannot always be visualized on a plot, poor solutions may be tricky to spot and resolve.
- The algorithm cannot run if there is missing data: You will need to remove these lines or estimate values to ensure the algorithm is able to run.
- The algorithm cannot run with different data types: When you use many different data types, it becomes difficult to compute a distance matrix. There is no simple formula that can work with both qualitative and numerical data at the same time.
- The dendrogram can be misinterpreted: The descriptors and composition of clusters may be difficult to interpret for all your business stakeholders involved with clustering.

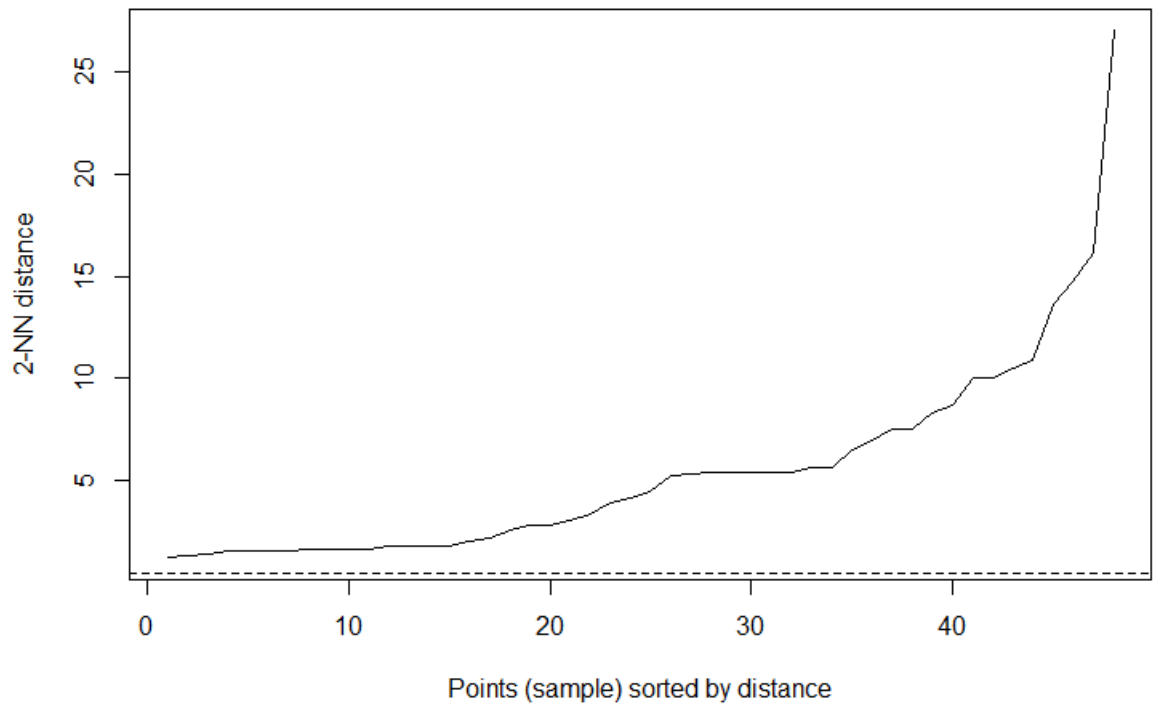
Density-based clustering

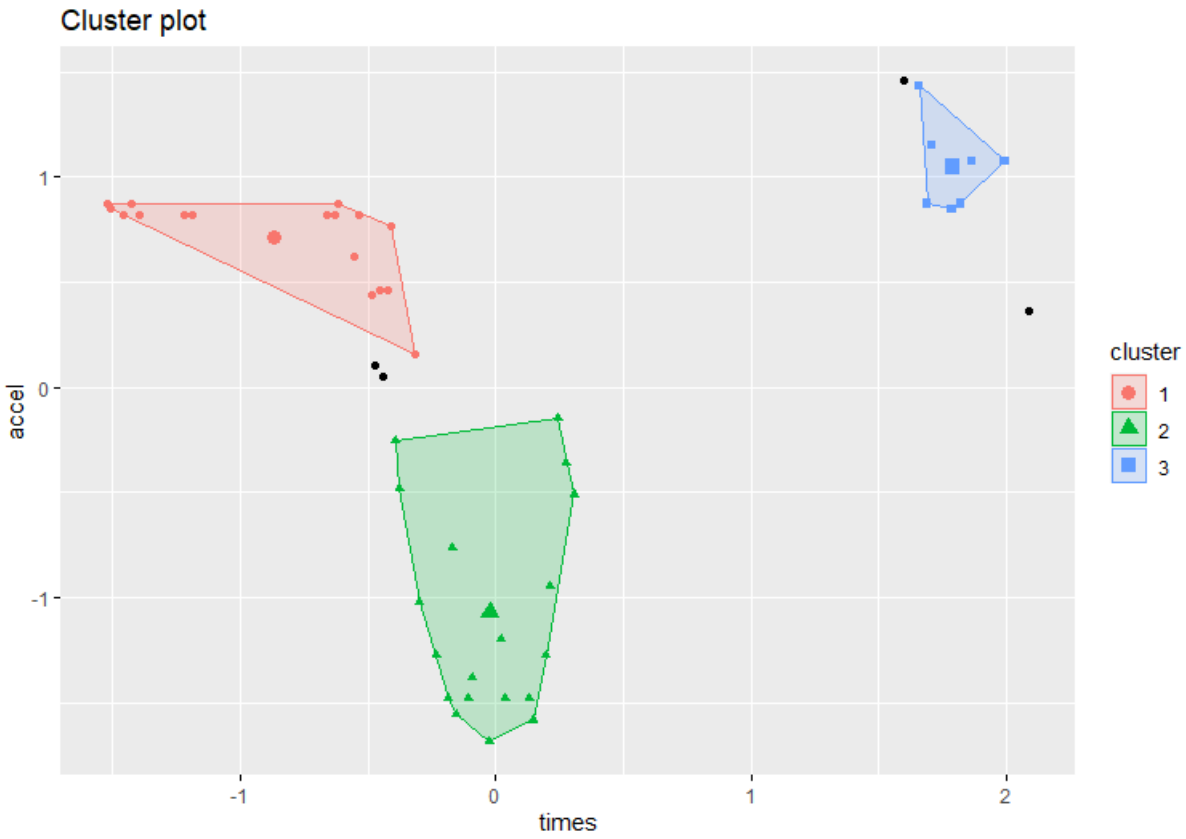
```r
# Density-based clustering

#Importing libraries
library(dbscan)
library(fpc)
library(pch)

# data.2 data
data.2 ← read.csv("data 2.enc")
str(data.2)
D2← data.2[,c(2,3)]
str(D2)

# Installing packages
if(!require(devtools)) install.packages("devtools")
devtools::install_github("kassambara/factoextra")
library(factoextra)

# Obtaining optimal eps value
kNNdistplot(D2, k=2)
abline(h = 0.45, lty=2)

# Density-based clustering with fpc & dbscan
set.seed(25)

f ← fpc::dbscan(D2, eps=15, MinPts =5)
f
d ← dbscan::dbscan(D2, 15, 5)
d
```

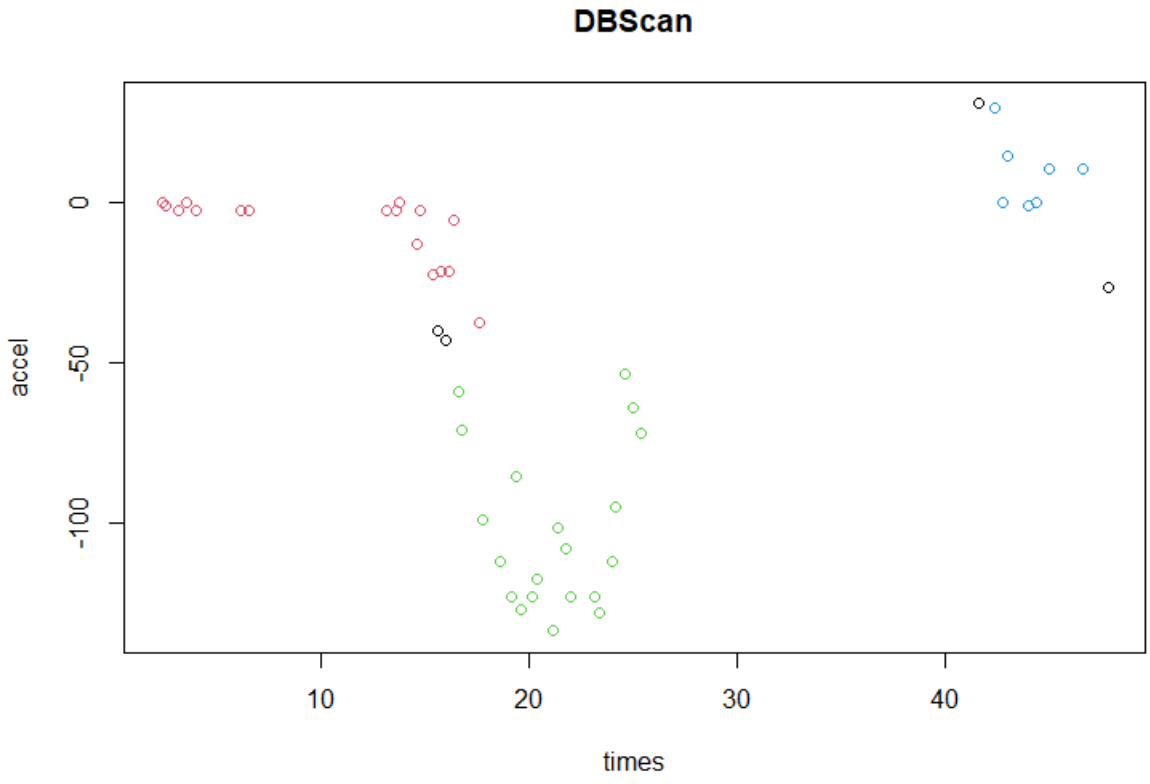
```r

# Cluster visualization
fviz_cluster(d, D2, geom = "point")

# Plotting Cluster
plot(d, D2[,1:2], main = "DBScan")

```

Results

```
DBSCAN clustering for 48 objects.
Parameters: eps = 15, minPts = 5
The clustering contains 3 cluster(s) and 4 noise points.

 0  1  2  3
 4 17 20  7

Available fields: cluster, eps, minPts
```




Cluster plot
accel
times
cluster
1
2
3
1
0
-1
-1
0
1
2


DBScan
accel
times
0
-50
-100
10
20
30
40

Comment

To get the best result of clustering my data, I tried to increase and decrease the number of minPoints and EPS to see which solution is the suitable .

When

```
f <- fpc::dbscan(data.2, eps=20, MinPts = 5)
f
```

output:

```
dbscan Pts=48 MinPts=5 eps=20
      0 1  2  3 4
border 5 0  3  1 2
seed   0 7  9 15 6
total  5 7 12 16 8
```

```
d <- dbscan::dbscan(data.2, 20, 5)
d
```

output:

DBSCAN clustering for 48 objects.

Parameters: eps = 20, minPts = 5

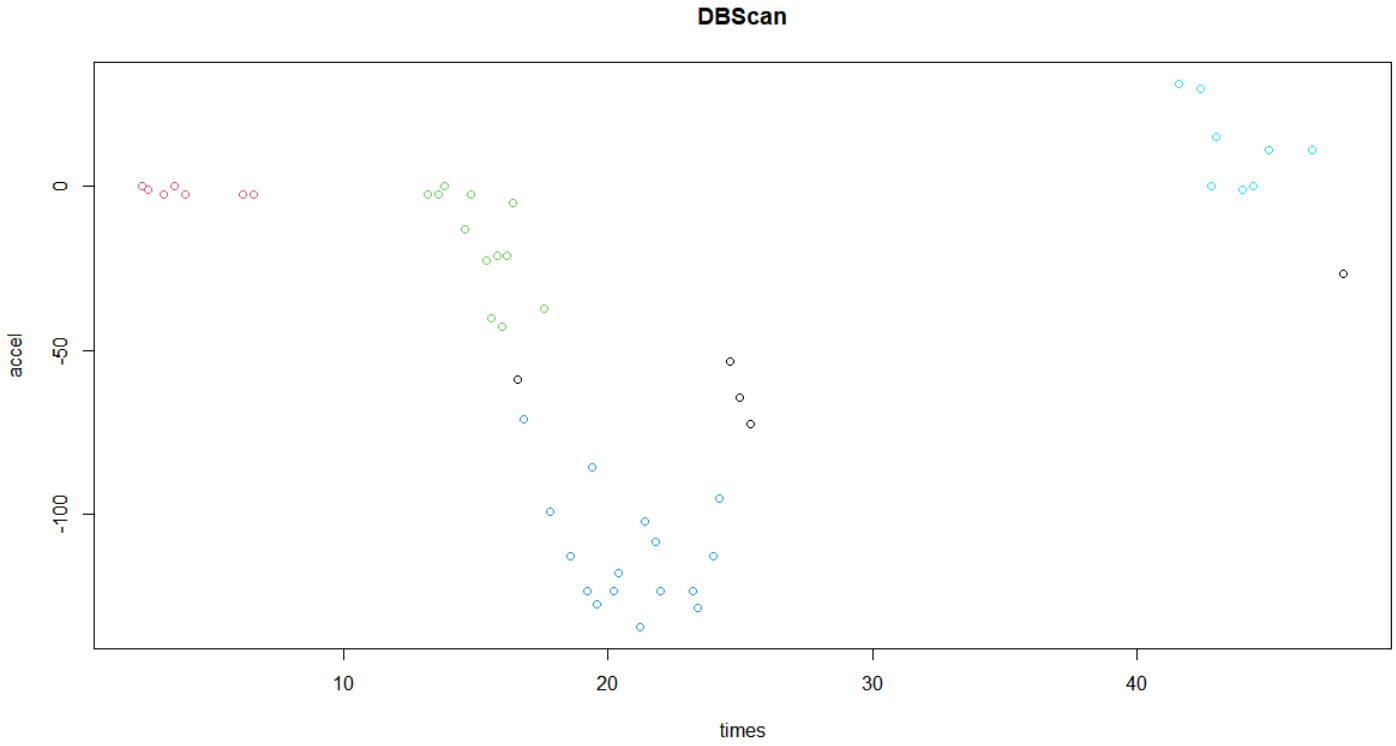
The clustering contains 4 cluster(s) and 5 noise points.

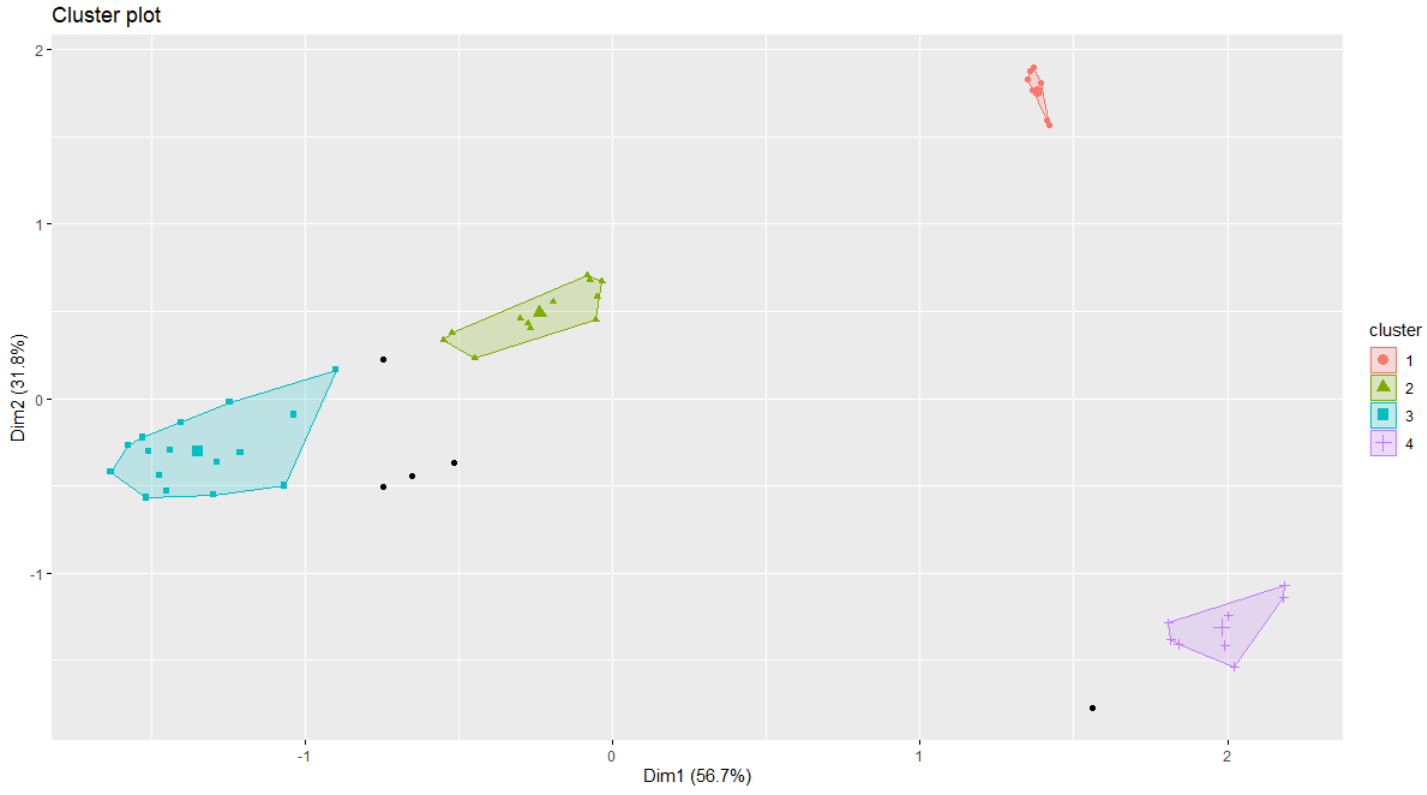
```
0  1  2  3  4
 5  7 12 16  8
```

We notice that the number of noise points incresed by 1 and the number of clusters increased to 4

When using Eps =20 and minpoints =5

```
plot(d, D2[,1:2], main = "DBScan")
```


DBScan
accel
0
-50
-100
10
20
30
40
times


Cluster plot
Dim2 (31.8%)
2
1
0
-1
-1
0
1
2
Dim1 (56.7%)
cluster
1
2
3
4

Advantages

- Can discover arbitrarily shaped clusters.
- Does not require a-priori specification of number of clusters.
- Able to identify noise data while clustering.
- Find cluster completely surrounded by different clusters.
- Require just two points which are very insensitive to the ordering of the points in the database.

Disadvantages

- Not partitionable for multiprocessor systems.
- Datasets with altering densities are tricky.
- Sensitive to clustering parameters minPoints and EPS.
- Fails to identify cluster if density varies and if the dataset is too sparse.
- Sampling affects density measures.

When this method is preferable >>

DBSCAN. DBSCAN is a clustering method that is used in machine learning to separate clusters of high density from clusters of low density. Given that DBSCAN is a density-based clustering algorithm, **it does a great job of seeking areas in the data that have a high density of observations, versus areas of the data that are not very dense with observations. DBSCAN can sort data into clusters of varying shapes as well, another strong advantage.**

Improve this method: when we increase Eps so that outliers decrease or the clusters increase and outliers increase , we get better normalization .