Unsupervised Learning Methods

Dr Sebnem Er

2020-10-12

Contents

Chapter 1

Introduction

This book will guide you through the R codes for the following Unsupervised Learning methods:

- Association Rules
- Cluster Analysis
- Self Organising Maps

The chapters will be made available on Tuesdays when we start a new topic. So please update your browser to access the codes for the relevant chapter.

Chapter 2

Association Rules

2.1 Prerequisites

You need to have the following R packages installed and recalled into your library:

```
library(datasets)
library(grid)
library(tidyverse)
library(readxl)
library(knitr)
library(ggplot2)
library(lubridate)
library(arules)
library(arulesViz)
library(plyr)
```

2.2 The Groceries Dataset

We shall mine Groceries dataset for association rules using the Apriori Algorithm. The Groceries dataset can be loaded from R. The steps for doing so are shown below. Note that you will only be able to load the data set once the package `arules` has been loaded into R. The Groceries dataset contains a collection of receipts with each line representing 1 receipt and the items purchased. Each line is called a transaction and each column in a row represents an item.

```
data(Groceries)
summary(Groceries)
```

```
## transactions as itemMatrix in sparse format with
##  9835 rows (elements/itemsets/transactions) and
##  169 columns (items) and a density of 0.02609146
##
## most frequent items:
##       whole milk other vegetables       rolls/buns             soda
##             2513             1903             1809             1715
##           yogurt          (Other)
##             1372            34055
##
## element (itemset/transaction) length distribution:
## sizes
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
## 2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77   55   46
##   17   18   19   20   21   22   23   24   26   27   28   29   32
##   29   14   14    9   11    4    6    1    1    1    1    3    1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   4.409   6.000  32.000
##
## includes extended item information - examples:
##        labels  level2           level1
## 1 frankfurter sausage meat and sausage
## 2     sausage sausage meat and sausage
## 3  liver loaf sausage meat and sausage
```

As you can see, the data is in "transactions" format with a density of 0.0261 (check slides to remember what this value means). There are 9835 transactions with 169 distinct items that can be bought in this database (D).

The summary function also provides the distribution of number items per transaction and the most popular items.

Now let us examine the first 3 transactions in D.

```
inspect(head(Groceries, 3))
```

```
##     items
## [1] {citrus fruit,
##      semi-finished bread,
##      margarine,
##      ready soups}
```

```
## [2] {tropical fruit,
##      yogurt,
##      coffee}
## [3] {whole milk}
```

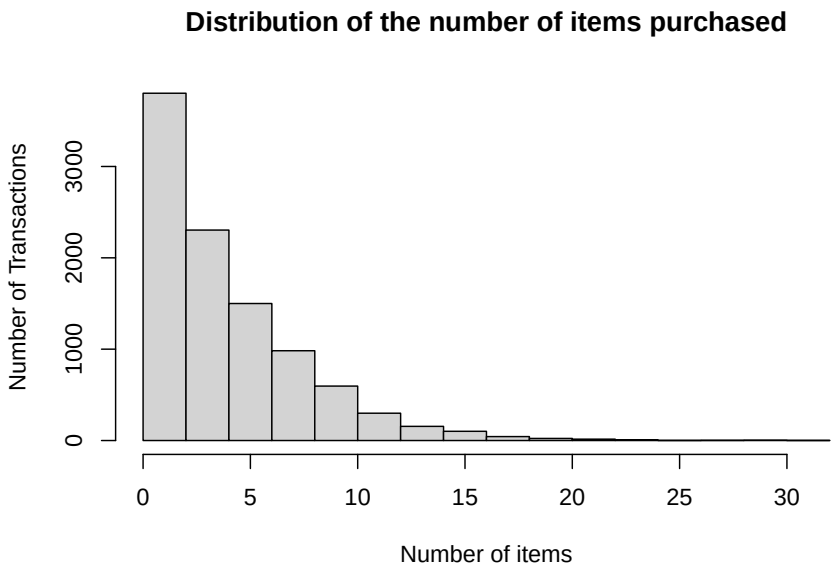
The first customer bought {citrus fruit,semi-finished bread,margarine,ready soups}, whereas the third customer bought only {whole milk}.

We can also find how many items each transaction contains, for the first 10 transactions:

```
head(size(Groceries), 10)
```

```
##  [1] 4 3 1 4 4 5 1 5 1 2
```

```
hist(size(Groceries), main = "Distribution of the number of items purchased", xlab = "Number of i
```



As it is clear, the distribution of the number of items is skewed to right, clearly most transactions inlcude fewer number of items, only very few have more than 10 items purchased together.

2.3 Support Count (Item Frequencies) and Item Frequency Plot

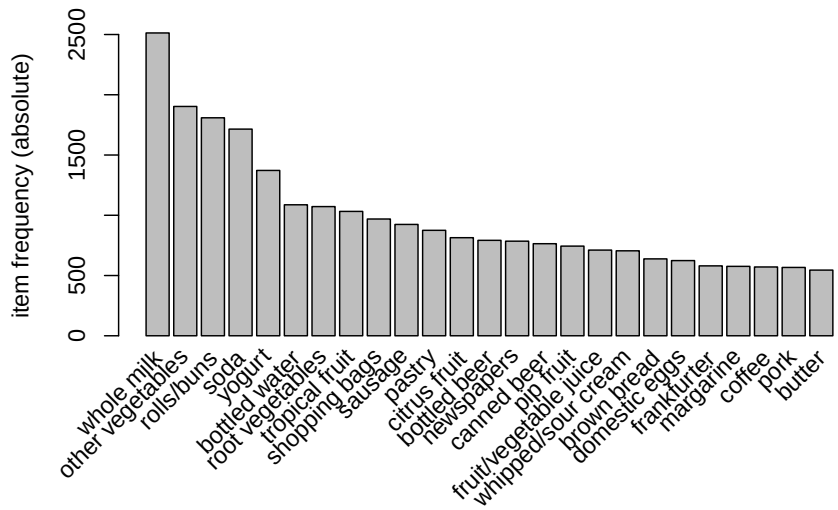
We can check the support count ($freq(A)$) for the top 25 products with the following R code:

```
itemSupportCount = itemFrequency(Groceries, type = "absolute") # obtain the counts for
itemSupportCount = sort(itemSupportCount, decreasing = TRUE) # sort the counts in desc
head(itemSupportCount, 25) # check the support count for the top 25 items
```

```
##            whole milk      other vegetables            rolls/buns
##                  2513                  1903                  1809
##                  soda                yogurt         bottled water
##                  1715                  1372                  1087
##       root vegetables        tropical fruit         shopping bags
##                  1072                  1032                   969
##               sausage               pastry          citrus fruit
##                   924                   875                   814
##          bottled beer           newspapers           canned beer
##                   792                   785                   764
##            pip fruit fruit/vegetable juice    whipped/sour cream
##                   744                   711                   705
##          brown bread         domestic eggs           frankfurter
##                   638                   624                   580
##            margarine                coffee                  pork
##                   576                   571                   567
##               butter
##                   545
```

We can also plot the support count, it is possible to change the colours of the bars as well.

```
itemFrequencyPlot(Groceries, topN = 25, type="absolute")
```



We can see that top purchased product is {whole milk} and it appears in 2513 transactions out of 9835. Therefore the support count for {whole milk} is 2513.

2.4 Support

Remember the support ($S(A)$) is calculated as follows:

$$S(A) = \frac{\texttt{freq(A)}}{n}$$

The support for {whole milk} would be

$$S(\texttt{whole milk}) = \frac{\texttt{freq(whole milk)}}{n} = \frac{2513}{9835} = 25.55\%$$

It is possible to obtain this information with the same code as shown previously by simply replacing `type="absolute"` with the `type="relative"` option:

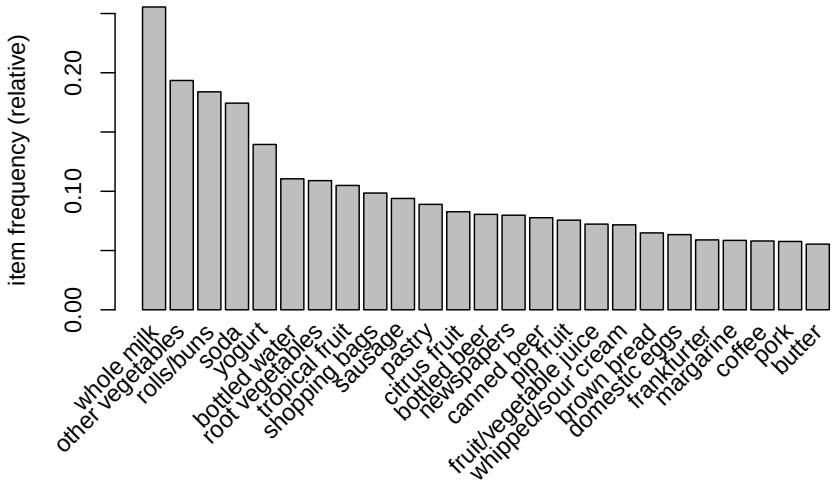
```
itemSupport = itemFrequency(Groceries, type = "relative") # obtain the counts for individual item
itemSupport = sort(itemSupport, decreasing = TRUE) # sort the counts in descending order
head(itemSupport, 25) # check the support count for the top 25 items
```

```
##            whole milk      other vegetables            rolls/buns
##            0.25551601            0.19349263            0.18393493
```

```
##                  soda                yogurt         bottled water
##            0.17437722            0.13950178            0.11052364
##       root vegetables        tropical fruit         shopping bags
##            0.10899847            0.10493137            0.09852567
##               sausage                pastry          citrus fruit
##            0.09395018            0.08896797            0.08276563
##          bottled beer            newspapers           canned beer
##            0.08052872            0.07981698            0.07768175
##             pip fruit fruit/vegetable juice    whipped/sour cream
##            0.07564820            0.07229283            0.07168277
##           brown bread         domestic eggs           frankfurter
##            0.06487036            0.06344687            0.05897306
##             margarine                coffee                  pork
##            0.05856634            0.05805796            0.05765125
##                butter
##            0.05541434
```

We can also plot the support.

```
itemFrequencyPlot(Groceries, topN = 25, type="relative")
```



Note that the maximum support is low. To ensure that the top 25 frequent items are included in the analysis the minimum support would have to be less than 0.10! (10%) Suppose we set the minimum support to 0.001 and minimum confidence to 0.8. We can mine some rules by executing the following R code:

2.5 Rule Generation with Apriori Algorithm

- We are going to use the Apriori algorithm within the `arules` library to mine frequent itemsets and association rules..
- Assume that we want to generate all the rules that satisfy the support threshold of 0.1% and confidence threshold of 80%, then we need to enter `supp=0.001` and `conf=0.8` values in the `apriori()` function. If you want stronger rules, you can increase the value of `conf` and for more extended rules give higher value to `maxlen`.
- It might be desirable to sort the rules according either confidence or support, here we chose sorting according to confidence in a descending manner.
- Finally we can examine the rules using `summary()` function.

```
rules <- apriori(Groceries, parameter = list(supp=0.001, conf=0.8))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5   0.001      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 9
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [157 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 done [0.04s].
## writing ... [410 rule(s)] done [0.06s].
## creating S4 object  ... done [0.00s].
```

```
rules <- sort(rules, by='confidence', decreasing = TRUE)
summary(rules)
```

```
## set of 410 rules
```

```
##
## rule length distribution (lhs + rhs):sizes
##   3   4   5   6
##  29 229 140  12
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.000   4.000   4.000   4.329   5.000   6.000
##
## summary of quality measures:
##     support           confidence        coverage             lift
##  Min.   :0.001017   Min.   :0.8000   Min.   :0.001017   Min.   : 3.131
##  1st Qu.:0.001017   1st Qu.:0.8333   1st Qu.:0.001220   1st Qu.: 3.312
##  Median :0.001220   Median :0.8462   Median :0.001322   Median : 3.588
##  Mean   :0.001247   Mean   :0.8663   Mean   :0.001449   Mean   : 3.951
##  3rd Qu.:0.001322   3rd Qu.:0.9091   3rd Qu.:0.001627   3rd Qu.: 4.341
##  Max.   :0.003152   Max.   :1.0000   Max.   :0.003559   Max.   :11.235
##      count
##  Min.   :10.00
##  1st Qu.:10.00
##  Median :12.00
##  Mean   :12.27
##  3rd Qu.:13.00
##  Max.   :31.00
##
## mining info:
##       data ntransactions support confidence
##  Groceries          9835   0.001        0.8
```

In this output we are provided with the following information:

- There are 410 rules based on 0.001 support and 0.8 confidence thresholds.
- The distribution of the number of items in each rule (rule length distribution): Most rules are 4 items long.

We need use the `inspect()` function to see the actual rules.

```
inspect(rules[1:5])
```

```
##     lhs                     rhs              support confidence    coverage     lift
## [1] {rice,
##      sugar}              => {whole milk} 0.001220132          1 0.001220132 3.91364
## [2] {canned fish,
##      hygiene articles}   => {whole milk} 0.001118454          1 0.001118454 3.91364
## [3] {root vegetables,
```

```
##       butter,
##       rice}               => {whole milk} 0.001016777          1 0.001016777 3.913649    10
## [4] {root vegetables,
##       whipped/sour cream,
##       flour}              => {whole milk} 0.001728521          1 0.001728521 3.913649    17
## [5] {butter,
##       soft cheese,
##       domestic eggs}      => {whole milk} 0.001016777          1 0.001016777 3.913649    10
```

If we look at the confidence we see that for the top 5 rules it is 1, this indicates 100% confidence:

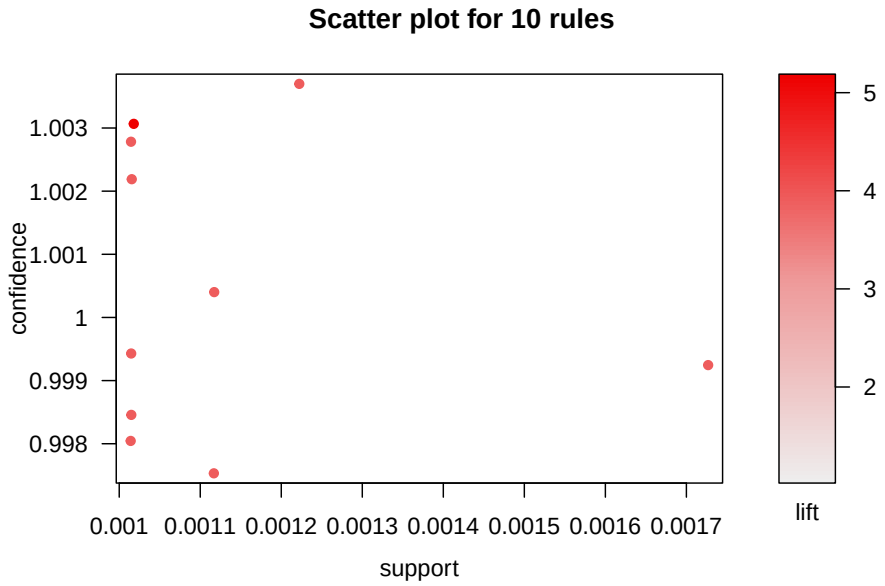
- 100% customers who bought "{rice, sugar}" end up buying "{whole milk}" as well.
- 100% customers who bought "{canned fish, hygiene articles}" end up buying "{whole milk}" as well.

In the following section we will look at visualizing the rules.

2.5.1 Visualisation of the Rules

```
topRules <- rules[1:10]
plot(topRules)
```

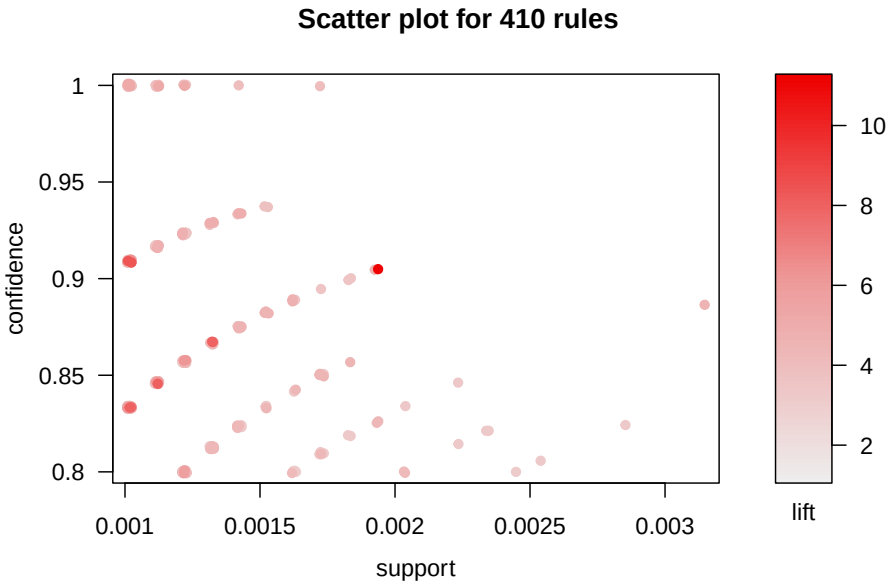
```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```



The scatter plot of support and confidence of the top ten rules shows us that high confidence rules have low support values.

```
plot(rules)
```

```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```



In the following section we will look at removing redundant rules.

2.5.2 Removing redundant rules

You may want to remove rules that are subsets of larger rules. Use the code below to remove such rules:

```
subset.rules <- which(colSums(is.subset(rules, rules)) > 1) # get subset rules in vector
# is.subset() determines if elements of one vector contain all the elements of other
length(subset.rules)
```

```
## [1] 91
```

```
subset.rules <- rules[-subset.rules] # remove subset rules.
```

2.5.3 Finding rules related to given items

In the case of specific product in interest, either as a precedent (LHS) or as a consequent (RHS) in the rule, you need to set the "`appearance=`" parameter in the apriori rule generating function:

Let us say we are interested in those transactions that end up in buying "root vegetables":

```
rveg.rules <- apriori(Groceries, parameter = list(supp=0.001, conf=0.8),appearance = li
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5   0.001      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 9
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [157 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 done [0.02s].
## writing ... [5 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
inspect(head(rveg.rules))
```

```
##     lhs                          rhs                  support confidence    coverage
## [1] {other vegetables,
##      whole milk,
##      yogurt,
##      rice}                    => {root vegetables} 0.001321810  0.8666667 0.001525165
## [2] {tropical fruit,
##      other vegetables,
##      whole milk,
##      oil}                     => {root vegetables} 0.001321810  0.8666667 0.001525165
## [3] {beef,
##      citrus fruit,
##      tropical fruit,
##      other vegetables}        => {root vegetables} 0.001016777  0.8333333 0.001220132
## [4] {citrus fruit,
##      other vegetables,
##      soda,
##      fruit/vegetable juice} => {root vegetables} 0.001016777  0.9090909 0.001118454
## [5] {tropical fruit,
```

```
##       other vegetables,
##       whole milk,
##       yogurt,
##       oil}                    => {root vegetables} 0.001016777  0.9090909 0.001118454 8.340400
```

2.6 Using your own dataset stored as a csv file

You might want to use a dataset from a csv file. The format of this file should be as follows:

- Transactions in the rows (remember in our small example, we had 5 transactions.)
- Items per transaction should be entered separately in different columns (items were A, B, C, D, E, and F)

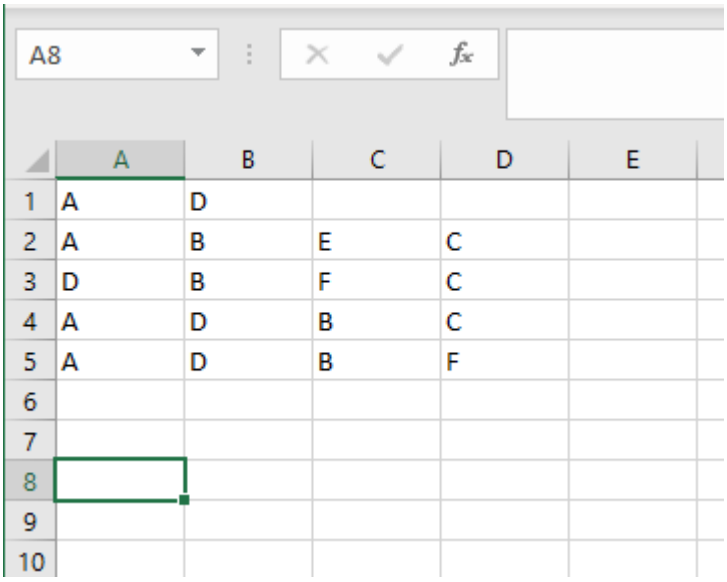


Figure 2.1: How the data looks like in csv format:

- The data should be extracted using the `read.transactions()` function.

```
slideExample <- read.transactions('C:/Users/01438475/Google Drive/UCTcourses/Analytics/Unsupervis
slideExample
```

```
## transactions in sparse format with
```

```
##  5 transactions (rows) and
##  6 items (columns)
```

```
inspect(head(slideExample, 6))
```

```
##     items
## [1] {A,D}
## [2] {A,B,C,E}
## [3] {B,C,D,F}
## [4] {A,B,C,D}
## [5] {A,B,D,F}
```

```
size(head(slideExample))
```

```
## [1] 2 4 4 4 4
```

I will leave all the rest for you to obtain.

2.7 References:

- R and Data Mining
- Susan Li - MBA
- Datacamp
- Dr Juwa Nyirenda's lecture notes

Chapter 3

Cluster Analysis

We will use the built-in R dataset USArrest which contains statistics, in arrests per 100,000 residents for assault, murder, and rape in each of the 50 US states in 1973. It includes also the percent of the population living in urban areas. It contains 50 observations on 4 variables:

3.1 Prerequisites

We will need the following packages:

```
library(cluster)
library(NbClust)
library(fpc)
```

Load the data set

```
data("USArrests")
# Remove any missing value (i.e, NA values for not available)
# That might be present in the data
df <- na.omit(USArrests)
# View the firt 6 rows of the data
head(df, n = 6)
```

```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
```

```
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
```

Before clustering is done, we can compute some descriptive statistics for the data

```
desc_stats <- data.frame(
Min = apply(df, 2, min), # minimum
Med = apply(df, 2, median), # median
Mean = apply(df, 2, mean), # mean
SD = apply(df, 2, sd), # Standard deviation
Max = apply(df, 2, max) # Maximum
)
desc_stats <- round(desc_stats, 1)
head(desc_stats)
```

```
##           Min   Med  Mean   SD   Max
## Murder    0.8   7.2   7.8  4.4  17.4
## Assault  45.0 159.0 170.8 83.3 337.0
## UrbanPop 32.0  66.0  65.5 14.5  91.0
## Rape      7.3  20.1  21.2  9.4  46.0
```

Note that the variables have large different means and variances. Therfore we need to standardise them.

```
df <- scale(df)
head(df)
```

```
##                Murder   Assault   UrbanPop         Rape
## Alabama    1.24256408 0.7828393 -0.5209066 -0.003416473
## Alaska     0.50786248 1.1068225 -1.2117642  2.484202941
## Arizona    0.07163341 1.4788032  0.9989801  1.042878388
## Arkansas   0.23234938 0.2308680 -1.0735927 -0.184916602
## California 0.27826823 1.2628144  1.7589234  2.067820292
## Colorado   0.02571456 0.3988593  0.8608085  1.864967207
```

For partition clustering methods we will assume that K=2 clusters

3.2 K-means clustering

We will use the kmeans() function in the stats package.

```
set.seed(123)
km.out <- kmeans(df, 2, nstart = 25)
# k-means group number of each observation
km.out$cluster
```

```
##        Alabama         Alaska        Arizona       Arkansas     California
##              1              1              1              2              1
##       Colorado    Connecticut       Delaware        Florida        Georgia
##              1              2              2              1              1
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              2              2              1              2              2
##         Kansas       Kentucky      Louisiana          Maine       Maryland
##              2              2              1              2              1
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##              2              1              2              1              1
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey
##              2              2              1              2              2
##     New Mexico       New York North Carolina   North Dakota           Ohio
##              1              1              1              2              2
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
##              2              2              2              2              1
##   South Dakota      Tennessee          Texas           Utah        Vermont
##              2              1              1              2              2
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming
##              2              2              2              2              2
```

3.3 K-medoids clustering

We will use the pam() in the cluster package.

```
pam.out <- pam(df, 2)
pam.out$cluster
```

```
##        Alabama         Alaska        Arizona       Arkansas     California
##              1              1              1              2              1
##       Colorado    Connecticut       Delaware        Florida        Georgia
##              1              2              2              1              1
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              2              2              1              2              2
##         Kansas       Kentucky      Louisiana          Maine       Maryland
##              2              2              1              2              1
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##              2              1              2              1              1
```

```
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey
##              2              2              1              2              2
##     New Mexico       New York North Carolina   North Dakota           Ohio
##              1              1              1              2              2
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
##              2              2              2              2              1
##   South Dakota      Tennessee          Texas           Utah        Vermont
##              2              1              1              2              2
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming
##              2              2              2              2              2
```

3.4 Hierarchical Clustering

Here the built-in R function hclust() is used:

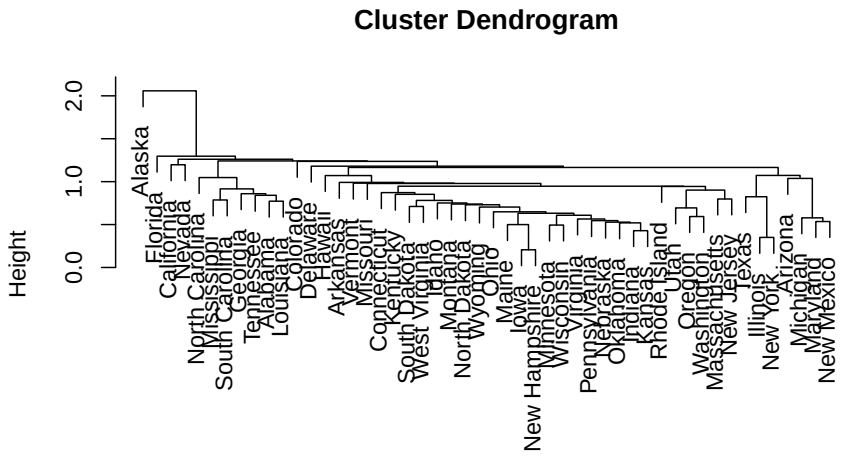
3.4.1 Compute pairewise distance matrices

```
dist.out <- dist(df, method = "euclidean")
```

3.4.2 Single Linkage

```
out.single.euc <- hclust(daisy(df,metric="euclidean"),method="single")

# try other metric="euclidean"
plot(out.single.euc)
```




```
 # decide to cut the tree at height 1
out.single.euc <- cutree(out.single.euc, h=1.5)
 # view cluster allocation
names (out.single.euc) <- rownames(df)
sort(out.single.euc)
```

```
##        Alabama        Arizona       Arkansas     California       Colorado
##              1              1              1              1              1
##    Connecticut       Delaware        Florida        Georgia         Hawaii
##              1              1              1              1              1
##          Idaho       Illinois        Indiana           Iowa         Kansas
##              1              1              1              1              1
##       Kentucky      Louisiana          Maine       Maryland  Massachusetts
##              1              1              1              1              1
##       Michigan      Minnesota    Mississippi       Missouri        Montana
##              1              1              1              1              1
##       Nebraska         Nevada  New Hampshire     New Jersey     New Mexico
##              1              1              1              1              1
##       New York North Carolina   North Dakota           Ohio       Oklahoma
##              1              1              1              1              1
##         Oregon   Pennsylvania   Rhode Island South Carolina   South Dakota
##              1              1              1              1              1
##      Tennessee          Texas           Utah        Vermont       Virginia
##              1              1              1              1              1
```

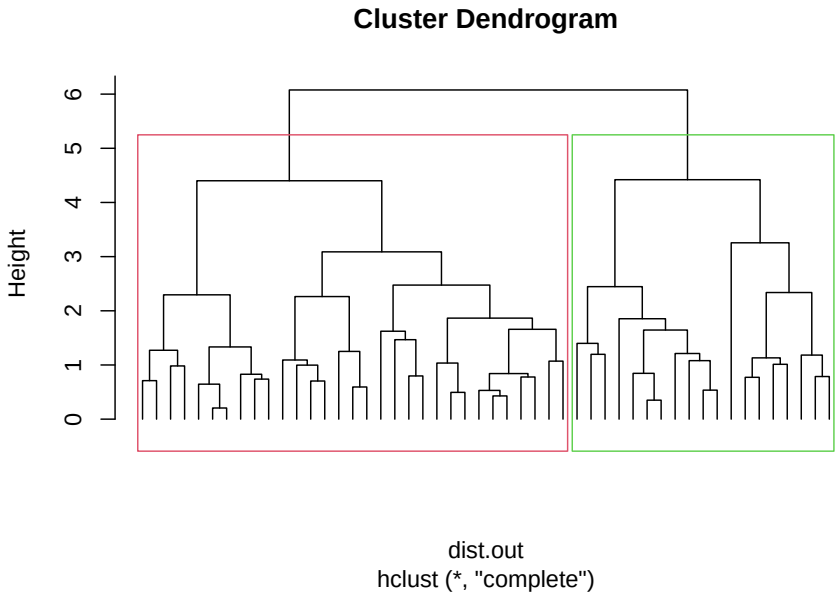
```
##     Washington  West Virginia      Wisconsin        Wyoming         Alaska
##              1              1              1              1              2
```

3.4.3 Complete Linkage

```
hc <- hclust(dist.out, method = "complete")
```

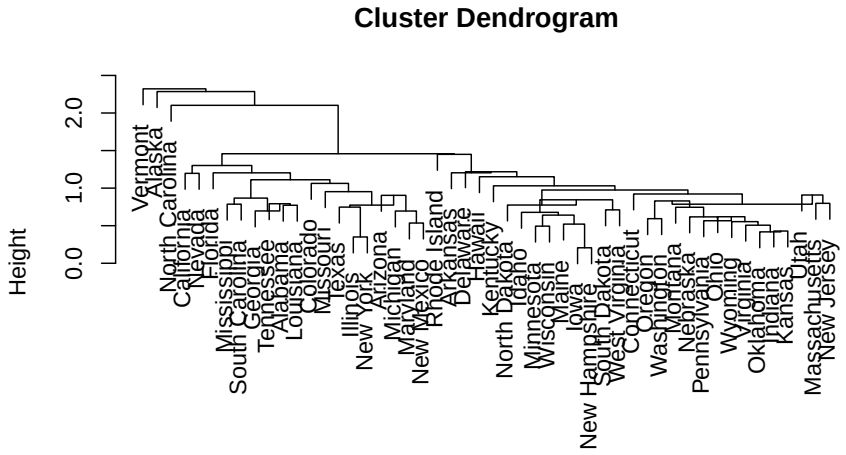
Visualization of hclust

```
plot(hc, labels = F,-1)
rect.hclust(hc, k = 2, border = 2:3) # Add rectangle around 2 clusters, try with 3?
```

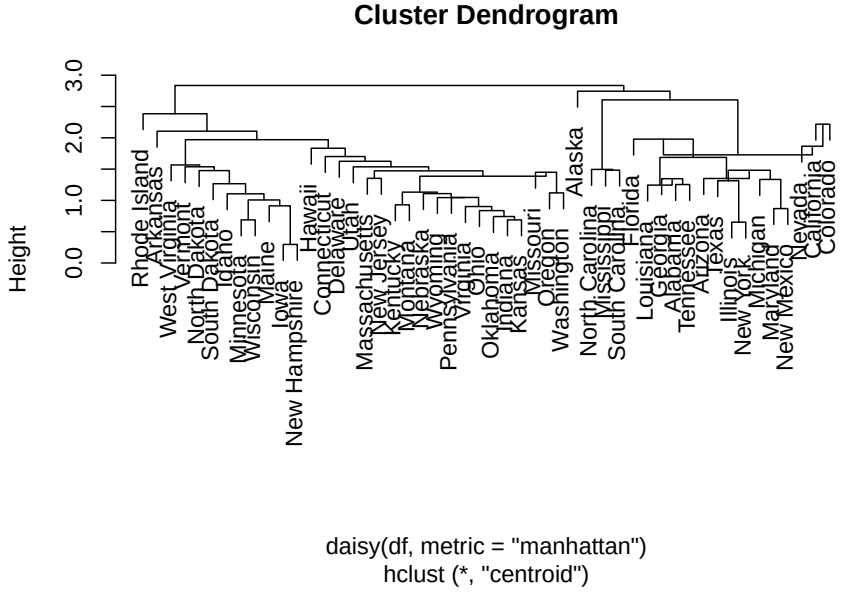


3.4.4 Centroid

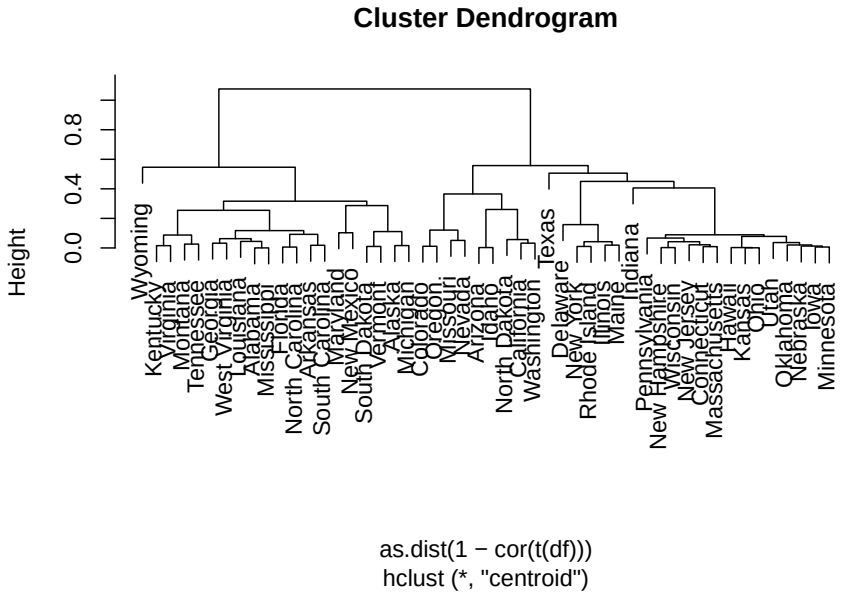
```
# Centroid clustering
out.centroid.euc <- hclust(daisy(df,metric="euclidean"),method="centroid")
plot(out.centroid.euc)
```




```
out.centroid.city <- hclust(daisy(df,metric="manhattan"),method="centroid")
plot(out.centroid.city)
```



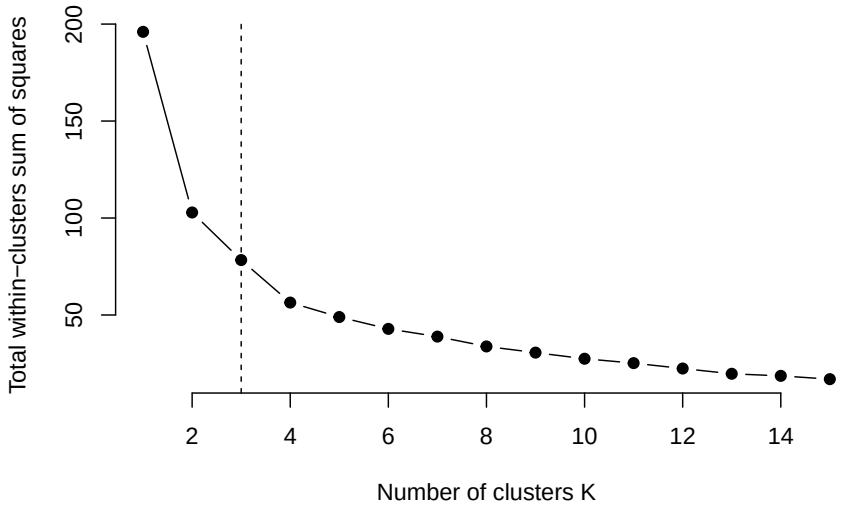
```
out.centroid.cor <- hclust(as.dist(1-cor(t(df))),method="centroid")
plot(out.centroid.cor)
```



3.5 Methods for determining number of clusters

3.5.1 Elbow method for k-means clustering

```
set.seed(123)
# Compute and plot wss for k = 2 to k = 15
k.max <- 15 # Maximal number of clusters
df.out <- df
wss <- sapply(1:k.max,
function(k){kmeans(df.out, k, nstart=10 )$tot.withinss})
plot(1:k.max, wss, type="b", pch = 19, frame = FALSE, xlab="Number of clusters K", ylab
abline(v = 3, lty =2)
```

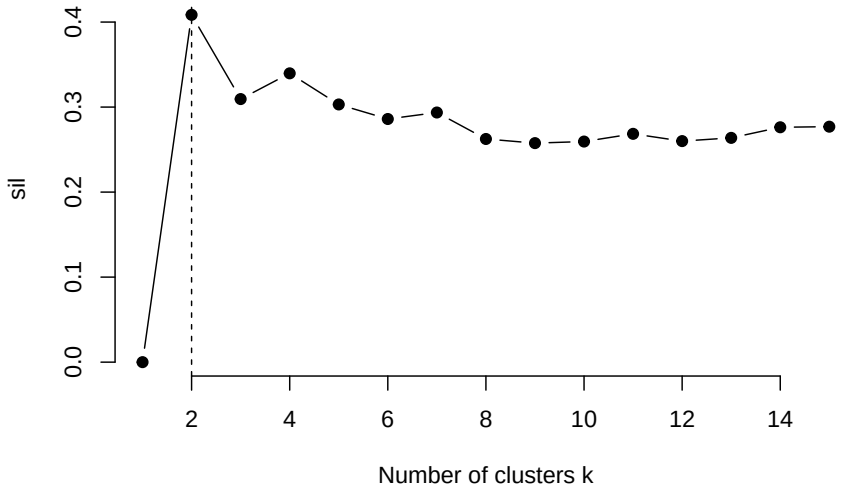


According to the elbow method, the optimal number of clusters suggested for the K-means algorithm is 3.

3.5.2 Average silhouette method for k-means clustering

```
k.max <- 15
data.out <- df
sil <- rep(0, k.max)
# Compute the average silhouette width for
# k = 2 to k = 15
for(i in 2:k.max){
km.res <- kmeans(df.out, centers = i, nstart = 25)
ss <- silhouette(km.res$cluster, dist(df.out))
sil[i] <- mean(ss[, 3])
}
```

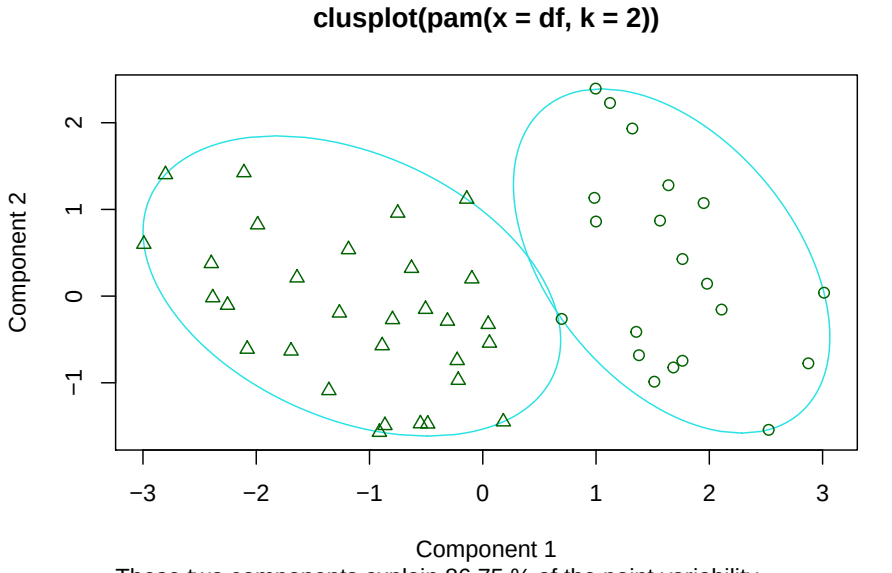
```
# Plot the average silhouette width
plot(1:k.max, sil, type = "b", pch = 19,
frame = FALSE, xlab = "Number of clusters k")
abline(v = which.max(sil), lty = 2)
```



According to the silhouette method the optimal number of clusters suggested for the Kmeans algorithm is 2.

3.5.3 Average silhouette method for PAM clustering

```
#clusplot(pam.out, main = "Cluster plot, k = 2", color = TRUE)
plot(pam.out)
```



These two components explain 86.75% of the point variability.

This table shows how to use the average silhouette width value:

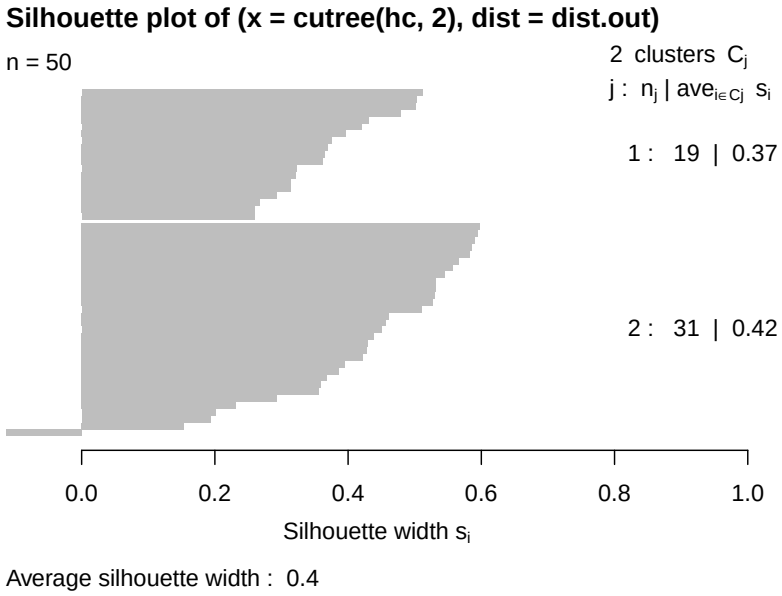
Range of SC : Interpretation 0.71-1.0 : A strong structure has been found 0.51-0.70 : A reasonable structure has been found 0.26-0.50 : The structure is weak

and could be artificial < 0.25 : No substantial structure has been found

According to the table, the fit is weak.

3.5.4 Average silhouette method for hierarchical clustering

```
plot(silhouette(cutree(hc,2),dist.out))
```



Average silhouette width : 0.4

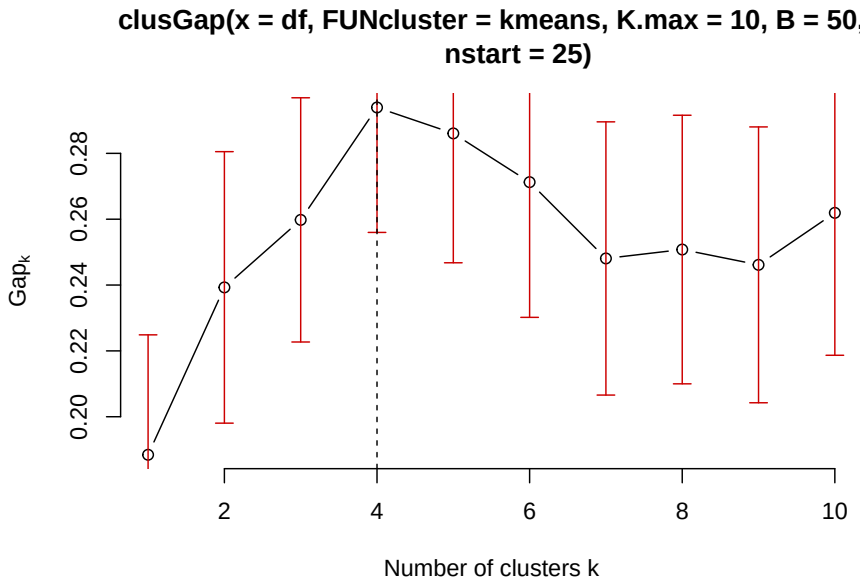
This table shows how to use the average silhouette width value:

Range of SC: Interpretation 0.71-1.0 : A strong structure has been found 0.51-0.70 : A reasonable structure has been found 0.26-0.50 : The structure is weak and could be artificial < 0.25 : No substantial structure has been found

The result for hierarchical clustering is similar to that of PAM. The conclusion we can make is that fit is weak.

3.5.5 Gap Statistic for K means clustering

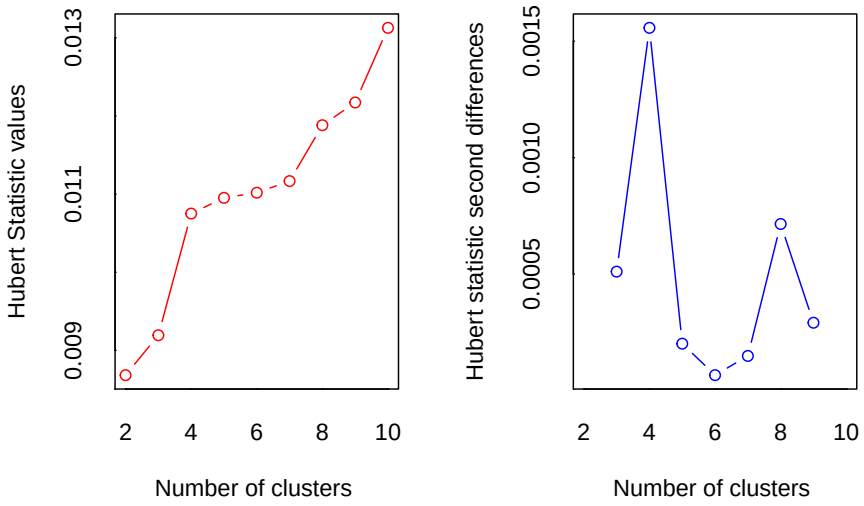
```
# Compute gap statistic
gap_stat <- clusGap(df, FUN = kmeans, nstart = 25, K.max = 10, B = 50)
# Print the result
plot(gap_stat, frame = FALSE, xlab = "Number of clusters k")
abline(v = 4, lty = 2)
```



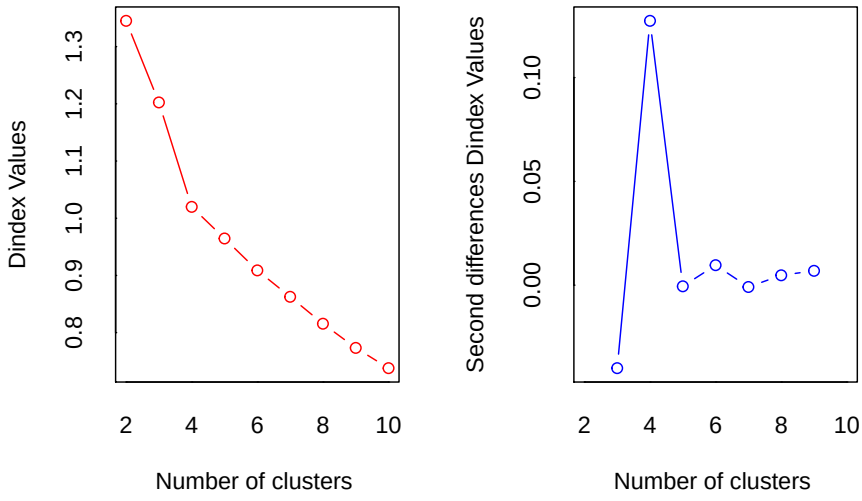
According to the Gap Statistic the 'optimal number of clusters chosen for the Kmeans algorithm is 4!

Using the NbClust package which uses a vote to chose the number of clusters. The following example determine the number of clusters using all statistics:

```
res.nb <- NbClust(df, distance = "euclidean",min.nc = 2, max.nc
= 10, method = "complete", index ="all")
```




```
## *** : The Hubert index is a graphical method of determining the number of clusters.
##                 In the plot of Hubert index, we seek a significant knee that corresp
##                 significant increase of the value of the measure i.e the significant
##                 index second differences plot.
##
```



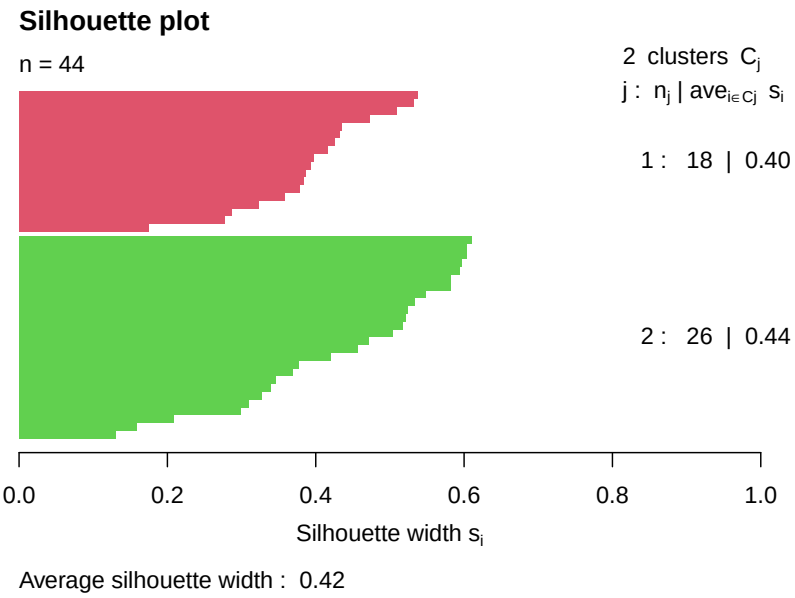

```
## *** : The D index is a graphical method of determining the number of clusters.
##                 In the plot of D index, we seek a significant knee (the significant peak in Di
##                 second differences plot) that corresponds to a significant increase of the val
##                 the measure.
##
## *******************************************************************
## * Among all indices:
## * 9 proposed 2 as the best number of clusters
## * 4 proposed 3 as the best number of clusters
## * 6 proposed 4 as the best number of clusters
## * 2 proposed 5 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 1 proposed 10 as the best number of clusters
##
##                     ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
##
##
## *******************************************************************
```

When all statistics in the NbClust package are allowed to vote, the majority (in this case 9 out of 23) propose that the 'optimal' number of clusters should be 2.

3.6 Clustering with CLARA

R function for computing CLARA is found in the in cluster package.

```
clarax <- clara(df, 2, samples=10)
# Silhouette plot
plot(silhouette(clarax), col = 2:3, main = "Silhouette plot")
```



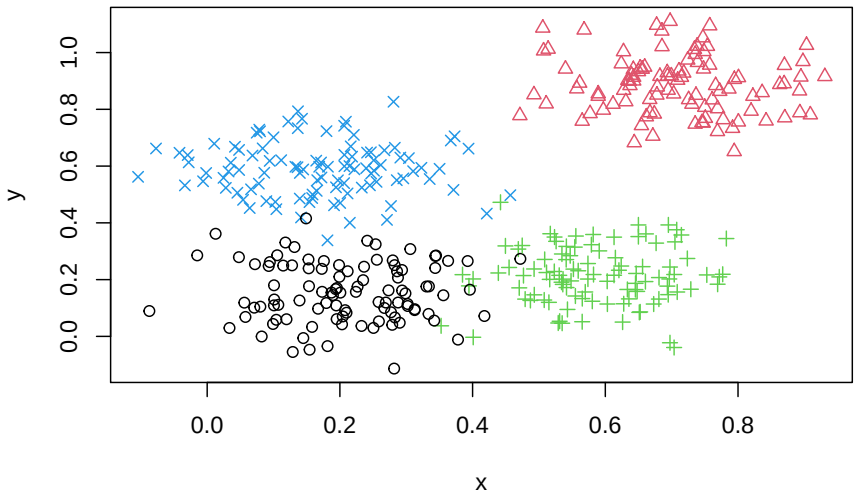
The overall average silhouette width is 0.42 meaning that the fit is weak (see table above showing range for Si and corresponding interpretation.

3.7 Clustering with DBSCAN

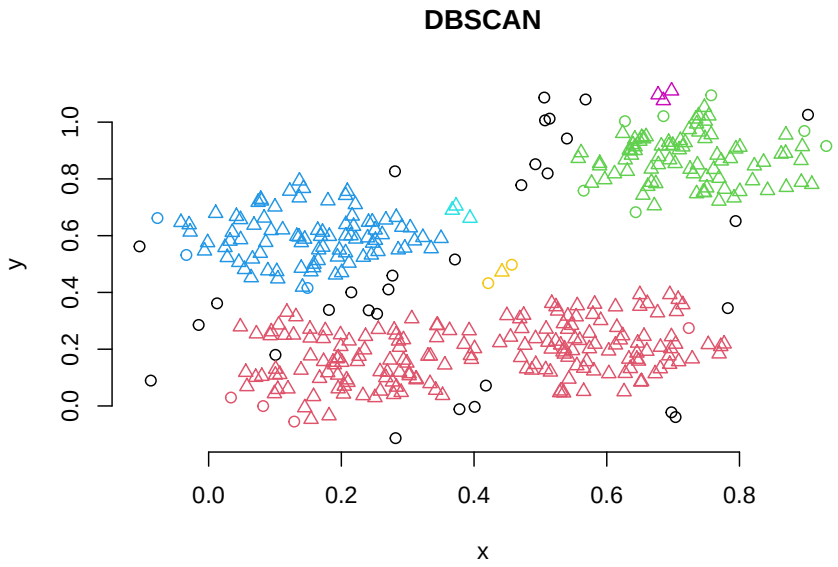
To illustrate the application of DBSCAN we will use a very simple artificial data set of four slightly overlapping Gaussians in two-dimensional space with 100 points each. We set the random number generator to make the results reproducible and create the data set as shown below. The function dbscan() is found in the fpc package.

```
set.seed(2)
n <- 400
x <- cbind(
x = runif(4, 0, 1) + rnorm(n, sd = 0.1),
```

```
y = runif(4, 0, 1) + rnorm(n, sd = 0.1)
)
true_clusters <- rep(1:4, time = 100)
plot(x, col = true_clusters, pch = true_clusters)
```




```
# To apply DBSCAN, we need to decide on the neighborhood radius eps
# and the density threshold minPts.
# The rule of thumb for minPts is to use at least the number of
# dimensions of the data set plus one. In our case, this is 3.
db <- fpc::dbscan(x, eps = .05, MinPts =3 )
# Plot DBSCAN results
plot(db, x, main = "DBSCAN", frame = FALSE)
```

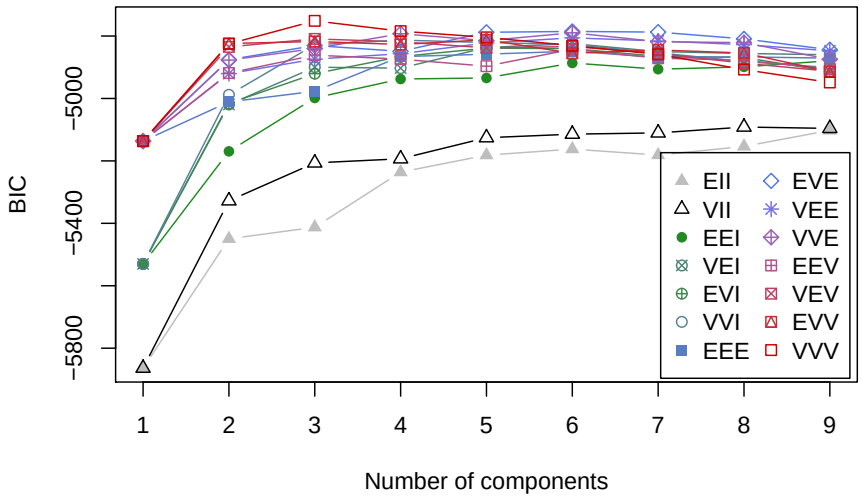


DBSCAN has found three clusters in the data.

3.8 Clustering using mixture models

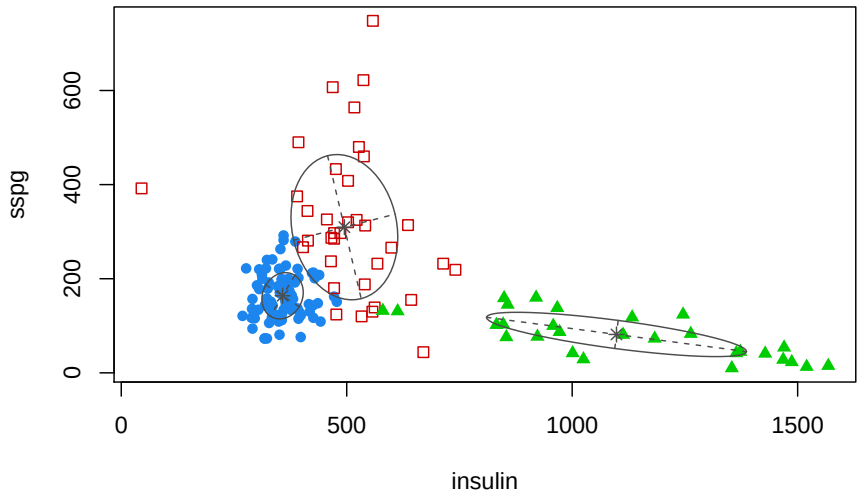
For this you need the function Mclust() in the mclust package. There are 14 model options available in the R package mclust. In one dimension though these collapse into only two models:E for equal variance and V for varying variance. In more dimensions, the model identifiers encode geometric characteristics of the model. For example, EVI denotes a model in which the volume of all clusters are equal(E), the shapes of the clusters may vary (V), and the orientation is the identity (I). That is, clusters in this model have diagonal covariances with orientation parallel to the coordinate axes.

```
library(mclust)
data("diabetes")
# Run the function to see how many clusters
# it finds to be optimal, set it to search for
# at least 1 model and up 20.
d_clust <- Mclust(diabetes[,-1])
plot(d_clust,diabetes[,-1],what="BIC")
```



The plot shows the results of mclust for the 10 available model parameterizations and up to 9 clusters for the diabetes dataset. The best model is considered to be the one with the highest BIC among the fitted models.

```
coordProj(diabetes[,-1],dimens = c(2,3),what="classification",classification
=d_clust$classification,parameters = d_clust$parameters)
```

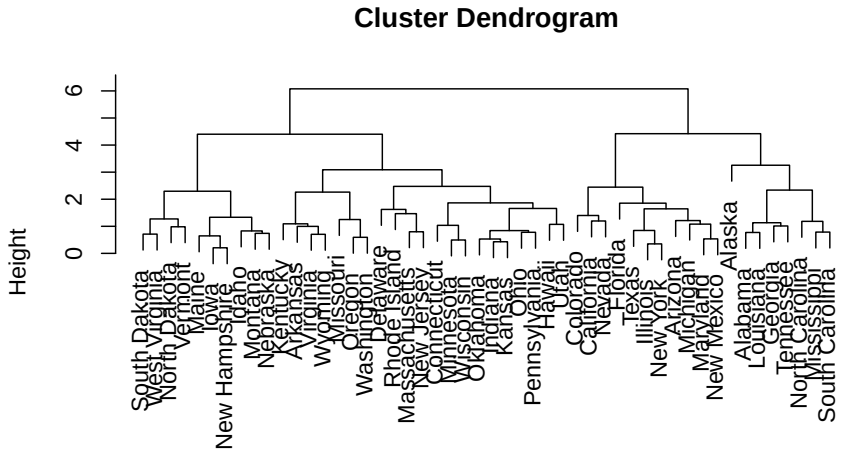


This plot shows the projection of the diabetes data with different symbols indicating the classification corresponding to the best model as determined by mclust. The component means are marked and ellipses with axes are drawn corresponding to their covariances. In this case there are three components, each with a different covariance. For more detailed interpretation see (C.Fraley and A.E. Raftery, Model based Methods of Classification:

Using the mclust Software in Chemometrics. Journal of Statistical Software, Vol. 18, 2007)

3.9 Cluster Profiling

```
out.complete.euc <- hclust(daisy(df,metric="euclidean"),method="complete")
plot(out.complete.euc)
```




```
out.complete.euc <- cutree(out.complete.euc, h=3.1)
clusvec <- out.complete.euc
```

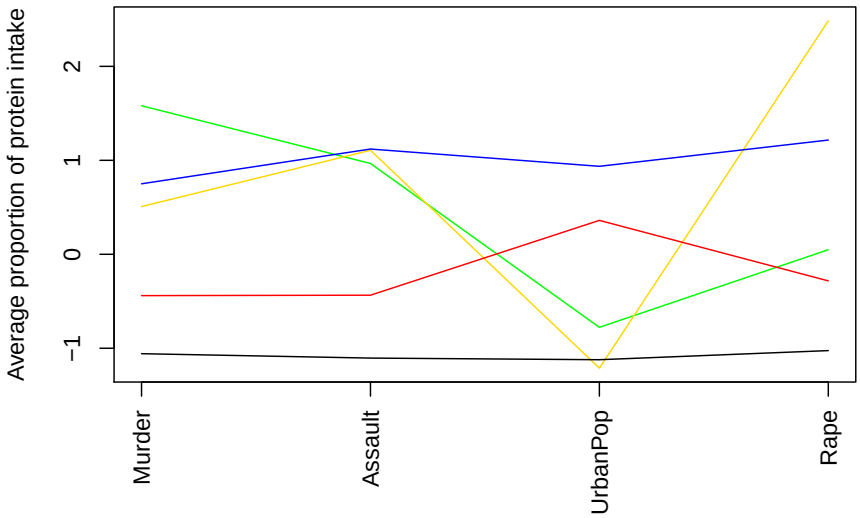
calculate means

```
class.means <- apply(df, 2, function(x) tapply (x, clusvec, mean))
class.means
```

```
##       Murder    Assault   UrbanPop        Rape
## 1  1.5803956  0.9662584 -0.7775109  0.04844071
## 2  0.5078625  1.1068225 -1.2117642  2.48420294
## 3  0.7499801  1.1199128  0.9361748  1.21564322
## 4 -0.4400338 -0.4353831  0.3607592 -0.28303852
## 5 -1.0579703 -1.1046626 -1.1219527 -1.02515543
```

plot means

```
plot (c(1,ncol(df)),range(class.means),type="n",xlab="",ylab="Average proportion of protein intak
axis (side=1, 1:ncol(df), colnames(df), las=2)
#ensure you list enough colours for the number of clusters
colvec <- c("green","gold","blue","red","black")
for (i in 1:nrow(class.means))
  lines (1:ncol(df),class.means[i,],col=colvec[i])
```


Average proportion of protein intake
−1
0
1
2
Murder
Assault
UrbanPop
Rape

Chapter 4

Self Organising Maps

For this section, please update your browser on the 20th of October Tuesday.