Modern statistics

October 28, 2025

Contents

Part I

Notes

Chapter 1

Introduction

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

Remark 1 (exam). Anche un esercizio teorico, con bassi punti, simile a quelli degli esercizi di casa.

1.1 Introduction

Remark 2. There is no unique definition of what a cluster is and research goes on even independently in many areas. Clustering is about finding groups in data.

Important remark 1. General intuition: clusters should be homogeneous, which could mean various things:

- observations within clusters being similar, and between different clusters dissimilar
- cluster based on high density regions
- every cluster generated by homogeneous probability model (e.g., Gaussian).

Remark 3. General **tasks** for which clustering is applied:

- exploratory data analysis looking for "interesting patterns",
- information reduction for simplification,
- comparing clustering in specific data with other (external) groupings or information.

Important remark 2. Clustering is often called **unsupervised classification**. There are also "supervised classification" methods (which rely on already classified units), such as Linear Discriminant Analysis, nearest neighbour classifier, for classifying new observations to classes, having training observations with already known true classes.

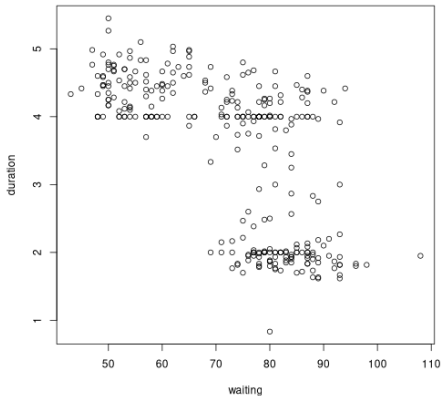


Figure 1.1: Old faithful geyser data

1.1.1 Example data sets

1.1.1.1 Old faithful geyser

Example 1.1.1. These are `waiting` times from previous eruption and `duration` of current eruption of geyser.

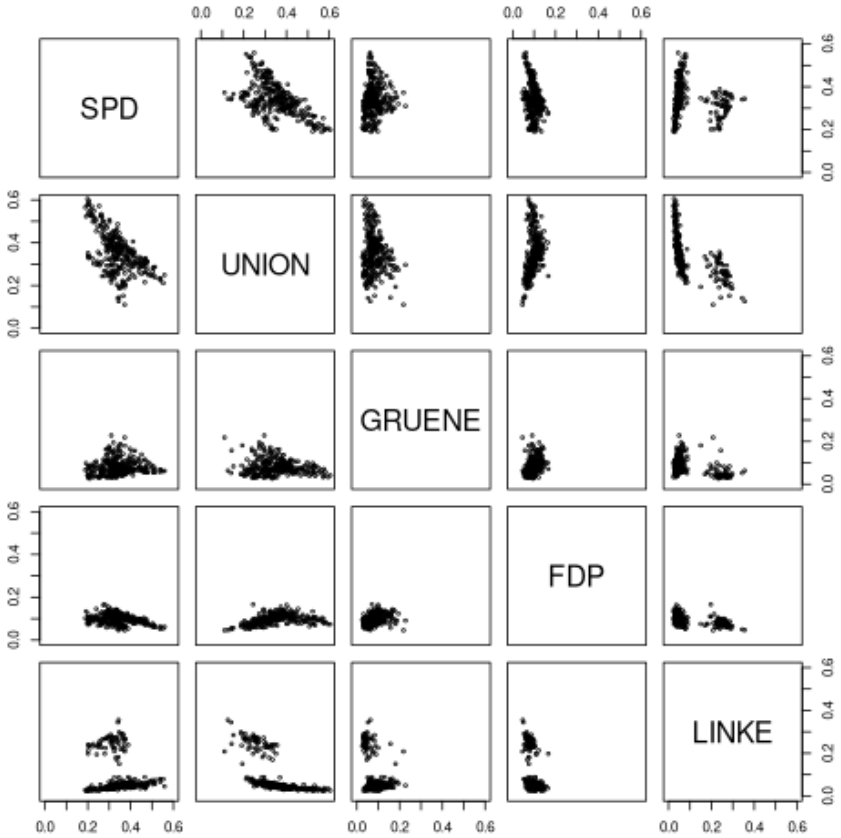
```
library(MASS)
data(geyser) # also available in read.table("data/geyser.dat", header = TRUE)
str(geyser)

## 'data.frame': 299 obs. of  2 variables:
##  $ waiting : num  80 71 57 80 75 77 60 86 77 56 ...
##  $ duration: num  4.02 2.15 4 4 4 ...

plot(geyser)
```

1.1.1.2 German bundestag election 2005

Example 1.1.2 (German bundestag election 2005)**.** Data on percentages of elections in Germany bundestag. `ewb` is east/west/berlin, `state` is federal state. Ogni riga è una regione di uno stato (regione che come rowname è andata persa nel salvataggio in csv ma pace)

```
p05 <- read.table("data/bundestag.dat", header = TRUE)
head(p05) # percentage of votes to parties, ewb (east/west/berlin area), state = fede

##         SPD     UNION     GRUENE        FDP      LINKE             state  ewb
## 1 0.3905845 0.3636154 0.08034580 0.09689123 0.04798175 Schleswig-Holstein West
## 2 0.3620308 0.4165538 0.06247530 0.10069150 0.03863251 Schleswig-Holstein West
## 3 0.3631324 0.3896104 0.06558515 0.10738025 0.04592932 Schleswig-Holstein West
## 4 0.3763323 0.3810651 0.08045083 0.09890993 0.04159996 Schleswig-Holstein West
```

```
## 5 0.4146330 0.2882370 0.12749007 0.08852824 0.05972804 Schleswig-Holstein West
## 6 0.3948488 0.3626132 0.07669667 0.09597278 0.04455005 Schleswig-Holstein West

pairs(p05[1:5], xlim = c(0, 0.6), ylim = c(0, 0.6), cex = 0.5)
```




```
## Note the xlim, ylim parameters, set to the same values for all
## plots in order to show which parties are big and which are small.
## cex=0.5 makes plot symbols smaller.
```

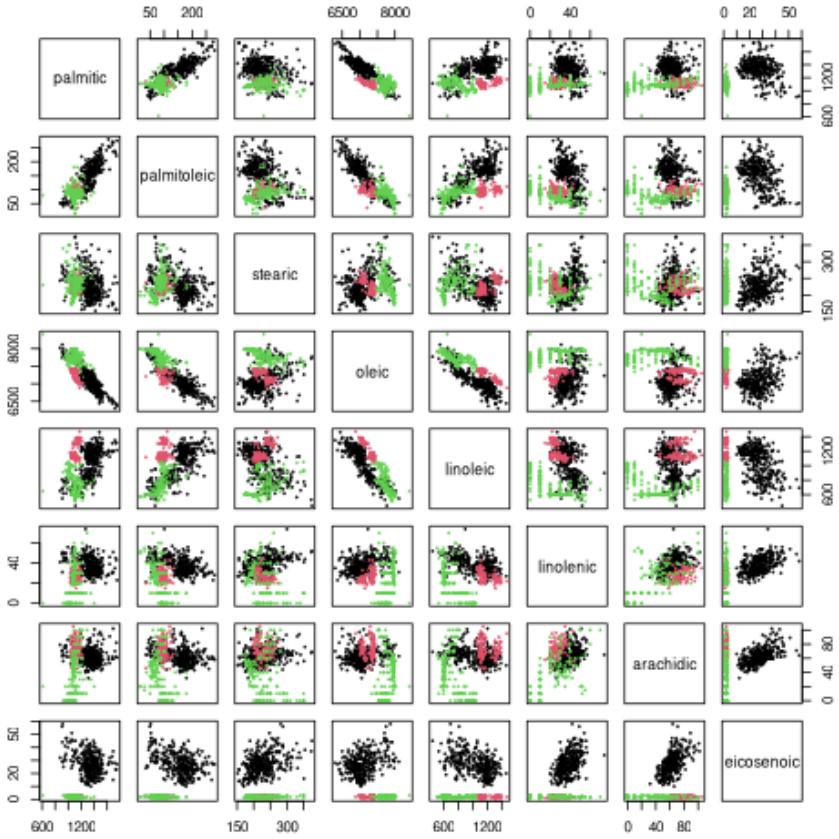
Il dataset è stato generato così

```
## ## how the dataset was generated
## library(flexclust)
## p05 <- bundestag(2005)
## state <- bundestag(2005, state = TRUE)
## ewb <- rep("West", 299)
## ewb[state == "Berlin"] <- "Berlin"
## ewb[state %in% c("Brandenburg", "Mecklenburg-Vorpommern", "Sachsen",
##                  "Sachsen-Anhalt", "Thueringen")] <- "East"
```

1.1.1.3 Olive oil

Example 1.1.3 (Olive oil)**.** Contrary to previous cases, here we have data on oil types with a grouping provided (`macro.area` and `region`). Here we may be interested in clustering and to see if the output overlap somehow the given group (`macro.area` or `region`); in machine learning this is one of the way to evaluate clustering algorithm

```
library(pdfCluster)

## pdfCluster 1.0-4

data(oliveoil) ## oliveoil <- read.table("data/oliveoil.dat", header=TRUE)
str(oliveoil)

## 'data.frame': 572 obs. of  10 variables:
##  $ macro.area : Factor w/ 3 levels "South","Sardinia",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ region     : Factor w/ 9 levels "Apulia.north",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ palmitic   : int  1075 1088 911 966 1051 911 922 1100 1082 1037 ...
##  $ palmitoleic: int  75 73 54 57 67 49 66 61 60 55 ...
##  $ stearic    : int  226 224 246 240 259 268 264 235 239 213 ...
##  $ oleic      : int  7823 7709 8113 7952 7771 7924 7990 7728 7745 7944 ...
##  $ linoleic   : int  672 781 549 619 672 678 618 734 709 633 ...
##  $ linolenic  : int  36 31 31 50 50 51 49 39 46 26 ...
##  $ arachidic  : int  60 61 63 78 80 70 56 64 83 52 ...
##  $ eicosenoic : int  29 29 29 35 46 44 29 35 33 30 ...

## plot of chemical variables
olive <- oliveoil[, 3:10]
## pairs(olive, cex = 0.3) #plot
pairs(olive, cex = 0.3, col = oliveoil[, 1]) # plot with coloring by "macro areas"
```

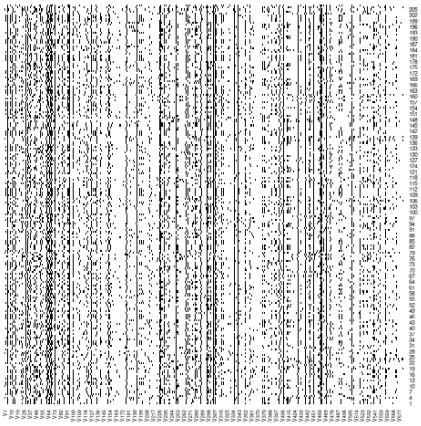


1.1.1.4 Veronica

Example 1.1.4 (Veronica dataset)**.** Veronica plants: rows are different type of veronica plants, all 0/1 variables (1 is dominant marker), very high dimensional dataset (583 variables, typical genetic dataset). The aim was to generate cluster to determine "species" of veronica plants based on dominant markers.

```
veronica <- read.table("data/veronica.dat")
dim(veronica)

## [1] 207 583

head(veronica[1:5], n = 5)

##   V1 V2 V3 V4 V5
## 1  0  0  1  0  1
## 2  0  0  1  0  1
## 3  0  0  0  0  1
## 4  1  0  1  0  1
## 5  0  0  0  0  0
```

```
## Some things can better be done on matrices:
veronicam <- as.matrix(veronica)
## below rows are plants, columns are variables
heatmap(veronicam, Rowv = NA, Colv = NA, col = grey(seq(1, 0, -0.01)), scale="none")
```



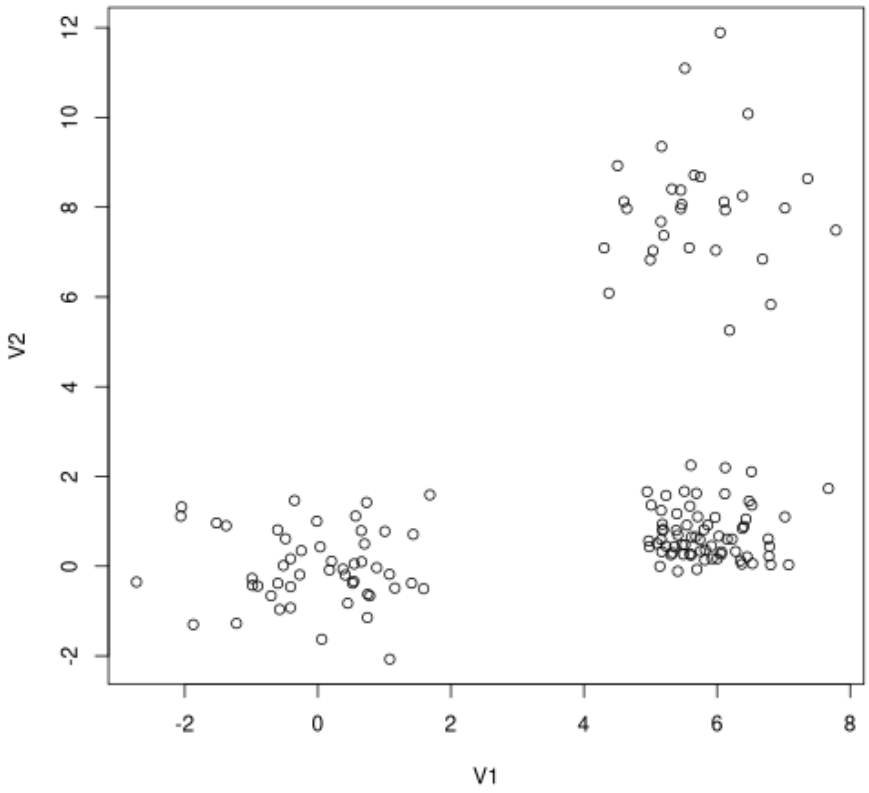
The 0/1 variable are not informative for creation of species/groups/cluster. This plot of all data is fairly uselesss (is like heatmap)

scale= none

1.1.1.5 Artificial datasets

Example 1.1.5 (Artificial dataset 1)**.** Simulated dataset

```
clusterdata1 <- read.table("data/clusterdata1.dat")
with(clusterdata1, plot(V1, V2)) # its a 3-cluster.
```



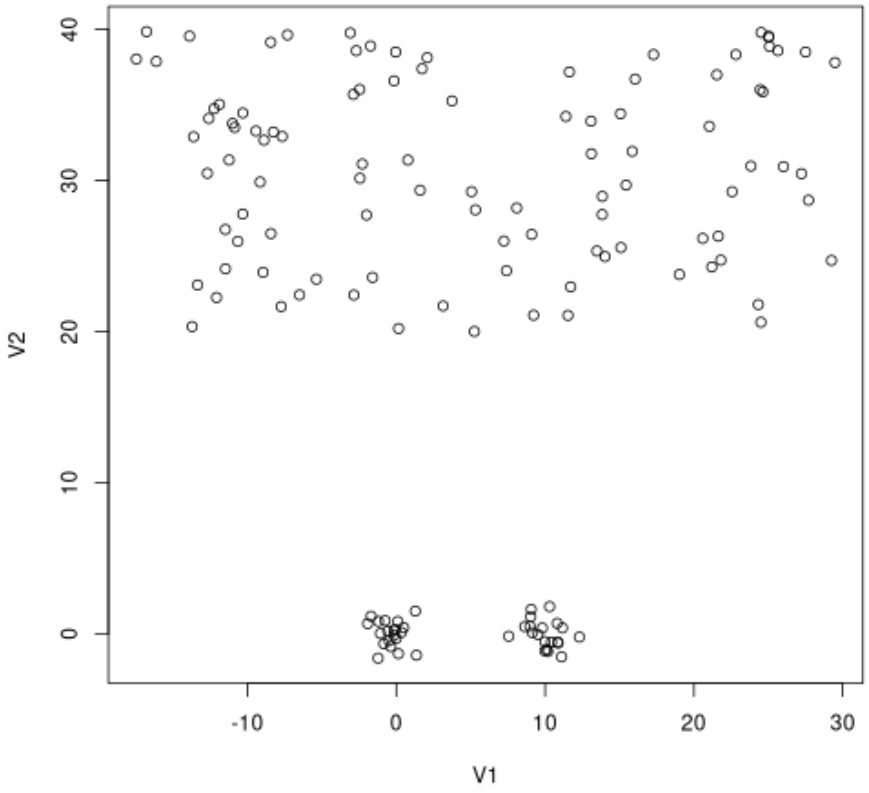

```
## This is how data set was generated.

## library(sn)
## set.seed(665544)
## v1 <- c(rnorm(50,0,1), rsn(70,5,1,8), rnorm(30,6,1))
## v2 <- c(rnorm(50,0,1), rsn(70,0,1,8), 8+rt(30,5))
## clusterdata1 <- cbind(v1,v2)
```

We use `clusterdata1.dat` because now the code above give other data since `rsn` (random number generator of skewed normal) has changed its precise handling of random numbers in an update, which means that the code above will produce a slightly different data set now, despite the `set.seed`.

Example 1.1.6 (Artificial dataset 2)**.** Another case of three cluster.

```
set.seed(77665544)
x1 <- rnorm(20)
y1 <- rnorm(20)
x2 <- rnorm(20,mean=10)
y2 <- rnorm(20)
```

```
x3 <- runif(100,-20,30)
y3 <- runif(100,20,40)
clusterdata2 <- data.frame("V1" = c(x1,x2,x3), "V2"= c(y1,y2,y3))
with(clusterdata2, plot(V1, V2)) # again its a 3-cluster
```

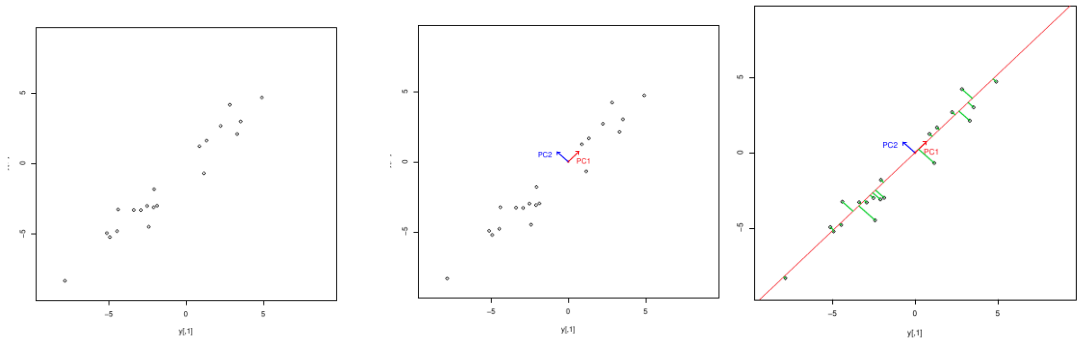


In this case the three cluster are characterized by different variability (two are strict, the above one has a lot of variability inside and some units are nearer to the cluster bolow than to units at the opposite side of the cluster).

1.2 PCA

If we want to visualize on two dimension a multidimensional dataset, one option is PCA. It's a dimension reduction technique aimed at finding most *informative* dimensions in data (where as information one consider data variance).
More precisely, after centering variables:

$$Z_1 = X_1 - \bar{X}_1$$
$$Z_2 = X_2 - \bar{X}_2$$
$$\dots$$
$$Z_p = X_p - \bar{X}_p$$



(a) (b) (c)

Figure 1.2: PCA with two variable and two components

we define:

- **PC1**: the first principal component is a linear combination (l.c.)

$$Y_1 = a_{11}Z_1 + a_{21}Z_2 + \ldots + a_{p1}Z_p$$

  so that the Var $[Y_1]$ max among all l.c. with $\|\mathbf{a}\| = 1$;

- **PC2**: the second principal component

$$Y_2 = a_{21}Z_1 + a_{22}Z_2 + \ldots + a_{2p}Z_p$$

  is *orthogonal* to the first Y_1 and choosen such as Var $[Y_2]$ is max among all combinations with $\|\mathbf{a}\| = 1$;

- **PC3** etc: same thing *orthogonal* to all what's already there.

Example 1.2.1. It amounts to changing the coordinate system in a dataset (see fig 1.2 for a toy example of two PC out of two variables). PC values are projections on PCs and the sum of variances of PCs is equal to the sum of all variances in the original variable. In the case of image, the variance of PC1 is 98.5% of overall sum.

Important remark 3 (Derivation). PCA can equivalently be derived as eigendecomposition of covariance matrix Σ:

$$\Sigma = D\Delta D^T$$

with:

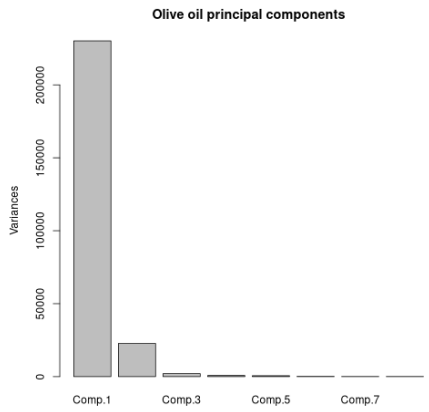
- D containing Principal Components (eigenvectors of Σ)
- Δ diagonal matrix of ordered eigenvalues (PC variances).

For standardised variables, Σ is the correlation matrix.

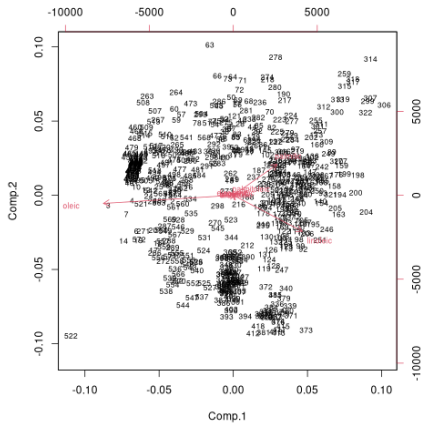
Important remark 4 (Problem with PCA). Are the Principal Components really the most informative dimensions? PCA identifies "information" with large variance. This is not necessarily appropriate; there is no guarantee that these dimensions are most informative for clustering or regression. However it makes sense as a "first guess".

Example 1.2.2 (Olive oil)**.** Lets see principal component with olive data

```
prolive <- princomp(olive)  ## Also prcomp does PCA: this is
                            ## preferred by some sources.
summary(prolive) ## Almost 90% of variance of the dataset is

## Importance of components:
##                             Comp.1       Comp.2       Comp.3       Comp.4
## Standard deviation     479.7299024 150.82827868 45.394449751 27.522646558
## Proportion of Variance   0.8970072   0.08866821  0.008031707  0.002952451
## Cumulative Proportion    0.8970072   0.98567544  0.993707152  0.996659603
##                             Comp.5       Comp.6       Comp.7       Comp.8
## Standard deviation     24.78169442 1.196956e+01 7.1390744088 6.9756965249
## Proportion of Variance  0.00239367 5.584168e-04 0.0001986489 0.0001896608
## Cumulative Proportion   0.99905327 9.996117e-01 0.9998103392 1.0000000000

                  ## synthesized in the first component
plot(prolive, main = "Olive oil principal components") ## Variance captured
```




```
biplot(prolive, cex = 0.7) ## Biplot with variable axes
```



Biplot show the scatterplot of the first two components (rowlabels instead of points ma amen); but shows the linear combination of original variables: the long arrow in negative direction for `oleic` is negatively associated to the first component.
It seems all information is in 2-d, mainly oleic vs. linoleic/palmitoleic. Same kind of info should be visualized using loadings

```
prolive$loadings

##
## Loadings:
##             Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## palmitic     0.284  0.637  0.451         0.452  0.146  0.238  0.160
## palmitoleic                0.165  0.574 -0.669  0.325  0.156  0.219
## stearic                   -0.724  0.397  0.405  0.254  0.213  0.208
## oleic       -0.843 -0.169  0.337         0.199  0.141  0.226  0.169
## linoleic     0.447 -0.744  0.304         0.247  0.107  0.220  0.170
## linolenic                         -0.293 -0.111 -0.216 -0.158  0.907
## arachidic                 -0.142 -0.636 -0.192  0.665  0.308
## eicosenoic                -0.113        -0.183 -0.537  0.808
##
##                Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## SS loadings     1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000
## Proportion Var  0.125  0.125  0.125  0.125  0.125  0.125  0.125  0.125
## Cumulative Var  0.125  0.250  0.375  0.500  0.625  0.750  0.875  1.000

sum(prolive$loadings[, 1]^2) # these are the a, i guess: here the sum of them squared is 1

## [1] 1
```

But these variables have larger values (and variance) in general,

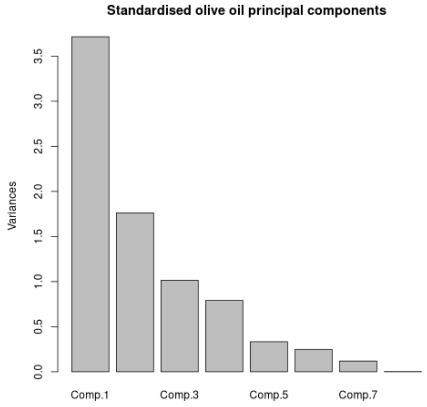
```
sapply(olive, sd)

##    palmitic palmitoleic     stearic       oleic    linoleic   linolenic
##   168.59226    52.49436    36.74494   405.81022   242.79922    12.96870
##   arachidic  eicosenoic
##    22.03025    14.08330
```

So surely first principal components (volendo sintetizzare varianza) sono ad esse legate. So in this plot we take factor which are associated to variavle with more variances if we dont's standardise.
Standardising all variables by dividing by sd changes PCs should be done whenever measurements are not of comparable, not same unit of measure (eg not standardize on the election data).

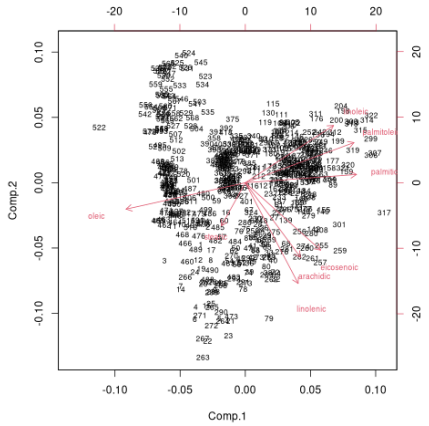
```
solive <- scale(olive) ## Standardise data
sprolive <- princomp(solive) ## PCA
summary(sprolive) # variance is more equally redistributed

## Importance of components:
```

```
##                             Comp.1    Comp.2    Comp.3     Comp.4     Comp.5
## Standard deviation     1.9274086 1.3276711 1.0072629 0.88966996 0.57726430
## Proportion of Variance 0.4651763 0.2207247 0.1270444 0.09911235 0.04172721
## Cumulative Proportion  0.4651763 0.6859009 0.8129454 0.91205772 0.95378493
##                             Comp.6     Comp.7       Comp.8
## Standard deviation     0.49838105 0.34440148 0.0455864323
## Proportion of Variance 0.03110233 0.01485251 0.0002602203
## Cumulative Proportion  0.98488727 0.99973978 1.0000000000
```

```
# PCA plotting
plot(sprolive, main = "Standardised olive oil principal components")
```



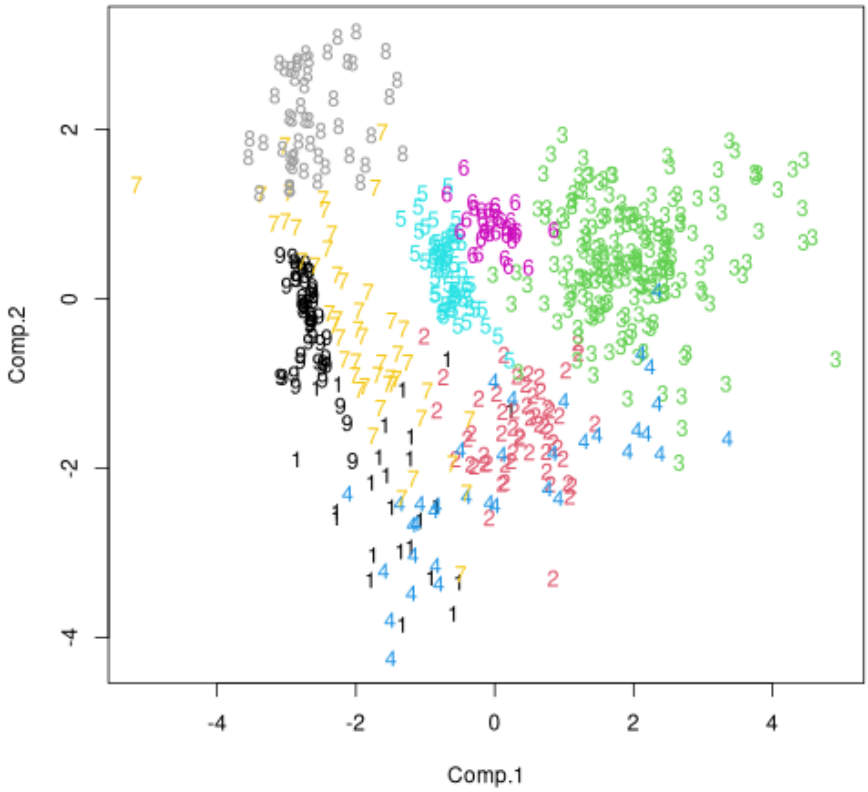

```
biplot(sprolive, cex = 0.7) # biplot
```



In the biplot post standardization the contribution to variances are not so unalanced anymore and all the variable have proper contribution to the first two components (all are not on zero as before); it seems to be 3/4 group of variables (oleic, steatic, linoleic/arachidic/eicosenoic, palmitic, palmitoleic, moleic) with respect to associations to the first two components.
Standardization can be done by `cor=TRUE` directly in `princomp` but here we did

with scale because standardized version will also be useful for other analyses.)
Let's plot groups of units (by `region`) and the first two components

```
## plotting with regions related coloring and symbols
## fpc::clusym is just a vector of plot/pch symbols
plot(sprolive$scores, col = oliveoil$region, pch = fpc::clusym[oliveoil$region])
```



Unfortunately standard R color is up to 8 and then recycle (so 1 and 9 are both black), so it was decided to change `pch` as well.
One point 6 seems very well concentrated in a single cluster; however we don't know if 6 is well separated in the all 8-dimensional space instead of 2.
In plot like this we can see the *least separation possible*, cluster in more dimension should be better: looking at 1 and 4 we cant rule out that in more dimensions they are well separated.
Bottom line: avoid pca before clustering? According to prof, for clustering purposes, *avoid PCA and information deletion before clustering* (unless clustering is difficult by the high number of variables, only in that case).

1.3 K-means

It's the most popular clustering method for multivariate data in $\mathbb{R}^p$, $(p \geq 1)$, first proposed by Steinhaus (1956, polish). It's based on least squares principle, "cluster analysis's LS-estimator".
Assume K clusters, find K centroid points, classify observations to closest centroid, so that sum of squared distances of observations to centroids is minimised.

1.3.1 Basic definitions

Important remark 5 (Notation). We have that:

- a dataset of n observation written as a set (we ignore the fact that two values can be the same): $D = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$;
- observation are p-dimensional real vector: $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip})' \in \mathbb{R}^p$, $i \in \mathbb{N}_n = 1, \ldots, n$;
- a clustering is a collection of disjoint subsets of D: $\mathcal{C} = \{C_1, \ldots, C_K\}$ is *a clustering* composed by $C_1, \ldots, C_K$ disjoint subsets of D.
- K-means clustering produces a **partition** of the dataset D, $C_1 \cup \ldots \cup C_K = D$ and $C_i \cap C_j = \emptyset$ for any two Ci, Cj with $i \neq j$.

For partitions, if $\mathbf{x}_i \in C_k$, the **label** of $\mathbf{x}_i$ is k;

$$c(i) = k, i \in \mathbb{N}_n$$

Knowing the labels $c(1), \ldots, c(n)$ of all points defines partition.

Remark 4. Obviously not every such C qualifies as "good" or "useful" clustering; what is demanded of "good" C depends on cluster analysis aim.

Definition 1.3.1 (Euclidean distance between points)**.** For $\mathbf{x} = (x_1, \ldots, x_p)$, $\mathbf{y} = (y_1, \ldots, y_p) \in \mathbb{R}^p$,

$$d_{L2}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\| = \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2}$$

is called the Euclidean distance (or L2-distance) between $\mathbf{x}$ and $\mathbf{y}$.

Definition 1.3.2 (K-means clustering)**.** The K-means clustering of D is defined by choosing the centroids $\hat{\mathbf{m}}_1^{Km}, \ldots, \hat{\mathbf{m}}_K^{Km}$ and the partition $c^{Km}(1), \ldots, c^{Km}(n)$ in such a way that it's minimised the distance between each units and the centroid of the cluster it's assigned to:

$$S(\mathcal{C}, \mathbf{m}_1, \ldots, \mathbf{m}_K) = \sum_{i=1}^{n} \left\|\mathbf{x}_i - \mathbf{m}_{c(i)}\right\|^2$$

This above is the k-means *objective function.*

1.3.2 Computation

Remark 5. So we need to find centroids $\hat{\mathbf{m}}_1^{Km}, \ldots, \hat{\mathbf{m}}_K^{Km}$ and the partition $c^{Km}(1), \ldots, c^{Km}(n)$ that minimize the objective function. This is not trivial at first; we show some easier supportive results.

Proposition 1.3.1. *For given:*

- *centroids* $\mathbf{m}_1, \ldots, \mathbf{m}_K$*, the objective function* $S(\mathcal{C}, \mathbf{m}_1, \ldots, \mathbf{m}_K)$ *is minimised by*

$$c(i) = \underset{j \in \{1,\ldots,K\}}{\arg\min} \left\| \mathbf{x}_i - \mathbf{m}_j \right\|, i \in \mathbb{N}_n$$

  that is by assigning each unit to one of the K *centroid/cluster which minimizes the distance from the considered units;*

- *partitioning* $c(1), \ldots, c(n) \in \mathbb{N}_k$*, the objective function* $S(\mathcal{C}, \mathbf{m}_1, \ldots, \mathbf{m}_K)$ *is minimised by chosing centroids for the* k*-th cluster the (multivariate/vector) mean of the units here classified*

$$\hat{\mathbf{m}}_k = \frac{1}{n_k} \sum_{c(i)=k} \mathbf{x}_i$$

  where $n_k = |C_k| = |\{i : c(i) = k\}|$ *is the number of observation in the cluster* k*.*

Proof. We show that the vector of first derivative of objective function with respect to the centroids is null considering the multivariate mean (second proposition, the first $c(i)$ are label not numbers so we can't work on it). Considering a single cluster k out of K to optimize:

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{m}_k} S(\mathcal{C}, \mathbf{m}_1, \ldots, \mathbf{m}_K) &= \frac{\partial}{\partial \mathbf{m}_k} \sum_{i=1}^{n} \left\| \mathbf{x}_i - \mathbf{m}_{c(i)} \right\|^2 \\
&\overset{(1)}{=} \frac{\partial}{\partial \mathbf{m}_k} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} (x_{ij} - m_{c(i)j})^2 \right) \\
&= \sum_{i=1}^{n} \sum_{j=1}^{p} 2(x_{ij} - m_{c(i)j})(-1) = -2 \sum_{i=1}^{n} \sum_{j=1}^{p} (x_{ij} - m_{c(i)j}) \\
&\overset{(2)}{=} -2 \sum_{c(i)=k} \sum_{j=1}^{p} (x_{ij} - m_{kj}) = -2 \sum_{j=1}^{p} \left(\sum_{c(i)=k} x_{ij} - n_k m_{kj} \right)
\end{aligned}$$

where:

- in (1) we just substituted the definition of euclidean distance so the square root goes away
- in (2) we focused only on class k being the other irrelevant to distance minimization

Equating the last to zero we obtain that, looking at one component of the resulting vector

$$m_{kj} = \frac{1}{n_k} \sum_{c(i)=k} x_{ij}, j \in \mathbb{N}_p$$

This is the reason why the method is called K-means.
It can be shown that taking the second derivative (Hessian matrix) this is positive definite, indicating this is a minimum □

Remark 6. Up to now we know that if we know either the clustering or the centroids we can determine the remaining; however we have to choose resolve/put togeter everything.

Important remark 6. Minimisation of $S(\mathcal{C}, \mathbf{m}_1, \ldots, \mathbf{m}_K)$ is hard (in CS it's NP hard, which basically means there is no efficient algorithm for solving this). Standard algorithms find only local minima (only for very small data sets global optimum can be found).

Definition 1.3.3 (K-means algorithm)**.** The classical K -means algorithm (Lloyd (1982)):

1. assuming K is given, start with an initial set of centroids $\mathbf{m}_1, \ldots, \mathbf{m}_K$. Then:

2. assign observations to closest centroid:

$$c(i) = \underset{j \in 1,\ldots,K}{\arg\min} \left\| \mathbf{x}_i - \mathbf{m}_j \right\|, i \in \mathbb{N}_n.$$

3. For each assigned cluster $k \in N_K$, calculate/update the best centroid by multivariate mean of the cluster items $\mathbf{m}_k = \frac{1}{n_k} \sum_{c(i)=k} \mathbf{x}_i$

4. stop if previous steps don't change clustering, otherwise go to step 2.

Important remark 7. Some remarks:

- the algorithm is guaranteed to converge (fastly), because every step decreases $S(\mathscr{C}, \mathbf{m}_1, \ldots, \mathbf{m}_K)$ and there are only finitely many partitions (not shown);

- **problem**: depends on *initial choice of centroids*. As initialization one can take K random data points, or assign every point to random cluster and take their means; furthermore as sensibility check one can run many times with different initialisations and take the best clustering (with respect to the objective function, the lowest one). Certainly with the lowest objective function/best cluster we cannot be sure to have the global optimal but experience tells that this leads to very good solutions;

- R `kmeans` offers more sophisticated algorithm (default used is "Hartigan-Wong"), but they still depends on initial means.

1.3.3 Examples

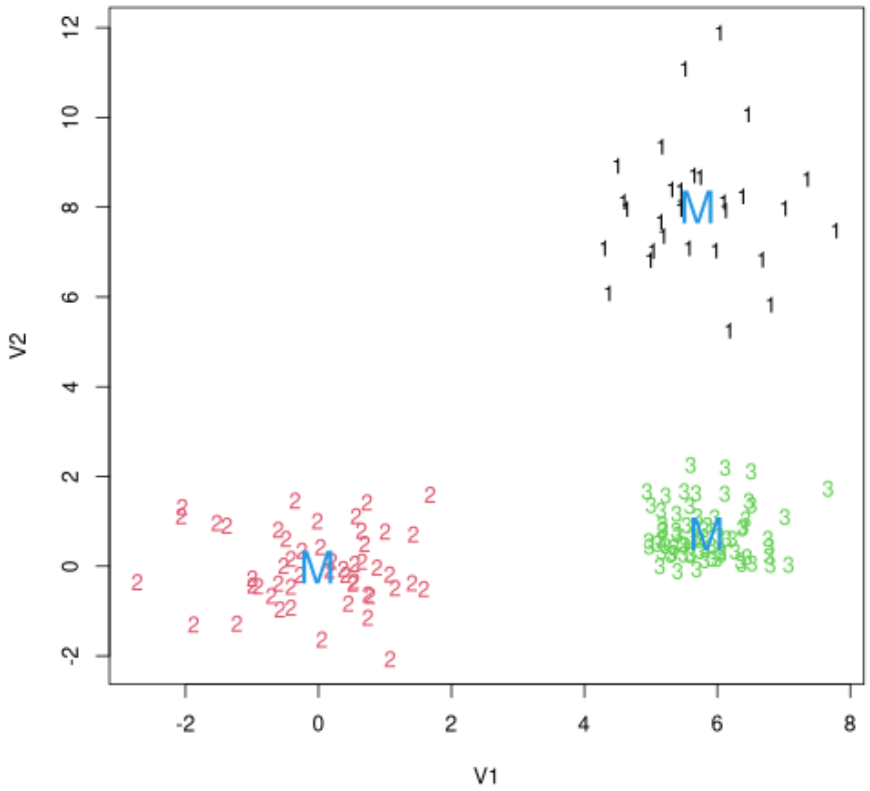
`kmeans` is used on the data.frame of observation:

- `centers`: specifies number K

- `nstart`: results of `kmeans` is random depending on initialization values. With nstart we specify the number of different possible random initialisations to start with (this is not the seed!). The default value is 1, and *this really should be changed* unless the algorithm will be run only 1 time with a single set of initial centroids.
  Depending on the dimensions of the dataset increasing this can be computationally cumbersome. For all the dataset we'll see here 100 will be fine;

- `iter.max` that gives the maximum number of iterations before the algorithm is stopped in any case. The default of this is 10, which is too low in my view, however for the mostly small datasets presented here it should be enough.
  In real application the prof suggest to set it to 100, if feasible, to be sure to get a good solution.

Example 1.3.1.
```
set.seed(665544)
c1k3 <- kmeans(clusterdata1,
               centers = 3,
               nstart = 100)
# usage of c1k3$cluster (labels) and c1k3$centers (final centroids)
plot(clusterdata1, col = c1k3$cluster, pch = fpc::clusym[c1k3$cluster])
points(c1k3$centers, pch = "M", cex = 2, col = 4) # add the cluster means
```



What do we have in the output object?

```
str(c1k3)

## List of 9
##  $ cluster     : int [1:150] 2 2 2 2 2 2 2 2 2 2 ...
##  $ centers     : num [1:3, 1:2] 5.7038 -0.0215 5.8367 8.0059 -0.0183 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:3] "1" "2" "3"
##   .. ..$ : chr [1:2] "V1" "V2"
##  $ totss       : num 2739
##  $ withinss    : num [1:3] 78.8 85.1 46.9
##  $ tot.withinss: num 211
##  $ betweenss   : num 2528
##  $ size        : int [1:3] 30 50 70
##  $ iter        : int 1
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

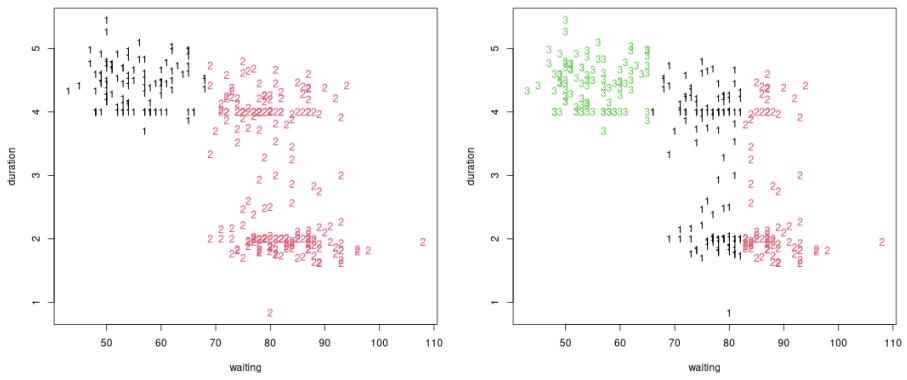
we have

- `cluster`: integers of groups

- `centers`: the centroids matrix
- `tot.withinss`: total within cluster sum of squares, this is the value of our objective function S
- `withinss`: is vector with the within cluster sum of square for each cluster (it sums to `tot.withinss`)
- `totss` is the objective function we get running k-means with one cluster (total variability with respect to the mean)
- `betweenss` is the variability accounted from clustering i guess (`totss - tot.withinss`): it should be computed as squared distances between the cluster means and the grand mean
- `size` gives number of observation in resulting cluster
- `iter` number of iteration used for the solution (but this is not very informative if we start with high `nstart`, this just mean we started with lucky initializations)
- `ifault` capture error (if different from 0) related eg to missingess or other problems

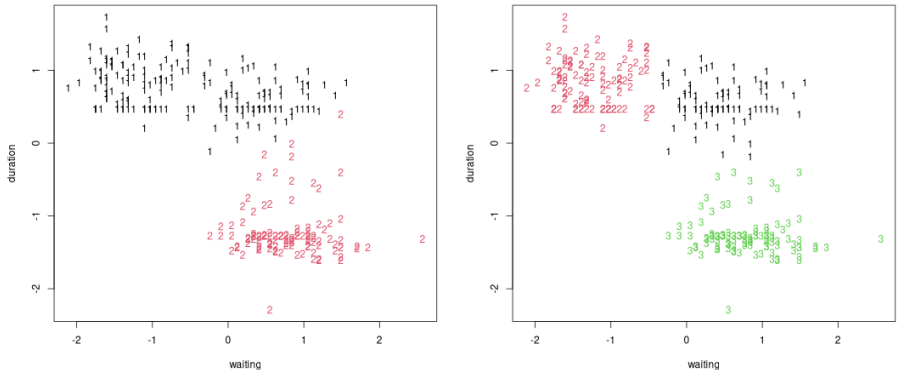
Example 1.3.2 (old faitful geyser)**.** Applying both 2-means and 3-means cluster to geyser data without scaling at first (very different)

```
set.seed(12345)

## for both 2 and 3 clusters
geyserk2 <- kmeans(geyser, 2, nstart = 100)
geyserk3 <- kmeans(geyser, 3, nstart = 100)

## plotting
par(mfrow = c(1,2))
plot(geyser, col = geyserk2$cluster, pch = fpc::clusym[geyserk2$cluster])
plot(geyser, col = geyserk3$cluster, pch = fpc::clusym[geyserk3$cluster])
```



The clustering solution seems more based on the x-axis, they're too vertical division with respect to three cloud of points that one could imagine; the reason is that x-axis variance is larger (look at the ticks) and, to optimize, its contribution to the variability/euclidead distance (which is dominated by the highest variability variables) is higher than the y-axis variable.
Often reasonable to scale variables to have unitary variance for all the variable and let spreads to be comparable; this affect the clustering as well, and the final solution found is more similar to what to expect looking at data points (three clouds)

```
sgeyser <- scale(geyser)
geysersk2 <- kmeans(sgeyser, 2, nstart = 100)
geysersk3 <- kmeans(sgeyser, 3, nstart = 100)

## below only ticks change btw
par(mfrow = c(1, 2))
plot(sgeyser, col = geysersk2$cluster, pch = fpc::clusym[geysersk2$cluster])
plot(sgeyser, col = geysersk3$cluster, pch = fpc::clusym[geysersk3$cluster])
```

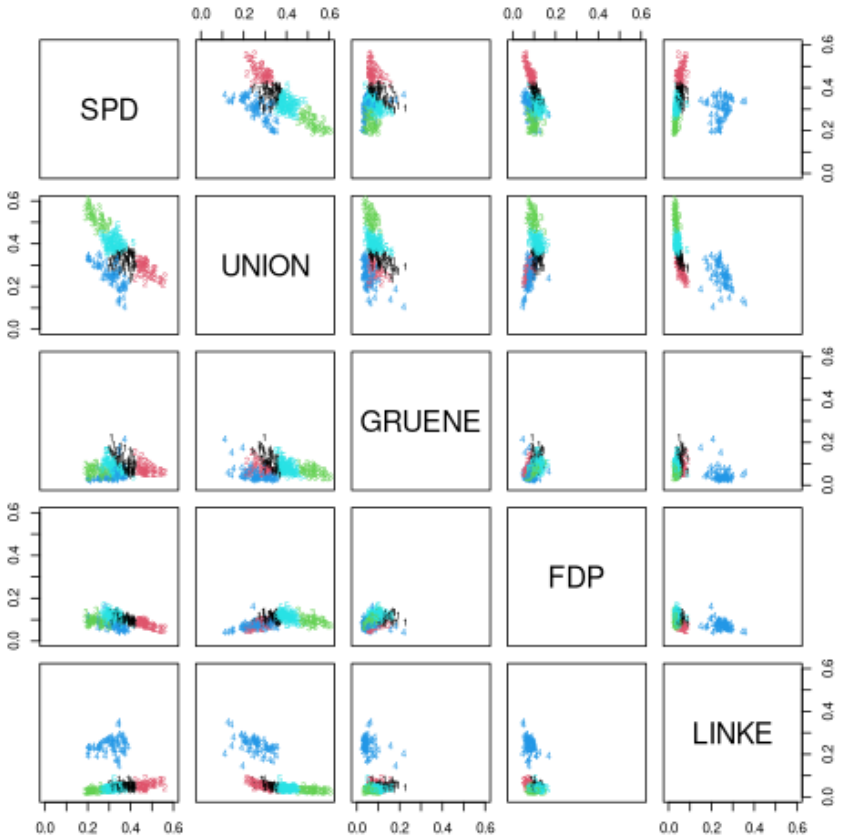


Important remark 8 (Scale equivariance and scaling). K-means is not *scale equivariant*: multiply variables by different constants and K-means result will change.
Scaling gives all variables same impact: scaling is not always good if all variables have same measurement unit and lower variation means that less meaningful stuff is going on. (eg in bundestag-data we don't need to scale since 0.01 refers to same number of voters for all parties, even if party with higher votes will have higher variance probably.)

Example 1.3.3 (bundestag)**.** Let's see 5 cluster (for interpreting clusters, equivalence with italian parties are SPD = PD, union = forza italia, gruene = verdi, linke = sinistra, fdp = liberal party credo)

```
set.seed(1234567)
bundestagk5 <- kmeans(p05[1:5], 5, nstart = 100)
pairs(p05[1:5],
      xlim = c(0, 0.6), ylim = c(0, 0.6),
```

```
      cex = 0.7,
      col = bundestagk5$cluster,
      pch = fpc::clusym[bundestagk5$cluster])
```



Most striking feature is the separation of clustering in the last column (where link party is strong); cluster n 4 (blue presumo) can be considered strongholds of the link party. Other cluster are less separated from the other ones
Is this partition in 5 groups a good one? yes and no

- yes because it easily interpretable (look at the graph by column: blue is where linke are strong (right part of the plot), red where spd is strong, green is where union is strong
- at the same time no: guene and fdp are less associable to regions

```
bundestagk5$centers # looking centroids 4 is stronghold of linke (25%, while otherwise very lou

##          SPD     UNION     GRUENE        FDP      LINKE
## 1 0.3709695 0.3258766 0.11039143 0.10678831 0.05418565
## 2 0.4755049 0.2809211 0.07737812 0.07977076 0.05684986
## 3 0.2382177 0.5235427 0.06558486 0.09447968 0.03217685
```

```
## 4 0.3076717 0.2555014 0.05407416 0.07920767 0.24680508
## 5 0.3219290 0.4031411 0.08349274 0.11502019 0.04082300
```

In exercises make an attempt to interpret the cluster/what can be learned considered the research question; do this by data visualization as above, looking at the centers. In this case we have natural geographical clusterings which can be overlapped to the k-means based on political vote just by using `table`

```
table(bundestagk5$cluster, p05$ewb)

##
##     Berlin East West
##   1      6    0   66
##   2      0    0   44
##   3      0    0   38
##   4      6   53    4
##   5      0    0   82

table(bundestagk5$cluster, p05$state)

##
##     Baden-Wuerttemberg Bayern Berlin Brandenburg Bremen Hamburg Hessen
##   1                  7      1      6           0      1       5     11
##   2                  0      0      0           0      1       1      4
##   3                  3     32      0           0      0       0      0
##   4                  0      0      6          10      0       0      0
##   5                 27     12      0           0      0       0      6
##
##     Mecklenburg-Vorpommern Niedersachsen Nordrhein-Westfalen Rheinland-Pfalz
##   1                      0            8                  21               6
##   2                      0           17                  20               0
##   3                      0            1                   2               0
##   4                      7            0                   0               0
##   5                      0            3                  21               9
##
##     Saarland Sachsen Sachsen-Anhalt Schleswig-Holstein Thueringen
##   1        0       0              0                  6          0
##   2        0       0              0                  1          0
##   3        0       0              0                  0          0
##   4        4      17             10                  0          9
##   5        0       0              0                  4          0
```

Looking at `ewb`, cluster 4 (where link is strong) is all from the east germany, which make sense; conversely all the east votes are in linke-strong groups. Berlin is split into two parts (cluster 4 and 1), maybe due to historical separation reasons.
Regarding `state`, ...

1.3.4 Probabilistic background

Remark 7. we now relate kmeans to statistical model. we postponed the model after the kmeans definition: this is how historically happened

Let's assume we have iid multivariately distributed observtion with mean vector $\mathbf{a}$ and covariance matrix Σ ($\mathbf{x}_1, \ldots, \mathbf{x}_n \sim \mathrm{N}(\mathbf{a}, \Sigma)$ iid;); let $\varphi_{\mathbf{a},\Sigma}$ be the multivariate normal density, then multivariate mean $\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$ is ML estimator for $\mathbf{a}$. Now let's complicate it a bit.

Theorem 1.3.2. *Assume we have observation $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are coming from groups we don't know and are jointly distributed according to multivariate density (density for an $n \times p$ matrix)*

$$f(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \prod_{i=1}^{n} \phi_{\mathbf{a}_{\gamma(i)}, b\mathbf{I}_p}(\mathbf{x}_i)$$

where:

- *the product is due to the fact that observation are independent*
- *they are not identically distributed: the mean for a single observation is $\mathbf{a}_{\gamma(i)}$ (where $\gamma(i)$ specifies the group from which the observation comes from) while the variance covariance matrix of its variables/components is $b\mathbf{I}_p$ (with $b > 0$ and $\mathbf{I}_p$ the $p \times p$ unit matrix): so here we assume covariance matrix is the same for all the cluster (and all the variables have the same variance b).*

Under these condition:

- *the k-means means/centroids $\hat{\mathbf{m}}_1^{Km}, \ldots, \hat{\mathbf{m}}_K^{Km}$ are ML estimator for the units mean vector $\mathbf{a}_1, \ldots, \mathbf{a}_K$*
- *the k-means cluster/groups $c^{Km}(1), \ldots, c^{Km}(n)$ are are ML for $\gamma(1), \ldots, \gamma(n)$.*

Remark 8. In a nutshell the model says if we have true cluster we don't know having different means but same varcov, kmeans can be written as ML estimator giving both means and groups

Proof. To derive ML estimator we need to show how to choose the $\mathbf{a}$s and the γs (here we're not interested in the b) that maximize the densities (assuming independence between observation). The log-density is easier to maximize and is:

$$\log f(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \sum_{i=1}^{n}\left(-\log\left(\sqrt{2\pi\det(b\mathbf{I}_p)}\right) - \frac{1}{2b}(\mathbf{x}_i - \mathbf{a}_{\gamma(i)})'(\mathbf{x}_i - \mathbf{a}_{\gamma(i)})\right)$$

Maximising this for $\mathbf{a}_1, \ldots, \mathbf{a}_K, \gamma(1), \ldots, \gamma(n)$ does not depend on b and amount to look at the second part and minimize just

$$\sum_{i=1}^{n}(\mathbf{x}_i - \mathbf{a}_{\gamma(i)})'(\mathbf{x}_i - \mathbf{a}_{\gamma(i)}) = \sum_{i=1}^{n}\left\|\mathbf{x}_i - \mathbf{a}_{\gamma(i)}\right\|^2$$

which is just the K-means definition.
So since k-means is defined to minimize this, is the ML estimator above as well. □

Important remark 9. Some people think that K-means is "nonparametric" without model assumptions.
Actually K-means:

- is implicitly based on Gaussian clusters with equal and spherical covariance matrices
- has difficulties finding clusters if covariance matrices differ strongly, are not spherical or nonlinear
- it is true that K-means but can be computed and can works well if assumptions are violated (see artificial data 1 where skewnormal was used)
  As for all statistical methods, it is an important issue to what extent real data can be tolerated to deviate from the formal model assumptions without doing much harm to the results of the analysis.
  Normality assumption linked with K-means doesn't say K-means can't be good otherwise, but it hints at in what situation it is best, and what kinds of clusters it can find.
  In general (not only K-means) all model-based methods will work well in some but not all cases in which the assumptions are violated: to distinguish between acceptable and unacceptable violations of the model assumptions is a major skill for a statistician, which only comes with conscious evaluation of experience (one source of such experience is to simulate data sets with model assumptions violated, and to run the methods and try to understand the results).
- finally, representing objects in clusters optimally by centroids makes sense in some applications (such as representing images in database for helping users to find images).
- K-means is inconsistent as ML-estimator: there's a theorem which says ML are generally consistent under conditions; these conditions are not respected for k-means. The idea is depicted in fig 1.3 with the density of two normal distributions (say different cluster). Here if $n \to \infty$ the histogram will perfectly match the theoretical distributions.
  What K-means will do is to split the data in the red line and group assign all observation to the left to one group and all to the right to another one. So it will missclassify some units generated from the other distribution to the one considered.
  In the graph the X are the means of the considered distribution, while M are the means/centroids that K-means will estimate; these are different and will stay different even if $n \to \infty$. This occurs because the observation on one group have mean according to the truncated normal distribution (instead of the standard gaussian) since k-means assign up to the red line

```r
## Normal densities
xpoints <- seq(-4, 4, by = 0.01)
norm1 <- dnorm(xpoints, mean = -1, sd = 1)
norm2 <- dnorm(xpoints, mean = 1, sd = 1)
plot(xpoints, norm1, type = "l", ylab = "density") # plot first group density
points(xpoints, norm2, type = "l") # add second group density
```

```
lines(c(0, 0), c(0, 0.4), col = 2) # red line
points(-1, 0, pch = "X", col = 4) # mean left group
points(1, 0, pch = "X", col = 4) # mean right group

## Computations for truncated distributions.  See
## https://en.wikipedia.org/wiki/Truncated_normal_distribution for
## formulae.

prob1 <- pnorm(0, mean = -1, sd = 1) # Probability of being below zero for left Gaussian
prob2 <- pnorm(0, mean = 1,  sd = 1) # Probability of being below zero for right Gaussian

te1 <- -1 - dnorm(1) / pnorm(1) ## Expected value of Gaussian with mean -1, truncated at 0.
te2 <- 1 - dnorm(-1) / pnorm(-1) ## Expected value of Gaussian with mean 1, truncated at 0.

## Expected value for left cluster combines the two according to
## their probabilities for their contributions to the left cluster.
## while it's symmetric for right cluster.
ecluster1 <- prob1 * te1 + prob2 * te2
ecluster2 <- - ecluster1

## Plot cluster means
points(ecluster1, 0, pch = "M", col = 3)
points(ecluster2, 0, pch = "M", col = 3)
```

Important remark 10 (More sophisticated asymptotic theory (Pollard (1981))). K-means is consistent for its own "canonical functional".
Let P a distribution with $E_P \|\mathbf{x}\|^2 < \infty$. Let

NB: Just for info, not at the exam. Require Lebesgue integrals.

$$(\mathbf{m}_1^{Km*}, \ldots, \mathbf{m}_K^{Km*}) = \underset{(\mathbf{m}_1,\ldots,\mathbf{m}_K)}{\arg\min} \int \min_{\mathbf{m}\in\{\mathbf{m}_1,\ldots,\mathbf{m}_K\}} \|\mathbf{x}-\mathbf{m}\|^2 \, \mathrm{d}P(\mathbf{x})$$

Then, a.s., for $n \to \infty$ and data i.i.d. from P

$$\{\mathbf{m}_1^{Km}, \ldots, \mathbf{m}_K^{Km}\} \to \{\mathbf{m}_1^{Km*}, \ldots, \mathbf{m}_K^{Km*}\}$$

This does not require a Gaussian distribution.

1.3.5 Estimating number of cluster

Remark 9. Classical K-means requires to specify K (rarely known): it's usually fixed by practical considerations; estimating it isn't always well defined problem (one should expect it needs user input).
How to obtain a "good" K from data?

1.3.5.1 Looking at the objective function

If we compute K-means for various K, we could look at where objective function is minimized:

$$S_K = S(\mathcal{C}, \hat{\mathbf{m}}_1^{km}, \ldots, \hat{\mathbf{m}}_K^{Km}) = \sum_{i=1}^{n} \left\| \mathbf{x}_i - \hat{\mathbf{m}}_{c(i)}^{Km} \right\|^2$$

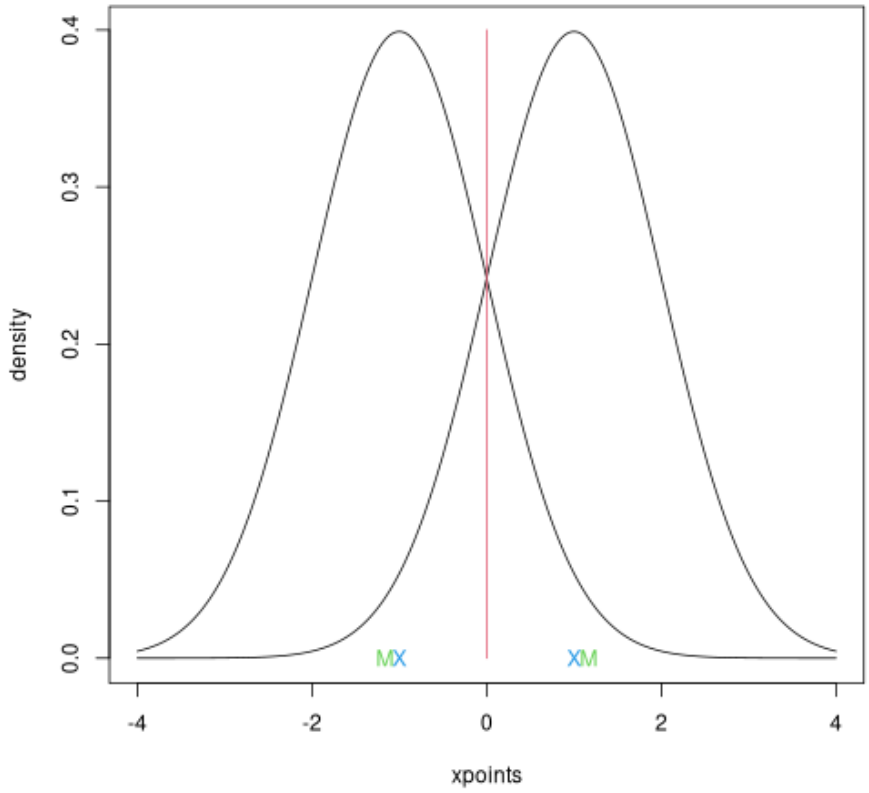


Figure 1.3: K-means is inconsistent

Problem is that S_k will always decrease with increasing K: for given K-means clustering it's always possible to construct a $(K+1)$-solution with lower objective function so at least we know that with $K+1$ the final solution among those attempted will have lower objective function.
We can construct that $(K+1)$-means solution using a single observation from a cluster and create a new cluster, while keeping everything else constant. This will lower value of S (the new cluster has zero difference from the centroid and the moving centroid of the old cluster will better the objective function) so the new solution will have S even smaller (if not equal), so $S_{K+1} \leq S_K$. (the only exception to this is if the centroid coincides with where the two solutions will have same objective function).

This above in theory: in practice we can only find local optima, and in very rare situations it might happen that algorithm based on random starts gives $S_K < S_{K+1}$

1.3.5.2 The elbow method

```
K <- 1:10
K <- setNames(as.list(K), K)
res <- lapply(K, function(k) kmeans(clusterdata1, k, nstart = 100)$tot.withinss)

par(mfrow = c(1, 2))
plot(clusterdata1)
plot(1:10, unlist(res), xlab = "k", ylab = "S_k", type = "l")
```

We can plot K against S_K and look for "elbows": in fig 1.4, with reference to simulated data where cluster are actually 3, we see that going from 3 to 4 does not diminish objective function that much .

Important remark 11 (Rationale for elbow). Assuming there is a true number K^* of well separated clusters, as long as $K < K^*$ some observations from different true clusters are put together and S can be improved strongly if they are separated. If $K \geq K^*$, new fitted clusters split up true (homogeneous) clusters, and this should not improve S much.

Important remark 12 (Problems with elbow method). We have that:

- it's subjective, hard to reproduce and investigate systematically;
- some datasets don't show clear elbows (eg bundestag);
- the rationale doesn't imply that for $K < K^*$ or $K \geq K^*$ the decrease of S would be constant;
- S is bounded from below by 0; S will be relatively low for large K in any case, with not much "space" to show an "elbow".

Important remark 13 (Take home message). Elbow method can give an orientation, but formal methods are preferable to the elbow method.

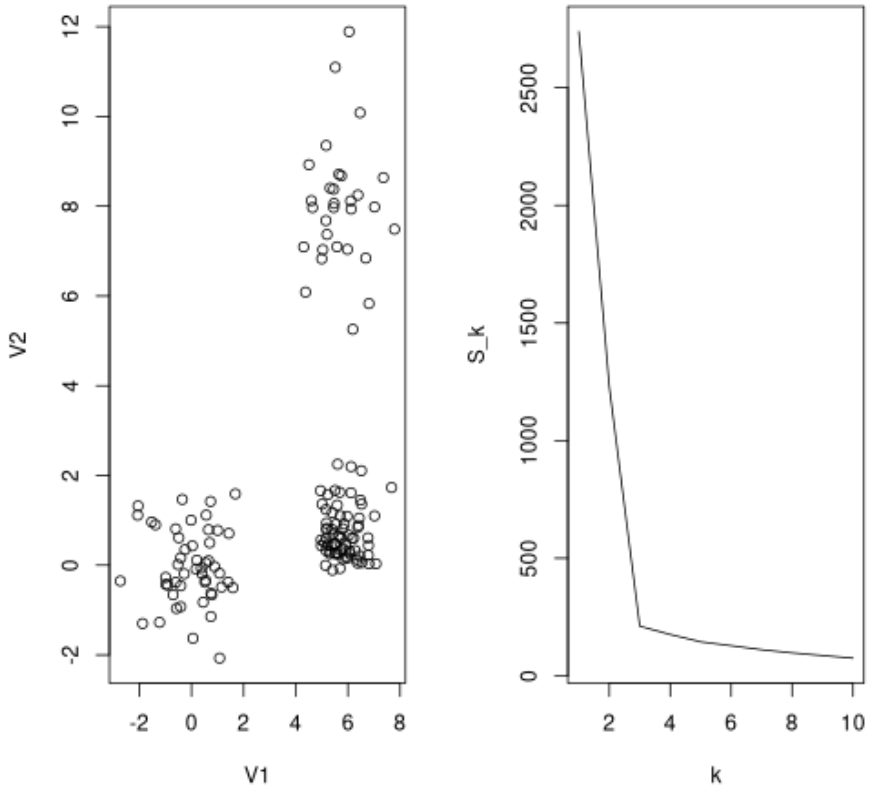


Figure 1.4: The elbow method

1.3.5.3 The gap statistic

Remark 10. There are many attempts at defining indexes, based on transformations of S_K , that have local optimum for "best" K; we see here the gap statistic (Tibshirani et al. (2001)).

Remark 11. It's based on $\log(S_K)$ rather than S_K:

- this does not solve the fact that the indicators goes always down as K increase
- if $S_K \to 0$ then $\log(S_K) \to -\infty$ so it's not bounded from below (differently from S_K) and can better differentiate in places where S_k is close to 0
- attenuate the fact that generally variation of S_K are strongly K-dependent

Important remark 14. The gap statistics compare $\log S_K$ in data with how it is expected to behave under uniform distribution for which K-means is applied in same way.
The idea is to set k where the gap between the two becomes bigger (and then stable).

Remark 12. The reason for Uniform distribution is that it can be seen as "borderline between clustering and no clustering", as it has a flat density and very small modifications can give it one or arbitrarily many density modes.
Tibshirani et al. (2001) have theory for $p = 1$.

Important remark 15 (Informal idea). So:

- Let $\mathbf{U} = (U_1, \ldots, U_n)$, with $U_1, \ldots, U_n$ i.i.d. distributed uniformly on rectangle between $\min_{i=1,\ldots,n} x_{ij}$ and $\max_{i=1,\ldots,n} x_{ij}$ for all the variables $j = 1, \ldots, p$ (or, better, between $\min_{i=1,\ldots,n} y_{ij}$ and $\max_{i=1,\ldots,n} y_{ij}$, where y_{ij} is score on j-th scaled principal component).
  Note **U is a matrix** simulating data under multivariate uniform assumption
- generate B (e.g., $B = 100$) data sets distributed as $\mathbf{U}$, and compute K-means clustering for all of them, then calculate construct its distribution of $\log S_K(\mathbf{U})$
- the heuristic idea: if the best number of clusters is K_0 and
  - we are under the optimal level ($K < K_0$): we expect that going from K to $K+1$ will diminish the log(obj.f) applied to the database, that is $\log S_K(D) >> \log S_{K+1}(D)$,
  - we are over or at the optimal level ($K \geq K_0$): we expect still a decrease associated to K, that is $\log S_K(D) > \log S_{K+1}(D)$, but only in same manner in which a decrease is seen $\log S_K(U) > \log S_{K+1}(U)$ in the uniformly generated dataset, because no "proper" new cluster is constructed.
    That is $\log S_K(U)$ levels act as comparison to rule out the natural decrease in the function in absence of actual clustering/association, credo.

Important remark 16 (Algorithm). The steps are:

1. for $K \in \mathbb{N}_{K_{max}}$, compute K -means clustering of D, $\log S_K(D)$;
2. generate B dataset $\mathbf{U}_1, \ldots, \mathbf{U}_B$, cluster them by K-means and compute $\log S_K(\mathbf{U}_1), \ldots, \log S_K(\mathbf{U}_B)$;
3. compute estimated mean and standard error of $\log S_k$

$$\bar{S}_K = \frac{1}{B}\sum_{i=1}^{B} \log S_k(\mathbf{U}_i)$$

$$\hat{sd}_K = \sqrt{\frac{1}{B-1}\sum_{i=1}^{B}\left(\log S_k(\mathbf{U}_i) - \bar{S}_k\right)^2}$$

4. compute the gap statistic defined as distance between statistics in the simulation and on the dataset:

$$Gap(K) = \bar{S}_k - \log S_K(\mathcal{D})$$

We can estimate the standard error of $Gap(K)$ (under say the null) assuming $\mathcal{D} \sim \mathbf{U}$ independent from $\mathbf{U}_1, \ldots, \mathbf{U}_b$, and it is:

$$s_K = \sqrt{\left(1 + \frac{1}{B}\right)}\hat{sd}_K$$

because:

$$\mathrm{Var}\left[\bar{S}_k - \log S_k(\mathcal{D})\right] = \mathrm{Var}\left[\bar{S}_k\right] + \mathrm{Var}\left[\log S_k(\mathbf{U})\right]$$

$$\hat{Var}\left[\bar{S}_k - \log S_k(\mathcal{D})\right] = \frac{\hat{sd}_K^2}{B} + \hat{sd}_K^2$$

5. Choose the optimal K_0 as smallest K so that the gap

$$Gap(K) > Gap(K^*) - 2s_k$$

where $K^* = \arg\max_L Gap(L)$ is the "raw best" K where the gap is maximized.
We would look just at the maximized gap, but there's random variation in this: so we can a smaller K_0 if its gap is only worse by random variation than K^* (that is $Gap(K)$ belong to the confidence interval of $Gap(K^*)$, being over its lower limit constructed using 2 time its standard error[1]).
In this way we keep K as low as possible (and occam razor works) compatible with the distribution of optimal K according to gap.
For too small K, $Gap(K) << Gap(K^*)$ (smaller $log(S_k(D)$: larger Gap)

6. alternatively (soluzione me, one could compare $Gap(K)$ to $Gap(K+1)$ rather than max gap, and choose the smallest K where the improvement of increasing the number of cluster "is not that high"

$$Gap(K) > Gap(K+1) - qs_{k+1}, \quad q > 0$$

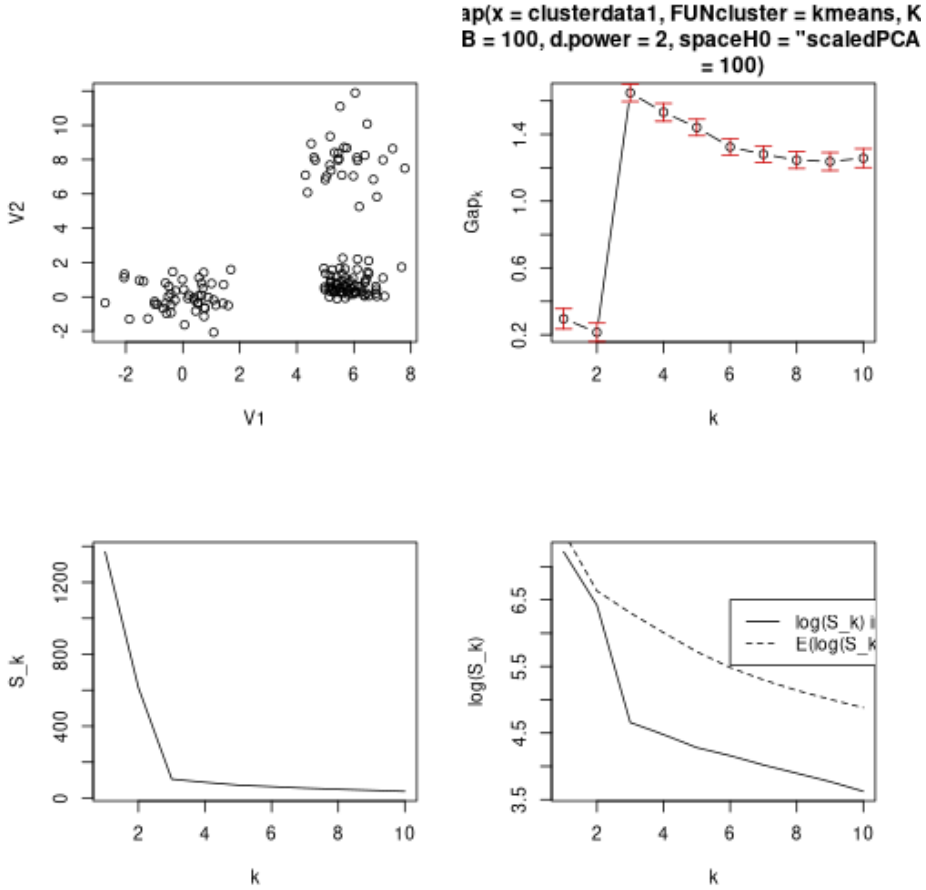
(in original paper with $q = 1$, but this can be set to 2). R-function `clusGap` in `cluster` offer all this.

[1] Here we rely on central limit theorem for mean, since we can choose B to be large; people use 2 instead of 1.96 to be "safer".

Example 1.3.4 (Artificial dataset)**.** In the artificial dataset we see that (third graph) the vertical difference between the two becomes max when $K = 3$ (number of cluster used to generate) and then is somewhat stable.
Dont look at the band since are not based on twice the standard error, the band are 1*se on each side; 3 is even the best K for which the gap is maximized; we could't choose 2 because if far from the 2*band (so yes imagine to double the width of the band)

```
library(cluster) # Has the clusGap function
set.seed(123456)

cg1 <- clusGap(clusterdata1,
               kmeans, # specify the function used for clustering
               K.max = 10,  #  maximum number of clusters investigated
               B = 100,  # n of simulations from uniform distribution
               d.power = 2, # specify we're optimizing squared
                            # Euclidean distances, as k-means does: so
                            # when we use kmeans here always 2
               spaceH0 = "scaledPCA", # specifies the way the uniform
                                      # distribution is simulated:
                                      # here we use the PCA solution
                                      # instead of original data
               nstart = 100) # options to clustering methods
## so in this case kmeans will be run for k=1:10 cluster, for 100
## sims, each using 100 random initialization (1 million of cycles)

# At this time we need another function to specify that we want the
# criterion for choosing K to be the first one (Gap(K) > Gap(K^*) - 2
# s_k); we use print with method = "globalSEmax" and SE.factor =2
print(cg1,
    method = "globalSEmax", # we compare with Gap(K^*) rather than K+1
    SE.factor = 2) # this is the q (of second criteria)

## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = clusterdata1, FUNcluster = kmeans, K.max = 10, B = 100, d.power = 2, spaceH0 = "
## B=100 simulated reference sets, k = 1..10; spaceH0="scaledPCA"
##  --> Number of clusters (method 'globalSEmax', SE.factor=2): 3
##           logW   E.logW       gap     SE.sim
##  [1,] 7.222172 7.518419 0.2962467 0.06073207
##  [2,] 6.419890 6.634534 0.2146432 0.05540815
##  [3,] 4.658166 6.306516 1.6483500 0.05224121
##  [4,] 4.475601 6.007479 1.5318780 0.05353712
##  [5,] 4.280162 5.721320 1.4411574 0.04925218
##  [6,] 4.159101 5.483455 1.3243545 0.04770478
##  [7,] 4.018231 5.298835 1.2806042 0.04809762
##  [8,] 3.894625 5.140342 1.2457174 0.05056464
##  [9,] 3.768839 5.005861 1.2370216 0.05406971
## [10,] 3.623056 4.880549 1.2574927 0.05641072

## - logW is the log S_k (for 1 to 10 cluster,
```

```
## - E.logW is the expected log S_k under uniform distribution
## - gap is the difference between the two
## - SE.sim is the standard error s_k

## plotting
par(mfrow = c(2, 2))
plot(clusterdata1) # 1: data
plot(cg1) # 2: Values of gap and +/- 1se bands
plot(1:10, exp(cg1$Tab[, 1]), # 3: elbow method: values of log(S_k)
   xlab = "k", ylab = "S_k", type = "l") # exponentiated so S_k
plot(1:10, cg1$Tab[, 1], # 4a) log(S_k) in the data
   xlab = "k", ylab = "log(S_k)", type = "l")
points(1:10, cg1$Tab[, 2], # 4b) log(S_k) expectation under uniform distribution
     xlab = "k", ylab = "log(S_k)", type = "l", lty = 2)
legend(6, 6.5, c("log(S_k) in data", "E(log(S_k)) uniform"), lty = 1:2)
```



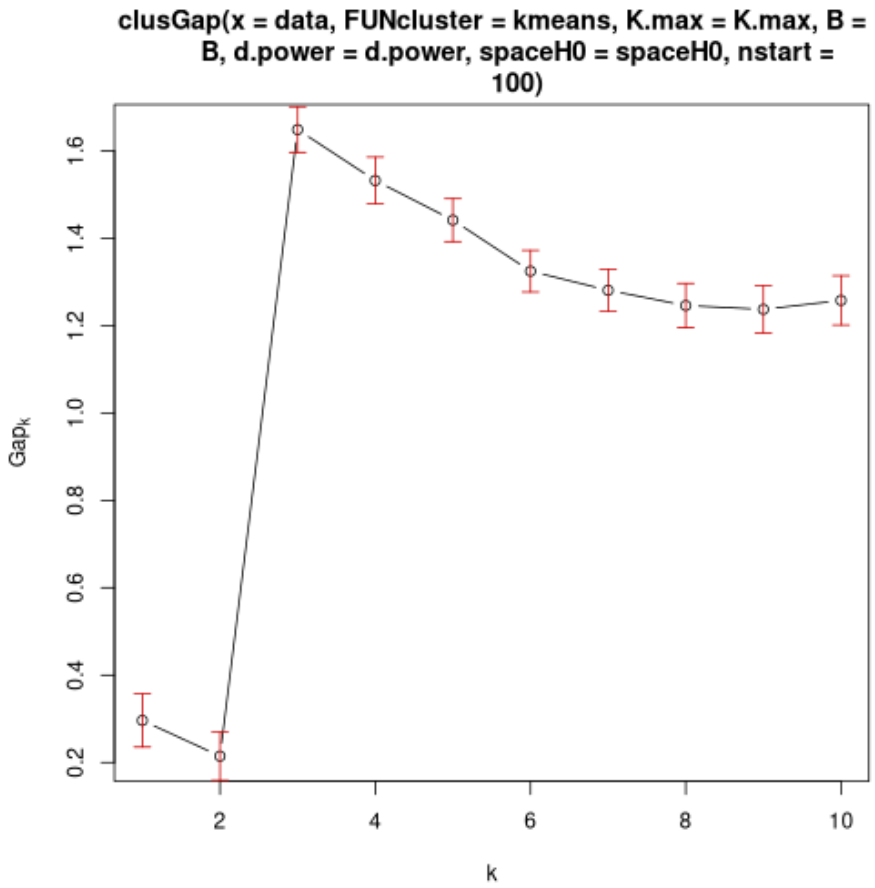
Remark 13. The `clusGap` function does not directly give out the optimal number of clusters K_0, and neither the resulting optimal clustering.
For extracting K_0 need to run `maxSE`, which operates on table entries of `clusGap` output. For the optimal clustering, need to re-run `kmeans` with optimal K_0 .

the following gapnc function that does these automatically.

```
gapnc <- function(data,
                  FUNcluster = kmeans,
                  K.max = 10,
                  B = 100,
                  d.power = 2,
                  spaceH0 = "scaledPCA",
                  method = "globalSEmax",
                  SE.factor = 2,
                  ...)
{
    ## As in original clusGap function the ... arguments are passed on
    ## to the clustering method FUNcluster (kmeans).
    ## Run clusGap
    gap1 <- clusGap(data, kmeans, K.max, B, d.power, spaceH0, ...)
    ## Find optimal number of clusters; note that the method for
    ## finding the optimum and the SE.factor q need to be specified here.
    nc <- maxSE(gap1$Tab[, 3], gap1$Tab[, 4], method, SE.factor)
    ## Re-run kmeans with optimal nc.
    kmopt <- kmeans(data, nc, ...)
    out <- list()
    out$gapout <- gap1
    out$nc <- nc
    out$kmopt <- kmopt
    out
}
## The output of clusGap is in component gapout.
## The optimal number of clusters is in component nc.
## The optimal kmeans output is in component kmopt.
```

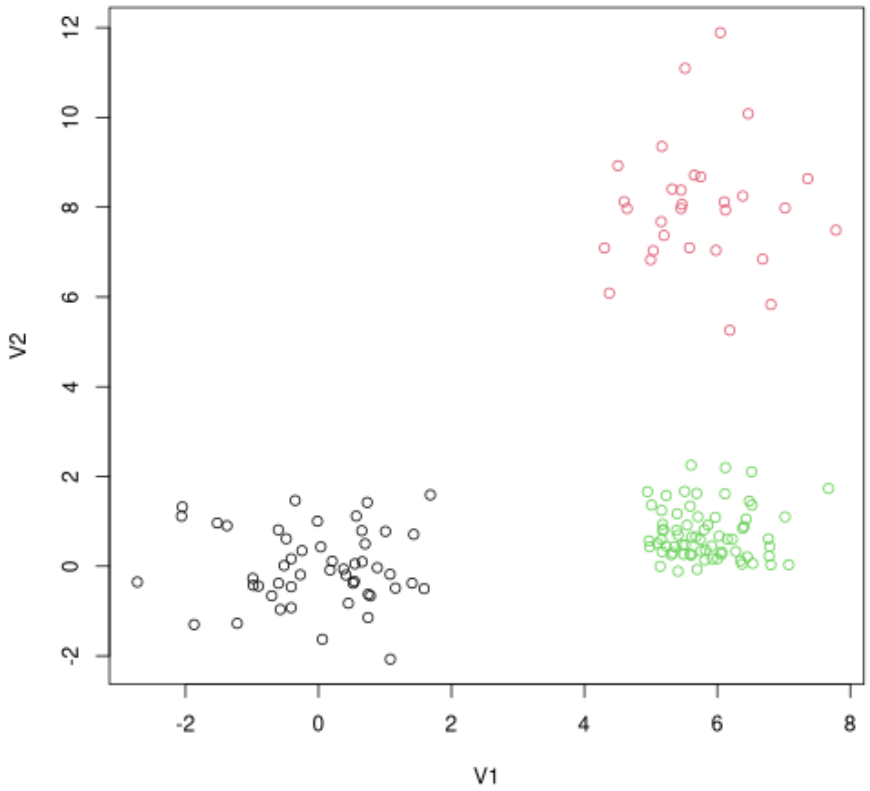
Example 1.3.5. The usage of gapnc with clusterdata

```
set.seed(123456)
cgnc1 <- gapnc(clusterdata1, nstart = 100)

## Could also specify K.max, B, d.power, spaceH0, method, SE.factor
plot(cgnc1$gapout) ## same clusgap plot as before
```




```
print(cgnc1$gapout, method = "globalSEmax", SE.factor = 2) #?

## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = data, FUNcluster = kmeans, K.max = K.max, B = B, d.power = d.power, sp
## B=100 simulated reference sets, k = 1..10; spaceH0="scaledPCA"
##  --> Number of clusters (method 'globalSEmax', SE.factor=2): 3
##           logW   E.logW       gap     SE.sim
##  [1,] 7.222172 7.518419 0.2962467 0.06073207
##  [2,] 6.419890 6.634534 0.2146432 0.05540815
##  [3,] 4.658166 6.306516 1.6483500 0.05224121
##  [4,] 4.475601 6.007479 1.5318780 0.05353712
##  [5,] 4.280162 5.721320 1.4411574 0.04925218
##  [6,] 4.159101 5.483455 1.3243545 0.04770478
##  [7,] 4.018231 5.298835 1.2806042 0.04809762
##  [8,] 3.894625 5.140342 1.2457174 0.05056464
##  [9,] 3.768839 5.005861 1.2370216 0.05406971
## [10,] 3.623056 4.880549 1.2574927 0.05641072

print(cgnc1$gapout) #?

## Clustering Gap statistic ["clusGap"] from call:
```

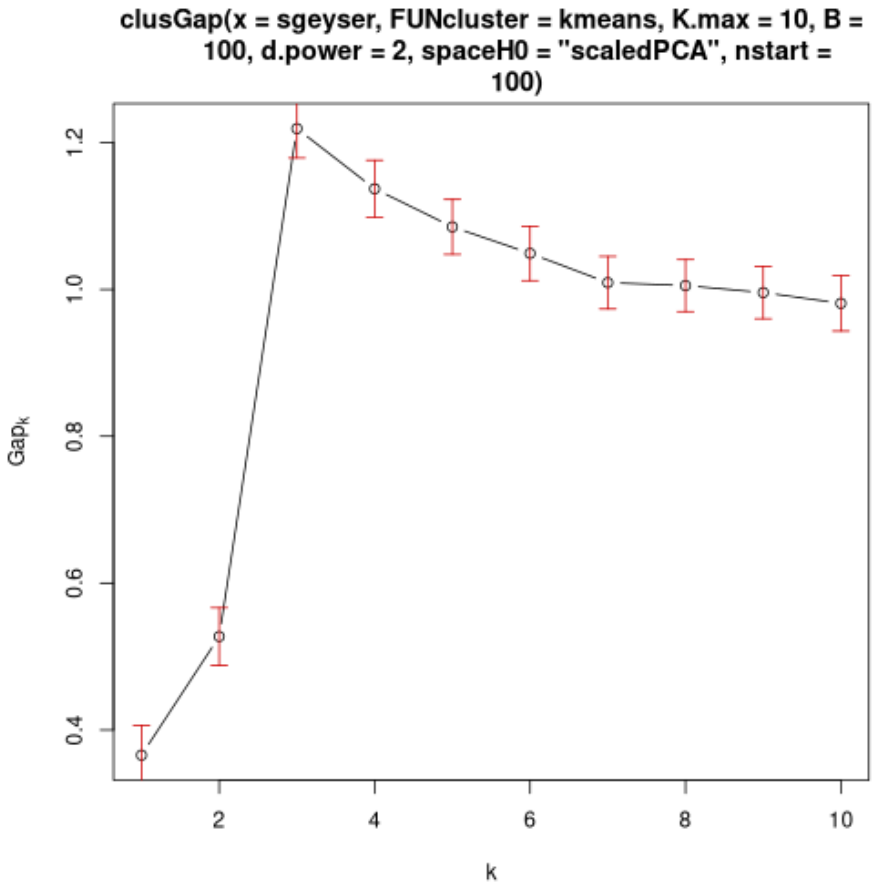
```
## clusGap(x = data, FUNcluster = kmeans, K.max = K.max, B = B, d.power = d.power, spaceH0 = sp
## B=100 simulated reference sets, k = 1..10; spaceH0="scaledPCA"
##  --> Number of clusters (method 'firstSEmax', SE.factor=1): 1
##          logW   E.logW       gap     SE.sim
##  [1,] 7.222172 7.518419 0.2962467 0.06073207
##  [2,] 6.419890 6.634534 0.2146432 0.05540815
##  [3,] 4.658166 6.306516 1.6483500 0.05224121
##  [4,] 4.475601 6.007479 1.5318780 0.05353712
##  [5,] 4.280162 5.721320 1.4411574 0.04925218
##  [6,] 4.159101 5.483455 1.3243545 0.04770478
##  [7,] 4.018231 5.298835 1.2806042 0.04809762
##  [8,] 3.894625 5.140342 1.2457174 0.05056464
##  [9,] 3.768839 5.005861 1.2370216 0.05406971
## [10,] 3.623056 4.880549 1.2574927 0.05641072

## Unfortunately need to specify method and SE.factor here
## once more to reproduce earlier output.
cgnc1$nc

## [1] 3

plot(clusterdata1, col = cgnc1$kmopt$cluster) ## As seen before
```



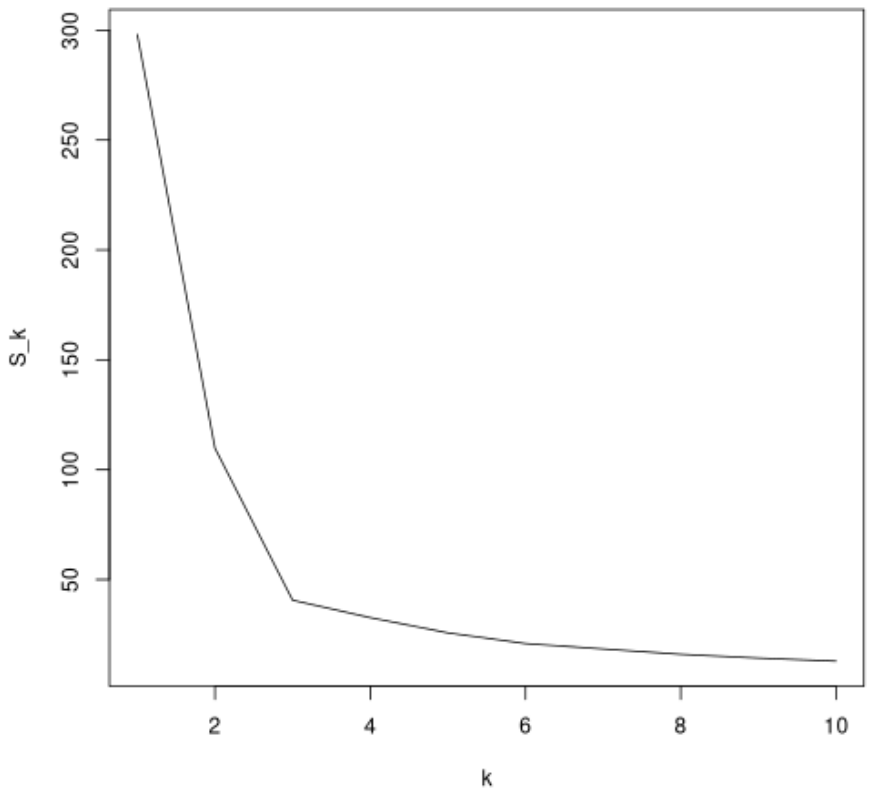
Example 1.3.6 (Geyser)**.** Other example of `gapnc` (non stampato l'output è la stessa roba di prima a vari pezzi, si sceglie chiaramente la soluzione a $k = 3$)

```
cgg <- clusGap(sgeyser,kmeans,K.max=10,B=100,d.power=2,spaceH0="scaledPCA",nstart=100
print(cgg,method="globalSEmax",SE.factor=2)

## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = sgeyser, FUNcluster = kmeans, K.max = 10, B = 100, d.power = 2, spaceH
## B=100 simulated reference sets, k = 1..10; spaceH0="scaledPCA"
##  --> Number of clusters (method 'globalSEmax', SE.factor=2): 3
##           logW   E.logW       gap     SE.sim
##  [1,] 5.697093 6.062757 0.3656638 0.04055419
##  [2,] 4.698517 5.225740 0.5272232 0.03933551
##  [3,] 3.703345 4.922320 1.2189751 0.03995289
##  [4,] 3.485951 4.622947 1.1369960 0.03882986
##  [5,] 3.247605 4.332688 1.0850837 0.03746564
##  [6,] 3.037916 4.086927 1.0490108 0.03710178
##  [7,] 2.912454 3.921610 1.0091563 0.03575295
##  [8,] 2.771421 3.776574 1.0051533 0.03544375
##  [9,] 2.656914 3.652573 0.9956591 0.03558674
## [10,] 2.554183 3.535168 0.9809853 0.03770200
```

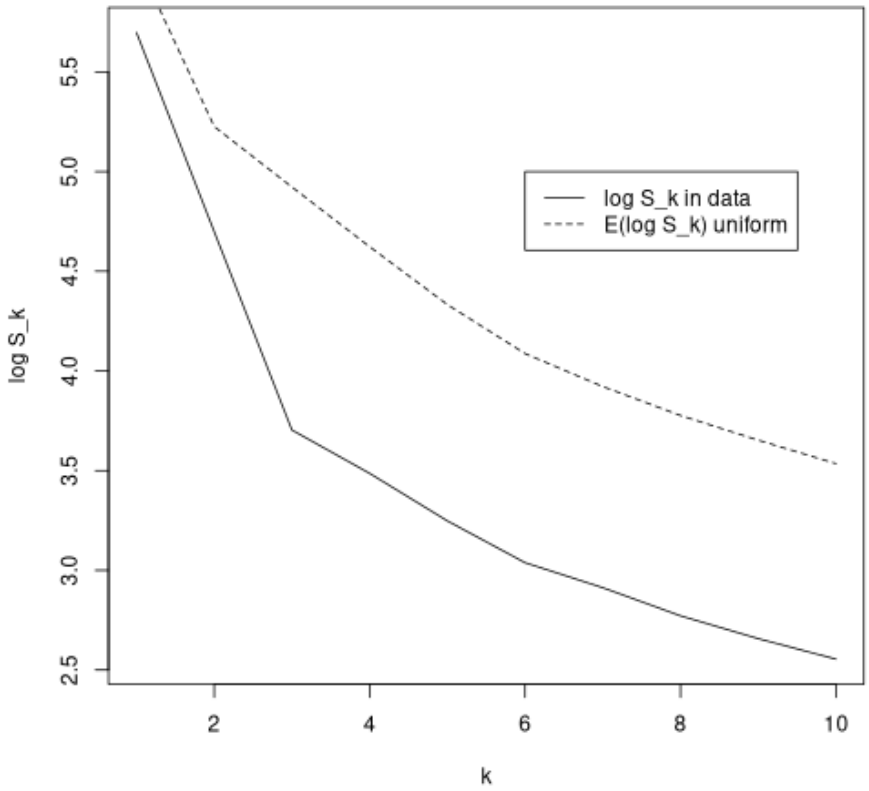
```
## --> Number of clusters (method <U+2019>globalSEmax<U+2019>, SE.factor=2): 3
plot(cgg)
```



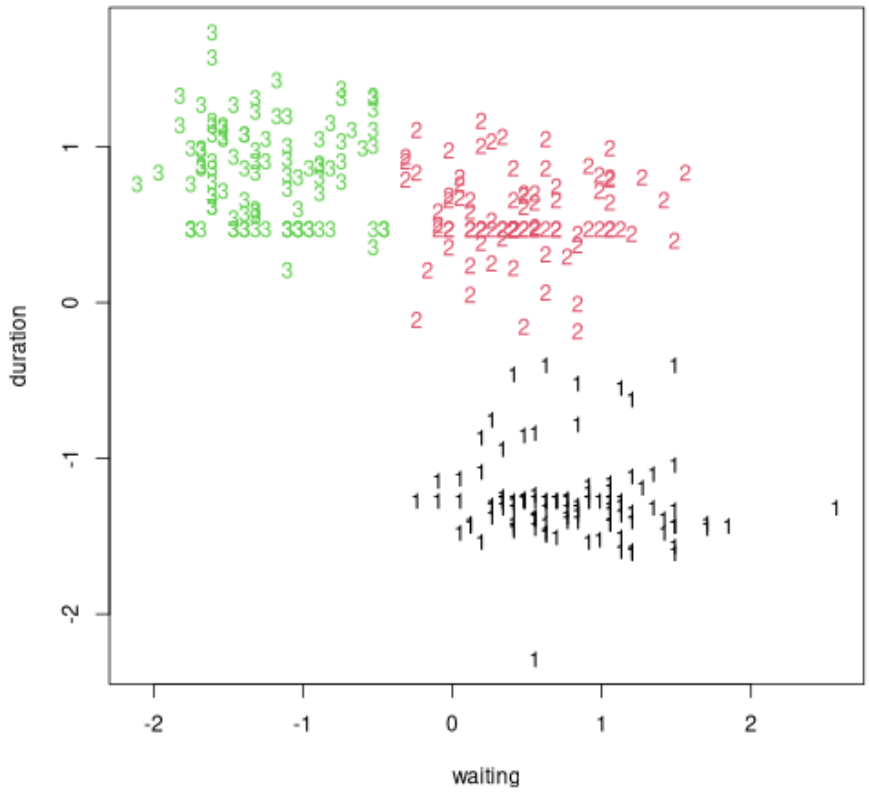

```
## Values of gap
plot(1:10,exp(cgg$Tab[,1]),xlab="k",ylab="S_k",type="l")
```




```
## Values of S_k
plot(1:10,cgg$Tab[,1],xlab="k",ylab="log S_k",type="l")
points(1:10,cgg$Tab[,2],xlab="k",ylab="log S_k",type="l",lty=2)

legend(6,5,c("log S_k in data","E(log S_k) uniform"),lty=1:2)
```




```
## Values of log(S_k) and its expectation under uniform distribution
## Unfortunately, the "clusGap"-function doesn<U+2019>t give out a clustering,
## so this has to be computed afterwards again.
geyser3 <- kmeans(sgeyser,3,nstart=100)
plot(sgeyser,col=geyser3$cluster,pch=clusym[geyser3$cluster])
```

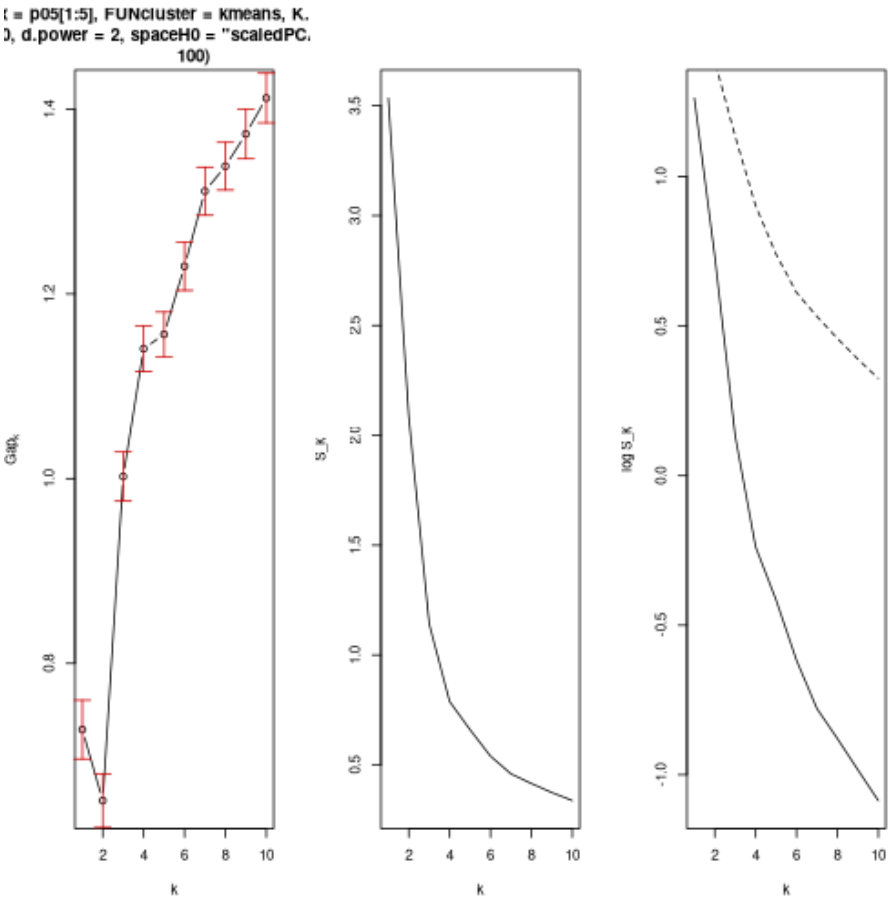



```
## Alternatively:
cgg2 <- gapnc(sgeyser,nstart=100)
```

Example 1.3.7 (bundestag)**.** Here the optimal cluster are not so convincing; visual clusters not spherical.

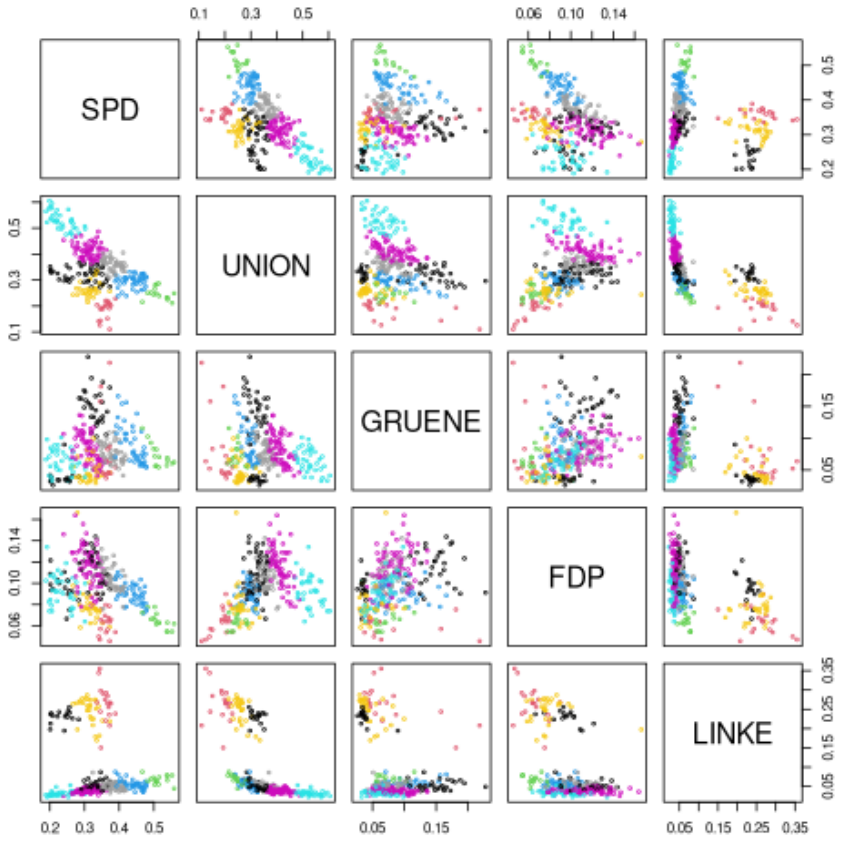
```
cgp05 <- clusGap(p05[1:5],kmeans,,K.max=10,B=100,d.power=2,spaceH0="scaledPCA",nstart=
print(cgp05,method="globalSEmax",SE.factor=2)

## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = p05[1:5], FUNcluster = kmeans, K.max = 10, B = 100, d.power = 2, spaceH
## B=100 simulated reference sets, k = 1..10; spaceH0="scaledPCA"
##  --> Number of clusters (method 'globalSEmax', SE.factor=2): 9
##            logW    E.logW       gap     SE.sim
##  [1,]  1.2627842 1.9911867 0.7284025 0.03182098
##  [2,]  0.7347556 1.3862288 0.6514732 0.02906250
##  [3,]  0.1326669 1.1349399 1.0022730 0.02658621
##  [4,] -0.2381619 0.9025250 1.1406869 0.02492804
##  [5,] -0.4155489 0.7407048 1.1562537 0.02421461
```

```
##  [6,] -0.6172533 0.6126186 1.2298718 0.02611286
##  [7,] -0.7789216 0.5322882 1.3112098 0.02607777
##  [8,] -0.8792488 0.4590052 1.3382539 0.02578382
##  [9,] -0.9834019 0.3897277 1.3731296 0.02680869
## [10,] -1.0864673 0.3255173 1.4119846 0.02693554

## --> Number of clusters (method <U+2019>globalSEmax<U+2019>, SE.factor=2): 9

## plots
par(mfrow = c(1, 3))
plot(cgp05)
plot(1:10,exp(cgp05$Tab[,1]),xlab="k",ylab="S_k",type="l")
plot(1:10,cgp05$Tab[,1],xlab="k",ylab="log S_k",type="l")
points(1:10,cgp05$Tab[,2],xlab="k",ylab="log S_k",type="l",lty=2)
legend(6,5,c("log S_k in data","E(log S_k) uniform"),lty=1:2)
```



Here the best performing cluster is for $K = 10$; 9 is the first (di poco ma ci entra, mentre 8 no) that is within 2 standard error from the best, and is the chosen solution. Now if we plot it

```
## log S_k values and expectation under uniform
## Re-compute 9-means
p059 <- kmeans(p05[1:5], 9, nstart=100)
pairs(p05[1:5],col=p059$cluster,cex=0.5)
```



A lot of cluster (1 and 9 have the same color btw)

Important remark 17 (Final remarks). Methods for estimating K are quite sensitive to model assumptions.
Estimating K is often ill-posed problem.
Could also use graphical diagnosis, subject-matter knowledge, or pragmatic considerations

1.3.6 K-means and big data

Compared to clustering methods introduced later:

- K-means is computationally simple, not memory intensive, and can be run for relatively large data sets. It's a big-data suitable method
- however really large data sets however require adaptations:

- **parallel computing**: different parts of an algorithm (eg iteration from different starting points) can be run independently at the same time on different processors.
- run K-means only **on a smaller random data subset**, then classify all remaining points to the closest mean fasten the process; however this involves some quality loss, and doesn't seem to be very popular in the literature
- a simple idea in Alguliyev et al. (2021):
  1. **split the data** set into q batches (sub datasets);
  2. run K-means on *each* batch independently in parallel (with smaller data sets often far fewer iterations are needed until convergence). This will produce a dataset with qK centroids/means;
  3. apply K-means on the qK centroids from previous step;
  4. assign all points in the original dataset to the closest "super-centroid" found in Step 3.

  Can use simulations on artificial (not so big but big enough) data sets to see how much quality is lost, and how different ideas compare.

Chapter 2

Dissimilarities

An intuitive definition of clustering: we want to find solutions where observations in same cluster should be similar, observations in different clusters should be dissimilar.
K-means is based on p-dimensional variables, but *many clustering methods are based on dissimilarity measures between object pairs.*
We can define dissimilarities for all kinds of data; dissimilarity depends on background/application.

Definition 2.0.1 (Dissimilarity)**.** Is a function $d : X \times X \to \mathbb{R}_0^+$ so that

- $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x}) \geq 0$ for $x, y \in X$
- $d(\mathbf{x}, \mathbf{x}) = 0$

Remark 14. For a dissimilarity some also require:

- $d(x, y) > 0$ for $\mathbf{x} \neq \mathbf{y}$
- triangle inequality:

$$d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}) \geq d(\mathbf{x}, \mathbf{z}), \quad \mathbf{x}, \mathbf{y}, \mathbf{z} \in X$$

but these twos are not always required

Definition 2.0.2 (Distance/metric)**.** A *dissimilarity* which fullfills the *triangle inequality.*

Important remark 18. There are also **similarities**, mostly equivalent to dissimilarities (e.g. $\max d - d$ can give similarity).
The term **proximity** is used for both.

2.1 Continuous variables

Definition 2.1.1 (Minkowski)**.** The Minkowski (L_q)-distance defined on units having p dimension

$$d_{Lq}(\mathbf{x}_i, \mathbf{x}_j) = \sqrt[q]{\sum_{i=1}^{p} d(x_{il}, x_{jl})^q}$$

where $d_i(x, y) = |x - y|$

Remark 15. In case:

- $q = 1$, d_{L1} we have the manhattan block distance
- $q = 2$, d_{L2} we have euclidean distance;
- in general higher q gives more weight to variables with larger distance; for $q \to \infty$, this converges to $\max_l |x_{il} - x_{jl}|$, $L\infty$ - or "maximum distance" (the distance between two units is the maximum distance among its variables).

Example 2.1.1. The distance between $(0, 0)$ and $(1, 1)$ or $(0, 2)$

- using manhattan is 2 in both cases one have to traverse two streets with manhattan
- it's $\sqrt{2}$ and 2 respectively using euclidean, which take the shortest path between two points

One should think what's the best distance for a certain (eg maybe if we're planning streets) distance based on manhattan is better, la butto li.

Important remark 19. Minkowski distances are not **scale equivariant**: they tends to weight more variables with larger variation so a standardization before applying it can be sensible in many cases.
Euclidean distance is **rotation invariant** (manhattan is not); if i rotate points the euclidean distances remains the same, while manhattan changes because

Definition 2.1.2 (Mahalanobis distance)**.** The (squared) Mahalanobis distance is defined by

$$d_M(\mathbf{x}_i, \mathbf{x}_j)^2 = (\mathbf{x}_i - \mathbf{x}_j)^\top S^{-1}(\mathbf{x}_i - \mathbf{x}_j),$$

where $\mathbf{S}$ is scatter matrix (the *sample variance-covariance matrix*).

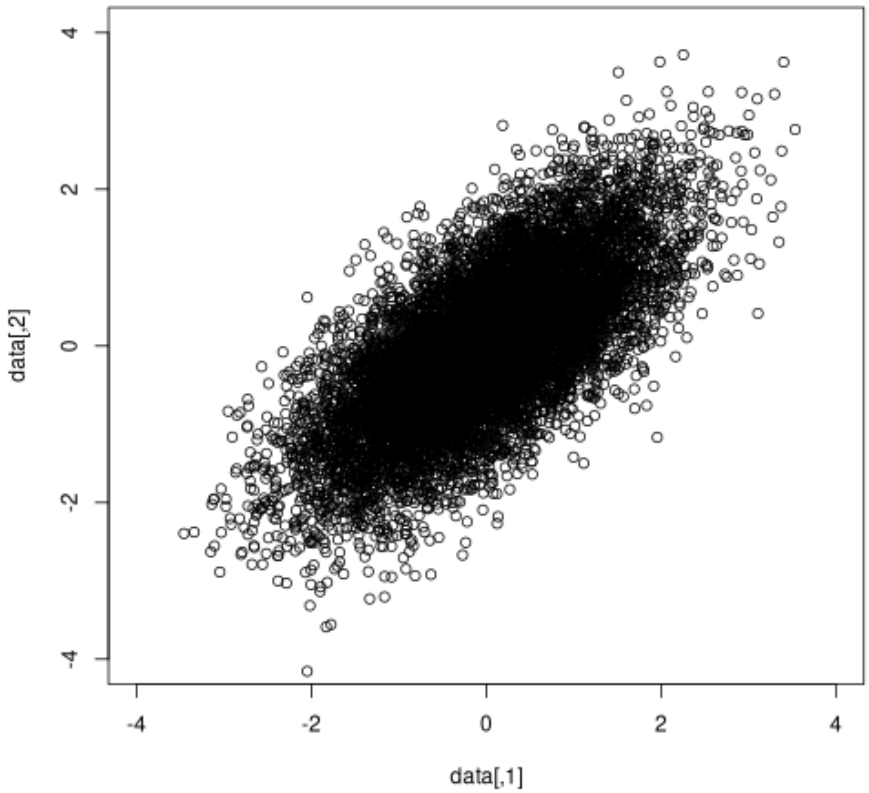
Important remark 20. Mahalanobis distance is **scale and rotation invariant**.

Remark 16. Fun fact; the gaussian mvn has some link to mahalanobis distance in the sense that the density (for p components) has some resembling components

$$f(\mathbf{x}) = (2\pi)^{-p/2} \det \Sigma^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu})^\top S^{-1}(\mathbf{x}_i - \boldsymbol{\mu})\right)$$

with Σ variance covariance matrix
The prof artificially generated data from two dimensional normal distribution: the ellipsoid having same malahnobis distance from the mean have same density/probability of being extracted.
Compared to euclidean distance weights the distance based on the correlation betweeen two variables so that one point that is far from the mean can have the same distance from it if involves correlated variables.
If two variables are correlated/share information we don't want this to weight to much in the distance calculation and mahalanobis diminish the impact of those
The same applies between malahanobis distance between units, the distance is is weighted verso il basso se proviene da un set di dati correlati

```r
set.seed(1)
mu <- c(0,0)
sigma <- matrix(c(1, 0.7, 0.7, 1), byrow = TRUE, ncol = 2)
data <- mvtnorm::rmvnorm(n = 10000, mean = mu, sigma = sigma)
plot(data, xlim = c(-4,4), ylim = c(-4,4))
```




```r
## d <- mahalanobis(data, colMeans(data), cov = cov(data))
## eps <- 0.00001
## d_one <- (d - 1) < eps & (d - 1) > 0
## d_two <- (d - 2) < eps & (d - 2) > 0
## mean(d_one)
## mean(d_two)
## no da generare come funzione 2 d
```

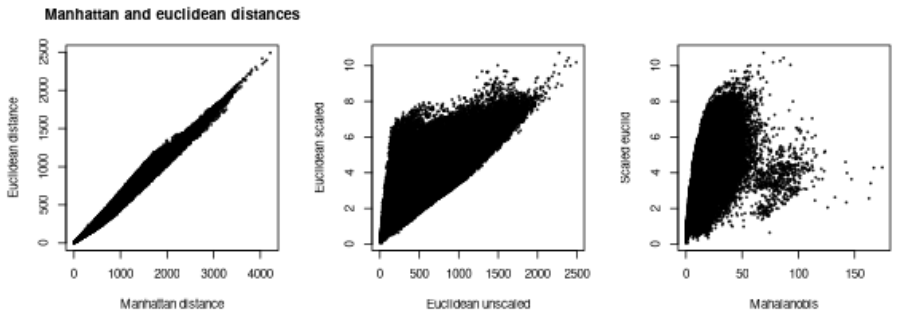
```r
library(pdfCluster)
data(oliveoil)
olive <- oliveoil[, 3:10]
solive <- scale(olive)
```

```r
dolive2 <- dist(olive, method = "euclidean") # one distance per every pair of object
dolive1 <- dist(olive, method = "manhattan")

## dist produces an object of class dist. One can a matrix of all
## distances (but this is memory intensive because dist matrix is
## simmetric)
## Finally distance matrix can be transformed into a dist-object by as.dist.

dolivematrix <- as.matrix(dolive2)
dolivematrix[1, 2] #  distance between observations 1 and 2.
```

```
## [1] 158.3667
```

```r
## Let's plot distances: these plot have one point per every pair of
## object

par(mfrow = c(1,3))
plot(dolive1, dolive2, cex = 0.3,
     xlab = 'Manhattan distance', ylab = 'Euclidean distance',
     main = "Manhattan and euclidean distances") # plot 1

dolives2 <- dist(solive, method = "euclidean")
plot(dolive2, dolives2, cex = 0.3, xlab = "Euclidean unscaled",
     ylab = "Euclidean scaled" )

## Manhattan and euclidean distances (plot1) are not so different in this case.
## Actually, it makes much more difference to scale the data (plot2)
## Distances are less related (ma boh diocane non puoi confrontare il
## second e il terzo con il primo plot per correlazione)

## The mahalanobis command can only compute a vector of Mahalanobis
## distances between one vector and one point.  So producing all
## distances is more tedious; here's how to make a distance matrix:

olivecov <- cov(olive)

mm <- apply(olive, 1, function(single_unit){
    mahalanobis(olive, single_unit,  olivecov)
})

## mahalm <- matrix(0, ncol = 572, nrow = 572)
## for (i in 1:572)
##     mahalm[i, ] <- mahalanobis(olive,
##                                as.numeric(olive[i, ]), # to have a vector
##                                olivecov)

## Plotting (plot3) we note that for the Mahalanobis distance it
## doesn't make a difference whether the data set is scaled or not.

plot(as.dist(mm), dolives2, cex=0.3,
```

```
    xlab = 'Mahalanobis', ylab = "Scaled euclid")
```



Important remark 21. Some remarks regarding euclidean vs mahalanobis distance.

- Euclidean distance with standardised variables can be written as a Mahalanobis distance with $\mathbf{S}$ diagonal matrix of variable's variances.
- in Euclidean distance using standardization will implicitly reduce the weight of variables with large variances

Mahalanobis distance

- will implicitly reduce the weight of *directions* in Euclidean space with large variance (i.e., the first principal components);
- will take into account correlations in the sense that correlated variables will be treated as carrying redundant information, which is downweighted. This makes sense if indeed several correlated variables share the same information that should only be used once. Mahalanobis distances are to be understood relative to the directions of large variance.

- Mahalanobis distances are not a good choice if correlation between variables constitutes relevant information itself (such as correlation between education level, savings, salary in social stratification).

2.2 Binary and categorical variables

Looking at a binary only dataset of $n \times p$ the notation

$$\mathbf{x}_i = (x_{i1}, \ldots, x_{ip}) \in \mathcal{B}_1 \times \ldots \times \mathcal{B}_p, i = 1, \ldots, n$$
$$\mathcal{B}_j = \{0, 1\}, \quad \forall j \in \mathbb{N}_p$$

Definition 2.2.1 (Simple matching distance)**.** Defined as the percentage of equal valued variables among the p of two units:

$$d_{SM}(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{p} \sum_{j=1}^{p} 1(x_{1j} \neq x_{2j})$$

Remark 17. Often for binary data, joint presences are meaningful but joint absences are not (eg Veronica: the joint presence of the gene is informative for plants to be classified similarly, the joint absence is not given the rarity). In these cases Jaccard can be useful.

Definition 2.2.2 (Jaccard distance)**.** Defined as:

$$d_J(\mathbf{x}_1, \mathbf{x}_2) = 1 - \frac{\sum_{j=1}^{p} 1(x_{1j} = 1 \text{ and } x_{2j} = 1)}{\sum_{j=1}^{p} 1(x_{1j} = 1 \text{ or } x_{2j} = 1)}$$

Example 2.2.1. If

$$\begin{aligned}
&\mathbf{x}_1 = (1,0,0,0,0,0,0,0,0,0)\\
&\mathbf{x}_2 = (0,0,0,1,0,0,0,0,0,0)\\
&d_{SM}(\mathbf{x}_1, \mathbf{x}_2) = 0.2\\
&d_J(\mathbf{x}_1, \mathbf{x}_2) = 1, \quad \text{(maximum)}
\end{aligned}$$

d_{SM} has observations with very few ones similar d_J depends on whether ones are in same places.

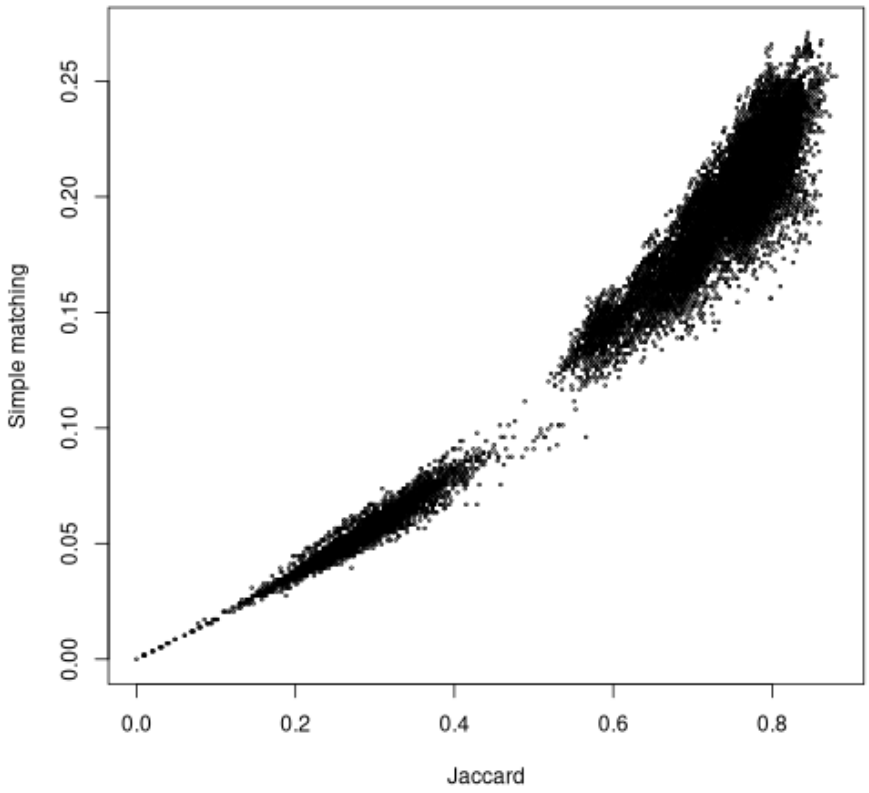
Example 2.2.2. If

$$\begin{aligned}
&\mathbf{x}_1 = (1,1,1,1,1,1,1,1,1,0)\\
&\mathbf{x}_2 = (0,1,1,1,1,1,1,1,1,1)\\
&d_{SM}(\mathbf{x}_1, \mathbf{x}_2) = d_J(\mathbf{x}_1, \mathbf{x}_2) = 0.2
\end{aligned}$$

Remark 18. we have that

- `sm` in R-package `nomclust` computes d_{SM}
- `dist` computes d_J
- d_{L1}/p is d_{SM} for binary variables.

```
veronica <- read.table("data/veronica.dat")
jveronica <- dist(veronica, method = "binary") # jaccard
smveronica <- dist(veronica, method = "manhattan")/583 # simple matching
plot(jveronica, smveronica, cex = 0.3, xlab = "Jaccard", ylab = "Simple matching")
```



Important remark 22. Simple matching distance is more generale and category doesn't need to be binary (eg party preference)

2.3 Correlation dissimilarity

Remark 19. In certain application we may be interested in clustering variable rather than units/observation.
Correlation measures how strongly variables are related (regardless of whether they take similar values)

Example 2.3.1. In Veronica data, could cluster AFLP bands instead of plants. In Bundestag data, could cluster parties: parties (variable) may be seen similar if strong in same constituencies.

Definition 2.3.1 (Correlation distance)**.** Given that the correlation is bounded $-1 \leq r(x_{\cdot 1}, x_{\cdot 2}) \leq 1$, we can set

$$d_C(x_{\cdot 1}, x_{\cdot 2}) = \frac{1}{2}(1 - r(x_{\cdot 1}, x_{\cdot 2}))$$

obtaining

$$d_C(x_{\cdot 1}, x_{\cdot 2}) = \begin{cases} 0 & r = 1 \\ 1 & r = -1 \end{cases}$$

Example 2.3.2 (Bundestag)**.** We have:

```
p05 <- read.table("data/bundestag.dat", header = TRUE)
corp05 <- cor(p05[1:5])
(cordistp05 <- 0.5*(1 - corp05)) # matrix of dissimilarities (not distances no triang

##              SPD     UNION    GRUENE       FDP     LINKE
## SPD    0.0000000 0.7831133 0.4553201 0.6650399 0.5668952
## UNION  0.7831133 0.0000000 0.5764791 0.3218088 0.8110412
## GRUENE 0.4553201 0.5764791 0.0000000 0.3306895 0.6822384
## FDP    0.6650399 0.3218088 0.3306895 0.0000000 0.7370640
## LINKE  0.5668952 0.8110412 0.6822384 0.7370640 0.0000000
```

Definition 2.3.2. Different definition

$$d_{CN}(x_{\cdot 1}, x_{\cdot 2}) = 1 - |r(x_{\cdot 1}, x_{\cdot 2})|$$

has largest dissimilarity for r = 0 and small if $|r|$ large.

Important remark 23. So:

- d_C treats two strongly negatively correlated variables as very dissimilar (opposite tendency).
- d_{CN} treats them as very similar (holding very similar information).

Depending on the application we should decide one or another

2.4 Mixed type of variables and missing values

Remark 20. For data sets with variables of mixed type (e.g., continuous, categorical, dichotomic, ordinal), can aggregate different dissimilarities for different types of variables (e.g. simple matching for categorical, L1 for continuous, Jaccard for presence/absence)

Definition 2.4.1 (Gower coefficient)**.** Distance between two units on p different variables ($l = 1, \ldots, p$ is the variable index)

$$d_G(\mathbf{x}_i, \mathbf{x}_j) = \frac{\sum_{l=1}^{p} \frac{w_l}{s_l} \delta_{ijl} d_l(x_{il}, x_{jl})}{\sum_{l=1}^{p} \delta_{ijl} w_l}$$

with

- d_l is the distance governing variable l
- δ_{ijl} helps dealing with missing values (it's an indicator function actually): $\delta_{ijl} = 1$ iff none of the two values x_{il}, x_{jl} are missing and 0 otherwise

- w_l is an optional weight for variable importance (often $w_l = 1$)
- s_l scaling/normalization of d_i: for Gower coefficient it is $s_l = \max d_l$ but other choices are possibile
- for continuous variables generally $d_l(x_{il}, x_{jl}) = |x_{il} - x_{jl}|$ is used, a generalization of d_{L1}

Remark 21. If w_l are 1 the maximum value it can take is 1 credo (d_l/s_l will be always ≤ 1)

Important remark 24. Some remarks:

- Note that this missing value treatments requires that variables are scaled so that differences in one variable can be meaningfully compared with differences in another. For example dissimilarity between $\mathbf{x}_1$ and $\mathbf{x}_2$ may be computed on variables 1, 2, 3 (where $\mathbf{x}_1$, $\mathbf{x}_2$ both are non-missing), dissimilarity between $\mathbf{x}_3$ and $\mathbf{x}_4$ may be computed on variables 3, 4, 5, but should give a comparable value;
- denominator can be zero: this occurs, and Gower is undefined, if there's no variables where all couple are not missing. In case one can set it to the most dissimilar value
- Jaccard d_J is undefined if $\mathbf{x}_i$ and $\mathbf{x}_j$ have only zeroes since the denominator is then $\sum_{l=1}^{q} 1(x_{il} = 1 \text{ or } x_{jl} = 1) = 0$
- Gower: for use of d_J with $q = 1$ as d_l for presence/absence data, set $\delta_{ijl} = 0$ if $x_{il} = x_{jl} = 0$, both treated as missing!

TODO: chiarissimo direi

- Gower could also be used with p *groups of* variables: e.g. if $l = 1$ Mahalanobis distance for all continuous variables, $l = 2$ Jaccard distance for all presence/absence variables.
  May then choose w_1 and w_2 as numbers of variables involved in d_1 and d_2 to give all variables same weight.
  Missing values may require treatment inside d_1 and d_2 (Jaccard can just treat them as absences).

Important remark 25 (R usage). The original Gower coefficient can be computed by function `cluster::daisy`:

- it uses Gower scaling $s_l = \max d_l$
- weights can be specified
- variable types can be specified using a list in `type` argument. Particularly in case of binary variable one want to specify whether to use simple matching (`symm`) or Jaccard (`asymm`).
  - unspecified: numerical with L_1.
  - `factors` are considered as nominal variables (simple matching is applied)
  - `ordered factors` are treated as ordinal, will be coded by progressive numeric (1, 2, 3, ...) and then treated as numerical.

  - `symm`: binary variables with simple matching (as will be used for categorical factors) ,
  - `asymm`: binary variables with Jaccard

  `symm` and `asymm` can also take vectors such as `type=list(asymm=c(2,4),symm=c(3,6))` if vars 2,3,4,6 are binary.

- General variable-wise distances are not supported.
- Any variable could have missing values, `NA`, which daisy handles as required by Gower.

Example 2.4.1 (Boston dataset)**.** 506 districts of Boston for which 14 variables are collected; most of these are interval-scaled, but there are

- `chas` (4th variable): *binary*
- `rad` (9th variable): ordinal variable, meaning that the ranking of the values is meaningful ($24 > 8 > 4$), but the absolute values of differences are not (the distance between 24 and 8 is in no meaningful way four times the distance between 8 and 4).

As said `daisy` takes factors as nominal variables (simple matching) and variables of type `ordered` as ordinal. Eg let's treat `rad` as **nominal** by transforming into a factor (this is not appropriate but let's see how changes?):

```
library(cluster)

housing <- read.table("data/Boston.dat", header = TRUE)
housing$rad <- as.factor(housing$rad)
head(housing)

##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
##   medv
## 1 24.0
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7

summary(housing)

##       crim                zn             indus            chas
##  Min.   : 0.00632   Min.   :  0.00   Min.   : 0.46   Min.   :0.00000
##  1st Qu.: 0.08205   1st Qu.:  0.00   1st Qu.: 5.19   1st Qu.:0.00000
```

```
##  Median : 0.25651   Median :  0.00   Median : 9.69   Median :0.00000
##  Mean   : 3.61352   Mean   : 11.36   Mean   :11.14   Mean   :0.06917
##  3rd Qu.: 3.67708   3rd Qu.: 12.50   3rd Qu.:18.10   3rd Qu.:0.00000
##  Max.   :88.97620   Max.   :100.00   Max.   :27.74   Max.   :1.00000
##
##       nox               rm             age              dis
##  Min.   :0.3850   Min.   :3.561   Min.   :  2.90   Min.   : 1.130
##  1st Qu.:0.4490   1st Qu.:5.886   1st Qu.: 45.02   1st Qu.: 2.100
##  Median :0.5380   Median :6.208   Median : 77.50   Median : 3.207
##  Mean   :0.5547   Mean   :6.285   Mean   : 68.57   Mean   : 3.795
##  3rd Qu.:0.6240   3rd Qu.:6.623   3rd Qu.: 94.08   3rd Qu.: 5.188
##  Max.   :0.8710   Max.   :8.780   Max.   :100.00   Max.   :12.127
##
##       rad           tax           ptratio          black           lstat
##  24     :132   Min.   :187.0   Min.   :12.60   Min.   :  0.32   Min.   : 1.73
##  5      :115   1st Qu.:279.0   1st Qu.:17.40   1st Qu.:375.38   1st Qu.: 6.95
##  4      :110   Median :330.0   Median :19.05   Median :391.44   Median :11.36
##  3      : 38   Mean   :408.2   Mean   :18.46   Mean   :356.67   Mean   :12.65
##  6      : 26   3rd Qu.:666.0   3rd Qu.:20.20   3rd Qu.:396.23   3rd Qu.:16.95
##  2      : 24   Max.   :711.0   Max.   :22.00   Max.   :396.90   Max.   :37.97
##  (Other): 61
##       medv
##  Min.   : 5.00
##  1st Qu.:17.02
##  Median :21.20
##  Mean   :22.53
##  3rd Qu.:25.00
##  Max.   :50.00
##
```

```
## Gower dissimilarity
## Specify that 4-th variable chas (binary) is treated as symmetric i.e.,
## simple matching distance.
vartype <- list(symm = 4)
gdhousing <- daisy(housing, metric = "gower", type = vartype) # don't print

## gdhousing is a dist object
class(gdhousing)

## [1] "dissimilarity" "dist"
```

Specifying `metric="gower"` is unnecessary, because this is automatically done if factor or ordered variables are in data. Note also that treating a binary variable as `symm` is mathematically equivalent to treating it as numerical.
Now treat

- `rad` as **ordinal**, which is appropriate: orginally has values `1,2,3,4,5,6,7,8,24`, as ordinal variables these are treated as `1,2,3,4,5,6,7,8,9` in dissimilarity computation;
- `chas` as asymmetric (Jaccard, not so appropriate)

```
housing$rad <- as.ordered(housing$rad)
vartype <- list(asymm = 4)
gdhousing2 <- daisy(housing, metric = "gower", type = vartype)
```

`daisy` also allows for `metric="euclidean"` or `metric="manhattan"`, in which case missing values are "outweighted", i.e.

$$d_{Lq}(\mathbf{x}_i, \mathbf{x}_j) = \sqrt[q]{\frac{p}{r} \sum_{x_{ij}, x_{jl} \neq \texttt{NA}} d_i(x_{il}, x_{jl})^q}$$

where r is the number of variables on which both $x_{il}, x_{jl} \neq$ `NA`. The factor p/r is there so that the sum is weighted as if it has p entries even if there are only $r < p$ because of missing values.

2.5 Multidimensional scaling (MDS)

NB: prof consiglia di fare mds per visualizzazione quando si lavora con distanze non euclidee, can be useful for clustering

It's not a cluster analysis method nor a dissimilarity but useful in connection with general dissimilarities that are not euclidean. It's basically a visualization method and also sometimes used as dimension reduction method.
Say we have a dataset $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ characterised by a dissimilarity d (either the dataset comes as dissimilarity matrix but we don't know data or we have data and computed dissimilarity which is not euclidean, so we don't have a natural visualization in the euclidean space as showed by `plot` in R).
The euclidean space is the space where our intuition works: it's not so clear how to visualize data with dissimilarities that are non euclidean. Basically what MDS does is that if we have obs characterised by this non euclidean dissimilarity, it finds new observations in the p-dimensional euclidean space $\mathbf{y}_1, \dots, \mathbf{y}_n \in \mathbb{R}^p$ such that their euclidean distance is approximately the same non euclidean on the original correspondent

$$d(\mathbf{x}_i, \mathbf{x}_j) \approx d_{L2}(\mathbf{y}_i, \mathbf{y}_j).$$

This would allow us to plot in a known reference (cartesian space). Di base possiamo anche essere interessati al fatto che non vi sia perfetta corrispondenza ma proporzionalità tra le due distanze, es with fixed $b > 0$,

$$bd(\mathbf{x}_i, \mathbf{x}j) \approx d_{L2}(\mathbf{y}_i, \mathbf{y}_j)$$

is enough, this approximates dissimilarities proportionally.
If infact the original dissimilarity is the euclidean one, that is $d = d_{L2}$, principal component does something like this (reducing the number of variables to eg 2 in order to visualize on the cartesian plane with the most information): PCA provides something like a low-dimensional approximation, e.g., $bdL2 \approx d_{L2}^{2-d}$.
We are interested in MDS which do not have euclidean dissimilarity because if we do have euclidean dissimilarity pretty much is something similar to what principal component does.
There are several MDS methods, that is to find actually b and $\mathbf{y}_1, \dots, \mathbf{y}_n$ in equation above, to make $bd(\mathbf{x}_i, \mathbf{x}j) \approx d_{L2}(\mathbf{y}_i, \mathbf{y}_j)$.
The **Ratio MDS** minimise "stress", defined as:

$$\sigma = \sqrt{\frac{\sum_{i<j} \left[bd(\mathbf{x}_i, \mathbf{x}_j) - d_{L2}(\mathbf{y}_i, \mathbf{y}_j)\right]^2}{n(n-1)/2}}$$

by

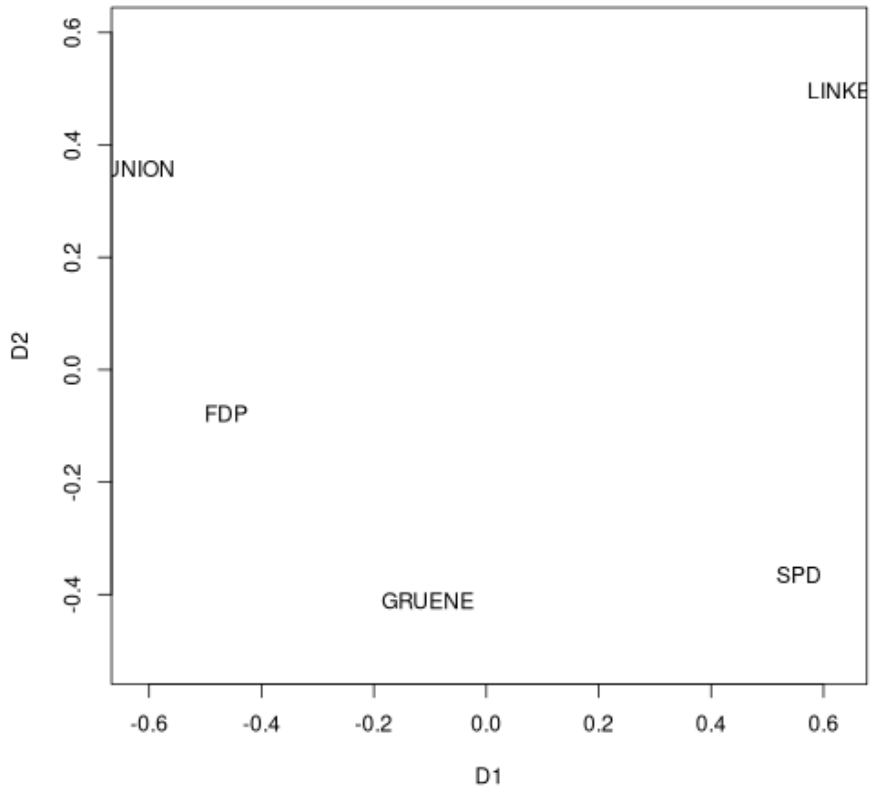
- choosing $\mathbf{b}$ and $\mathbf{y}_1, \dots, \mathbf{y}_n$: at numerator we have a least square principle, which is minimized (denominator is a normalizing constant, counting the number of unique i, j couples)
- minimization is conducted under the constraint $\sum_{i<j} b^2 d(\mathbf{x}_i, \mathbf{x}_j)^2 = \frac{n(n-1)}{2}$. If the constraint holds and we choose $\mathbf{y}_i, \mathbf{y}_j$ to be all equal, then $\sigma = 1$ (which is the max value we can obtain).
  Thus this constraint allows a percentage-like interpretation of stress squared approximation error relative to $\sum_{i<j} bd(\mathbf{x}_i, \mathbf{x}_j)^2$.
  So stress is **percentage approximation error** related to changing data and distance function (the lower the better).

Numerically it's quite complicated but R-package `smacof` does this for us

Example 2.5.1 (Mds on bundestag)**.** Exercise out of nowhere: how the correlation distance MDS should look like?

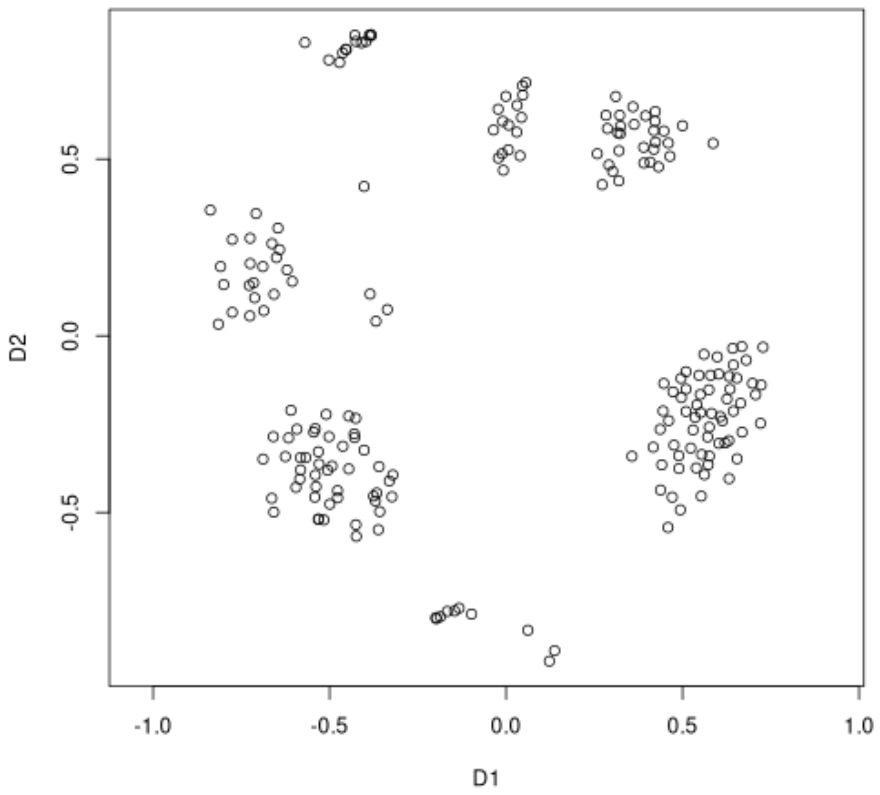
```
## this is our distance
p05 <- read.table("data/bundestag.dat", header = TRUE)
corp05 <- cor(p05[1:5])
(cordistp05 <- 0.5*(1 - corp05)) # matrix of distances

##              SPD     UNION    GRUENE       FDP     LINKE
## SPD    0.0000000 0.7831133 0.4553201 0.6650399 0.5668952
## UNION  0.7831133 0.0000000 0.5764791 0.3218088 0.8110412
## GRUENE 0.4553201 0.5764791 0.0000000 0.3306895 0.6822384
## FDP    0.6650399 0.3218088 0.3306895 0.0000000 0.7370640
## LINKE  0.5668952 0.8110412 0.6822384 0.7370640 0.0000000

# we have five objects, we want to plot them (five points) on a two dimensional
# plot and place them such that the lower  correlation distance the closer are the
# points on the cartesian plane.
# The lowest distance we have is between FDP and UNION (0.321) which will
# probably be nearer than other.

# MDS on distance matrix: we have to choose dimensionality p (of R^p vectors)
# as well, tipically it's 2 for plotting
library(smacof)

## Caricamento del pacchetto richiesto:  plotrix
## Caricamento del pacchetto richiesto:  colorspace
## Caricamento del pacchetto richiesto:  e1071
##
## Caricamento pacchetto:  'smacof'
## Il seguente oggetto è mascherato da 'package:base':
##
##     transform

# ndim below is the number of dimensions  (2 is default)
(mdsparties <- mds(cordistp05, ndim = 2))
```

```
##
## Call:
## mds(delta = cordistp05, ndim = 2)
##
## Model: Symmetric SMACOF
## Number of objects: 5
## Stress-1 value: 0.06
## Number of iterations: 15

mdsparties$conf # placement on the two dimension (conf for configuration)

##                D1          D2
## SPD     0.5565181 -0.36521372
## UNION  -0.6171174  0.35821180
## GRUENE -0.1038678 -0.41143844
## FDP    -0.4629498 -0.07802883
## LINKE   0.6274169  0.49646919

mdsparties$stress # 6%, not much information missing, this is a good approximatio/rep

## [1] 0.05975537

# plotting
# below we setup the figure: asp=1 is the "aspect ratio" and means that
# distances are correctly represented because x and y axis scaling are the
# same.
plot(mdsparties$conf, type = "n", asp = 1) #just setup the figure, dont plot
text(mdsparties$conf, labels = rownames(mdsparties$conf)) # we plot names not point
```



Example 2.5.2 (MDS on veronica)**.** Doing on veronica data and jaccard distance

```
## MDS on veronica
mdsveronica <- mds(jveronica)
plot(mdsveronica$conf, asp = 1)
```




```
mdsveronica$stress # [1] 0.2577986 # bad 26%, quite a bit of information not represen

## [1] 0.2577986
```

Problem here is that veronica data has a lot of variables and reducing all to two variables causes higher information loss. When stress has high value one could dimension try ndim higher than 2 (which lower stress tipically); problem is that standard images are 2-d.
The plot is however useful and seems to generate some like 5-6 clusters of species and some other information that are less clearly clustered

Important remark 26. Note: plot command for mds exists, i.e., can use `plot(veronicamds)` rather than `plot(veronicamds$conf)`, but the latter works better with user's graphics parameters.

2.6 Similarity between clusterings

We may beinterested in:

- compare different clustering solutions between them;

- compare generated clustering with external grouping (for interpretation and validation);
- even use "true" external grouping to compare different clustering methods.

Other applications:

- Investigate effect of variables or outliers (if removed);
- Investigate stability of clustering by comparing clusterings on resampled data sets;
- To cluster clusterings.

Definition 2.6.1 (Rand index)**.** Let $\mathcal{C}_1 = C_{11}, \ldots, C_{1K_1}$ and $\mathcal{C}_2 = C_{21}, \ldots, C_{2K_2}$ be partitions of our dataset $\mathcal{D}$ (two alternative clustering, composed of K_1 and K_2 clusters respectively, which are just a set/collection of units). For $\mathbf{x}, \mathbf{y} \in \mathcal{D}$ and $\mathrm{j} = 1, 2$, define the indicator function

$$i_j(\mathbf{x}, \mathbf{j}) = 1(\mathbf{x}, \mathbf{y} \text{ are in the same cluster in } C_j)$$

Rand index (Rand (1971)) is defined as simple matching similarity of i_1, i_2:

$$R(C_1, C_2) = \frac{1}{n(n-1)/2} \sum_{\mathbf{x},\mathbf{y} \in D} 1(i_1(\mathbf{x}, \mathbf{y}) = i_2(\mathbf{x}, \mathbf{y}))$$

That is the percentage of couples of units which are classified in the same way in the two clustering solutions (either together in the same cluster or in different clusters).

Important remark 27. This indicator

- pro: doesn't require equal clusters size $K_1 = K_2$
- pro: it doesn't require *cluster matching* either (cluster are treated just as label and it's not important that labels in a clustering solutions are different from the other clustering solutions)
- cons: typical values depend on number of clusters / cluster size K_1, K_2 (easier to find pairs that are consistently classified in different solutions if the number of cluster is lower).

Remark 22. For cons reason what is used in the literature is the following: it standardize the original given its expected value given the number of clusters in the two solutions K_1 and K_2, and the number of elements in each cluster of each solution $a_1, \ldots, a_{K_1}$ and $b_1, \ldots, b_{K_2}$

Definition 2.6.2 (Adjusted rand index)**.**

$$ARI(\mathcal{C}_1, \mathcal{C}_2) = \frac{R(\mathcal{C}_1, \mathcal{C}_2) - ER}{1 - ER}$$

where:

- $R(\mathcal{C}_1, \mathcal{C}_2)$ is Rand index as above

- ER is expected value of rand index, $R(\mathcal{C}_1^*, \mathcal{C}_2^*)$, where $\mathcal{C}_1^*, \mathcal{C}_2^*$ are generated using random partitions on $\mathcal{D}$ with n objects $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$:

The algorithm goes like this:

- Fix the number of clusters in each solution $K_j = |\mathcal{C}_j| = |\mathcal{C}_j^*|$ with $j = 1, 2$; let $a_1, \ldots, a_{K_1}$ be cluster sizes of $\mathcal{C}_1$ while $b_1, \ldots, b_{K_2}$ cluster sizes of $\mathcal{C}_2$
- $\mathcal{C}_1^*$: draw randomly without replacement clusters with sizes $a_1, \ldots, a_{K_1}$ from $\mathcal{D}$;
- $\mathcal{C}_2^*$: independently of $\mathcal{C}_1^*$, draw randomly without replacement clusters with sizes $b_1, \ldots, b_{K_2}$ from $\mathcal{D}$.

NB: In one exam asked to show this

Under this model the expected value can be shown

$$\mathbb{E}\left[ARI(\mathcal{C}_1^*, \mathcal{C}_2^*)\right] = \frac{ER - ER}{1 - ER} = 0$$

Important remark 28. ARI takes values between -1 and 1:

- 1 if clusterings are identical,
- 0 is its expected value for two independent random clusterings (very low value; would expect reasonable clusterings on same data to be somehow positively related). The clustering solution does not have much in common except communality due to randomness
- negative values are for mainly discordant clustering solutions

Can be seen as consistency excess compared to random consistency

Remark 23. There's also an explicit formula on how to compute ARI, which is the following

Theorem 2.6.1.

$$ARI(\mathcal{C}_1, \mathcal{C}_2) = \frac{\sum_{ij}\binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}\right]/\binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}\right]/\binom{n}{2}}$$

where i runs over N_{K_1}, j runs over N_{K_2}, $n_{ij} = |\mathcal{C}_{1i} \cap \mathcal{C}_{2j}|$, $a_i = \sum_{j=1}^{K_2} n_{ij}$ and $b_j = \sum_{i=1}^{K_1} n_{ij}$.

Example 2.6.1 (Olive oil)**.** On olive data

```
library(cluster)
library(mclust) # This has the adjustedRandIndex command.
set.seed(1234)

# How much difference is there in scaling vs not scaling data?
solive <- scale(olive)
olive3 <- kmeans(olive, 3, nstart = 100)
olive3s <- kmeans(solive, 3, nstart = 100)
```

```
## Adjusted Rand indexes comparing solution on scaled and unscaled:
## coefficient is 0.4587804 so these clusterings are somewhat different.
## remember 0 is for random consistency while 1 is for equality
adjustedRandIndex(olive3$cluster, olive3s$cluster)

## comparation with actual geographical area
## unscaled: 0.3182057
## scaled: 0.448355
# both OK but not great, with scaling clearly better
adjustedRandIndex(olive3$cluster, oliveoil$macro.area)
adjustedRandIndex(olive3s$cluster, oliveoil$macro.area)

## The gap statistic suggests a higher number of clusters (not run below for
## timing: they're actually 18/19)
if (FALSE) {
  cgolive <- clusGap(solive, kmeans, 20, B = 100,
                     d.power = 2, spaceH0 = "scaledPCA",
                     nstart = 100)
  print(cgolive, method="globalSEmax", SE.factor=2)
  # B=100 simulated reference sets, k = 1..20; spaceH0="scaledPCA"
  # --> Number of clusters (method 'globalSEmax', SE.factor=2): 19
}
## apply kmeans with 19 clusters
kmolive19 <- kmeans(solive, 19, nstart = 100)
## check if the 19-cluster solution performance compared to region and
## macroarea. It's not that better.
adjustedRandIndex(kmolive19$cluster, oliveoil$region)
adjustedRandIndex(kmolive19$cluster, oliveoil$macro.area)
```

2.7 Further methods for dissimilarities

The following are methods which take as input a dissimilarity matrix (euclidean or not) as well.

2.7.1 Partitioning Around Medoids (PAM, Kaufman and Rousseeuw 1990))

This is like k-means for dissimilarity data: on fixes the number of cluster k and then choose k-centroids of the cluster and then assign other object to the cluster to the closest centroid.

The only difference is that, with a general dissimilarity matrix we cannot generally compute means (we may not know the original data and its space/type of variable as well).

For PAM, centroids ("medoids") are data objects.

Definition 2.7.1 (PAM clustering)**.** Considering the dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, $\mathbf{x}_i \in \mathcal{X}$, and a dissimilarity measure $d : \mathcal{X}^2 \to \mathbb{R}_0^+$
The PAM clustering for given chosen K uses *medoids* which are data object

$\mathbf{m}_1^{PK}, \ldots, \mathbf{m}_K^{PK} \in \mathcal{D}$ define group clustering $c^{PK}(1), \ldots, c^{PK}(n)$ which minimize the objective function T which depends on clustering solutions and chosen medoids by minimizing distance of each object from its medoid:

$$T(\mathcal{C}, \mathbf{m}_1^{PK}, \ldots, \mathbf{m}_K^{PK}) = \sum_{i=1}^{n} d(\mathbf{x}_i, \mathbf{m}_{c(i)}^{PK})$$

For k-means d is squared euclidean, which can be used here as well but does not need to be

Important remark 29. Some remarks:

- more flexible than K-means (we can run it with any kind of dissimilarity);
- can be run for Euclidean data: using squared or unsquared euclidean distance for d (or Mahalanobis as well);
  - unsquared Euclideans don't penalise large within-cluster distances that much, so it will be better at finding non-spherical clusters (but not perfect);
  - in case of squared Euclidean, K-means is better (because means are better than medoids).

Computation of PAM Problem with PAM, like k-means many possible clustering solutions so one can't really find the global optimal (especially if data is large).

Kaufman and Rousseeuw proposed deterministic algorithm for *local optimum*, which is fairly good but slow.

Handling dissimilarities is bad for large data sets ($n > 4000$); anyway one can use function `clara` ("clustering large applications") which does PAM on subsets of data and then assign all the remaining data to nearest medoid.
Works only for euclidean data though, so one should not use PAM with large dataset

NB: non lo chiede all'esame, è per informazione

The algorithm works like that: . There are two parts: the first does a fairly good clustering from scratch (which is similar to kmeans++). In second phase clustering will be used/finetuned and some object will be swapped/exchange and then it will be checked if the changes improves the objective function

1. **build phase**
   (a) start with $k = 1$ (start with 1 cluster)
   (b) take/fix the medoids from previous step (if available) $(\mathbf{m}_1^{lk}, \ldots, \mathbf{m}_{k-1}^{lk}) = (\mathbf{m}_1^{l(k-1)}, \ldots, \mathbf{m}_{k-1}^{l(k-1)})$ then choose as the k-th one the object which minimizes sum of distances

$$\mathbf{m}_k^{lk} = \arg\min_{\mathbf{m}_k \in \mathcal{D}} \sum_{i=1}^{n} d(\mathbf{x}_i, \mathbf{m}_{c^k(i)})$$

   For $k = 1$ we check all the dataset and choose the object minimizing sum of distances with respect to all other objects. For $k > 1$ take another medoid, excluding the first, and do the same by minimizing

distances on all the remaining.
Clustering in k groups is done then by assigning each i-th object to the nearest medoid where $c^k(i) = \arg\min_{j\in\mathbb{N}_k} d(\mathbf{x}_i, \mathbf{m}_j^{lk})$.

(c) **END**: if $k = K$ otherwise $k = k + 1$ and go to previous step

2. **swap phase**, at this phase the assignment of objects to medoid is not necessarily overall optimal so we make

(a) we have a clustering solution $T^0 = T(c^k(1), \dots, c^k(n))$, and our medoids $(\mathbf{m}_1^*, \dots, \mathbf{m}_K^*) = (\mathbf{m}_1^{lK}, \dots, \mathbf{m}_K^{lK})$, this swap iteration is set to $q = 1$

(b) considering all object which are not medoids $\mathbf{x}_i \notin \{\mathbf{m}_1^*, \dots, \mathbf{m}_K^*\}$, for all pairs (i, k) with clusterings/medoids replace non-medoids with medoids and recompute the clustering/association:

$$T_{ik} = T(\mathcal{C}^{ik}, \mathbf{m}_1^{ik}, \dots, \mathbf{m}_K^{ik})$$

where $(\mathbf{m}_1^{ik}, \dots, \mathbf{m}_K^{ik})$ are $(\mathbf{m}_1^*, \dots, \mathbf{m}_K^*)$ but with $\mathbf{m}_k$ replaced by $\mathbf{x}_i$ and $\mathcal{C}^{ik}$ assigns every object to the cosest centroid in $\{\mathbf{m}_1^{ik}, \dots, \mathbf{m}_K^{ik}\}$

(c) we check if there are miglioramenti: we check among all (i, k) the one that minimizes the objective function $(j, l) = \arg\min_{(i,k)} T_{ik}, T^q = T_{jl}$ and take it as new best

(d) **END**; if $T^q \geq T^{q-1}$ otherwise $(\mathbf{m}_1^*, \dots, \mathbf{m}_K^*) = (m_1^{jl}, \dots, m_K^{jl})$, $q = q + 1$

TODO: va in mona ci mollo

Example 2.7.1.

```
library(cluster)
library(mclust)
## Package 'mclust' version 6.1.1
## Type 'citation("mclust")' for citing this R package in publications.

library(smacof)

bundestagk5 <- kmeans(p05[1:5], 5, nstart=100)
# By default, if pam is called with a data set that is not a dist-object, the
# Euclidean distance is used (raw, not squared euclidean):
bundestagp5 <- pam(p05[1:5], 5)
adjustedRandIndex(bundestagk5$cluster, bundestagp5$cluster)

## [1] 0.7249983

## [1] 0.7249983

# Alternatively, pam can be started with a dist-object, which allows
# computing it eg with the Manhattan-distance:
p05manhattan <- dist(p05[1:5], method = "manhattan")
bundestagp5m <- pam(p05manhattan,5)

## Euclidean vs manhattan PAM: quite different (0.6607979)
adjustedRandIndex(bundestagp5$cluster, bundestagp5m$cluster)
```

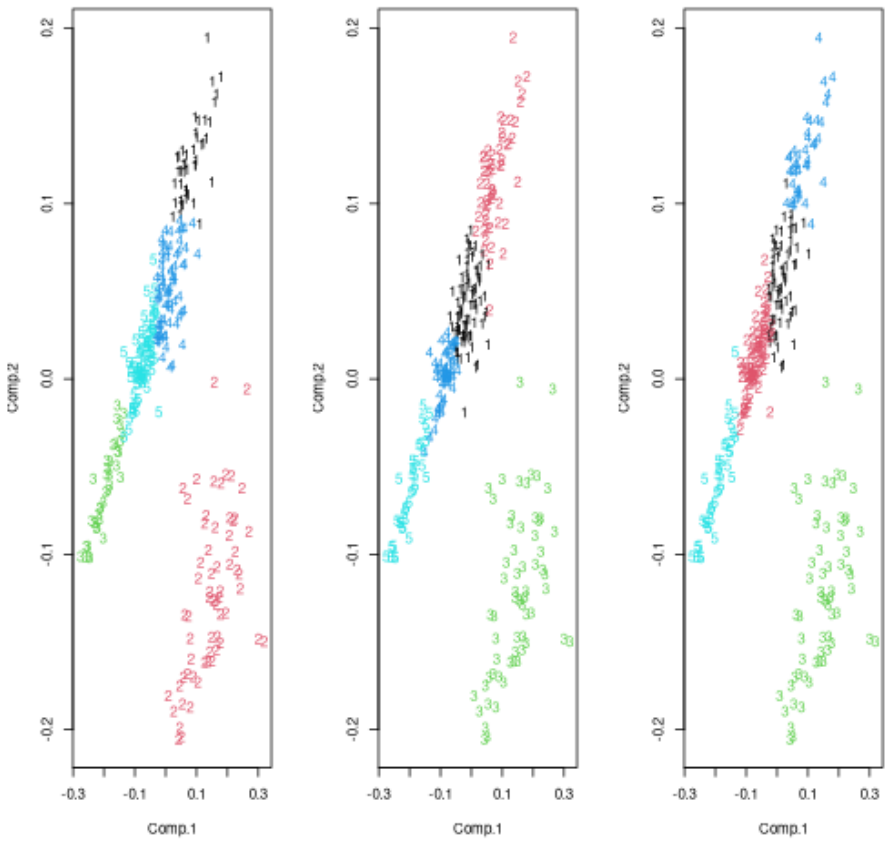


Figure 2.1: PCA-k-means, PAM/Euclidean, PAM/Manhattan

```
## [1] 0.6607979

## K-means vs manhattan PAM: similar interestingly 0.8863854
adjustedRandIndex(bundestagk5$cluster, bundestagp5m$cluster)

## [1] 0.8863854
```

visualizing cluster with principal components and MDS

```
## Could also use PCA to visualise this.
prp05 <- princomp(p05[1:5])
par(mfrow = c(1,3))
plot(prp05$scores,col=bundestagk5$cluster,pch=fpc::clusym[bundestagk5$cluster])
plot(prp05$scores,col=bundestagp5$cluster,pch=fpc::clusym[bundestagp5$cluster])
plot(prp05$scores,col=bundestagp5m$cluster,pch=fpc::clusym[bundestagp5m$cluster])
```

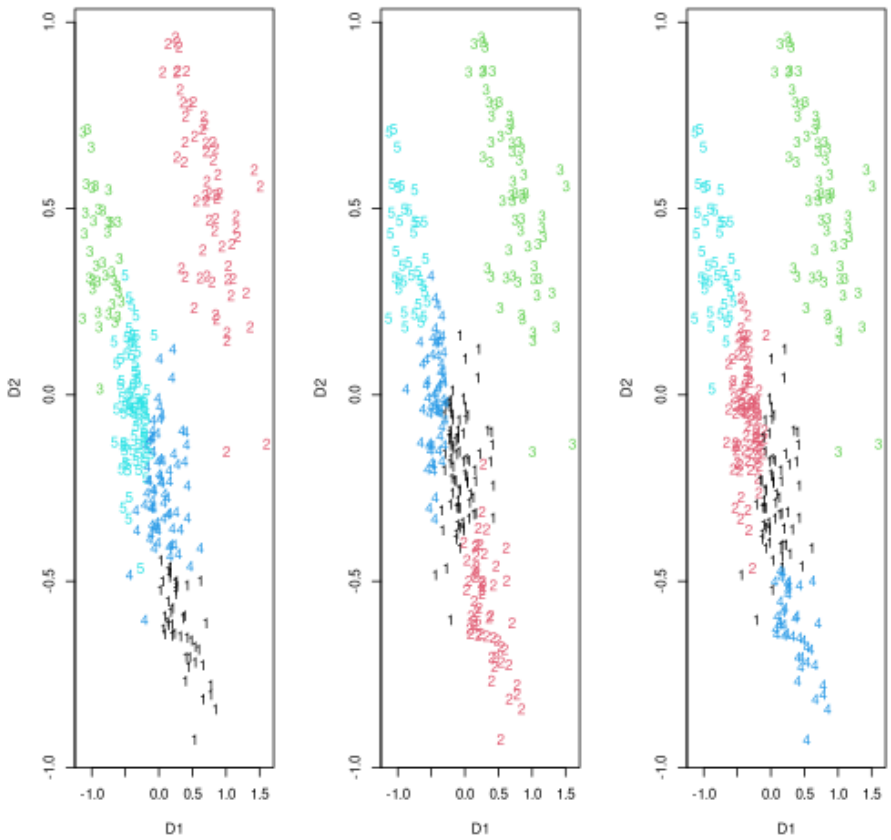


Figure 2.2: MDS manhattan-k-means, PAM/Euclidean, PAM/Manhattan

```
## we try to express manhattan distances in terms of euclidean distances using
## mds
mdsp05m <- mds(p05manhattan)

## visualization of cluster
par(mfrow = c(1,3))
plot(mdsp05m$conf, col=bundestagk5$cluster, pch=fpc::clusym[bundestagk5$cluster])
plot(mdsp05m$conf, col=bundestagp5$cluster, pch=fpc::clusym[bundestagp5$cluster])
plot(mdsp05m$conf, col=bundestagp5m$cluster, pch=fpc::clusym[bundestagp5m$cluster])
```

solutions are similar in a sense

2.7.1.1 CLARANS (Ng and Han (2002))

An algorithm for applying PAM on bigger data, CLARANS means Clustering Large Applications based on RANdomized Search.
It's actually another algorithm for approximation of optimum of PAM objective

function T for large data.
Repeat q times (and then use best found solution):

1. Draw k medoids at random.
2. Repeat r times:
   (a) Draw a non-medoid at random.
   (b) Tentatively replace one of the medoids by it.
   (c) If this improves T , keep the replacement.

Each iteration is very fast, but need large q, r to find a good solution. Quality of solution (and time providing it) depends on how large q, r are.

2.7.2 Average Silhouette Width (ASW)

The ASW is a quality measure for a clustering; computed for given dissimilarity data d with given clustering $(c(1), \ldots, c(n))$; can be used to define the best number of cluster K for dissimilarities or to compare different clustering solutions (eg different methods).
It's based on measuring how well each object $\mathbf{x}_i$ fits into its own cluster:

Definition 2.7.2. for each object we can compute its silhouette. Considering $\mathbf{x}_i$, $i \in \mathbb{N}_n$ it's defined as

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

where

$$a(i) = \frac{1}{n_{c(i)} - 1} \sum_{c(i)=c(j), i \neq j} d(\mathbf{x}_i, \mathbf{x}_j)$$

$$b(i) = \min_{c(i) \neq k} \frac{1}{n_k} \sum_{c(j)=k} d(\mathbf{x}_i, \mathbf{x}_j)$$

so

- $a(i)$ is an average of distances between the object and all the other objects of the same cluster; a small $a(i)$ is good for the clustering solution
- $b(i)$ is the average of distance of the object with the ones from the cluster which have smallest mean distance; a large $b(i)$ is good for clustering solution at hand
- $b(i) - a(i)$ is a unnormalized indicator of clustering goodness for the unit; it is compared to a normalizing denominator in order to make the silhouette have a maximum value of 1.
- It may be the case that the silhouette is negative (due to numerator) However Note: Points with negative silhouette width aren't necessarily. "misclassified". Putting them in neighbouring cluster will change silhouette values of other points and may lead to worse ASW.

Finally as convention $s(i) = 0$ for $\{\mathbf{x}_i\} \in \mathscr{C}$ one point cluster

Definition 2.7.3 (Average silhouette width)**.** It's defined as

$$ASW(\mathscr{C}_k) = \frac{1}{n}\sum_{i=1}^{n} s(i)$$

Important remark 30. In comparing clustering with different number of cluster, the optimal k is the one obtained by maximizing average silhouette width:

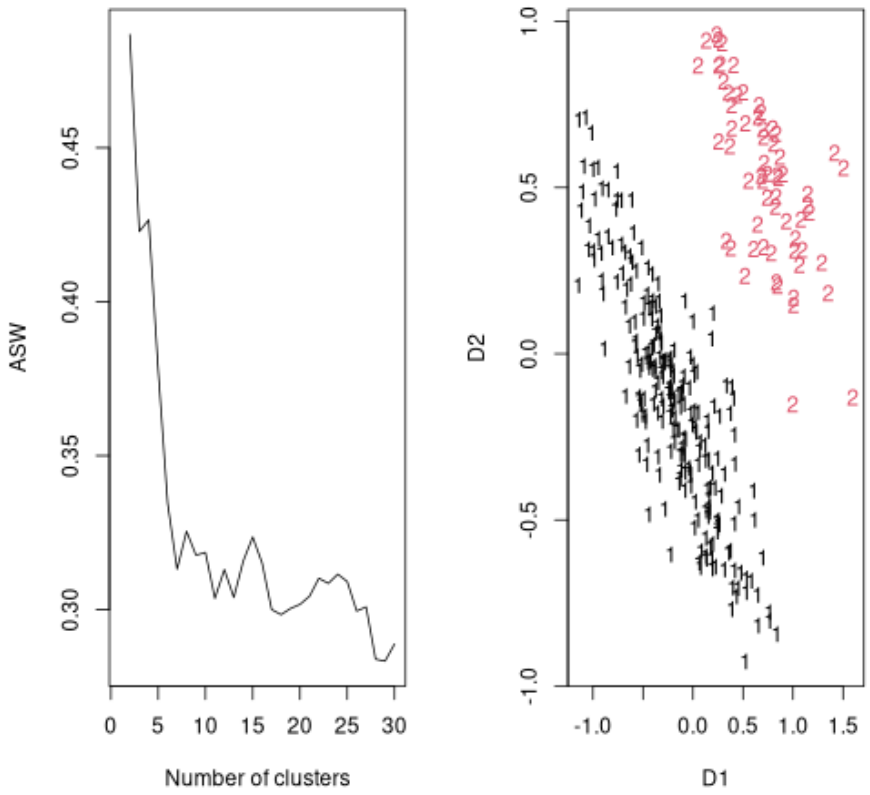
$$K_{ASW} = \underset{k=2,\dots,K_{max}}{\arg\max} ASW(\mathscr{C}_k)$$

Some disadvantage of ASW

- particularly compared to gap statistics is that it cannot compute for $K = 1$ (we don't have cluster other than one so it's impossible to compute $b(i)$).
- In some applications there's tendency of ASW to favour $K = 2$; local optima for $K > 2$ may also be worth exploring (the idea is to use both ASW and cluster plotting to choose the most convincing solution)

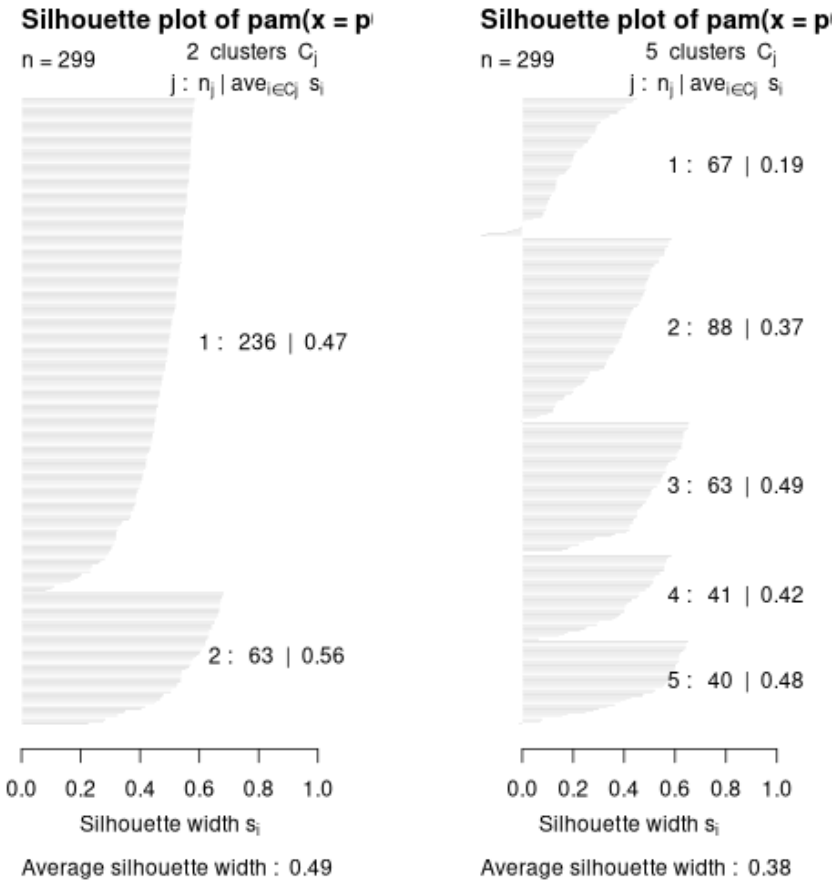
Example 2.7.2.

```
## to compare k we have to do it manually. Let's check the bundestag with
## manhattan distance for k between 2 and 30
pasw <- NA           # ASW for k
pclusk <- list()     # clustering objects
psil <- list()       # silhouettes objects

for (k in 2:30){
  # PAM clustering
  pclusk[[k]] <- pam(p05manhattan, k)
  # Computation of silhouettes
  psil[[k]] <- silhouette(pclusk[[k]], dist = p05manhattan)
  # ASW needs to be extracted from summary of a silhouette object
  pasw[k] <- summary(psil[[k]])$avg.width
}

# Plot the ASW-values against K:
par(mfrow=c(1,2))
plot(1:30, pasw, type = "l", xlab = "Number of clusters", ylab = "ASW")
pasw # Best value at K=2

##  [1]        NA 0.4867459 0.4228157 0.4265560 0.3779154 0.3345530 0.3131574
##  [8] 0.3254449 0.3177023 0.3185734 0.3036558 0.3130183 0.3039691 0.3157744
## [15] 0.3235948 0.3151002 0.3000481 0.2983881 0.3003727 0.3017265 0.3043202
## [22] 0.3101771 0.3085357 0.3115082 0.3091192 0.2995540 0.3008825 0.2839130
## [29] 0.2832789 0.2888274

# plotting the best solution on MDS-plot:
plot(mdsp05m$conf,
     col = pclusk[[2]]$cluster,
     pch = fpc::clusym[pclusk[[2]]$cluster])
```



Plotting of individual silhouette per cluster can be done plotting the silhouette objects

```
# Silhouette plots for K=2 and K=5 clusters
par(mfrow = c(1,2))
plot(psil[[2]]) # best solution according to ASW, w
plot(psil[[5]])
```

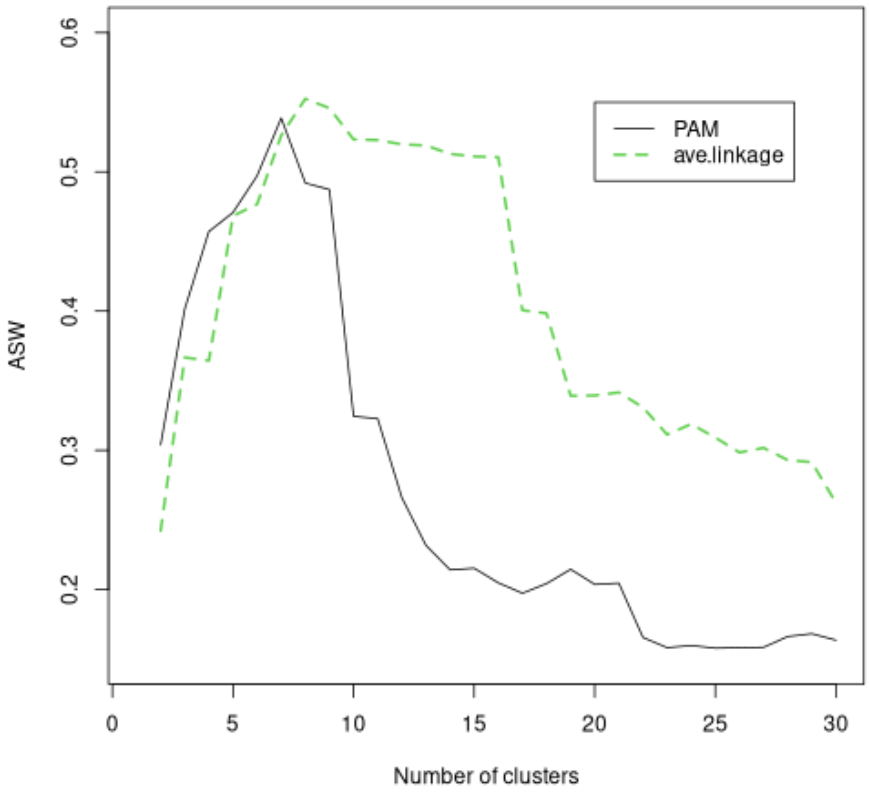


In the K=2 solution we can see we have 236 observation in the first cluster (average silhouette for its object is 0.47) and 63 in the second (average silhouette 0.56); objects are ordered with $s(i)$ in decreasing order. ASW is plotted as well 0.49 cluster.
If we split in $k = 5$ cluster the first one will have some points with negative $s(i)$.

Example 2.7.3. Another example on veronica

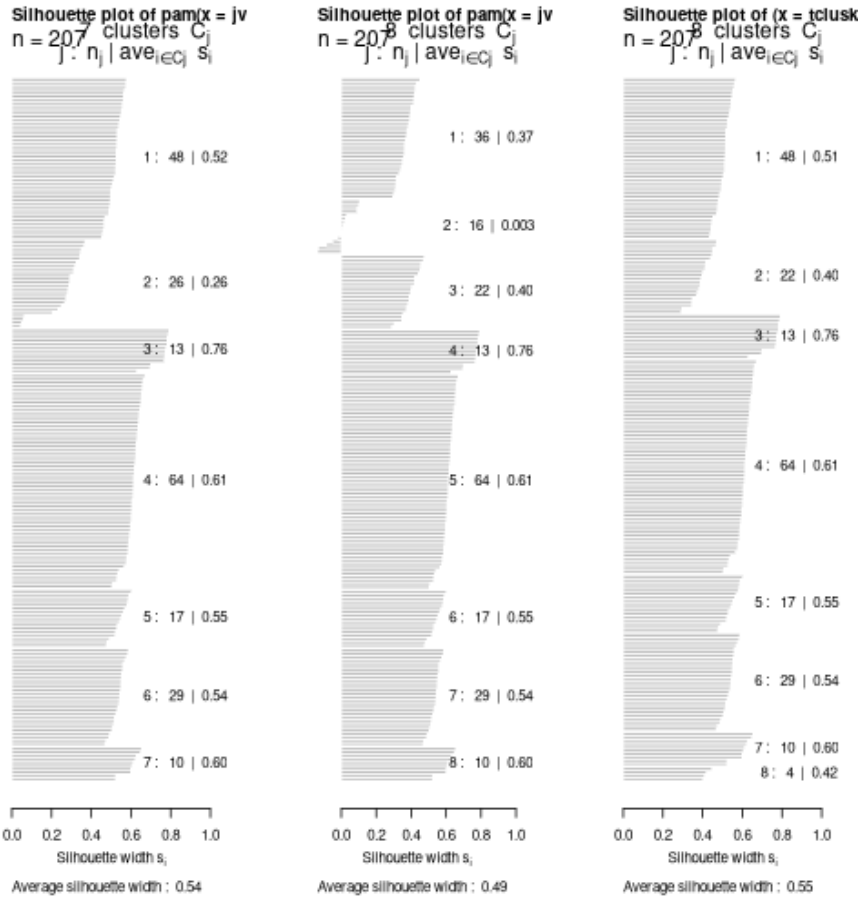
```
# PAM
pasw <- NA
pclusk <- list()
psil <- list()
for (k in 2:30){
  pclusk[[k]] <- pam(jveronica,k)
  psil[[k]] <- silhouette(pclusk[[k]])
  pasw[k] <- summary(psil[[k]])$avg.width
}
which.max(pasw) # Maximum at K=7, ASW=0.5386146

## [1] 7

pasw[7]
```

```
## [1] 0.5386146

# Average Linkage
plantclust <- hclust(jveronica,method="average")
tasw <- NA
tclusk <- list()
tsil <- list()
for (k in 2:30){
  tclusk[[k]] <- cutree(plantclust,k) #produce the clustering by cut
  tsil[[k]] <- silhouette(tclusk[[k]],dist=jveronica)
  tasw[k] <- summary(silhouette(tclusk[[k]],dist=jveronica))$avg.width
}
which.max(tasw) # Maximum is at K=8, ASW=0.5524769

## [1] 8

tasw[8]

## [1] 0.5524769

# comparing solutions
plot(1:30, pasw, type="l",xlab="Number of clusters",ylab="ASW",ylim=c(0.15,0.6))
points(1:30,tasw,type="l",col=3,lty=2,lwd=2)
legend(20,0.55,legend=c("PAM","ave.linkage"),col=c(1,3),lwd=c(1,2),lty=c(1,2))
```



Let's see pam and average linkage visually

```
par(mfrow=c(1,3))
plot(psil[[7]]) # pam 7
plot(psil[[8]]) # pam 8
plot(tsil[[8]]) # average link 8 (highest)
```



In the average linkage with 8 all the cluster has solid positive mean; in posizione centrale il cluster 2 non è un gran che .
volendo qui aggiungere i plot delle nuvole usando MDS: tenendo conto che se le cose non tornano potrebbe essere un problema di MDS che è cluster agnostico

Chapter 3

Hierarchical clustering

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

3.1 Introduction

Remark 24. The **aim** is to set up *hierarchy of clusters*, a sequence of groupings at different levels (fig 3.1)
Hierarchy is how we think about biological classification like family trees.

Definition 3.1.1 (Hierarchy)**.** Mathematically, it's a

- ordered sequence of partitions $\mathcal{C} = \cup_{j=1}^{m} \mathcal{C}_j$ (each $\mathcal{C}_j$ is a set of set with no overlap, think the cut of a dendrogram)
- partitions $\mathcal{C}_j$ have decreasing numerosity $K_1 = |\mathcal{C}_1| > \ldots > K_m = |\mathcal{C}_m|$ (so $\mathcal{C}_1$ first is the set of singletons subset of the dataset $\mathcal{D}$)
- Lower level sets are partitions of higher level sets: for $C_j \in \mathcal{C}_j$ and $C_k \in \mathcal{C}_k$ with $j < k$ either $C_j \cap C_k = C_j$ or $C_j \cap C_k = \emptyset$.

NB: sembra consistent con l'agglomerative clustering, a livello di progressione di indici

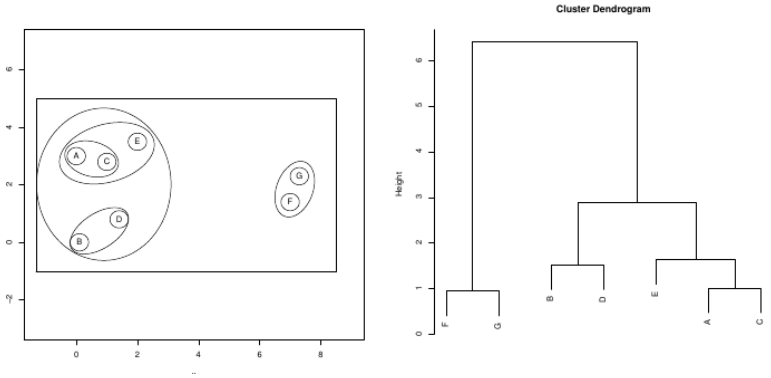


Figure 3.1: Hierarchical clustering

Important remark 31 (Approaches). Hierarchical clustering approaches:

- *Agglomerative* methods: we put together units up to obtaining clusters and cluster to obtain the whole. The most similar units/cluster are grouped togheter before;
- *Divisive* methods: we start from the whole and split in cluster and subcluster

Most divisive methods are computationally cumbersome (and not necessarily better); *we see agglomerative methods*

Important remark 32 (Distances between cluster). We need the concept of *dissimilarity between cluster* (which can be composed of 1 or more units) to choose the most similar cluster to be merged.
Dissimilarity *between clusters* is a function $D : (\mathcal{P}(D))^2 \to \mathbb{R}_0^+$, normally based on dissimilarity d *between units* of the cluster.

Important remark 33 (Agglomerative clustering algorithm). we have

1. **Inizialization**: setting $k = 1$ and the first clustering solution is composed by singletons

$$\mathcal{C}_1 = \left\{C_1^1, \ldots, C_n^1\right\} = \left\{\{\mathbf{x}_1\}, \ldots, \{\mathbf{x}_n\}\right\}$$

   so we have that the numerosity of cluster within solution $K_1 = n$ and level/height of dendrogram set to $H_0 = 0$

2. find cluster with minimum distance

$$D(C_i^k, C_j^k) = \min_{(C_l^k, C_m^k)} D(C_l^k, C_m^l)$$

3. merge clusters C_i^k, C_j^k with minimum distance:

$$\mathcal{C}_{k+1} = \mathcal{C}_k \cup \left\{C_i^k \cup C_j^k\right\} \setminus \left\{C_i^k, C_j^k\right\}$$

   then

   - set actual level/height to $H_k = D(C_i^k, C_j^k)$
   - update number of cluster composing solution to $K_{k+1} = |\mathcal{C}_{k+1}|$
   - renumber clusters in $\mathcal{C}_{k+1}$ as $C_1^{k+1}, \ldots, C_{K_{k+1}}^{k+1}$

4. if $K_{k+1} = 1, \mathcal{C}_{k+1} = \{\mathcal{D}\}$ stop; otherwise increase k by 1 and go to step 2

Remark 25. In the simple case where two cluster are composed of single units the distance between cluster coincides with distance between units

$$D(\{\mathbf{x}_i\}, \{\mathbf{x}_j\}) = d(\{\mathbf{x}_i\}, \{\mathbf{x}_j\})$$

Other than that, dissimilarity between clusters can be defined differently, producing different methods of hierarchical clustering.

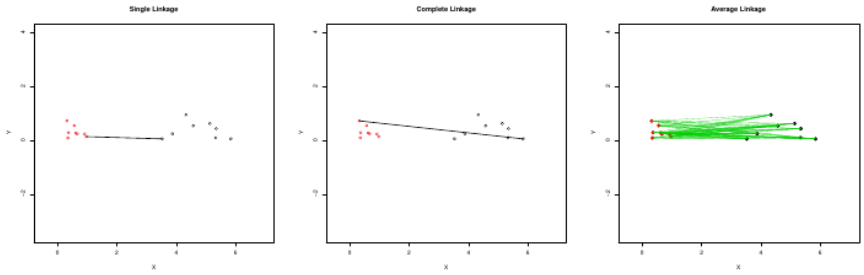


Figure 3.2: Linkage methods

3.2 Standard methods

Definition 3.2.1 (Single Linkage (or nearest neighbour) clustering)**.** It defines the dissimilarity D between two cluster as the minimum dissimilarity d between their components

$$D(B,C) = \min_{\mathbf{x}\in B,\mathbf{y}\in C} d(\mathbf{x},\mathbf{y})$$

Definition 3.2.2 (Complete linkage (or furthest neighbour) clustering)**.** The maximum dissimilarity between their component

$$D(B,C) = \max_{\mathbf{x}\in B,\mathbf{y}\in C} d(\mathbf{x},\mathbf{y})$$

Definition 3.2.3 (Average linkage (or UPGMA) clustering)**.** We check all the combinatorial differences between objects of the two clusters and take the mean

$$D(B,C) = \frac{1}{|B|\,|C|} \sum_{\mathbf{x}\in B,\mathbf{y}\in C} d(\mathbf{x},\mathbf{y})$$

Important remark 34. In case B and C are singletons all the three distances between cluster coincides with the distance between the two elements.
Otherwise different methods can give quite different results (fig 3.2)

Remark 26. Some results that suggest how complete linkage and single linkage may be useful. In some applications one may be interested in specific aspects:

- complete linkage is a clustering methods such that if one cut at a given height the dendrogram, one has the guarantee that all within cluster distances are smaller than their height

Proposition 3.2.1 (Within-cluster homogeneity guarantee)**.** *With* complete linkage $d(x,y) \leq H_{k-1}$ *(where* H_{k-1} *is the height where the two items were put toghether) for all* $\mathbf{x},\mathbf{y}$ *in same cluster in* $\mathcal{C}_k$*.*

Proof. Proof by complete induction (show for the first level of hierarchi and then that if it holds for up to a level k, it will holds for $k+1$ too)

- at first step the height is the distance between the two most similar object $H_1 = d(\mathbf{x}_1,\mathbf{x}_2)$, we have only one cluster with two two different points for which it holds (for the remaining sigleton cluster it holds), OK

- now let's assume $d(\mathbf{x}, \mathbf{y}) \leq H_{k-1}$ for $\mathscr{C}_k$. We need to show it will holds to the next level as well.
  Let's join two cluster going to the next level $C_1^* = C_1 \cup C_2$; we have $H_k = D(C_1, C_2)$ is max d within C_1^* (by definition of complete linkage), max d in all other clusters was already smaller before because $H_{k-1} \leq H_k$ (monotonicity, see Section 4.4).

□

Remark 27. So complete linkage is useful if we want to guarantee that elements within cluster have distances below to a certain value.
With single linkage the opposite occurs

Proposition 3.2.2 (Between-cluster separation guarantee)**.** *With* single linkage $d(x, y) \geq H_{k-1}$ *for all* $\mathbf{x}, \mathbf{y}$ *in same cluster in* $\mathcal{C}_k$*.*

Proof. pairs of clusters with $\exists d(x, y) < H_{k-1}$ for $\mathbf{x}, \mathbf{y}$ in different clusters should have been merged before step k. □

Important remark 35 (Final remarks). Clustering is having object within cluster as homogeneous as possible while object belonging to different cluster as diverse as possible.
Complete and single linkage make a choice in one or other direction

- **complete linkage** enforces within-cluster homogeneity disregarding separation (Complete Linkage clusters are often not properly separated at all);

- **single linkage** enforces between-cluster separation disregarding homogeneity (Single Linkage clusters can be extremely heterogeneous).

Methods above can be extreme: **average linkage** is a compromise and in practice often preferable.

Important remark 36 (R usage). Clustering is made by the following steps:

- obtain a `stats::dist` object starting from a numeric matrix or dataframe. Some methods of interest are `euclidean`, `manhattan`, `binary`, `minkowski`

- `stats::hclust` will take a `stats::dist` and produce the clustering; the methods applied can be specified as

  - `method = "average"`
  - `method = "single"`
  - `method = "complete"`

- once obtained the dendogram we can cut at a given height (distance) or specifying the number of cluster (K, which btw just by looking at the dendogram and choosing can be questionable). The `cutree` function does it fixing K as number of cluster or H as height

3.3 Examples

Example 3.3.1 (Veronica)**.** To have a non standard dataset on which start we use binary data from veronica, using jaccard distance and average/subgke/complete linkages

```
veronica <- read.table("data/veronica.dat")

# here we use jaccard distance and average linkage
jveronica <- dist(veronica, method = "binary") # jaccard
plantclust_avg <- hclust(jveronica, method = "average") # average linkage
plantclusts_single <- hclust(jveronica, method="single") # single linkage
plantclustc_compl <- hclust(jveronica, method="complete")# complete linkage

## Plot the dendrogram: 8 looks a good number
par(mfrow = c(2, 2))
plot(plantclust_avg, main = 'Average linkage')
plot(plantclusts_single, main = 'Single linkage') ## 8 looks still OK
plot(plantclustc_compl, main = 'Complete linkage') ## Could use 8 or 9.

## obtain a vector clusters by cutting, eg with average linkage
## 8 seems a good number of cluster looking at avg linkage dendrogram
## Estimating the number of cluster from looking at the dendrogram is somewhat
## questionable though.
plantclust8 <- cutree(plantclust_avg, k = 8)
head(plantclust8)

## [1] 1 1 1 2 3 4

## Visualisation using MDS in last plot
mdsveronica <- smacof::mds(jveronica)
plot(mdsveronica$conf,
     col = plantclust8,
     pch = fpc::clusym[plantclust8]) ## This looks good.
```

They look quite similar, but this is a coincidence: this gives us confidence this to be a good clustering.
MDS show that cluster 7 and 8 seems strange: MDS looses information by trying to squeeze distances in a 2d space (we don't say that 8 are bad cluster due to displaying of MDS).
Then the prof decided to use the Jaccard distance also for genes (**clustering of the variables**).
heatmap is major application of hierarchichal clustering:

- one produce hierarchical clustering of observation
- hierarichical clustering of variables
- then uses clustering to *sort* observation based on pattern of variables on the heatmap based on the two clustering.
- however the order in not unique and the order does not make sure that two neighbor units are similar

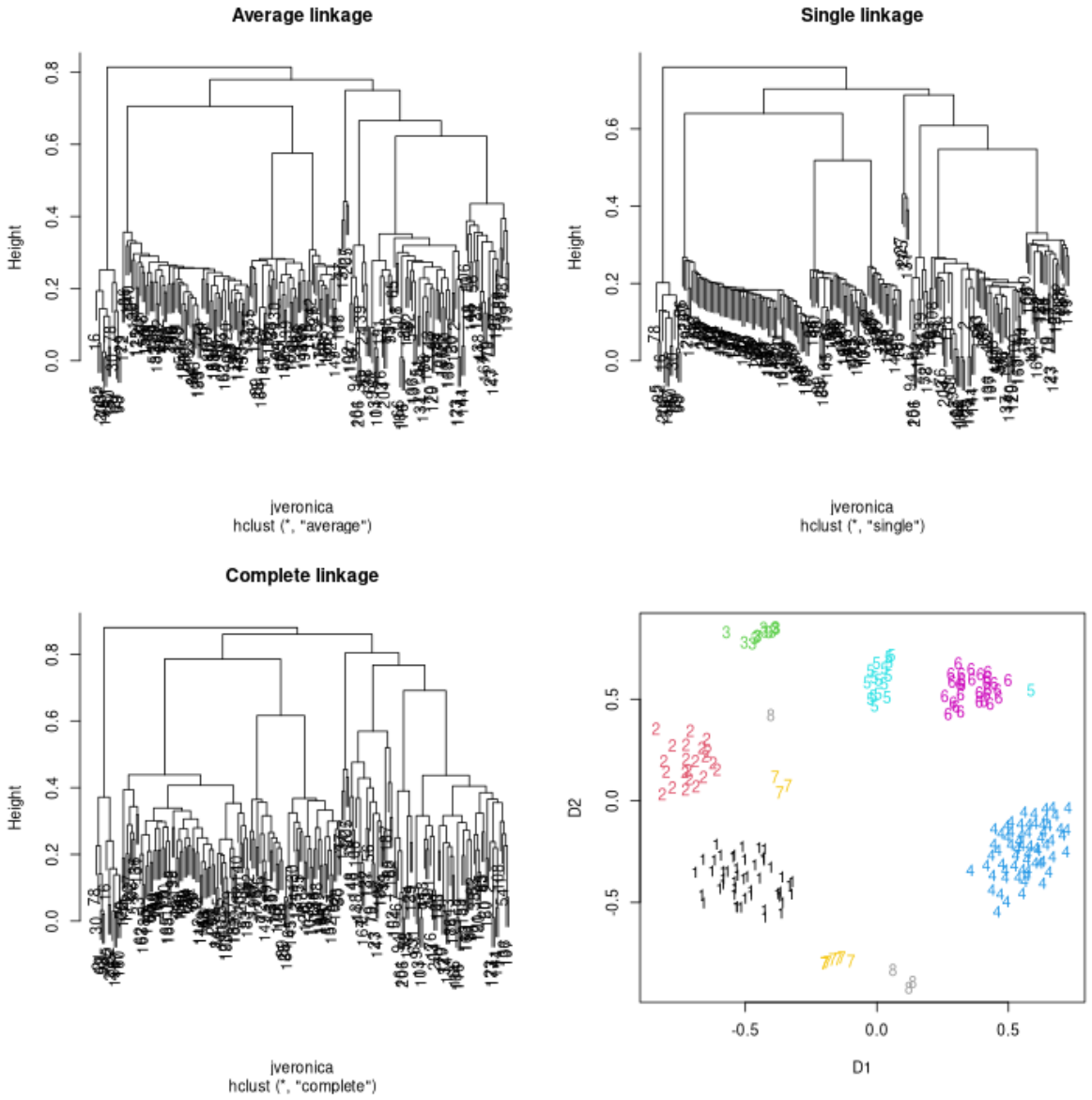


Figure 3.3: Veronica clusterings

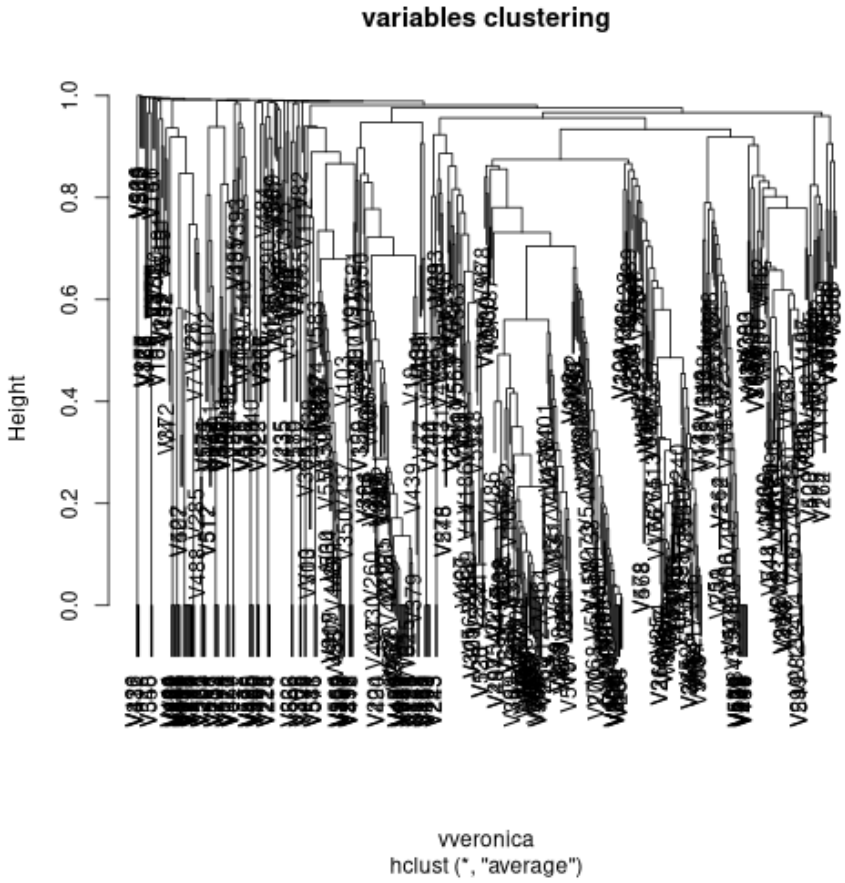


Figure 3.4: veronica heatmap

```
# clustering of variables
veronicam <- as.matrix(veronica)
vveronica <- dist(t(veronicam), method="binary") # distance of variables (note t)
varclust <- hclust(vveronica, method="average")  # clustering of variables

## par(mfrow=c(1,2))
plot(varclust, main = "variables clustering") # messy but can be used to order genes
```

```
## heatmap
heatmap(veronicam,
        Rowv = as.dendrogram(plantclust_avg),
        Colv = as.dendrogram(varclust),
        col = grey(seq(1, 0, -0.01)))
```

Example 3.3.2 (Geyser data)**.** Another example using geyser data (from MASS), with

TODO: spiegazione verso minuto 28 del 3 ottobre 2024; spiegazione a 1:03 del 10 ottobre anche

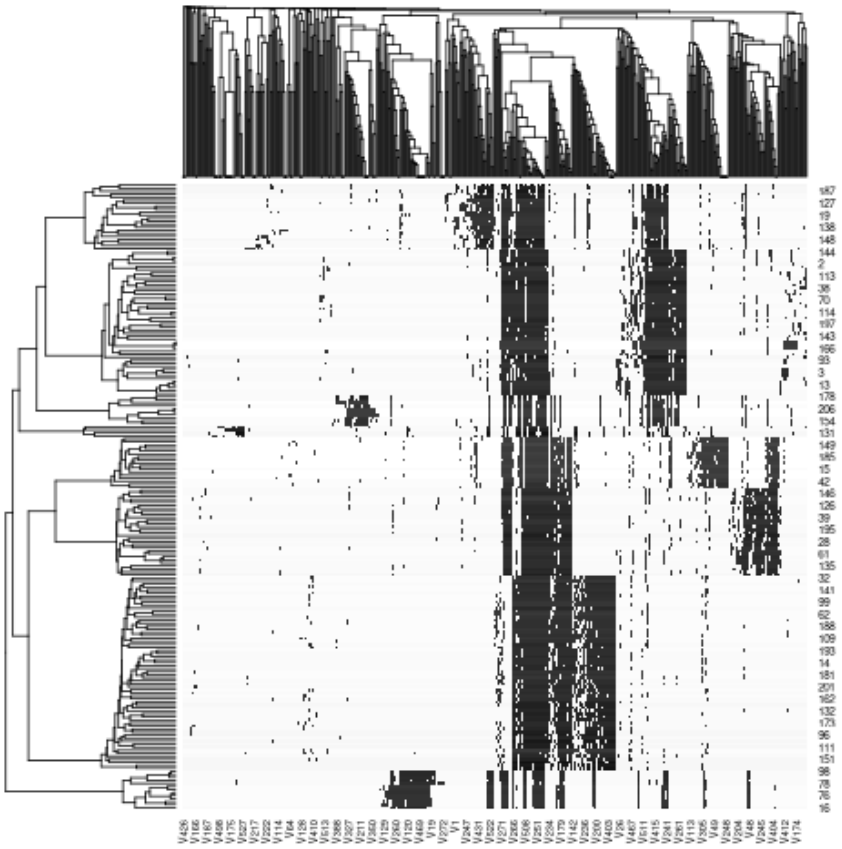


Figure 3.5: veronica heatmap

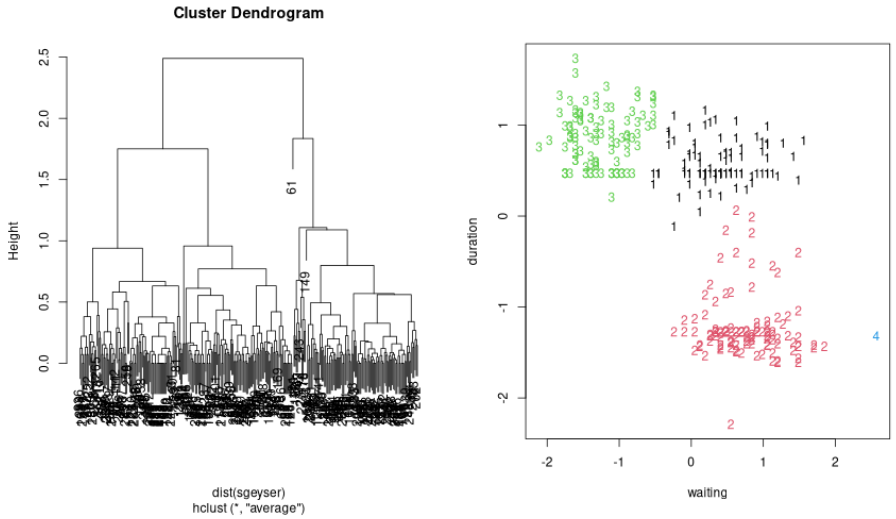


Figure 3.6: Geyser (average linkage)

- **average linkage** we obtain a fine solution, although isolating point 61 is somewhat odd.

```
geyser <- MASS::geyser
sgeyser <- scale(geyser)

# Average Linkage (based on euclidean distange of scaled data
geyclust <- hclust(dist(sgeyser), method="average")
par(mfrow = c(1,2))
plot(geyclust)    # 4 looks like a good K, will isolate one outlier
geyclust4 <- cutree(geyclust, k=4)
plot(sgeyser, col=geyclust4, pch = fpc::clusym[geyclust4])
```

- **single linkage** we have need at least K = 7 to not only have bulk and outliers. Single Linkage often "chains" big heterogeneous subsets and isolates small/one-point clusters.

```
# Single Linkage
geyclusts <- hclust(dist(sgeyser),method="single")
par(mfrow = c(1,2))
plot(geyclusts) # here 7 looks like a good K
geyclusts7 <- cutree(geyclusts, 7)
plot(sgeyser,col=geyclusts7,pch = fpc::clusym[geyclusts7])
```

- **complete linkage** tries to have all maximum within-cluster dissimilarities small. Probably $K = 4$ looks better on dendrogram; merge no. 1 and 4 next.

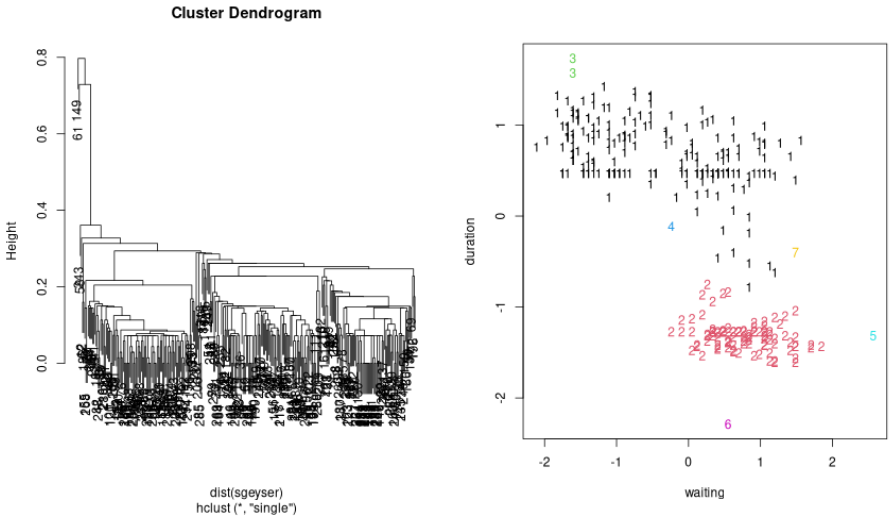


Figure 3.7: Geyser (single linkage)

```
# Complete Linkage
geyclustc <- hclust(dist(sgeyser),method="complete")
par(mfrow = c(1,2))
plot(geyclustc)      # 5 looks like a good K
geyclustc5 <- cutree(geyclustc,5)
plot(sgeyser,col=geyclustc5,pch=fpc::clusym[geyclustc5])
```

3.4 Additional remarks

3.4.1 Monotonicity

Remark 28. All standard methods defined here are monotonic, that is the heights/distances always get higher

$$0 = H_0 \leq H_1 \leq \ldots \leq H_{K-1}$$

There are versions of Agglomerative Hierarchical Clustering (AAHC) that aren't monotonic.

Proposition 3.4.1. *Complete Linkage is monotonic because whenever two clusters C_1, C_2 are merged in step k, i.e. $C_1^* = C_1 \cup C_2$, $H_k = D_{complete}(C_1, C_2)$ (this is thecomplete link dissimilarity between the two clusters), then the complete linkage dissimilarity between the new cluster and any cluster we already have will be $\geq H_k$ (its the max among maximum distances between object of C_1 and C_j and object of C_2 and C_j):*

$$D_{complete}(C_1^*, C_j) = \max\left[D_{complete}(C_1, C_j), D_{complete}(C_2, C_j)\right] \geq H_k$$

for all remaining clusters C_j, also for $i \neq j \in \{1, 2\}$: $D_{complete}(C_i, C_j) \geq H_k$ (otherwise we have merged differently), so that merging height in the next step H_{k+1} cannot be smaller than H_k.

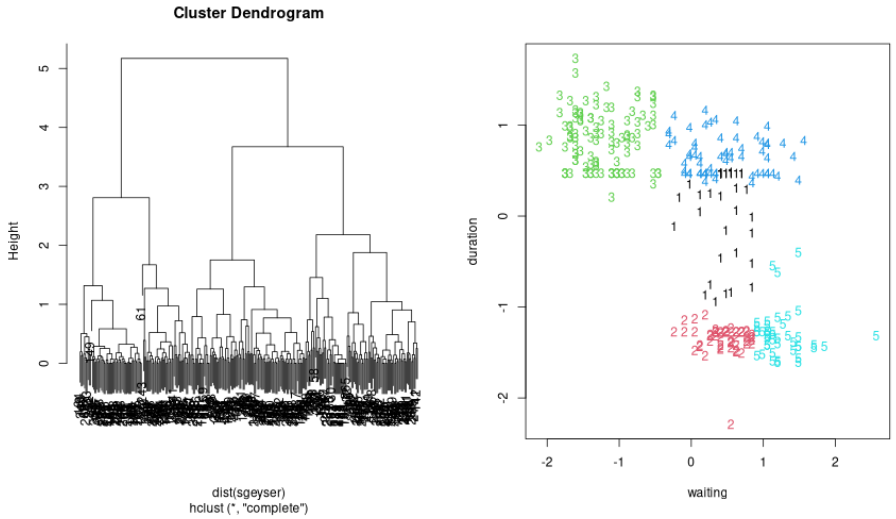


Figure 3.8: Geyser (complete linkage)

Proposition 3.4.2. *Single Linkage is monotonic because whenever two clusters C_1, C_2 are merged in step k, then*

$$D_{single}(C_1^*, C_j) = \min\left[D_{single}(C_1, C_j), D_{single}(C_2, C_j)\right] \geq D_{single}(C_1, C_2) = H_k$$

Once more, no smaller D_{single}-value has been produced, so $H_{k+1} \geq H_k$.

3.4.2 Large data

On large data sets we can't draw dendrogram, it's difficult to handle all dissimilarities (matrix memory I think).
One idea would be hierarchically cluster "microclusters", for example produced by faster k-means with large $k << n$, then either cluster the k means, or define distance between distributions within micro-clusters.

3.5 Ward's method

NB: Prof doesn't like this method

It's another hierarchical agglomerative method and in a sense Hierarchical version of K-means. The idea is to merge clusters optimising the objective $S(\mathcal{C}, m_1, \ldots, m_K)$ of k-means:

$$D(B, C) = D_k(B, C) = S(\mathcal{C}^*, \mathbf{m}_1^*, \ldots, \mathbf{m}_{K_{k+1}}^*)$$

Above dissimilarity between two cluster B and C depend on the dataset objctive function of k means applied to the clustering that results from merging B and C ($\mathbf{m}_1^*, \ldots, \mathbf{m}_{K_{k+1}}^*$ are the means of clusters when we put B and C together).
It optimises S on each level among clusterings produced by merging.
Disadvantage:

- Achieved S in most cases *worse* than K-means.

Advantages:

- Hierarchy;
- Deterministic (can also be used to initialise K-means);
- Has done well in a number of benchmark experiments

Finally one can write the objective function of k-means using dissimilarities alone Needs $n \times p$ Euclidean points and variables, but

$$S(\mathcal{C}, \mathbf{m}_1, \ldots, \mathbf{m}_K) = \sum_{k=1}^{K} \frac{1}{2\,|C_k|} \sum_{\mathbf{x}_i, \mathbf{x}_j \in C_k} d^2_{L_2}(\mathbf{x}_i, \mathbf{x}_j)$$

Can be generalised to other dissimilarities.

Example 3.5.1. Applied to bundestag

```
p05 <- read.table("data/bundestag.dat", header = TRUE)

# kmeans with 5 clusters
set.seed(12345)
kmbundestag5 <- kmeans(p05[1:5], 5, nstart = 100)

# Ward's method
wbundestag <- hclust(dist(p05[1:5]), method = "ward.D2")

# plot(wbundestag, labels=FALSE) # Dendrogram not shown
# Run with labels=FALSE because otherwise constituency names
# will dominate the plot.
# Same as wbundestag <- agnes(p05,method="ward")
# With K=5:

## check consistency between kmeans and cutted ward at 5 clusters
wbundestag5 <- cutree(wbundestag, 5)
table(kmbundestag5$cluster, wbundestag5) # Fairly different.

##    wbundestag5
##      1  2  3  4  5
##   1  8  0 36  0  0
##   2 72  0  0  0  0
##   3 27  0  0 55  0
##   4  0  0  0 16 22
##   5  0 63  0  0  0

mclust::adjustedRandIndex(kmbundestag5$cluster, wbundestag5)

## [1] 0.6362054

# [1] 0.6362054

## let's look at the value of objective function produced by this cluster
```

```
## using cluster.stats

## obj function of k-means
kmb <- fpc::cluster.stats(dist(p05[1:5]), kmbundestag5$cluster)
kmb$within.cluster.ss # 1.319956

## [1] 1.319956

# This is the same as kmbundestag5$tot.withinss

## obj function of cutted ward dendogram
wmb <- fpc::cluster.stats(dist(p05[1:5]), wbundestag5)
wmb$within.cluster.ss # 1.534854, Quite a bit worse.

## [1] 1.534854
```

Chapter 4

Mixture models (model-based cluster analysis)

This is the typical way statisticians intend clustering: one have a theoretical model for its data and cluster correspond to different parameters for the model.

4.1 Basic concepts

The most popular model assumption for clustring is mixture model we have a dataset $\mathscr{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, $\mathbf{x}_i \in \mathcal{X}$ assumed distributed iid with density

$$f_\eta(\mathbf{x}) = \sum_{k=1}^{K} \pi_k f_{\theta_k}(\mathbf{x}) \tag{4.1}$$

we assume that

- there are $k \in \mathbb{N}_k$ different cluster and k different distribution
- all distribution $f_{\theta_k} \in f_\theta : \theta \in \Theta$ comes from some parametric family, having parameter from some parameter set (eg some from normal density with mean vector and variance covariance matrix, but can be other thing as well). Parameters are $\theta_1, \ldots, \theta_k$
- pi_k are the proportion parameter: $0 < \pi_k \leq 1$, $\sum_{k=1}^{K} \pi_k = 1$ (every cluster has a probability of occurrence)
- our unknown are thus $\eta = (\pi_1, \ldots, \pi_K, \theta_1, \ldots, \theta_K)$.

Remark 29. Idea: Clusters correspond to subpopulations distributed as f_{θ_k} with different parameter values θ_k

Important remark 37. There are different way to write this down that will be useful later. We could write this down as a two-step version/random experiment:

$$\begin{aligned} &i \in \mathbb{N}_n : Z_i \text{ i.i.d. } \sim Multinomial(1, \pi_1, \ldots, \pi_K) \\ &\mathbf{x}_i | Z_i = k \sim f_{\theta_{Z_i}} \end{aligned}$$

where first we draw a single random variable Z_i from iid from a multinomial distribution (which extract the group, number from 1 to k). In the second step, given we know the cluster label $Z_i = k$, the distribution of the object is $f_{\theta_{Z_i}}$. In such case the marrginal density of $\mathbf{x}$ will be as in 4.1.
The major use of this is Bayes' formula, for classification. If we have points at the end we want to say something about the group they belong and this can be done by assigning the point to the class k which maxizes the following probability

$$P(Z_i = k|\mathbf{x}_i) = \frac{\pi_k f_{\theta_k}(\mathbf{x}_i)}{\sum_{j=1}^{K} \pi_j f_{\theta_j}(\mathbf{x}_i)}$$

To apply this we need to estimate the underlying parameter of distributions and the π_k as well.

In general what we're going to do is to ML-estimate our parameters to maximize the likelihood which under independent observation is product of densities:

$$\hat{\eta} = (\hat{\pi}_1, \ldots, \hat{\pi}_K, \hat{\theta}_1, \ldots, \hat{\theta}_K) = \arg\max_{\eta} \prod_{i=1}^{n} \left(\sum_{k=1}^{K} \pi_k f_{\theta_k}(\mathbf{x}_i) \right)$$

After we've done this we plug our estimate in the Bayes formula to obtain estimated probability for labels. p_{ik} is the estimated probability that $\mathbf{x}_i$ comes from cluster k:

$$p_{ik} = \hat{P}(z_i = k|\mathbf{x}_i) = \frac{\hat{\pi}_k f_{\hat{\theta}_k}(\mathbf{x}_i)}{\sum_{j=1}^{K} \hat{\pi}_j f_{\hat{\theta}_j}(\mathbf{x}_i)}$$

These can be used for classification, assigning each observation to its most likely cluster

$$\hat{Z}_i = \arg\max_{k \in N_k} p_{ik}$$

Important remark 38 (Number of clusters K). before we looked at K fixed and find a way to estimate parameters, however one could look the problem to find the number of K cluster (this is not known).
One could use the average silhouette width, but this doesn't use the modelling information, isn't related to statistical modelling.
This can be done by computing the BIC.

Definition 4.1.1 (Bayesian Information Criterion)**.**

$$BIC(n, K, \eta) = 2 \log \hat{L}_{n,K,\eta} - v(K, \eta) \log(n)$$

where

- likelihood of data is as usual function of number of units, number of cluster and parameters vector. The hat means that this is the likelihood with ML estimated parameter plugged in (we've said if we have k fixed we can estimate parameter from likelihood (we'll see how to do that))

$$\hat{L}_{n,K,\eta} = \prod_{i=1}^{n} \left(\sum_{k=1}^{K} \hat{\pi}_k f_{\hat{\theta}_k}(\mathbf{x}_i) \right)$$

- $v(K, \eta)$ is the number of free (1-d) parameters in model: can be seen as a penalty for model with too many parameters (as in K-means, likelihood will improve always for increasing K as long as $K < n$).
  If the $K$is larger the number of parameter will be larger.

Example 4.1.1 (number of free parameter)**.** If $\theta = (\mathbf{a}, \mathbf{\Sigma})$ for (unconstrained) p-variate Gaussian then overall parameter will be

$$\eta = (\pi_1, \dots, \pi_K, (\mathbf{a}_1, \mathbf{\Sigma}_1), \dots, (\mathbf{a}_K, \mathbf{\Sigma}_K))$$

then

$$v(K, \eta) = (K - 1) + K\left(p + \frac{p(p+1)}{2}\right)$$

because

- in the first part $K - 1$ freely settable π_k (-1 because the last has to be set to make sum $= 1$)
- in the second part, for each of the K groups we have a vector of p means and $\frac{p(p+1)}{2}$ unique elements on the variance-covariance matrix (eg diagonal for variances and upper triangular matrix part for covariances)

from here it's clear that the bigger K becomes, the biggere the number of parameter becomes

Important remark 39. Regarding BIC:

- BIC-optimal model: is the one who maximise $BIC(n, K, \eta)$ (in this formulation higher is better). Thus **BIC model selection** consist in fitting models with all K of interest or more generally all models of interest. Choose the one with largest BIC.
- BIC Can compare models with different K: given η we can find optimal K we maximize BIC over K.
  One particular application is to include $K = 1$ in the comparison to check if there are actually different cluster at all.
- we can compare model with different η: given K we maximize over η.
  This is interesting if we want to check eg if the covariance matrix are all equal or can change between clusters; in the first case we can fit a model with a lower number of parameters;
- Note: In the literature BIC is defined sometimes so that large is good (as here) and sometimes so that small is good (with changed signs).
  Particular attention for interpretation is needed because some software package implement one other the other.

Some theoretical ideas for BIC Originally (Schwarz (1978)), the formula for BIC has been derived in a Bayesian setup as approximation for the following likelihood of data given number of cluster K

$$p(\tilde{\mathbf{x}}|K) = \int h(\eta) \cdot l_{n,K}(\eta, \tilde{\mathbf{x}}) d\eta,$$

where

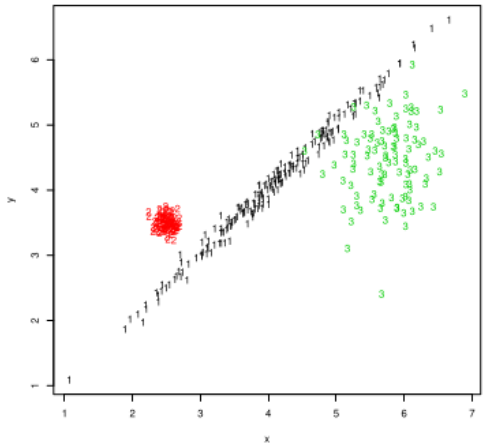


Figure 4.1: mixture1

- h is uniform prior for η
- if all K have the same prior probability $p(\tilde{\mathbf{x}}|K)$ is proportional to the posterior for K

Another motivation for BIC comes from **Keribin** (2000): (by simulation I think) they showed that if a "real" K exists, chosing K by maximizing BIC estimates the "real" K *consistently* (good for large n) in mixture model under some assumptions (which are fulfilled for a 1-d Gaussian mixture with equal variances bounded from below). This still seems to be best existing consistency result.
Problem with consistency of the estimator: If assumed mixture model does not hold precisely, for large n (for which it's consistent) estimated K will become larger and larger in order to give optimal mixture approximation.

Remark 30. The AIC is even worse than the BIC in this respect; if n isn't very small, will estimate at least as many mixture components as BIC, and BIC for large n already tends to estimate too many for the needs of cluster analysis. AIC is not recommended for clustering.

4.2 The Gaussian mixture model

This is the most popular mixture model; most distributions can be fitted arbitrarily well using Gaussian mixtures with large enough K, see Nguyen et al. (2020).

$$f_\eta(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \phi_{\mathbf{a}_k, \Sigma_k}(\mathbf{x})$$

where ϕ is the Gaussian p-variate density and the parameter of the models are the following $\eta = (\pi_1, \ldots, \pi_K, (\mathbf{a}_1, \mathbf{\Sigma}_1), \ldots, (\mathbf{a}_K, \mathbf{\Sigma}_K))$ with different vectors of mean and variance-covariance matrix per cluster.

Some remarks:

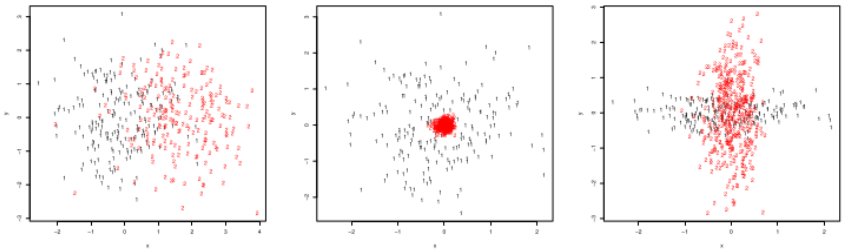


Figure 4.2: mixture2

- We are here assuming that all the cluster are normally distributed (which is different from assuming all the data is normally distributed)
- Gaussian populations are elliptical with *flexible* shapes; within-cluster distances may not be small. Dataset in fig 4.1 has ben generated using a gaussian mixed model: in cluster 1 there are the greatest differences within same cluster. In these situations of different variability k-means and PAM usually perform worst
- Gaussian mixtures may be unimodal and not heterogeneous. Sometimes that's desired, sometimes not. Eg in figure 4.2

Remark 31. There are methods which does density based clustering: `pdfCluster/dbscan` do density-based clustering, `mergenormals` in `fpc` merges Gaussians for clustering.

4.3 Covariance matrix models

$$f_\eta(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \phi_{\mathbf{a}_k, \Sigma_k}(\mathbf{x})$$

I guess in the context of gaussian mixture, one could (depending on application, eg plotting of data if possible) make some assumptions on the variance covariance matrix

- have fully flexibility $\mathbf{\Sigma}_k$, each different. *Problem* is that sometimes there are too many parameters to be estimated (too instable estimates).
- otherwise one can either assume just a common varcov matrix $\Sigma_k = \Sigma$ or even more restrictive $\forall k \in \mathbb{N}_K : \Sigma_k = b\mathbf{I}_p$ (same varcov matrix between groups, with zero correlation among variables).

  Within-cluster distances are often smaller with constrained models.

This is what originates several different possible models, actually, to be fittable.

`mclust` is the most popular package to fit gaussian mixture. One can show that a symmetric matrix (as covariance) can be decomposed using spectral decomposition in the following components

$$\mathbf{\Sigma}_k = \lambda_k \mathbf{D}_k \mathbf{A}_k \mathbf{D}_k^T \quad k = 1, \dots, K$$

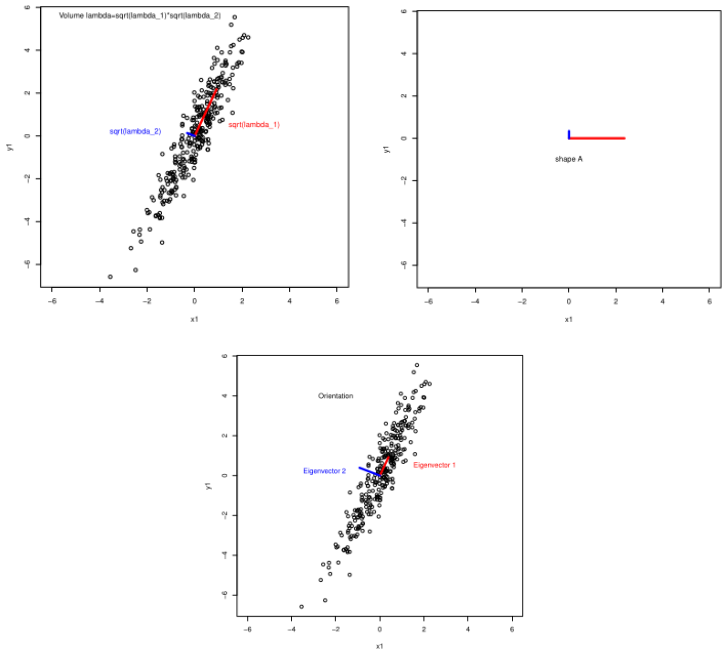


Figure 4.3: Connection to pca: volume, shape and orientation

where standard spectral decomposition is taken but worked a little more in the sense that the eigenvalue matrix is decomposed in one parameter (hypervolume) that tells how big the product of the eigenvector is and another that has only relative eigenvalue

TODO: 1:19 del 17 ottobre 2024

- $(\lambda_{k1}, \dots, \lambda_{kp})$ are the eigenvalues
- $\lambda_k = \prod_{i=1}^{p} (\lambda_{ki})^{1/p}$ hypervolume,
- $\mathbf{D}_k$ is the matrix of eigenvectors,
- the matrix $\mathbf{A}_k = \frac{1}{\lambda_{1k}} \mathbf{Diag}(\lambda_{k1}, \dots, \lambda_{kp})$ "shape" is constrainted to have $\det A_k = 1$ (product of diagonal elements is assumed to be 1)

Remark 32. Note connection to PCA (within clusters) 4.3. These things have different interpretation.

- *hypervolume* is how much is spread the point cloud, how big is it (it correspond to the 2d volume of high density region);
- *shape* is relative length of eigenvalue with respect to each other (compact or stretched cloud)
- *eigenvectors* in the D matrix formalize the *orientation* (first eigen is the direction of greater variability);

Remark 33. One or more of these can be assumed equal between clusters. Shape can be assumed to be the unit matrix (like k-means)

`mclust` package uses a coding system. Models are defined by three letter codes which in order indicates

1. volume
2. shape
3. orientation

In each one can specify

- "V" variable (things can be different between clusters)
- "E" equal (things are forced to be equal between clusters)
- "I" unit matrix.

From `?mclustModelNames` for univariate mixture

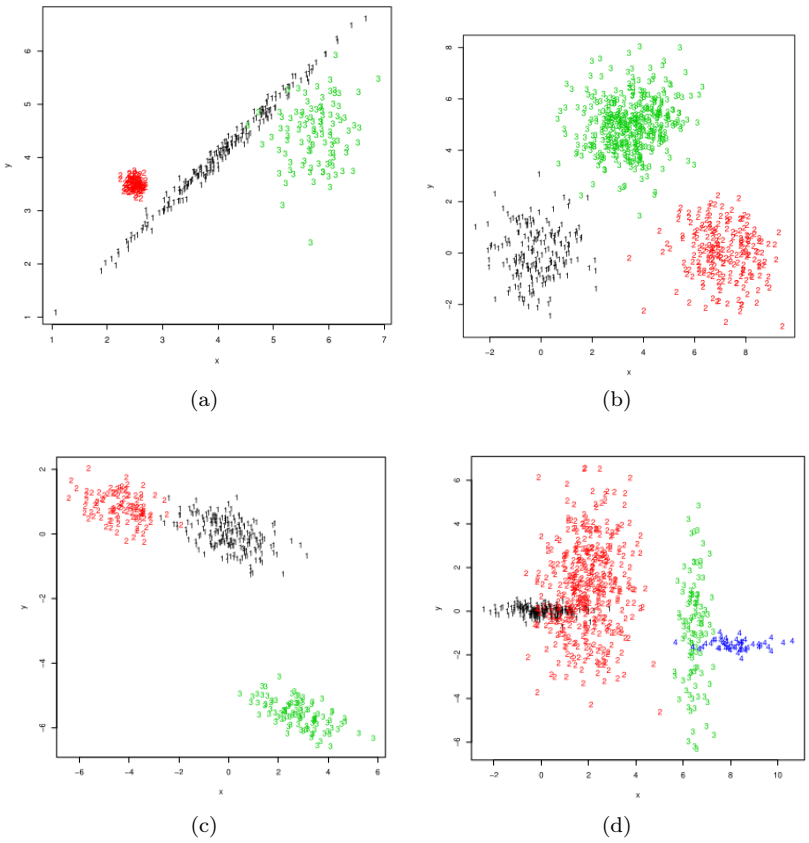
- "E" apply equal variance to all k groups (one-dimensional/one variable, there's no covariance between variables of the same group)
- "V": let variable variance (one-dimensional)

For multivariate mixture (in case we have more than one variable al the three component volume/shape/orientation are specified using letters). The shape of fitted cluster is described as

```
"EII": spherical, equal volume    (spherical due to unit matrix shape and orientation?)
"VII": spherical, unequal volume
"EEI": diagonal, equal volume and shape
"VEI": diagonal, varying volume, equal shape
"EVI": diagonal, equal volume, varying shape
"VVI": diagonal, varying volume and shape
"EEE": ellipsoidal, equal volume, shape, and orientation
"VEE": ellipsoidal, equal shape and orientation
"EEV": ellipsoidal, equal volume and equal shape
"VEV": ellipsoidal, equal shape
"EVV": ellipsoidal, equal volume
"VVV": ellipsoidal, varying volume, shape, and orientation
```

Some notable/classical model are in fig

- "VVV" (high left): fully flexible model. Two are spherical with different sizes and the third black fully different, so we need a VVV model to fit this
- "EII" (high right): equal volume, all spherical (k-means);
- "EEE" (low left): equal (but flexible) volume, shape and orientation (covariance matrix are equal but can be non spherical, oriented in strange way in space). Related/Assumptions of linear discriminant analysis ;
- "VVI" (low right): diagonal ("local independence", within cluster there's no correlation between units?); components can be interpreted in terms of marginals. All eigenvector are parallel to axes



(a) (b)

(c) (d)

Figure 4.4: several models

Constraints used for estimation:

- if we assume **Equal volume**: clusters are similar in terms of within-cluster dissimilarity/variation (in some application useful)
- in case of **Non-unit shape** (covariance matrix is not diagonal with all entries the same): the clustering will be invariant against variable scaling .
- if we assume **Non-diagonal orientation** (covariance matrix is not diagonal): clustering will be rotation invariant.

What `mclust` does by default is optimising over all models using BIC: we'll have some solutions that are not rotation and scale invariant (which may be an issue, **think if we should scale**).
Models are not required to be true, but determine implications for clustering.

4.4 Computation: the EM-algorithm

Here we assume K fixed: the way to deal with different K is to fit different model using different K and then choose the one optimizing BIC. But here we assume K fixed and we want to maximise the log-likelihood

$$\log L_{n,K}(\eta) = \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} \pi_k \phi_{\mathbf{a}_k, \Sigma_k}(\mathbf{x}_i) \right)$$

under conditions $\pi_k > 0 \forall k$, $\sum_{k=1}^{K} \pi_k = 1$.

Remark 34. There is no straightforward analytic solution; we need an algorithm to find local optima.

4.4.1 EM in general

Important remark 40 (EM-algorithm (Expectation-Maximization)). The algorithm used is the EM-algorithm (Dempster et al. (1977)), which is general principle to find ML-estimator if *information is incomplete*. In mixture model missing information is cluster memberships $Z_1, \ldots, Z_n$. In EM:

- we observed data object $\mathbf{x}_i$
- for each observation we have an unobserved random variable Z_i (in our case the cluster group) which somehow influences likelihood
- we call mixture the observed $\mathbf{y}_i = (Z_i, \mathbf{x}_i)$
- the unobserved complete data is composed by $\tilde{\mathbf{y}} = \{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$
- $\tilde{\mathbf{x}} = T(\tilde{\mathbf{y}})$: T can be thought as the function extracting observed data (i suppose $\tilde{\mathbf{x}} = \{\mathbf{x}_1, \ldots, \mathbf{y}_n\}$)

We want to maximise the loglikelihood

$$L_{n,K}(\eta) = \sum_{i=1}^{n} log f_\eta(\mathbf{x}_i)$$

Let's define

$$L_{n,c}(\eta) = \sum_{i=1}^{n} log f_{\eta,c}(\mathbf{y}_i)$$

where the c subscript stands for "complete": this is the likelihood we would like to maximize (complete data) but we don't have some information.
We assume some initial value for the parameters η_0 initialisation. The general formulation for the EM algorithm is the following:

- **E-step**: compute expected complete likelihood (given parameter estimates we already have η_{t-1}, η_0 eventually)

  $$q(\eta|\eta_{t-1}) = E_{\eta_{t-1}}(L_{n,c}(\eta)|T = \tilde{\mathbf{x}})$$

  This is the expected value, assuming the parameter η_{t-1} we had from previous iteration is correct of the complete loglikelihood, under the condition that the observed data is actually the data that we have. For the moment this is misterious but we will see how this act in the gaussian case.
  Because we don't know all the data we cannot compute the likelihood of the complete data; best we can do is to take the expected value of complete likelihood assuming that the x are the one we observed

- **M-step**: maximise conditional likelihood

  $$\eta_t = \arg\max_{\eta} q(\eta|\eta_{t-1})$$

  So we choose parameter which maximise the previously obtained expected value

Theorem in dempster et al. (1977): Both steps never decrease $L_{n,K}(\eta)$ (so if it never decrease it can only increase and will find a local optima somewhere)

4.4.2 EM in the Gaussian mixture model

In the Gaussian mixed model our parameter "vector" (something strange having a matrixes inside vector, but just for notation) is

$$\eta = (\pi_1, \ldots, \pi_k, \mathbf{a}_1, \ldots \mathbf{a}_K, \mathbf{\Sigma}_1, \ldots, \mathbf{\Sigma}_K)$$

Complete loglikelihood with Z_i known would be (take the two step multinomial of remark 37 where first we decide the cluster according to the multinomial and then have the density from normal):

NB: ultima equazione da tenere le prime due cerco di intendere quel che dice a lezione 9:20 del 25 ott

$$\begin{aligned} L_{n,K,c}(\eta) &= \log \prod_{i=1}^{n} \sum_{k=1}^{K} 1(Z_i = k) \cdot \pi_k \cdot \phi_{\mathbf{a}_k, \mathbf{\Sigma}_k}(\mathbf{x}_i)) \\ &= \sum_{i=1}^{n} \log \sum_{k=1}^{K} 1(Z_i = k) \cdot \pi_k \cdot \phi_{\mathbf{a}_k, \mathbf{\Sigma}_k}(\mathbf{x}_i)) \\ &= \sum_{i=1}^{n} \sum_{k=1}^{K} 1(Z_i = k)(\log \pi_k + \log \phi_{\mathbf{a}_k, \mathbf{\Sigma}_k}(\mathbf{x}_i)) \end{aligned}$$

Particularly in the last step we could take log inside sum because the inner sum it's actually not a sum (using the selector function to consider only the probability/density from the cluster of the observation, and setting the other to 0).
In the **E-step** its expected value is obtained (here $\mathbf{x}$ are fixed)

$$E_{\eta_{t-1}}(L_{n,K,c}(\eta)|T=\tilde{\mathbf{x}}) = E_{\eta_{t-1}}\left(\sum_{i=1}^{n}\sum_{k=1}^{K} 1(Z_i=k)(\log\pi_k + \log\phi_{\mathbf{a}_k,\mathbf{\Sigma}_k}(\mathbf{x}_i))|T=\tilde{\mathbf{x}}\right)$$
$$= \sum_{i=1}^{n}\sum_{k=1}^{K} P(Z_i=k|\eta_{t-1},\mathbf{x}_i)(\log\pi_k + \log\phi_{\mathbf{a}_k,\mathbf{\Sigma}_k}(\mathbf{x}_i))$$

where we put the expected value into the sum and the expected value of an indicator variable is the probability (everything else does not have random components and is treated constant-like). This probability $P(Z_i=k|\eta_{t-1},\mathbf{x}_i) = p_{ik}^{(t-1)}$, assuming we know η_{t-1} and we know $\mathbf{x}_i$ can be plugged in/substituted applying Bayes theorem, as seen

$$p_{ik}^{(t-1)} = P(Z_i=k|\eta_{t-1},\mathbf{x}_i) = \frac{\pi_k^{t-1}\phi_{\mathbf{a}_k^{(t-1)},\mathbf{\Sigma}_k^{(t-1)}}(\mathbf{x}_i)}{\sum_{h=1}^{K}\pi_h^{t-1}\phi_{\mathbf{a}_h^{(t-1)},\mathbf{\Sigma}_h^{(t-1)}}(\mathbf{x}_i)}$$

In the **M-step** we search to optimize parameter for the next iteration (so we choose $\pi_k, \mathbf{a}_k, \mathbf{\Sigma}_k$ for the various cluster

$$\eta^{(t)} = \arg\max_{\eta}\sum_{i=1}^{n}\sum_{k=1}^{K} p_{ik}^{(t-1)}(\log\pi_k + \log\phi_{\mathbf{a}_k,\mathbf{\Sigma}_k}(\mathbf{x}_i))$$

Considering the most free/unconstrained (on covariance matrix)x VVV model, the point is that one can separately maximise the two part of the sum (because they're actually independent).
Regarding the probability that unit belongs to k-th cluster it can be shown that maximization (multivariate derivatives wrt to various π_k i guess)

$$\sum_{i=1}^{n}\sum_{k=1}^{K} p_{ik}^{(t-1)}\log\pi_k \implies \pi_k^{(t)} = \frac{1}{n}\sum_{i=1}^{n} p_{ik}^{(t-1)}$$

Regarding densities part, maximizing the term

$$\sum_{i=1}^{n}\sum_{k=1}^{K} p_{ik}^{(t-1)}\log\phi_{\mathbf{a}_k,\mathbf{\Sigma}_k}(\mathbf{x}_i)$$

yields weighted Gaussian ML estimators for $(\mathbf{a}_k, \mathbf{\Sigma}_k)$:

$$\mathbf{a}_k^{(t)} = \frac{1}{\sum_{i=1}^{n} p_{ik}^{(t-1)}}\sum_{i=1}^{n} p_{ik}^{(t-1)}\mathbf{x}_i$$
$$\mathbf{\Sigma}_k^{(t)} = \frac{1}{\sum_{i=1}^{n} p_{ik}^{(t-1)}}\sum_{i=1}^{n} p_{ik}^{(t-1)}(\mathbf{x}_i - \mathbf{a}_k^{(t)})(\mathbf{x}_i - \mathbf{a}_k^{(t)})^T$$

One can iterate these step until "convergence", normally defined by "too small increase" in $L_{n,K}$ (eg $<$ 10e-06).
Note similarity to Lloyd's algorithm for K-means:

- E-step probabilitically assigns points to clusters given parameters,
- M-step computes weighted parameter estimators given assignments.

Initialisation EM-algorithm depends on initialisation; better initialisation yields better local optimum at the end.
EM-algorithm can actually be started:

- from *initial parameters*
- an initial set of p_{ik}^0
- a partition of the data, in which case p_{ik}^0 is either 0 or 1.

We could either:

- start EM q times from random partitions and choose solution that maximises $l_{n,K}$
- try to find a single "intelligent" starting partition; regarding various alternatives have been proposed in literature

`mclust` package by default initialize EM using hierarchical clustering (function `hc`) which is done by

1. starting with every data point as cluster.
2. merging the two "closest" clusters (actually those which lead to maximum $l_{n,K}$)
3. going back to step 2 until there are K clusters (or a single one, to compute a whole hierarchy).

TODO: chiarissima sta nota e anche la sua spiegazione

Can be computed from pairwise dissimilarity matrix, which requires much memory and time for large n. For large n do this on subset and extract parameters.

Ci sono altri problemi (tipically with gaussian mixture but with some other as well) which is a gegenerating likelihood! We want to maximize loglikelihood choosing η for $\log \hat{L}_{n,K,\eta}$ become maximum. Problem is that this likelihood can become infinity so we can't really properly maximize this.
The reason why loglikelihood can become infinity, is as example as follows; in 1-d (single variable observations from k gaussian distributions), for $K \geq 2$ loglikelihood is:

$$\log \hat{L}_{n,K,\eta} = \sum_{i=1}^{n} \log \left[\sum_{k=1}^{K} \hat{\pi}_k \frac{1}{\sqrt{2\pi\hat{\sigma}_k^2}} \exp\left(-\frac{(x_i - \hat{\mu}_k)^2}{2\hat{\sigma}_k^2} \right) \right]$$

Now we want to choose $\hat{\pi}_k, \hat{\sigma}_k^2, \hat{\mu}_k$ such that $\log \hat{L}_{n,K,\eta}$ is maximum. Thing is that we could choose this in such a way that loglikelihood could become ∞. How to do this?

Let's choose as mean of the first cluster $\hat{\mu}_1 = x_1$ (first observation), and probability of first cluster $\hat{\pi}_1 = \frac{1}{n}$, say. Thus,

$$\hat{\pi}_1 \frac{1}{\sqrt{2\pi\hat{\sigma}_1^2}} \exp\left(-\frac{(x_1 - \hat{\mu}_1)^2}{2\hat{\sigma}_1^2} \right) = \frac{1}{n\sqrt{2\pi\hat{\sigma}_1^2}} \exp(0)$$

Now if $\hat{\sigma}_1 \to 0$, this converges to ∞. Considering the fact that we have other cluster than $k = 1$, say with $\hat{\sigma}_2$, $\hat{\mu}_2$ constant (other k as well), all other terms in the sum are bounded away from 0, the sum $\sum_{k=1}^{K}$ will converge to ∞, its log as well and the sum on units as well. This means loglikelihood is unbounded. Thus an optimizer will chose that in a way that loglikelihood is actually infinity (then the best clustering is always $\hat{\mu}_1 = x_1$, $\hat{\pi}_1 = \frac{1}{n}$, $\sigma_1^2 = 0$, so the first cluster fit exactly the first point while the rest is not so important). This is not very useful.
Loglikelihood is unbounded also in the multivariate case for $p > 1$ (if one eigenvalue of $\hat{\mathbf{\Sigma}}_k \to 0$).
Some cases where this *does not* happen are:

- For simple Gaussian ($K = 1$) this cannot happen; we cannot choose x_i equal to the mean $\hat{\mu}_1$ and then let $\hat{\sigma}_1^2$ goes to zero (the only cluster variance) because if one does this while $\sum_{k=1}^{1}$ for x_i converges to ∞ (as happen above), but it converges to zero for all other points ($\exp(-\infty)$ because $\hat{\sigma}_1^2 \to 0$ while the numerator will be something that is not zero); thus, after applying log, it can be shown that the loglikelihood overall will go to $-\infty$.

- It also doesn't, for $K \geq 2$, if all σ_k are assumed equal, because then $\hat{\sigma}_1 \to 0$ forces them all to 0, and again loglikelihood goes to $-\infty$ (and the maximization problem can be solved if we assume all the variances are equal, or in the p-dimensional cases all variances/covariances are equal).
  Thus there should not be problem with EII, EEI, EEE (p-dim analogue to all σ_k equal): basically the model with same volumes works (those not starting with V).

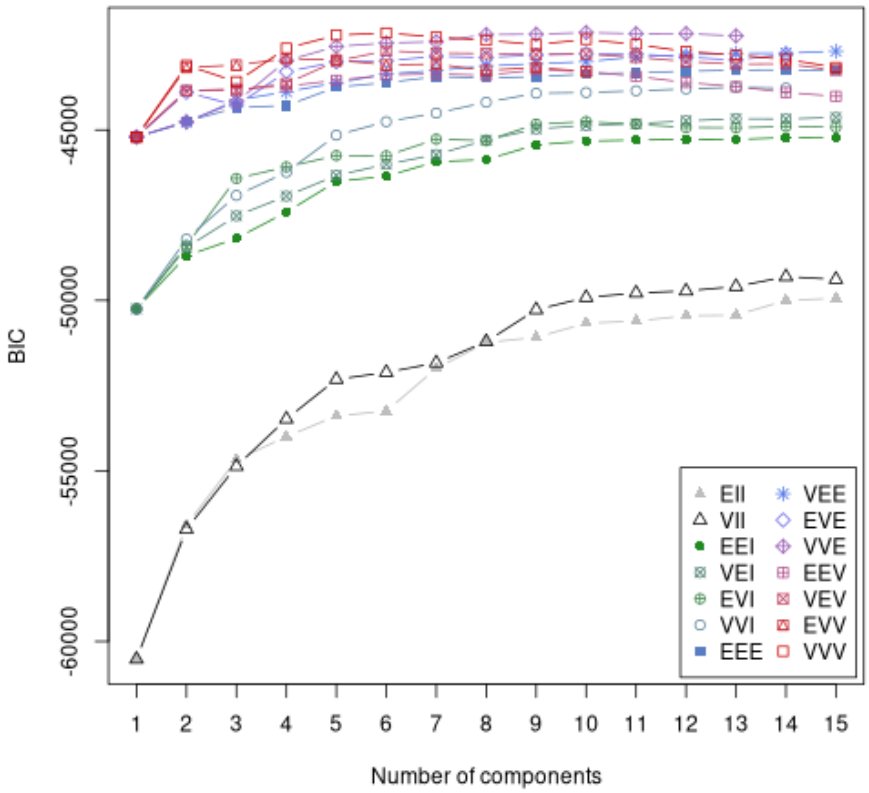
NB: ok non va a più infinito ma va a meno in una massimizzazione equivale a ignorare la soluzione che dato che è balzana si può fare, credo

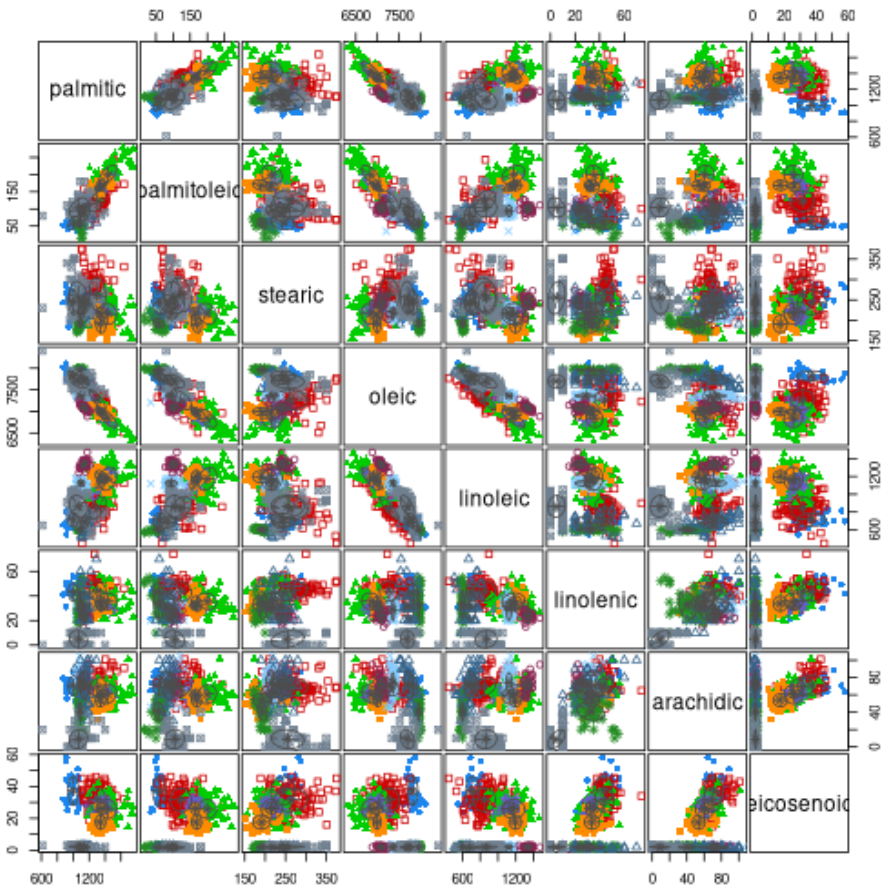
In these cases there are no problem.
Problem applies to VVV (with $K \geq 2$) and some other models (thos starting with V): in practice for VVV we try to find local optimum with all eigenvalues of all $\mathbf{\Sigma}_k$ bounded away from zero.
EM-algorithm only finds local optimum anyway; solution may often be fine but can degenerate (eg in some cases it can goes to infinity). For example in the plot vve solution for K=14/15 BIC plotting where not given because loglik were degenerate, se ho compreso bene.
If impossible (having free variances/covariances), `mclust` choose cov-matrix model that "works" .

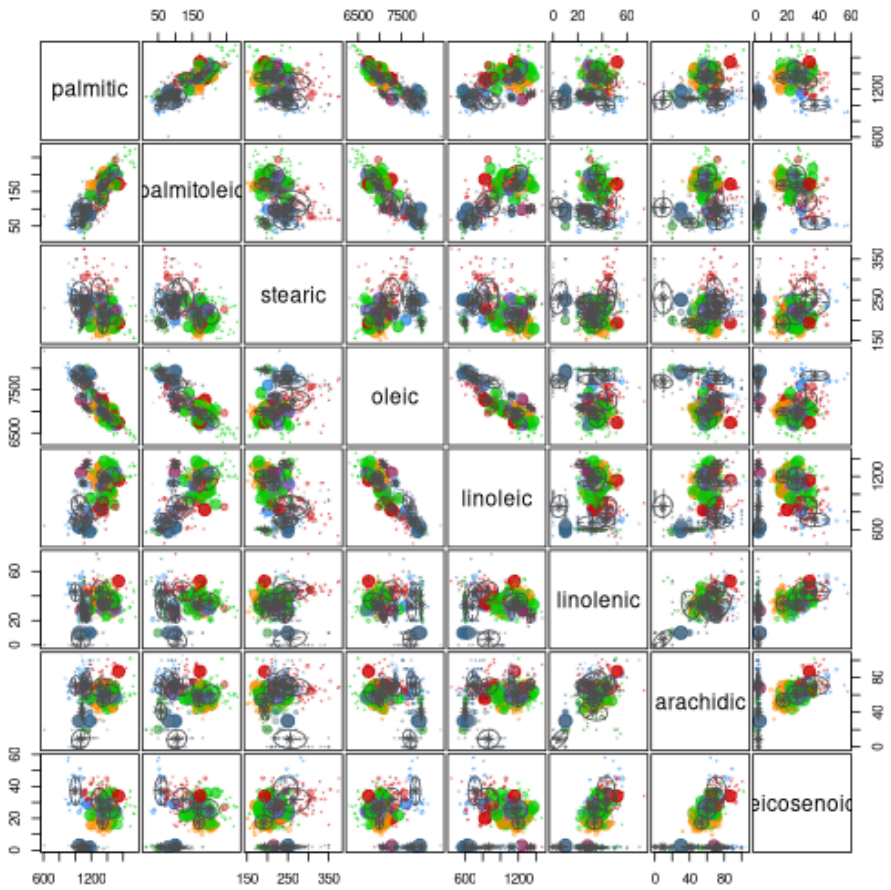
Important remark 41. There are consistency and asymptotic normality results for ML-estimators in the mixture case (Redner and Walker, 1984).
Results state that (under assumptions) consistent local optimum of the likelihood exists, bypassing the degenerating likelihood issue.
Mixture-ML doesn't suffer from inconsistency problem of K -means! (But not every local optimum is a good one.)

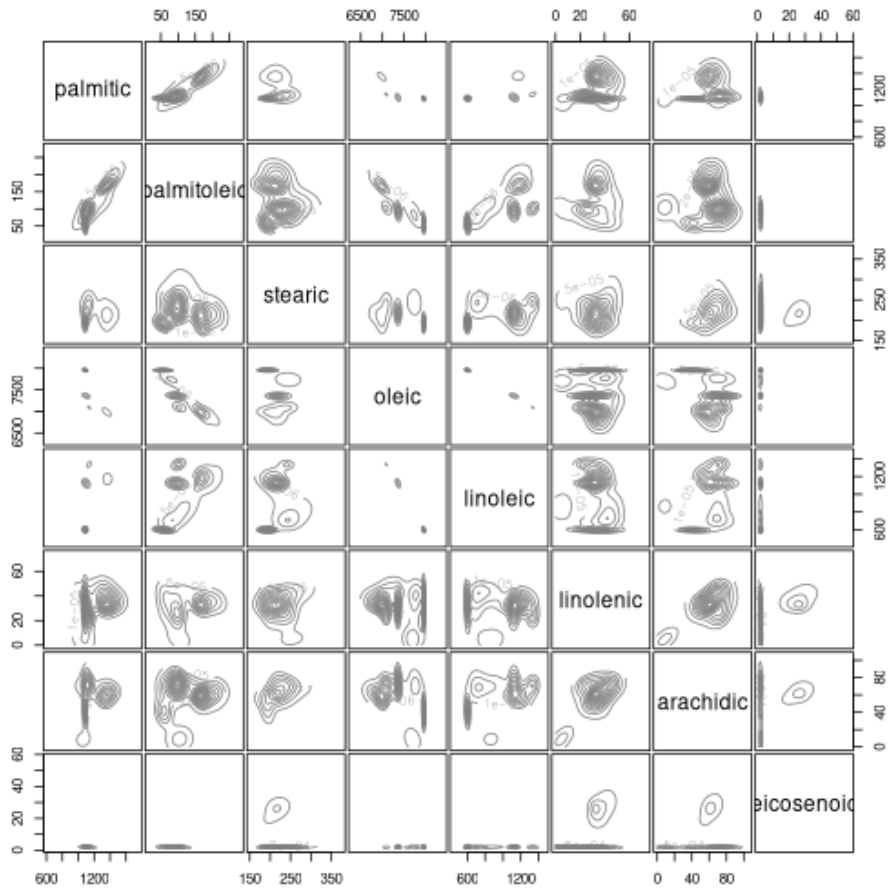
4.5 Example

```
library(pdfCluster)
library(mclust)
library(fpc)
data(oliveoil)
olive <- oliveoil[, 3:10]
molive <- Mclust(olive,G=1:15) # G is number of cluster to check, default is 1:9
summary(molive) # see results

## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VVE (ellipsoidal, equal orientation) model with 10 components:
##
##  log-likelihood   n  df       BIC       ICL
##       -20448.03 572 197 -42146.84 -42193.07
##
## Clustering table:
##   1   2   3   4   5   6   7   8   9  10
##  30  96 110  37  50  64  34  49  45  57

# above VVE model was the best (elippsoid with equal orientation and flexible
# volume) with 10 component (number of cluster). The loglikelihood is given, n
# is number of observartion, df is the  number of free parameters in the model
# BIC is given, ICL is a different criterion not treated. The frequency of each
# cluster is given


## summary of BIC gives the 3 best solutions (in this case lower is better..)
## and comparison with the best one
summary(molive$BIC)

## Best BIC values:
##             VVE,10        VVV,6       VVE,12
## BIC      -42146.84 -42158.49981 -42176.74491
## BIC diff      0.00    -11.65605    -29.90115

# check correspondance to regions
adjustedRandIndex(molive$classification, oliveoil$region) # [1] 0.5914548 Fairly good

## [1] 0.5914548

# Diagnostic plots
# ----------------
plot(molive)
```


BIC
-45000
-50000
-55000
-60000
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Number of components
EII
VII
EEI
VEI
EVI
VVI
EEE
VEE
EVE
VVE
EEV
VEV
EVV
VVV


palmitic
palmitoleic
stearic
oleic
linoleic
linolenic
arachidic
eicosenoic


palmitic
palmitoleic
stearic
oleic
linoleic
linolenic
arachidic
eicosenoic




```
# 1) in the first graph the number of components and BIC with different models
# fitted (we can se VVE has up to 13 components while we specified 15); this is
# due to the fact that there can be problems with flexible models estimation if
# one has too many components. Within one cluster one can have degenerate
# covariance matrix (in which case the model is not fitted).  Furthermore in the
# upper part there are no big differences between models; there could be other
# cluster that may be interesting (and that is open for interpretation)
# Gaussian mixture often produce solution that are similar in quality

# 2) the pairs plot (available if the variable are low enough) plots ellipsoid
# of the orientation of the first two dimension for the cluster solution; vi sono anc

# 3) in the third the uncertainty plot plots with size proportional 1 - p_{i,
#  Z_i} (where probability is probability to be belong to be assigned): the plot
# give bigger points for the objects where the probability is low (classification
# in that group is ambiguous)

## 4) high densities region projected on the two dimensional hyperplan ?
```

Uncertainty plot plots $1 - p_{i\hat{Z}_i}$ (with $p_{i\hat{Z}_i}$ the estimated probability thatob-

servation $\mathbf{x}_i$ was generated by the mixture component it was classified, so it's the maximum value of the π_{ik} - we assigned to the maximum one for k); if we're sure of classification of $\mathbf{x}_i$, then $p_{i\hat{Z}_i}$ should be high and $1 - p_{i\hat{Z}_i}$ small. The point dimension is proportional to this measure so smallest point are most sure points, while largest are one which we're less confident with.
If we have three cluster the smallest value $p_{i\hat{Z}_i}$ can take is $1/3$ (for K cluster $1/K$); for $K = 3$ the max value is $2/3$ and normally this bigger observation are to be found between cluster, on the "threshold".
Extracting information from the estimated object

```
# Specifying other models (here not done)
# If you want one or more specific covariance matrix models,
# you could specify modelNames, e.g.,
if (FALSE){
  moliveVVV <- Mclust(olive, G=1:10, modelNames="VVV")
  molivex <- Mclust(olive, G=1:10, modelNames=c("EEE","VVV"))
}



# Information available in output object, e.g.
# ---------------------------------------------
# Clustering vector
table(molive$classification) # $classification is the cluster group vector

##
##   1   2   3   4   5   6   7   8   9  10
##  30  96 110  37  50  64  34  49  45  57

## Estmated parameters are here (long): we find component proportions, means
## and covariance matrices (so with 10 cluster there will be 10 proportions, 10
## vector of means)
molive$parameters

## $pro
##  [1] 0.05189067 0.16863402 0.19068027 0.06496598 0.08851502 0.11251129
##  [7] 0.05881687 0.08528561 0.07747271 0.10122757
##
## $mean
##                  [,1]       [,2]       [,3]       [,4]       [,5]        [,6]
## palmitic    1001.52477 1304.61617 1433.90827 1356.05887 1342.55150 1098.001253
## palmitoleic   58.97520  123.05836  199.33528  164.32651  168.52916   94.519712
## stearic      244.46985  262.74334  216.17703  216.25869  191.17814  217.252030
## oleic       7825.94151 7271.05447 6843.81655 6972.87607 6993.78527 7361.332525
## linoleic     712.68068  846.31023 1173.08619 1143.30603 1196.89523 1123.938283
## linolenic     42.98128   43.85255   35.70144   31.94589   33.73972   28.793145
## arachidic     70.86104   69.28304   60.56436   61.10308   53.81116   73.842363
## eicosenoic    37.19240   31.49887   25.76015   26.37944   17.63412    1.959161
##                   [,7]        [,8]        [,9]       [,10]
## palmitic    1136.876071 1086.007654 1143.386356 1065.026402
```

```
## palmitoleic  101.001278   58.586753   84.434124  104.388141
## stearic      243.269159  193.052241  240.475802  255.201244
## oleic       7089.522575 7958.601127 7735.959969 7697.268543
## linoleic    1335.393427  601.348662  686.156178  864.211616
## linolenic     23.837425   32.738059   31.904206    4.820207
## arachidic     71.894178   40.868804   71.395622    8.918803
## eicosenoic     1.899771    1.983766    1.880432    2.036112
##
## $variance
## $variance$modelName
## [1] "VVE"
##
## $variance$d
## [1] 8
##
## $variance$G
## [1] 10
##
## $variance$sigma
## , , 1
##
##                palmitic  palmitoleic    stearic       oleic     linoleic
## palmitic     4369.87482   604.126726 -370.18679 -4941.17165   417.258559
## palmitoleic   604.12673   355.297859 -113.77951 -1009.84734   262.345338
## stearic      -370.18679  -113.779511  940.02058  -276.96960  -192.151218
## oleic       -4941.17165 -1009.847336 -276.96960 12948.43338 -6116.956568
## linoleic      417.25856   262.345338 -192.15122 -6116.95657  5607.820110
## linolenic     -40.34313   -25.554933   33.57098   -43.59423    12.040659
## arachidic      62.90767   -11.628973  -34.38529  -220.66116    49.012810
## eicosenoic    -11.08353    -2.352424    4.68502    10.31669    -1.789603
##               linolenic    arachidic  eicosenoic
## palmitic    -40.343128   62.9076700 -11.0835302
## palmitoleic -25.554933  -11.6289729  -2.3524243
## stearic      33.570978  -34.3852863   4.6850201
## oleic       -43.594232 -220.6611588  10.3166877
## linoleic     12.040659   49.0128100  -1.7896025
## linolenic    54.271169   42.6959197   0.6182790
## arachidic    42.695920  120.3625766  -0.4374788
## eicosenoic    0.618279   -0.4374788  80.9005584
##
## , , 2
##
##                 palmitic  palmitoleic     stearic       oleic     linoleic
## palmitic     10728.69950  1678.236715 -617.898380 -19645.9032   7536.92260
## palmitoleic   1678.23671   933.991036 -294.359048  -4100.9153   1646.22279
## stearic       -617.89838  -294.359048 1744.981471   -596.2035   -426.90797
## oleic       -19645.90318 -4100.915298 -596.203453  51858.3461 -25620.26916
## linoleic      7536.92260  1646.222788 -426.907972 -25620.2692  16263.78552
## linolenic      -30.17050   -35.029914   75.092130   -126.8126     40.62039
```

```
## arachidic      283.69657    57.673400  -31.051443   -805.7855    336.81070
## eicosenoic     -26.22706    -6.846377    9.253301     43.2054    -18.82047
##                linolenic    arachidic  eicosenoic
## palmitic     -30.1704966  283.6965727 -26.2270619
## palmitoleic  -35.0299140   57.6734002  -6.8463772
## stearic       75.0921298  -31.0514431   9.2533007
## oleic       -126.8126444 -805.7854930  43.2054023
## linoleic      40.6203925  336.8106952 -18.8204657
## linolenic     63.2007309   15.9275803   0.5338579
## arachidic     15.9275803  103.7729278  -0.7855519
## eicosenoic     0.5338579   -0.7855519  54.8722683
##
## , , 3
##
##                 palmitic palmitoleic    stearic        oleic      linoleic
## palmitic     12213.58338  1821.08113 -378.99046 -17622.80234   3676.076576
## palmitoleic   1821.08113   745.27051 -147.21413  -3618.98542   1148.931639
## stearic       -378.99046  -147.21413  664.70278   -207.90336    110.467715
## oleic       -17622.80234 -3618.98542 -207.90336  45401.65538 -22330.122815
## linoleic      3676.07658  1148.93164  110.46771 -22330.12282  16786.883493
## linolenic      -48.07423   -30.60606   20.36834    -95.20737     82.867100
## arachidic      269.21436    35.67819  -40.21859   -714.29133    266.589028
## eicosenoic     -28.81451    -6.03385    3.85824     39.96858     -8.046188
##                linolenic    arachidic  eicosenoic
## palmitic    -48.0742268  269.2143594 -28.8145065
## palmitoleic -30.6060571   35.6781937  -6.0338497
## stearic      20.3683448  -40.2185900   3.8582396
## oleic       -95.2073733 -714.2913257  39.9685847
## linoleic     82.8671003  266.5890283  -8.0461880
## linolenic    47.1290403   37.9460639   0.2531958
## arachidic    37.9460639  117.0136942  -0.9268589
## eicosenoic    0.2531958   -0.9268589  35.0665592
##
## , , 4
##
##                 palmitic  palmitoleic     stearic        oleic      linoleic
## palmitic     4780.609761   831.427431 -126.407198  -8903.00585   3195.013662
## palmitoleic   831.427431   205.809454  -30.809324  -1807.97316    770.255777
## stearic      -126.407198   -30.809324  248.584548   -107.01520     -2.415328
## oleic       -8903.005852 -1807.973163 -107.015202  23167.62770 -11567.131668
## linoleic     3195.013662   770.255777   -2.415328 -11567.13167   7257.177601
## linolenic      -6.527096    -5.180364    8.690618    -47.84308     29.225839
## arachidic     136.038392    20.000866   -8.669656   -355.33915    157.220570
## eicosenoic    -11.375993    -2.127067    1.343425     20.09929     -7.428891
##                linolenic    arachidic  eicosenoic
## palmitic     -6.5270959  136.0383922 -11.3759930
## palmitoleic  -5.1803638   20.0008658  -2.1270666
## stearic       8.6906178   -8.6696557   1.3434248
## oleic       -47.8430783 -355.3391497  20.0992854
```

```
## linoleic     29.2258393  157.2205702  -7.4288906
## linolenic    11.8055397    6.8772705   0.1143033
## arachidic     6.8772705   28.2392900  -0.3395832
## eicosenoic    0.1143033   -0.3395832  13.6752643
##
## , , 5
##
##                palmitic palmitoleic     stearic        oleic     linoleic
## palmitic     4395.02689  547.524491 -282.062389 -3141.344483 -1481.201403
## palmitoleic   547.52449  226.106830  -77.143113  -632.386265   -40.499608
## stearic      -282.06239  -77.143113  471.115293  -132.159774    -1.612667
## oleic       -3141.34448 -632.386265 -132.159774  7747.495438 -3514.434900
## linoleic    -1481.20140  -40.499608   -1.612667 -3514.434900  4970.945961
## linolenic     -45.83531  -19.588761   13.216108   -25.377231    26.165568
## arachidic      52.08607   -5.502743  -21.255070  -134.232796    14.469595
## eicosenoic    -10.87425   -1.918671    2.730905     6.776474     3.332226
##                linolenic    arachidic  eicosenoic
## palmitic    -45.8353147   52.0860679 -10.8742474
## palmitoleic -19.5887609   -5.5027429  -1.9186711
## stearic      13.2161078  -21.2550705   2.7309049
## oleic       -25.3772310 -134.2327957   6.7764745
## linoleic     26.1655683   14.4695953   3.3322256
## linolenic    37.5020678    4.6658685   0.1131067
## arachidic     4.6658685   54.5720996  -0.1088962
## eicosenoic    0.1131067   -0.1088962  19.5204141
##
## , , 6
##
##                palmitic palmitoleic    stearic       oleic     linoleic
## palmitic    1145.906415   86.404400 -47.108358 -620.779863 -393.9209381
## palmitoleic   86.404400  272.379402 -35.378040 -144.601736  -44.2342803
## stearic      -47.108358  -35.378040 236.506126  -15.700532   16.7954993
## oleic       -620.779863 -144.601736 -15.700532 1677.639583 -675.1257389
## linoleic    -393.920938  -44.234280  16.795499 -675.125739 1240.2795368
## linolenic     -2.755129   -1.208089  15.148485    3.177687   14.8265437
## arachidic     11.293458    7.821923  -5.748355  -25.975054    2.1191026
## eicosenoic    -2.812816   -1.532190   0.920855    1.175240    0.7881368
##                linolenic   arachidic eicosenoic
## palmitic    -2.7551286  11.2934577 -2.8128161
## palmitoleic -1.2080885   7.8219230 -1.5321898
## stearic     15.1484851  -5.7483554  0.9208550
## oleic        3.1776870 -25.9750540  1.1752396
## linoleic    14.8265437   2.1191026  0.7881368
## linolenic   71.6422644  54.1478757 -0.6693485
## arachidic   54.1478757 132.8978768 -0.6741646
## eicosenoic  -0.6693485  -0.6741646  0.5859710
##
## , , 7
##
```

```
##                palmitic palmitoleic    stearic       oleic     linoleic
## palmitic     1300.735440  179.197449 -75.725855 -2125.80333   731.271371
## palmitoleic   179.197449  169.970204 -42.489099  -450.91340   159.552256
## stearic       -75.725855  -42.489099 251.562769   -69.78317   -48.218780
## oleic       -2125.803331 -450.913395 -69.783168  5646.63839 -2754.483559
## linoleic      731.271371  159.552256 -48.218780 -2754.48356  1888.084010
## linolenic      -6.504038   -8.298073   7.411640   -15.88080     3.124839
## arachidic      26.093297    1.616520  -9.870580   -92.32091    30.247843
## eicosenoic     -3.166069   -1.073061   1.315281     4.64579    -1.850987
##               linolenic   arachidic eicosenoic
## palmitic     -6.5040382  26.0932972 -3.1660692
## palmitoleic  -8.2980733   1.6165197 -1.0730605
## stearic       7.4116404  -9.8705799  1.3152809
## oleic       -15.8808009 -92.3209150  4.6457904
## linoleic      3.1248394  30.2478426 -1.8509867
## linolenic    33.8265457   6.4676691 -0.2137485
## arachidic     6.4676691  46.8032953 -0.1301896
## eicosenoic   -0.2137485  -0.1301896  0.5890748
##
## , , 8
##
##                palmitic palmitoleic     stearic        oleic     linoleic
## palmitic     296.0066654 -41.2916774  -0.6956206 -120.9904495  -26.8971680
## palmitoleic  -41.2916774 240.0688066 -44.3107546  -64.3522194  -44.6844446
## stearic       -0.6956206 -44.3107546 138.5110136   -2.7709775    4.1114258
## oleic       -120.9904495 -64.3522194  -2.7709775  443.0891877 -145.4399938
## linoleic     -26.8971680 -44.6844446   4.1114258 -145.4399938  309.5035648
## linolenic    -14.0057585 -22.5460702 -16.5579032  -14.8487547  -12.7419302
## arachidic    -20.3291519 -20.5221626 -25.5849468  -27.1525124  -21.2488142
## eicosenoic    -0.3575256  -0.8834544   0.8322708    0.4431739    0.2926338
##               linolenic  arachidic eicosenoic
## palmitic    -14.005758 -20.329152 -0.3575256
## palmitoleic -22.546070 -20.522163 -0.8834544
## stearic     -16.557903 -25.584947  0.8322708
## oleic       -14.848755 -27.152512  0.4431739
## linoleic    -12.741930 -21.248814  0.2926338
## linolenic   125.483433 -30.129548 -1.0040024
## arachidic   -30.129548 106.697007  0.3477470
## eicosenoic   -1.004002   0.347747  0.5618801
##
## , , 9
##
##                palmitic palmitoleic     stearic       oleic     linoleic
## palmitic     3484.664636  492.984775 -106.704235 -4633.24402   944.323317
## palmitoleic   492.984775  440.912983  -28.274437  -943.70031   309.541249
## stearic      -106.704235  -28.274437  448.787600   -60.36336    24.172112
## oleic       -4633.244023 -943.700314  -60.363356 12183.21378 -5864.941209
## linoleic      944.323317  309.541249   24.172112 -5864.94121  4686.969120
## linolenic     -46.499659  -58.207094  -29.306135   -58.31685   -11.849858
```

```
## arachidic       3.017629  -85.490297  -99.651046  -257.43378    -1.537328
## eicosenoic     -8.140538   -2.382401    2.265783    10.47428    -2.067022
##               linolenic   arachidic eicosenoic
## palmitic    -46.4996594    3.017629 -8.1405382
## palmitoleic -58.2070937  -85.490297 -2.3824013
## stearic     -29.3061352  -99.651046  2.2657834
## oleic       -58.3168503 -257.433783 10.4742787
## linoleic    -11.8498581   -1.537328 -2.0670220
## linolenic   141.6244438   66.734409 -0.9324717
## arachidic    66.7344089  271.294700 -0.3757820
## eicosenoic   -0.9324717   -0.375782  0.4930746
##
## , , 10
##
##                 palmitic  palmitoleic     stearic        oleic      linoleic
## palmitic     12853.08117  1575.468059 -862.946027 -10208.38311  -3269.567407
## palmitoleic   1575.46806   807.135781 -288.857259  -2102.83295      4.799549
## stearic       -862.94603  -288.857259 1595.045124   -461.66183   -102.284464
## oleic       -10208.38311 -2102.832946 -461.661829  25589.30386 -11755.752810
## linoleic     -3269.56741     4.799549 -102.284464 -11755.75281  14898.716583
## linolenic     -115.05896   -45.204114   70.409079    -68.62581     85.477202
## arachidic      179.88276    22.435824  -41.502224   -419.51158     84.590607
## eicosenoic     -32.05615    -6.728209    9.176439     21.79128      6.886280
##                linolenic    arachidic  eicosenoic
## palmitic    -115.0589573  179.8827615 -32.0561530
## palmitoleic  -45.2041136   22.4358244  -6.7282089
## stearic       70.4090785  -41.5022241   9.1764385
## oleic        -68.6258126 -419.5115791  21.7912766
## linoleic      85.4772019   84.5906066   6.8862800
## linolenic     50.3698863   43.7875051   0.2920138
## arachidic     43.7875051  120.8752748  -0.8490498
## eicosenoic     0.2920138   -0.8490498   0.5356093
##
##
## $variance$scale
##  [1] 422.81689 574.01086 475.72086 123.03157 209.50818 140.90960 102.09911
##  [8]  82.90581 259.14889 320.28618
##
## $variance$shape
##             [,1]         [,2]         [,3]         [,4]        [,5]
## [1,]  2.97562472   4.13963432   1.77151992   2.72419937  2.98969242
## [2,]  0.49496010   0.27892207   0.42210758   0.36836749  0.47361709
## [3,]  0.68366324   1.37014141   1.10585362   0.46439501  0.83658759
## [4,] 42.61266688 126.61120755 133.98537632 264.77268047 51.44981385
## [5,] 10.69973600   9.77657535  22.26746903  21.72174355 29.49659213
## [6,]  0.07025763   0.09120721   0.05471849   0.06652252  0.17009984
## [7,]  0.16212119   0.05868168   0.10074066   0.05057518  0.03516674
## [8,]  0.19123368   0.09541114   0.07353024   0.11089662  0.09299145
##              [,6]        [,7]        [,8]         [,9]        [,10]
```

```
## [1,]  1.818028735  3.19929801 1.705986720  1.925796124 6.668227e+00
## [2,]  1.139739783  0.66815866 1.796309602  1.987321931 6.584585e-01
## [3,]  2.091356498  1.62275899 3.633685877  1.412670934 2.232046e+00
## [4,] 16.098220242 77.28646163 6.856334382 65.607067445 1.111965e+02
## [5,] 11.300197376  8.10614931 3.963912609 11.890777313 5.356086e+01
## [6,]  0.267629045  0.29749630 1.776200979  0.434779862 6.930009e-02
## [7,]  1.187946697  0.27628016 0.282766457  0.313270672 1.917828e-01
## [8,]  0.003989986  0.00559838 0.006578998  0.001740726 1.289069e-03
##
## $variance$orientation
##                palmitic  palmitoleic      stearic         oleic     linoleic
## palmitic     0.29970533  0.266544231  0.374177514 -0.3210073512 -0.719124335
## palmitoleic  0.08089671  0.400475297 -0.830214026 -0.0654792280 -0.078499554
## stearic     -0.83873066  0.354018592  0.231899622 -0.0014363542  0.034866537
## oleic        0.29031521  0.264601064  0.211638693  0.8424730224  0.068015438
## linoleic     0.33773040  0.278824113  0.263724994 -0.4274822392  0.686110366
## linolenic   -0.04236638 -0.288353472  0.025408578 -0.0015497349  0.008328883
## arachidic   -0.00225188 -0.642136789 -0.044289981 -0.0126447304 -0.006029019
## eicosenoic  -0.00431184  0.000745507  0.003068714  0.0007340541  0.001755535
##                linolenic    arachidic    eicosenoic
## palmitic    -0.037143960  0.278473418  0.0030707083
## palmitoleic -0.015486700  0.364833753  0.0051686011
## stearic     -0.090989933  0.328508465 -0.0030923340
## oleic       -0.044968626  0.290691605  0.0013186186
## linoleic    -0.046407744  0.288098719  0.0011709941
## linolenic    0.858224161  0.421548892  0.0093711163
## arachidic   -0.499318250  0.579808659  0.0007978116
## eicosenoic  -0.007618308 -0.006859039  0.9999313624
##
##
## $Vinv
## NULL
```

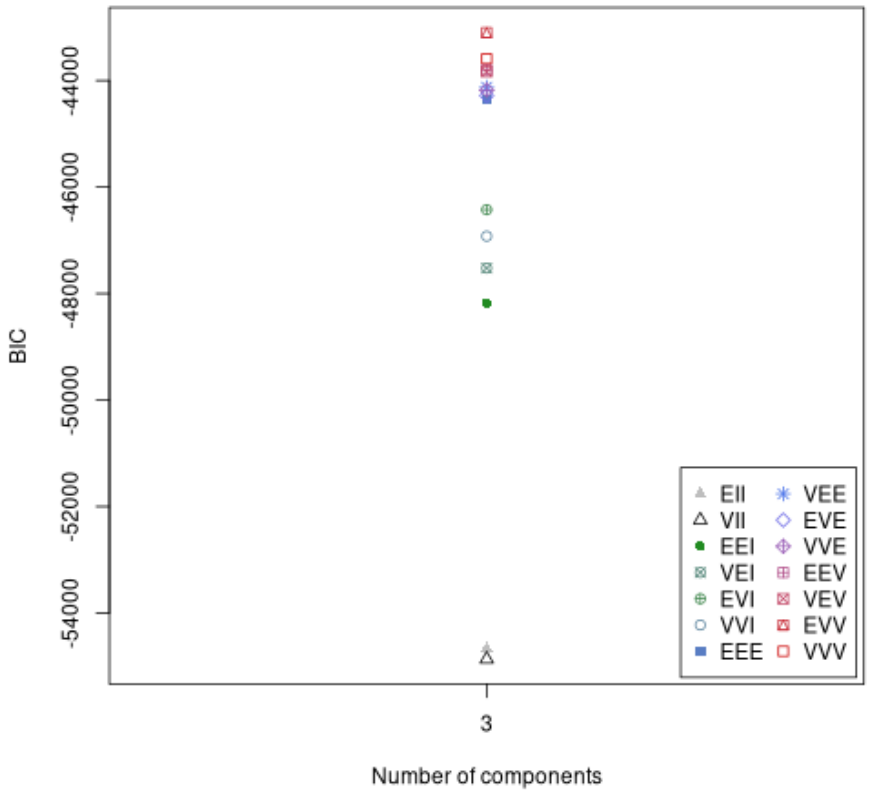
```
# Matrix of posterior probabilities p_ik that point i was generated
# by mixture component k
head(molive$z) # very long
```

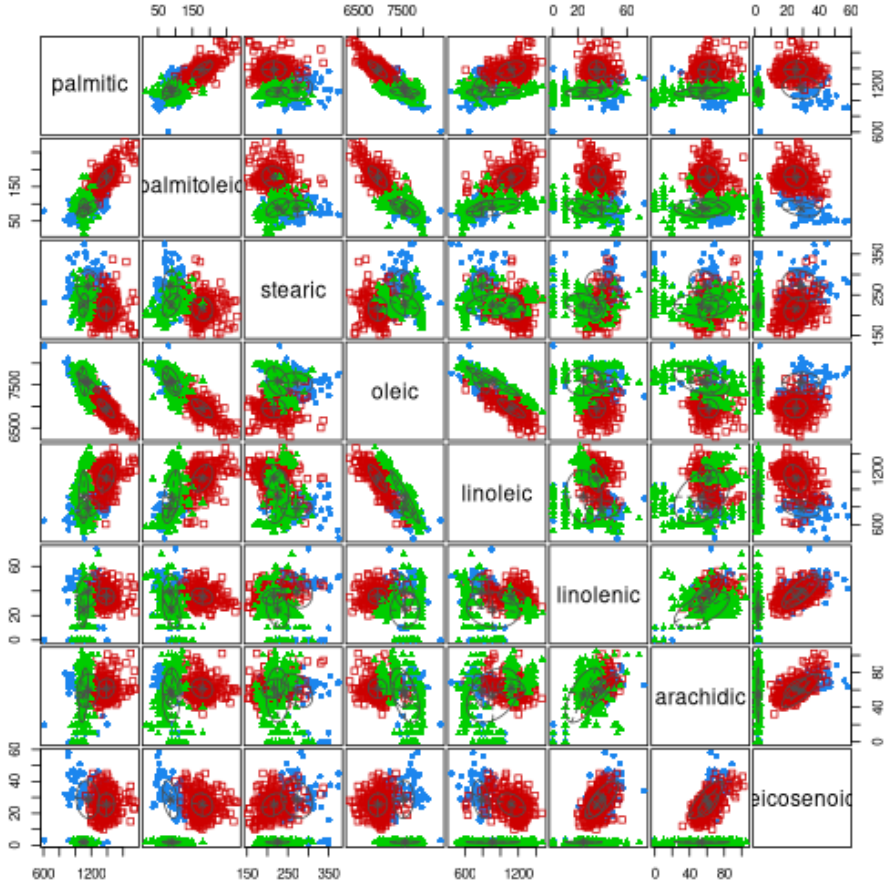
```
##           [,1]         [,2]         [,3]         [,4]         [,5]
## [1,] 0.9828720 0.0171210339 6.972116e-06 1.957467e-09 1.551460e-24
## [2,] 0.9890972 0.0108816316 2.117382e-05 9.481121e-10 5.352192e-19
## [3,] 0.9988935 0.0011064565 4.249024e-09 6.354440e-16 2.349993e-41
## [4,] 0.9998020 0.0001980478 6.850152e-10 1.063673e-22 2.439681e-39
## [5,] 0.9979165 0.0020834806 1.087652e-10 3.839161e-25 1.405407e-37
## [6,] 0.9998807 0.0001193443 4.230648e-11 1.048340e-23 6.407496e-42
##               [,6]          [,7]          [,8] [,9] [,10]
## [1,] 9.881313e-324  0.000000e+00 1.212380e-298    0     0
## [2,] 3.166612e-303 6.775122e-320  0.000000e+00    0     0
## [3,]  0.000000e+00  0.000000e+00 9.241535e-307    0     0
## [4,]  0.000000e+00  0.000000e+00  0.000000e+00    0     0
## [5,]  0.000000e+00  0.000000e+00  0.000000e+00    0     0
```

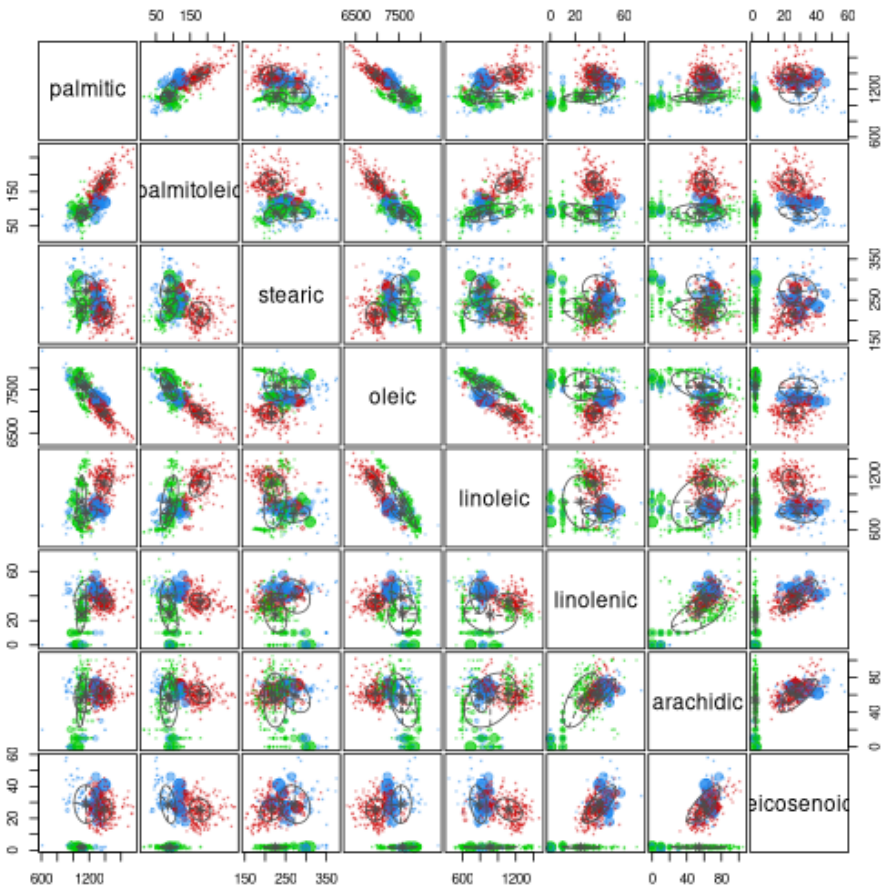
```
## [6,]  0.000000e+00  0.000000e+00  0.000000e+00    0     0
```

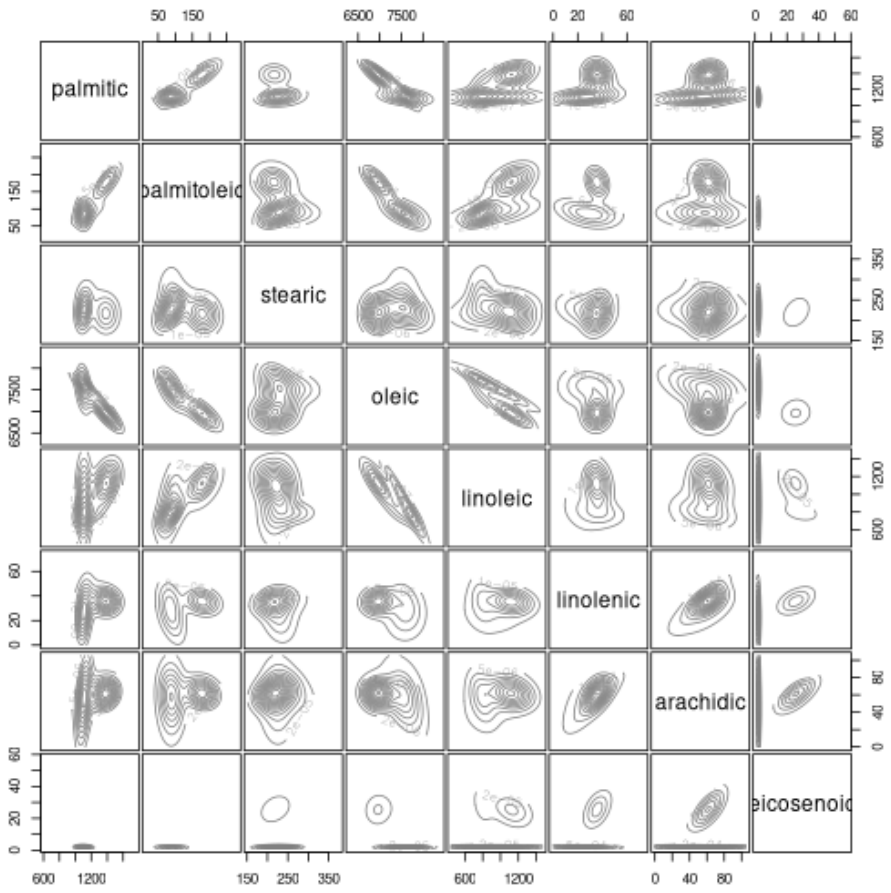
Could try model with 3 clusters and compare with Macro areas

```
molive3 <- Mclust(olive, G = 3)
adjustedRandIndex(molive3$classification, oliveoil$macro.area) # [1] 0.5349465, Also

## [1] 0.5349465

# Set of diagnostic plots offered by mclust
plot(molive3) #first is BIC for only three groups
```



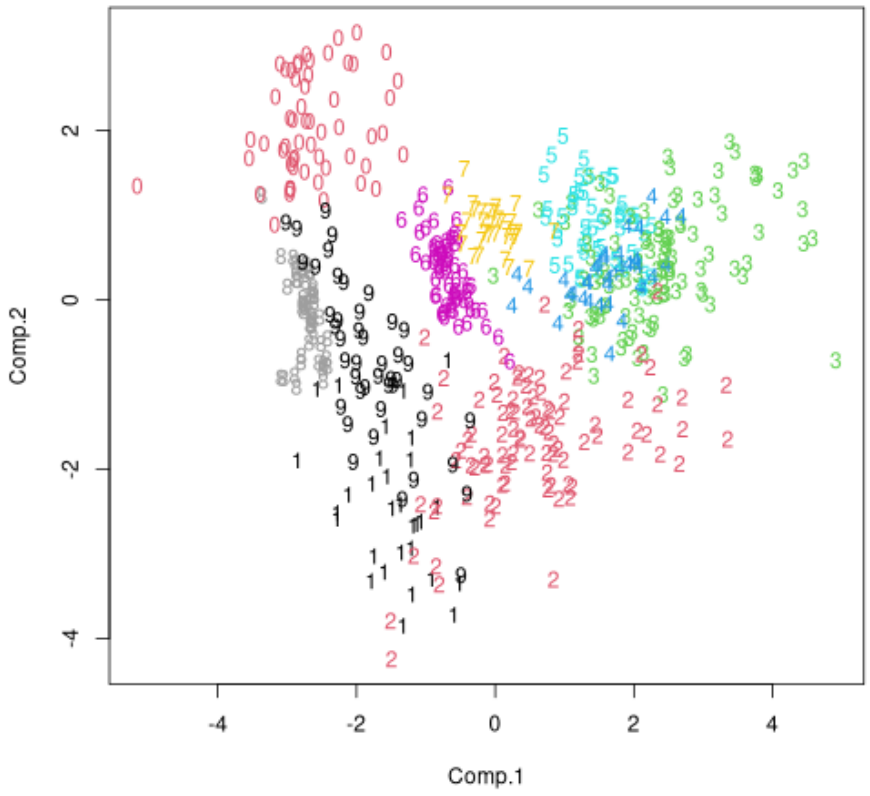

palmitic
palmitoleic
stearic
oleic
linoleic
linolenic
arachidic
eicosenoic


palmitic
palmitoleic
stearic
oleic
linoleic
linolenic
arachidic
eicosenoic



PCA can be clearer for not too low dimensional data here we plot the optimal ten cluster solution above

```
solive <- scale(olive) ## Standardise data
sprolive <- princomp(solive)
plot(sprolive$scores, col=molive$classification, pch=clusym[molive$classification])
```



It make sense to the prof

4.6 Big data sets

If we try to apply EM-algorithm and hierarchichal initialisation to big dataset:

- all observations are in memory (there could be memory problem)
- algorithm can be too *slow*

Fraley et al. (2005) propose the following strategy:

1. take random subset of large dataset (recommended size 2000; this may miss small clusters);
2. apply standard EM to random subset of data;
3. use results as inizialization for a single EM iteration on all observations;
4. to get smaller cluster, take 1% of data with lowest density values as additional cluster; with this start a single EM iteration on all observations, keep only if BIC is improved.

This (and individual steps) can be repeated several times.

4.7 Mixture model with skew and heavy-tailed distributions

Mixtures are general; not only Gaussians can be mixed. Could be interested in clusters that have heavier tails (higher probability of observation away from mean) or are skewed. An issue is that there are lots of possibilities(see, e.g., Lee and McLachlan (2013) with discussion).
We start with some distribution theory on some families to be mixed; note that there are alternative definitions for the skew distributions.

4.7.1 Some families

4.7.1.1 The t-distribution

Definition 4.7.1. If $Z \sim N(0,1)$ and $V \sim \chi^2_\nu$ are independent, then the ratio $\frac{Z}{\sqrt{V/\nu}} \sim t_\nu$.
Density has form:

$$f(x) = const\left(1 + \frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

compared to $const \exp\left(-\frac{x^2}{2}\right)$ for $N(0,1)$.

Remark 35. Some features:

- The t-distribution has heavier tails than Gaussian (it goes down to tail much slower than gaussian): observations scatter further away from center with larger probability. The lower df ν, the heavier tails are;
- if $\nu \to \infty$ T distribution converges to a gaussian distribution (we're interested in clustering from T with lower degrees of freedom otherwise we would use a normal)
- The standard t distribution is centered around 0; a general t-distribution with general location a and scale s can be computed in the following way. Let $T \sim t_\nu$ and let $Z \sim N(0, s^2)$ then the linear combination

$$a + sT \sim t_{\nu,a,s}.$$
$$a + \frac{Z}{\sqrt{V/\nu}} \sim t_{\nu,a,s}.$$

  is centered in a, with spread s.
  We will be interested to estimating a and s for different cluster.
- Finally
  - t_1 (also called Cauchy distribution) doesn't have expected value (integral of $x \cdot f(x)$ diverges) and thus no variance either (given definition of variance counting on expected value);
  - t_2 has expected value, but ∞ variance;
  - t_3 has expected value and variance but not third moment $\mathbb{E}\left[(X - \mathbb{E}[X])^3\right]$

– etc

Definition 4.7.2 (The multivariate t-distribution)**.** For the generalization, if $\mathbf{Z} \sim N_p(\mathbf{0}, \mathbf{\Sigma})$ and $V \sim \chi^2_\nu$, independent, then

$$\mathbf{a} + \frac{\mathbf{z}}{\sqrt{V/\nu}} \sim t_{p;\nu}(\mathbf{a}, \mathbf{\Sigma})$$

Here $\mathbf{\Sigma}$ is called scatter matrix. The covariance-matrix of this is not $\mathbf{\Sigma}$, but $\frac{\nu}{\nu-2}\mathbf{\Sigma}$, and only exists for $\nu > 2$ (otherwise it cannot be computed).

Remark 36. Multivariate t is elliptical distribution symmetric about $\mathbf{a}$ and has t-marginals distributions (where the multivariate Gaussian has Gaussian marginals).

4.7.1.2 The skew-normal distribution

Let

- $\boldsymbol{\lambda} \in \mathbb{R}^p$,
- $\mathbf{\Sigma}$ a $p \times p$-covariance matrix,
- $\mathbf{\Sigma}^{1/2}$ so that $\mathbf{\Sigma}^{1/2}\mathbf{\Sigma}^{1/2} = \mathbf{\Sigma}$ (can be constructed from spectral decomposition)
- $\boldsymbol{\delta}^* = \frac{\lambda^T}{\sqrt{1+\boldsymbol{\lambda}^T\boldsymbol{\lambda}}}\mathbf{\Sigma}^{1/2}$

Let $Z_0 \sim N(0,1)$, $\mathbf{Z}_1 \sim N_p(\mathbf{a}, \mathbf{\Sigma})$, and $Cov(Z_0, \mathbf{Z}_1) = \boldsymbol{\delta}^*$ (a p-dimensional vector of covariances between components of $\mathbf{Z}_1$ and Z_0).
Then $\mathbf{Z}_1 | Z_0 > 0$ (distribution of $\mathbf{Z}_1$ under the condition that $Z_0 > 0$) is said to have a (multivariate) **skew-normal distribution** (SN_p) with parameters

- $\mathbf{a}$, a parameter of location in p-dimensional space (but not the mean of the distribution),
- $\mathbf{\Sigma}$ covariance matrix of the normal on which it's based (not the covariance of the resulting skew-normal distribution)
- skewness parameter $\boldsymbol{\lambda}$ tells how the distribution is skewed in the p directions (zero: no skewness, positive and negative: left and right skewness).

NB: presumo usi le densità di due normali multivariate come base

The density can be shown to be the following

$$g_{\mathbf{a},\mathbf{\Sigma},\lambda}(\mathbf{x}) = 2\phi_{\boldsymbol{a},\mathbf{\Sigma}}(\mathbf{x})\Phi_{\mathbf{0},\mathbf{I}_p}\left(\boldsymbol{\lambda}^T\mathbf{\Sigma}^{-1/2}(\mathbf{x}-\mathbf{a})\right).$$

An equivalent definition of skew normal is the following. If we have another random normal variable $\mathbf{Z}_2 \sim N_p(\mathbf{0}, \mathbf{\Sigma} - \delta^*\delta^{*T})$, then the distribution of $\mathbf{Z} = \mathbf{a} + \boldsymbol{\delta}^*|Z_0| + \mathbf{Z}_2$ is still SN_p ($\mathbf{a}$ is still a positioning parameter added to $\mathbf{Z}_2$, then the skewness is added by considering the absolute value of Z_0 and $\boldsymbol{\delta}^*$).

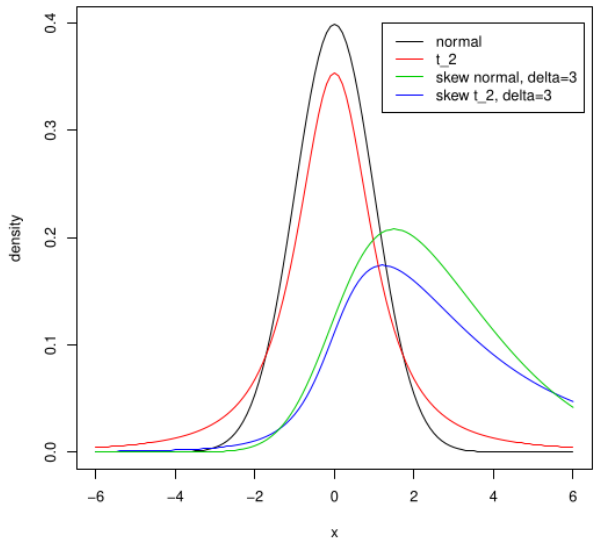


Figure 4.5: 1d skew dens

4.7.1.3 The skew t-distribution

It relates to the t-distribution in the same way the skew normal relates to the classical normal; it generates from SN and χ_2 in the same way as t is generated from N and χ_2 .
If $\mathbf{Z} \sim SN_p(\mathbf{0}, \mathbf{\Sigma}, \boldsymbol{\lambda})$, $V \sim \chi^2_\nu$, independent, then

$$\mathbf{a} + \frac{\mathbf{Z}}{\sqrt{V/\nu}} \sim st_{p;\nu}(\mathbf{a}, \mathbf{\Sigma}, \boldsymbol{\lambda})$$

has a multivariate skew-t distribution with ν df. It's heavy tailed as t-distribution but allows for more skewness and extreme points.
Also, a skew-t can also be equivalently generated as the distribution of $\mathbf{Z} = \mathbf{a} + \boldsymbol{\delta}^*|Z_0| + \mathbf{Z}_1$ where $Z_0 \sim t_\nu$, $\mathbf{Z}_1 \sim t_{p;\nu}(\mathbf{0}, \mathbf{\Sigma} - \delta^*\delta^{*T})$.

4.7.1.4 Graphs

Classical/symmetric and positively skewed (delta=3) densities (in 1 dimension) are shown in 4.5

4.7.2 Mixture modelling with skewed distribution

Mixture modelling based on any of these is done by fitting with skewness parameter as well (other than the others that were previously already presented) ;

$$f\eta(\mathbf{x}) = \sum_{k=1}^{K} \pi_k f_{\theta_k}(\mathbf{x})$$

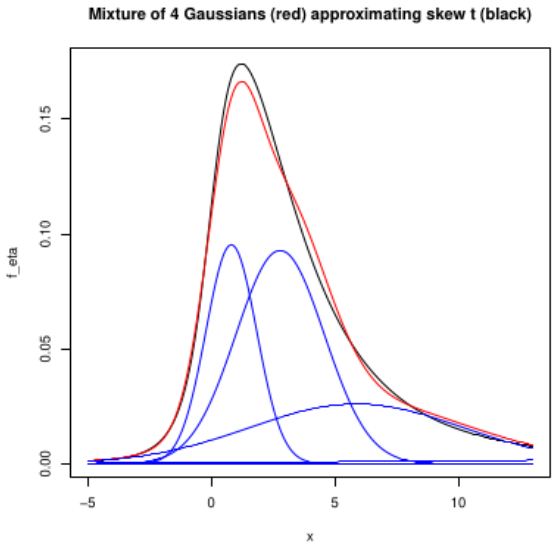


Figure 4.6: asd

where $\eta = (\pi_1, \ldots, \pi_K, \theta_1, \ldots, \theta_K)$, e.g., $\theta_k = (\mathbf{a}_k, \mathbf{\Sigma}_k, \nu_k, \lambda_k)$, $k = 1, \ldots, K$ for multivariate skew-t.
Here we have more parameter respect to multivariate normal to be estimated (ν_k-s for t and λ_k-s for skew distribution) and skew t is actually the most flexible model. More parameters allows a more flexible estimate/shape. To manage number of parameters we can choose to fit less more flexible distribution or more cluster of gaussian distribution. I think 2d visualization is key nell'ottica del prof on what to choose.
Estimation can use ML, EM-algorithm and BIC as previous: it's numerically more complicated and software sometimes doesn't give solution.
Like in the gaussian (VVV-III shit) one could define more parsimonious models by constraining $\mathbf{\Sigma}_k$ to be equal, diagonal or spherical; here also one can set ν_k or λ_k to be equal.
One can use BIC to compare different distributional shapes, but truth is that different mixtures fit data sets in different ways and it is often hard to say what's best; visual or other validation can help.
The true kind of mixture is hard to identify from data. Mixtures can well approximate each other (t can be very similar to normal), so it's hard to say which mixture to choose: depends on whether e.g. skew shape should be interpreted as one cluster, or as one symmetric core cluster plus others (fig **??**).

For data if figure 4.7 BIC with mixture of skew-t gives one cluster (left), while BIC with Gaussian mixtures (mclust) gives 2 (right). Both of these can make sense, depending on application (what my cluster are desidered to be).

4.7.3 Examples

We see three software packages for three family of mixture

- `teigen::teigen` is a function for multivariate t mixture (similar to `mclust`)

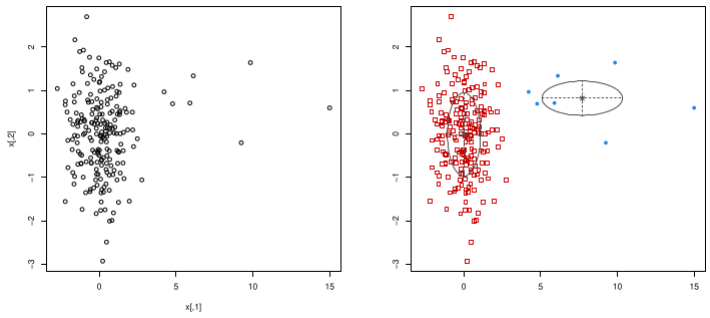


Figure 4.7: foo

- `smsn.mmix` and `smsn.search` in package `mixsmsn` for skew-normal
- `MGHM` in package `MixtureMissing` for skew-t

4.7.3.1 Mixture of T

Fitted by function teigen in package teigen.

```
library(teigen)
set.seed(776655) # set seed bc there's random inizialization of algorithm
capture.output(tolive <- teigen(olive, Gs=1:12)) # capture.output to avoid latex complaining

## [1] "\bTime taken:???  |  Approx. remaining:???  |  0% complete\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\

# Gs is numbers of searched clusters, optimum decided by BIC
#
# teigen has a parameter scale with default scale=TRUE.
# This standardises data before clustering, so result should be
# same as teigen(solive, Gs=1:12, scale=FALSE)
#
# See teigen help page for more options.

summary(tolive)

## ------------- Summary for teigen -------------
##              ------   RESULTS   ------
##              Loglik:          -1817.584
##              BIC:             -5063.723
##              ICL:             -5072.88
##              Model:           UUUC
##              # Groups:        5
##
##
## Clustering Table:
##
##   1   2   3   4   5
##  60 100  87 223 102
```

Like mclust, teigen fits various "covariance" matrix models based on spectral decomposition using possible constraints of volume, shape, orientation.
Letters used are

- "U" unconstrained (equivalent to `mclust` V)
- "C" constrained to be equal for all clusters (E),
- "I" unit matrix

There is also a *fourth letter* for the degrees of freedom of the t-distribution.
Above in the example, model "UUUC" as found optimal for olive oil data has unconstrained volume, shape, orientation, but equal degrees of freedom of the t-distributions.
To check all the BIC for all the models (large is good)

```
head(tolive$allbic) # show only some
```

```
##             G=1       G=2       G=3       G=4       G=5       G=6       G=7
## UUUU -7889.168 -5720.133 -5680.123 -5275.945 -5075.159 -5222.871 -5383.248
## UUUC      -Inf -5724.228 -5669.654 -5262.440 -5063.723 -5211.953 -5342.531
## CUCU      -Inf -6277.511 -6112.269 -5819.137 -5566.531 -5693.455 -5564.644
## CUCC      -Inf -6275.323 -6105.870 -5807.910 -5552.134 -5673.826 -5577.603
## CUUU      -Inf -5783.868 -5950.907 -5449.103 -5269.403 -5387.851 -5159.616
## CUUC      -Inf -5777.513 -5946.490 -5430.813 -5254.264 -5363.843 -5155.408
##             G=8       G=9      G=10      G=11      G=12
## UUUU -5164.612 -5259.720      -Inf      -Inf      -Inf
## UUUC -5130.624 -5258.010      -Inf      -Inf      -Inf
## CUCU -5754.682 -5677.640 -5874.410 -5847.167 -5925.770
## CUCC -5724.199 -5651.541 -5823.315 -5725.638 -5707.712
## CUUU -5131.501 -5218.922      -Inf      -Inf      -Inf
## CUUC -5148.231 -5217.414      -Inf      -Inf      -Inf
```

To have more look at

```
str(tolive)
```

```
## List of 13
##  $ iter          : num 32
##  $ fuzzy         : num [1:572, 1:5] 1.77e-17 5.24e-17 1.09e-15 2.04e-18 3.83e-20 .
##  $ parameters    :List of 9
##   ..$ df     : num [1:5] 7.68 7.68 7.68 7.68 7.68
##   ..$ mean   : num [1:5, 1:8] -0.971 -0.154 -0.764 0.921 -0.719 ...
##   ..$ lambda : num [1:5] 0.0542 0.1141 0.0376 0.1073 0.0368
##   ..$ d      : num [1:8, 1:8, 1:5] 0.1013 0.0102 -0.9745 0.1017 -0.1085 ...
##   ..$ a      : num [1:8, 1:8, 1:5] 19.5 0 0 0 0 ...
##   ..$ weights: num [1:572, 1:5] 0.00645 0.00653 0.00685 0.00434 0.00251 ...
##   ..$ sigma  : num [1:8, 1:8, 1:5] 0.3136 0.0163 -0.0513 -0.0631 -0.1265 ...
##   ..$ pig    : num [1:5] 0.105 0.176 0.153 0.389 0.177
##   ..$ conv   : logi TRUE
##  $ allbic        : num [1:28, 1:12] -7889 -Inf -Inf -Inf -Inf ...
##   ..- attr(*, "dimnames")=List of 2
```

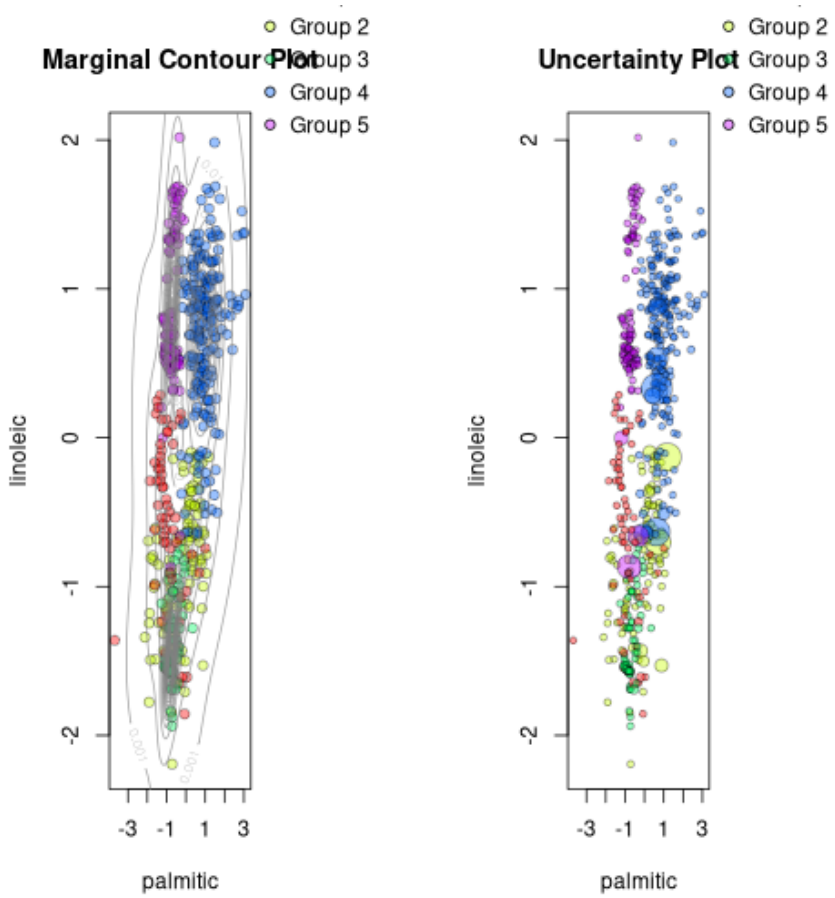
```
##   .. ..$ : chr [1:28] "UUUU" "UUUC" "CUCU" "CUCC" ...
##   .. ..$ : chr [1:12] "G=1" "G=2" "G=3" "G=4" ...
##  $ bic           : num -5064
##  $ bestmodel     : chr "The best model (BIC of -5063.72) is UUUC with G=5"
##  $ modelname     : chr "UUUC"
##  $ classification: int [1:572] 2 2 2 2 2 2 2 2 2 2 ...
##  $ G             : int 5
##  $ x             : num [1:572, 1:8] -0.93 -0.853 -1.902 -1.576 -1.072 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:8] "palmitic" "palmitoleic" "stearic" "oleic" ...
##   ..- attr(*, "scaled:center")= Named num [1:8] 1232 126 229 7312 981 ...
##   .. ..- attr(*, "names")= chr [1:8] "palmitic" "palmitoleic" "stearic" "oleic" ...
##   ..- attr(*, "scaled:scale")= Named num [1:8] 168.6 52.5 36.7 405.8 242.8 ...
##   .. ..- attr(*, "names")= chr [1:8] "palmitic" "palmitoleic" "stearic" "oleic" ...
##  $ logl          : num -1818
##  $ iclresults    :List of 10
##   ..$ iter          : num 32
##   ..$ fuzzy         : num [1:572, 1:5] 1.77e-17 5.24e-17 1.09e-15 2.04e-18 3.83e-20 ...
##   ..$ allicl        : num [1:28, 1:12] -7889 -Inf -Inf -Inf -Inf ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:28] "UUUU" "UUUC" "CUCU" "CUCC" ...
##   .. .. ..$ : chr [1:12] "G=1" "G=2" "G=3" "G=4" ...
##   ..$ parameters    :List of 9
##   .. ..$ df     : num [1:5] 7.68 7.68 7.68 7.68 7.68
##   .. ..$ mean   : num [1:5, 1:8] -0.971 -0.154 -0.764 0.921 -0.719 ...
##   .. ..$ lambda : num Inf
##   .. ..$ d      : num Inf
##   .. ..$ a      : num Inf
##   .. ..$ weights: num [1:572, 1:5] 0.00645 0.00653 0.00685 0.00434 0.00251 ...
##   .. ..$ sigma  : num [1:8, 1:8, 1:5] 0.3136 0.0163 -0.0513 -0.0631 -0.1265 ...
##   .. ..$ pig    : num [1:5] 0.105 0.176 0.153 0.389 0.177
##   .. ..$ conv   : logi TRUE
##   ..$ icl           : num -5073
##   ..$ bestmodel     : chr "The best model (ICL of -5072.88) is UUUC with G=5"
##   ..$ classification: int [1:572] 2 2 2 2 2 2 2 2 2 2 ...
##   ..$ modelname     : chr "UUUC"
##   ..$ G             : int 5
##   ..$ logl          : num -1818
##  $ info          :List of 5
##   ..$ univar    : logi FALSE
##   ..$ gauss     : logi FALSE
##   ..$ scalelogic: logi TRUE
##   ..$ scalemeans: Named num [1:8] 1232 126 229 7312 981 ...
##   .. ..- attr(*, "names")= chr [1:8] "palmitic" "palmitoleic" "stearic" "oleic" ...
##   ..$ scalesd   : Named num [1:8] 168.6 52.5 36.7 405.8 242.8 ...
##   .. ..- attr(*, "names")= chr [1:8] "palmitic" "palmitoleic" "stearic" "oleic" ...
##  - attr(*, "class")= chr "teigen"
```

of interest can be `parameters` with df. Note df=7.68; degrees of freedom don't have to be integer numbers! `pig` are the component proportions (sums up to 1)
The plot command for `teigen` allows for two kinds of plots (contour e uncertainty):

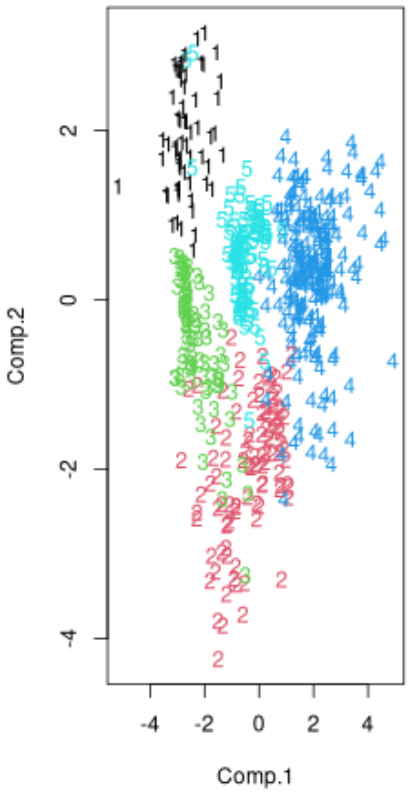
- Contour gives density contour overall and color for clustering (in the graph there are 5 clustering but legend is)
- uncertainty is like above where biggere observation are the one where we are more uncertain about classification (observations on border of clusters)

Both plot are done on *two variables*, and xmarg, ymarg (where one has to specify the numbers of the variables used). (it should do the matrix plot but it doesn't)

```
# teigen plots
par(mfrow=c(1,2))
plot(tolive, xmarg=1, ymarg=5, what="contour") # use variable 1 and 5-th
plot(tolive, xmarg=1, ymarg=5, what="uncertainty")
```



```
# Plot the clustering done on PCs
plot(sprolive$scores,
     col = tolive$classification,
     pch = clusym[tolive$classification])

# ARI with region and macro area are clearly better than from mclust.
adjustedRandIndex(tolive$classification, oliveoil$region)

## [1] 0.7727825

adjustedRandIndex(tolive$classification, oliveoil$macro.area)

## [1] 0.6189345
```



4.7.3.2 Mixture of skew-normal distributions

Important remark 42. there are problems with this package, so this part is just for information. It's ok to emulate skew-normal by skew-t with very large degrees of fredom Take home: don't use this

These can be fitted by functions `smsn.mmix` and `smsn.search` in package

`mixsmsn`. The first fit a mixture with given k number of components, the second select number of clusters by BIC.

```
library(mixsmsn)
set.seed(67543)
# WARNING! The mixsmsn package is slow and not very stable
# and can produce errors easily.
# Sometimes only very limited range of g.min, g.max will work.
estg <- smsn.search(solive,
                    nu = 1, # not needed for a skew normal mixture (it's df for
                            # T) but has to be specified.
                    g.min = 1,  # min. number of clusters
                    g.max = 12, # max. number of clusters
                    family = "Skew.normal") # fits a mixture of skew t-distributions,
                                            # with nu degrees of freedom,

# Can also specify uni.Gama=TRUE, which constrains
# Sigma-delta^* delta^*T to be constant over clusters.

str(estg)

# ARI is about the same as from teigen, slightly better.
adjustedRandIndex(estg$best.model$group, oliveoil$region)
adjustedRandIndex(estg$best.model$group, oliveoil$macro.area)

# BIC by number of cluster and PCA plot clustering
plot(1:12, estg$criteria,type="l", ylab="BIC", xlab="Number of clusters")
plot(sprolive$scores,col=estg$best.model$group,pch=clusym[estg$best.model$group])
```

As opposed to `mclust` and teigen, `mixsmsn` uses "small is good" version of BIC (as does `flexmix` in the next subsection).

4.7.3.3 Mixture of skew-t distributions

These can be fitted by `MixtureMissing::MGHM` (unfortunately does not fit skew-normal but np). It's a very new package, but seems more stable and faster than `mixsmsn`.

```
library(MixtureMissing)
# still takes a bit of time
stolive <- MGHM(solive,
                G = 1:12, # number of clusters
                model = "St", #"St" for the skew t-distribution.
                criterion = "BIC") # use BIC, small is good version

##
## Mixture: Skew-t (St)
## Data Set: Complete
## Initialization: kmedoids
```

```
##
## Fitting G = 1 was successful with 1/20 iterations
## Fitting G = 2 was successful with 20/20 iterations
## Fitting G = 3 was successful with 20/20 iterations
## Fitting G = 4 was successful with 20/20 iterations
## Fitting G = 5 was successful with 20/20 iterations
## Fitting G = 6 was successful with 20/20 iterations
## Fitting G = 7 was successful with 20/20 iterations
## Fitting G = 8 was successful with 20/20 iterations
## Fitting G = 9 was successful with 20/20 iterations
## Fitting G = 10 was successful with 20/20 iterations
## Fitting G = 11 was successful with 20/20 iterations
## Fitting G = 12 was successful with 20/20 iterations
##
## According to BIC, the best mixture model is based on G = 6
## Model rank according to BIC:
##   1. G = 6: 5036.434
##   2. G = 5: 5098.621
##   3. G = 8: 5198.211
##   4. G = 7: 5255.367
##   5. G = 9: 5311.645
##   6. G = 4: 5347.216
##   7. G = 3: 5488.514
##   8. G = 10: 5585.83
##   9. G = 11: 5684.283
##   10. G = 12: 5930.828
##   11. G = 2: 6342.618
##   12. G = 1: 7891.56
```

```
## Best solution is with 6 components
summary(stolive)
```

```
##
## Model: 6-Component Skew-t Mixture with Complete Data
##
## Iterations: 20 / 20
##
## Initialization: kmedoids
##
## Component frequency table:
##  comp1 comp2 comp3 comp4 comp5 comp6
##    100    48    48   223   100    53
##
## Mixing proportions:
##      comp1      comp2      comp3      comp4      comp5      comp6
## 0.17450367 0.08308004 0.08389103 0.39018759 0.17496936 0.09336830
##
## Component location vectors:
##         palmitic palmitoleic     stearic      oleic   linoleic  linolenic
## comp1  0.9318401  -0.1984367  1.21089037 -0.5693817 -0.2142292  1.1068527
```

```
## comp2 -0.6052322  -0.6916621  0.42341857  1.0190179 -1.0566990 -1.1238889
## comp3 -0.8637187  -1.1634296 -1.02054844  1.5670891 -1.5570326 -0.2916482
## comp4  0.5948569   0.6908524 -0.80885058 -0.7044957  0.7929298 -0.1635678
## comp5 -0.7195797  -0.6140842 -0.08157407 -0.1378401  0.9699509 -0.3297054
## comp6 -1.2473650  -0.5241897 -0.08398243  0.9769803 -0.1711506 -2.3305284
##        arachidic eicosenoic
## comp1  0.3991536  1.0595263
## comp2  0.4084455 -1.0052331
## comp3 -0.5565152 -1.0142753
## comp4 -0.3540394  0.3058477
## comp5  0.7648742 -1.0301563
## comp6 -2.6319964 -1.0145188
##
## Component dispersion matrices:
## , , comp1
##
##                  palmitic  palmitoleic       stearic       oleic    linoleic
## palmitic     0.5173071601  0.315310864  0.0008520692 -0.31237702  0.07921858
## palmitoleic  0.3153108635  0.311563290 -0.0706345300 -0.21114508  0.06179520
## stearic      0.0008520692 -0.070634530  1.0158033178 -0.04567005 -0.08198503
## oleic       -0.3123770228 -0.211145077 -0.0456700507  0.23528731 -0.10619777
## linoleic     0.0792185781  0.061795198 -0.0819850328 -0.10619777  0.11923784
## linolenic    0.0037600974  0.006097752  0.1040739628 -0.03645304  0.01364502
## arachidic   -0.0826166619 -0.071773003 -0.0385385045  0.04783988 -0.01457164
## eicosenoic  -0.2291654564 -0.158317121  0.0587120437  0.10520478 -0.01647632
##                linolenic   arachidic  eicosenoic
## palmitic     0.003760097 -0.08261666 -0.22916546
## palmitoleic  0.006097752 -0.07177300 -0.15831712
## stearic      0.104073963 -0.03853850  0.05871204
## oleic       -0.036453041  0.04783988  0.10520478
## linoleic     0.013645024 -0.01457164 -0.01647632
## linolenic    0.225459762  0.05801199  0.07359649
## arachidic    0.058011989  0.11966107  0.08853980
## eicosenoic   0.073596493  0.08853980  0.35617892
##
## , , comp2
##
##                  palmitic  palmitoleic      stearic        oleic     linoleic
## palmitic     0.0743227775 -0.006649220 -0.034265588 -0.021022963 -0.001977981
## palmitoleic -0.0066492200  0.063049323  0.046129716 -0.023384220  0.013489148
## stearic     -0.0342655880  0.046129716  0.235524877 -0.024371400  0.019925791
## oleic       -0.0210229634 -0.023384220 -0.024371400  0.065862301 -0.063440165
## linoleic    -0.0019779814  0.013489148  0.019925791 -0.063440165  0.078031398
## linolenic   -0.0497577478  0.037693432  0.040198831 -0.146324996  0.150364775
## arachidic   -0.0561935966  0.073688427  0.079529287 -0.163631257  0.171807994
## eicosenoic  -0.0004430403 -0.004394841  0.005635455  0.003310946 -0.004541598
##                linolenic    arachidic    eicosenoic
## palmitic    -0.0497577478 -0.056193597 -0.0004430403
## palmitoleic  0.0376934315  0.073688427 -0.0043948413
```

```
## stearic      0.0401988308  0.079529287  0.0056354553
## oleic       -0.1463249957 -0.163631257  0.0033109455
## linoleic     0.1503647754  0.171807994 -0.0045415985
## linolenic    0.8947358759  0.670169080 -0.0005817708
## arachidic    0.6701690797  0.863655311 -0.0050355315
## eicosenoic  -0.0005817708 -0.005035531  0.0023105470
##
## , , comp3
##
##                 palmitic   palmitoleic       stearic         oleic
## palmitic     0.0068025151 -0.0111879846  0.0171972008 -0.0032882607
## palmitoleic -0.0111879846  0.0266804467 -0.0286917093  0.0046329595
## stearic      0.0171972008 -0.0286917093  0.0599349612 -0.0075930647
## oleic       -0.0032882607  0.0046329595 -0.0075930647  0.0026529110
## linoleic    -0.0013376191  0.0028256603 -0.0034204053  0.0004553283
## linolenic    0.0322184475 -0.0632407885  0.0879711739 -0.0161069848
## arachidic   -0.0147980587  0.0225833861 -0.0704936808 -0.0018272371
## eicosenoic  -0.0009909411  0.0005738982 -0.0004752579  0.0007000540
##                 linoleic    linolenic    arachidic    eicosenoic
## palmitic    -0.0013376191  0.032218447 -0.014798059 -0.0009909411
## palmitoleic  0.0028256603 -0.063240788  0.022583386  0.0005738982
## stearic     -0.0034204053  0.087971174 -0.070493681 -0.0004752579
## oleic        0.0004553283 -0.016106985 -0.001827237  0.0007000540
## linoleic     0.0005372628 -0.006060849  0.003659609  0.0001820076
## linolenic   -0.0060608493  0.266558080 -0.100833744 -0.0017521348
## arachidic    0.0036596093 -0.100833744  0.222080611 -0.0042669503
## eicosenoic   0.0001820076 -0.001752135 -0.004266950  0.0024115739
##
## , , comp4
##
##                palmitic palmitoleic     stearic        oleic      linoleic
## palmitic     0.297310202  0.18294224  0.02430563 -0.181182599  0.0557342558
## palmitoleic  0.182942240  0.23953418 -0.01353399 -0.139132525  0.0726026651
## stearic      0.024305631 -0.01353399  0.46985717 -0.049449157 -0.0291699010
## oleic       -0.181182599 -0.13913252 -0.04944916  0.186155376 -0.1517072036
## linoleic     0.055734256  0.07260267 -0.02916990 -0.151707204  0.2228784217
## linolenic   -0.018983083 -0.03926113  0.02732867  0.002423270 -0.0001945538
## arachidic    0.007132415 -0.05547385  0.05201378  0.007790920 -0.0522391348
## eicosenoic   0.017502766 -0.02629561  0.08501689  0.008261777 -0.0731401501
##                linolenic    arachidic   eicosenoic
## palmitic    -0.0189830835  0.007132415  0.017502766
## palmitoleic -0.0392611262 -0.055473852 -0.026295609
## stearic      0.0273286671  0.052013778  0.085016886
## oleic        0.0024232702  0.007790920  0.008261777
## linoleic    -0.0001945538 -0.052239135 -0.073140150
## linolenic    0.1695522600  0.053412482  0.053197195
## arachidic    0.0534124820  0.187931133  0.116360267
## eicosenoic   0.0531971948  0.116360267  0.218518551
##
```

```
## , , comp5
##
##                 palmitic   palmitoleic       stearic         oleic
## palmitic     0.0515492927  0.0034079720  0.0342017394 -5.268917e-02
## palmitoleic  0.0034079720  0.0440956036  0.0455316190 -2.561527e-02
## stearic      0.0342017394  0.0455316190  0.2298920564 -1.238852e-01
## oleic       -0.0526891659 -0.0256152667 -0.1238852243  1.240506e-01
## linoleic     0.0481424236  0.0256699121  0.1478992727 -1.499558e-01
## linolenic   -0.0110296331 -0.0092662986 -0.1063934085  6.502613e-02
## arachidic    0.0055887502 -0.0081209048 -0.0439288854 -2.168202e-05
## eicosenoic   0.0002758025  0.0002326421  0.0005328294 -2.829127e-04
##                 linoleic     linolenic     arachidic    eicosenoic
## palmitic     0.0481424236 -0.0110296331  5.588750e-03  0.0002758025
## palmitoleic  0.0256699121 -0.0092662986 -8.120905e-03  0.0002326421
## stearic      0.1478992727 -0.1063934085 -4.392889e-02  0.0005328294
## oleic       -0.1499557979  0.0650261338 -2.168202e-05 -0.0002829127
## linoleic     0.1971698092 -0.0925019717 -9.309007e-03  0.0001748889
## linolenic   -0.0925019717  0.1539982880  7.915048e-02 -0.0007762328
## arachidic   -0.0093090073  0.0791504793  2.500546e-01  0.0007931003
## eicosenoic   0.0001748889 -0.0007762328  7.931003e-04  0.0025841149
##
## , , comp6
##
##                  palmitic  palmitoleic      stearic        oleic     linoleic
## palmitic     0.257422242  0.064503121 -0.019114954 -0.105861286 -0.020331745
## palmitoleic  0.064503121  0.144293637 -0.078149559 -0.062684720  0.038710398
## stearic     -0.019114954 -0.078149559  0.962381855 -0.070709591 -0.004851733
## oleic       -0.105861286 -0.062684720 -0.070709591  0.108978724 -0.085204920
## linoleic    -0.020331745  0.038710398 -0.004851733 -0.085204920  0.160822143
## linolenic    0.029458325  0.003195769  0.080781764  0.010114489 -0.063311050
## arachidic    0.031125826  0.022665353 -0.060497094  0.016833925 -0.063963828
## eicosenoic  -0.007819719 -0.005081572  0.016029281  0.001901206  0.001291440
##                linolenic     arachidic    eicosenoic
## palmitic     0.029458325  0.0311258262 -0.0078197186
## palmitoleic  0.003195769  0.0226653528 -0.0050815719
## stearic      0.080781764 -0.0604970940  0.0160292805
## oleic        0.010114489  0.0168339255  0.0019012058
## linoleic    -0.063311050 -0.0639638276  0.0012914401
## linolenic    0.130865235  0.0541857271 -0.0021652000
## arachidic    0.054185727  0.1297513262 -0.0008568493
## eicosenoic  -0.002165200 -0.0008568493  0.0026055972
##
##
## Final log-likelihood: -1492.831
##
## Total parameters: 323
##
## Information Criteria:
##       AIC      BIC      KIC     KICc     AIC3     CAIC    AICc      ICL
```

```
##  3631.662 5036.434 3957.662 4964.318 3954.662 5359.434 4475.63 5038.669
##       AWE      CLC
##  8060.676 2981.191

## Iterations is the number of iterations of EM, there's a max iteration
## parameter which can be set. Here all the 20 set iterations were used (it may
## not be the case if we converge before); this parameter can be set to higher
## level to obtain better solutions (maybe).
## Initialization: kmedoids is used for initialize EM
## Then the frequency table of clusters/components and corresponding
## percentages
## then the mean vector (vector a in course notes)
## dispersion matrices (Sigma in course notes)
## Total parameters is total parameters fitted (a lot 500 osservazioni e 323 parametri)

stolive$beta # skewness lambda in notes

##            palmitic palmitoleic     stearic       oleic     linoleic
## comp1 -1.0128419591 -0.34441091 -0.24058739  0.89406557 -0.553160696
## comp2  0.0926357238 -0.09260465 -0.11545319  0.04045863 -0.151927574
## comp3 -0.0032248837 -0.06358493  0.02267357  0.01353871 -0.001687739
## comp4  0.2995004191  0.29540056  0.31245761 -0.20550390 -0.078789568
## comp5 -0.0003341598  0.03938281  0.01374057  0.03020716 -0.069339444
## comp6  0.1445141066  0.08776968  0.58703488 -0.02235299 -0.179144785
##         linolenic   arachidic    eicosenoic
## comp1 -0.10654805 -0.01403276  0.0561509801
## comp2  0.79576202  0.02408988 -0.0174016948
## comp3  0.20534571 -0.11094776  0.0001083197
## comp4  0.33294217  0.41362528  0.2631260373
## comp5 -0.04825998 -0.09333079  0.0081865557
## comp6  0.16633214  0.26864928  0.0036840789

stolive$df # Degrees of freedom for t-distributions

##     comp1     comp2     comp3     comp4     comp5     comp6
## 15.302164 10.256666  3.561058 10.052606  6.379184  5.685475

# With plotting 4 plots are offered.
# - No. 4 shows the classification of points with very light density contours
# and is almost the same as No. 1 "classification".
# - No. 2 "parallel" could also be interesting.
# Can plot only no. 4 with plot(stolive,what="density")
plot(stolive) # need to specify 1-4 at comand line
plot(stolive,what="density")

## Warning in par(oldpar):  chiamata per par(new=TRUE) senza alcun
plot

# You can of course also use the clustering vector stolive$clusters
# for creating your own plot, e.g., based on Principal Components.
adjustedRandIndex(stolive$clusters, oliveoil$region) # 0.8127416 # best we've seen!
```

```
## [1] 0.8127416

adjustedRandIndex(stolive$clusters, oliveoil$macro.area)

## [1] 0.5945189
```

Important remark 43. **Comparing BIC from different models can be difficult** for different BIC formulation (not only the sign things, but there may be other differences regarding degrees of freedom)

NB: detto durante giovedi 14 novembre 2024

Remark 37. During exam one question could be to count number of degrees of freedom. BTW for df, for 6 mixture components we have:

- 6 parameters for mixing proportions, $6 - 1 = 5$ df for mixing (they have to sum up) proportion
- location vectors: 6 vectors for 8 variables each are 48 df
- covariance matrices: 6 matrices having each 8 variances and $\binom{8}{2} = 8*7/2 = 28$ unique covariances so 6 * (28 + 8) = 216 df
- t distribution df ν: in multivariate t ν is one scalar but here we have 6 distributions/components/cluster so 6 df
- skewness parameters $\boldsymbol{\lambda}$ is composed of 8 elements (one per variable) and we have 6 of it so 48

Parameter estimated are actually $6+48+216+6+48 = 324$: degrees of freedom are $5 + 48 + 216 + 6 + 48 = 323$ (1 is fixed due to mixing proportion), which is the number reported by software.
These are a lot of parameters for 572 observations

4.8 A mixture model for categorical data

Let

- $\mathbf{X}_1, \ldots, \mathbf{X}_n$ be iid
- each observation is composed of p variables $\mathbf{X}_i = (X_{i1}, \ldots, X_{ip})$, $i \in \mathbb{N}_n$
- each variable can assume a set of categories: for $j \in \mathbb{N}_p$ $X_{ij} \in \left\{c_1^{(j)}, \ldots, c_{m_j}^{(j)}\right\}$ set of $m_j \in \mathbb{N}$ categories (m_5 is number of categories for 5-th variable)
- a standard distribution for 1-d categorical data with m categories (and just one extraction) is the $Multinomial(1; \zeta_1, \ldots, \zeta_m)$ so each $\zeta_l = \P X = c_l$ is probability that observation assumes l-th category (equivalent of bernoulli for two categories)

If we want to have a mixture like done before that is

$$f_\eta(\mathbf{x}) = \sum_{k=1}^{K} \pi_k f_{\theta_k}(\mathbf{x}) \tag{4.2}$$

we need a distribution f_{θ_k}.
We could specify the probability for every possible combination of variables categories (but lot of parameters given by number of variables and categories).
Instead we do a mix of *locally independent multinomial distributions*: considering just a cluster, for an observation with p-variables $\mathbf{x} = (x_1, \dots, x_p)$ if we define $x^{jl} = 1(x_j = c_l^{(j)}$ as the indicator that j-th variables assumes its l-th category, we can define a single distribution with local independence where x^{jl} acts as "switch/selector" to choose the proper probability from the multinomial distribution of the single variable

$$f_\theta(\mathbf{x}) = \prod_{j=1}^{p} \prod_{l=1}^{m_j} \zeta_{jl}^{x^{jl}}$$

Here above $\zeta_{jl} = ¶X_j = c_l$ is the l-th probability parameter of the j-th variable multinomial, and we have the constraint that $\sum_{l=1}^{m_j} \zeta_{jl} = 1 \ \forall j$. All the parameters can be gathered as $\theta = (\zeta_{11}, \dots, \zeta_{pm_p})$.
So putting all together a distribution for a multivariate categorical observation can be summarized:

$$f_\theta(\mathbf{x}) = \prod_{j=1}^{p} \prod_{l=1}^{m_j} \zeta_{jl}^{x^{jl}} \quad \text{where} \quad \sum_{l=1}^{m_j} \zeta_{jl} = 1, \forall j$$
$$\theta = (\zeta_{11}, \dots, \zeta_{pm_p})$$

The number of degrees of freedom (parameters free to vary) are the number of category - 1 for each variable (probability of all categories but 1 for all the variables), that is $\sum_{j=1}^{p}(m_j - 1)$ ($\sum_{j=1}^{p} m_j$ parameters, p sum 1-constraints).
We may think that they're a big number of parameter but if we don't assume local independence (that is independence between variables within unit) the number of parameter to be estimated would be a lot more (think combinatorially to the product of the number of categories - 1 because probability must sums up to 1, that is $\left(\prod_{j=1}^{p} m_j\right) - 1$ degrees of freedom would be needed to model all possible probabilities .)
Assumption that within units, variables are independent is restrictive btw (one wouldn't want to assume this); we can relax/bring dependence if we define things as a mixture model of *locally* independent components.
Independence is just within cluster/component (but in the overall mixture the category may not be independent).
Note that n vectors of dimension p can fit up to np df: eg if we have just 1 observation on 583 variables we can fit model up to 1*583 df!
The Locally independent multinomial mixture model is defined as:

NB: utile a pensare due variabili di 4 categorie ciascuna, la distribuzione congiunta ha 16 slot di cui 15 riempibili a piacere

$$f_\eta(\mathbf{x}) = \sum_{k=1}^{K} \pi_k f_{\theta_k}(\mathbf{x})$$

where

- $\sum_{l=1}^{m_j} \zeta_{jlk} = 1, \ \forall j, k$, where $\zeta_{jlk}$is the probability that variable j takes the value c_l within mixture component number k; for every variable this probability has to sum up to 1 over all the categories (because everyone needs to pickup 1 category)

- $\sum_{k=1}^{K} \pi_k = 1$
- $\eta = (\pi_1, \ldots, \pi_K, \theta_1, \ldots, \theta_K)$ where θ_k is the set of ζ_{jlk} for varying j and l

With K groups the number of degrees of freedom is

$$(K-1) + K\sum_{j=1}^{p}(m_j - 1)$$

where

- $K-1$ is due to the π_k (which sums to 1)
- in every cluster, for every variable we have number of categories - 1 probabilities (think ζ_{jlk})

The number of df is still much smaller than $\left(\prod_{j=1}^{p} m_j\right) - 1$ for p, m_j big enough.

Remark 38. Note: Normally there are no distances between categories. Values are equal or different but not "close" or "far". Geometrical cluster intuition from Euclidean space doesn't work.
One could do MDS before and then go as well? but here we do not and work on original data

4.8.0.1 Illustrative (artificial) example

In marketing there's interest in finding groups of consumers defined by their product choices, e.g. for targeting advertising.
We ask people for:

- last purchased local food product that could be Parmigiano Reggiano, Mortadella Bolognese, Aceto Balsamico, None/Don't Know: $X_{i1} \in \{PR, MB, AN, N\}$, $m_1 = 4$
- last attended event which can be Sports, Music, Cinema, Political, None/Don't Know: $X_{i2} \in \{S, M, C, P, N\}$, $m_2 = 5$

Overall distribution ($4 \cdot 5 - 1 = 19$ df, parameter to be actually estimated) is in figure 4.8 (let's say these are real probability in the population). Here product and event are dependent (otherwise joint would coincide with product of marginals; here there are greater square in north east and south ovest part of the table, some values of one variable are more associated with some values of the other).
This actually is a mixture of two *locally independent* components with parameters as follows:

- Variable 1: PR, MB, AB, N,
  - Component 1 (3 df): $(\zeta_{111}, \ldots, \zeta_{141}) = (0.4, 0.4, 0.1, 0.1)$,
  - Component 2 (3 df): $(\zeta_{112}, \ldots, \zeta_{142}) = (0.2, 0.1, 0.4, 0.3)$,
- Variable 2: S, M, C, P, N,
  - Component 1 (4 df): $(\zeta_{211}, \ldots, \zeta_{251}) = (0.5, 0.2, 0.1, 0.1, 0.1)$,

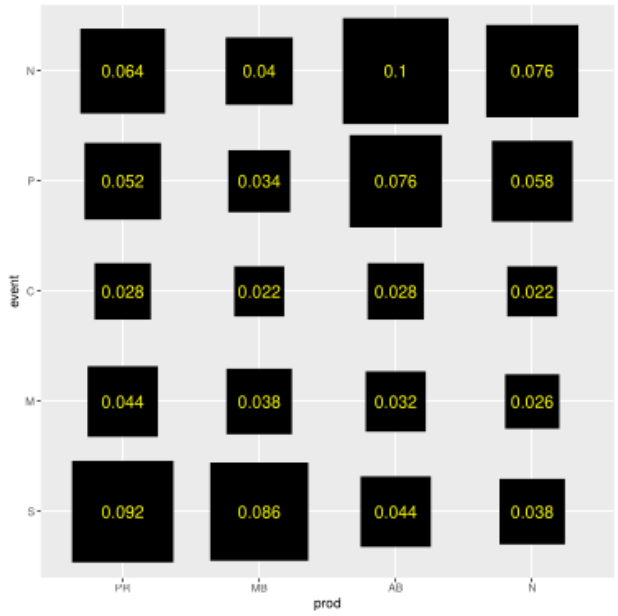


Figure 4.8: Overall distribution

– Component 2 (4 df): $(\zeta_{212}, \ldots, \zeta_{252}) = (0.1, 0.1, 0.1, 0.3, 0.4)$.

Mixing proportion $\pi_1 = 0.4$, $\pi_2 = 0.6$ (1 df). Thus overall 15 df (compared to 19 of the overall distribution); written down as local independent we writed down the model in a more parsimonious way.
The two locally independent component can be visualized as

- Cluster 1 with *independence* (strong Sports, PR, MB) is in fig 4.9 (vertical proportion are proportional: the ratio of probability within column is kept constant). Computation e.g. for entry (MB,C): $\zeta_{121}\zeta_{231} = 0.04$.

- Cluster 2 with *independence* (strong Politics, AB, N) in fig 4.10 (where vertical proportion are proportional: the ratio of probability within column is kept constant and knowing values of one variable does not give hint on the possible value of the other.
  Computation e.g. for entry (AB,N): $\zeta_{132}\zeta_{252} = 0.16$.

- The final mixture is determined as $0.4f_{\theta_1} + 0.6f_{\theta_2}$ and already presented in fig **??**.
  There, the entry (PR,N): $0.4\zeta_{111}\zeta_{251} + 0.6\zeta_{112}\zeta_{252} = 0.064$.
  The two component/cluster are *defined* by local independence. Overall distribution is "decomposed" into locally independent components.

Important remark 44 (Pros/cons). We have:

- **Advantage**: Clusters can be characterised/interpreted by marginal distributions. Parsimonious way of representing the distribution.

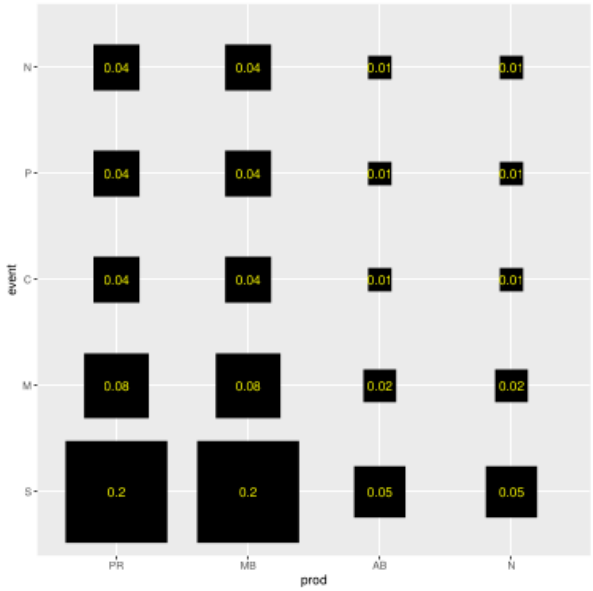


Figure 4.9: First component

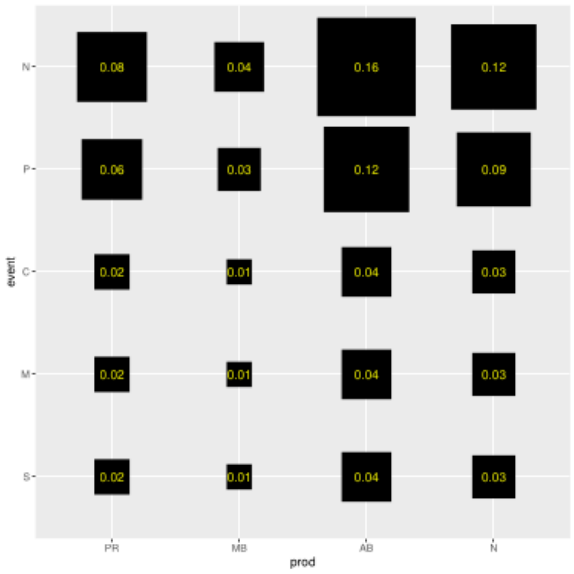


Figure 4.10: Second component

- **Disadvantage**: Not guaranteed that objects within clusters are "similar" (similarity in "simple matching" sense if marginal distributions are concentrated) ... but often these clusters are very homogeneous.

Remark 39. Categorical mixture models are often called "Latent Class Analysis" in the literature, see Celeux and Govaert (2015).

Remark 40. Cosi out of the blue dice che nei mixture se vogliamo classificare una osservazione a un cluster o all'altro di fatto stimiamo il modello e i suoi parametri, dopodiché stimiamo la probabilità a posteriori che di avere una unità del gruppo tal dei tali, dato il suo valore, l'unità al gruppo con la probabilità maggiore

4.8.0.2 Fit with ML, EM-algorithm

EM works like in the Gaussian mixture model.
The complete loglikelihood with Z_i known:

$$L_{n,K,c}(\eta) = \sum_{i=1}^{n}\sum_{k=1}^{K} 1(Z_i = k)(\log \pi_k + \log f_{\theta_k}(\mathbf{x}_i))$$

In the

- **E-step** we have always the same, we compute the expected value of complete likelihood given that we already have a parameter vector (maybe the initialization one)

$$E_{\eta_{t-1}}(L_{n,K,c}(\eta)|T = \tilde{\mathbf{x}}) = \sum_{i=1}^{n}\sum_{k=1}^{K} P(Z_i = k|\eta_{t-1}, \mathbf{x}_i)(\log \pi_k + \log f_{\theta_k^{(t-1)}}(\mathbf{x}_i)),$$

$$p_{ik}^{(t-1)} = P(Z_i = k|\eta_{t-1}, \mathbf{x}_i) = \frac{\pi_k^{(t-1)} f_{\theta_k^{(t-1)}}(\mathbf{x}_i)}{\sum_{h=1}^{K} \pi_h^{(t-1)} f_{\theta_h^{(t-1)}}(\mathbf{x}_i)}$$

  where p_{ik} is the probability that i-th observation comes from the k-th component. This (for multivariate t, skew normal and skew t) is the same as previous (multivariate gaussian)

- **M-step**: we maximise the following by choosing parameters

$$\sum_{i=1}^{n}\sum_{k=1}^{K} p_{ik}^{(t-1)}(\log \pi_k + \log f_{\theta_k}(\mathbf{x}_i))$$

  We can actually maximise separatedly (even for general mixtures) the first and second addend

$$\sum_{i=1}^{n}\sum_{k=1}^{K} p_{ik}^{(t-1)} \log \pi_k \implies \pi_k^{(t)} = \frac{1}{n}\sum_{i=1}^{n} p_{ik}^{(t-1)}$$

$$\sum_{i=1}^{n}\sum_{k=1}^{K} p_{ik}^{(t-1)} \log f_{\theta_k}(\mathbf{x}_i)$$

The mixing proportions are always estimated the same; what is different is the maximization of the second addend (in the gaussian here we had a weighted mean and weighted covariance matrix for each cluster, with weights being $p_{ik}^{(t-1)}$).
Qui out of the blue (prima non abbiamo visto sta roba per skew-t etc): in general estimation of parameters for multivariate t, skew normal and skew t is much more complicated and there are no close formula for that (so numerical approximation in needed).
But for the categorical data this is fairly simple and yields (for locally independent latent class model) weighted ML estimators for the marginal probabilities $(\zeta_{jlk})_{j\in\mathbb{N}_p, l\in\mathbb{N}_{m_j}, k\in\mathbb{N}_k}$:

$$\zeta_{jlk}^{(t)} = \frac{1}{\sum_{i=1}^n p_{ik}^{(t-1)}} \sum_{i=1}^n p_{ik}^{(t-1)} x_i^{jl}$$

Remembering that ζ_{jlk} is the probability that variable j take the value category l in the k-th mixture component.
If one want to estimate probability in a (say discrete) joint distribution what does is to estimate the relative frequency of the event/cell. So basically estimates are frequency; if we assume indipendence between the two/more variable we wouldn't estimate separate combination probability of each cell, we would estimate the two marginal distribution and all the joint distribution it would come from product. (With single local independent distribution ($k = 1$) is very simple to estimate these probabilities).
Here what we do is similar to what seen with gaussian but

- here we use as weight the probabilities $p_{ik}^{(t-1)} = P(Z_i = k|\eta_{t-1}, \mathbf{x}_i)$
- the weighted stuff is $x_i^{jl} = 1(x_{ij} = c_l^j)$ which is the indicator that for i-th observation, the j-th variables assumes its l-th category, so actually works as selector
  Cycling on units...

NB: boh incomprensibile for now 18 novembre poco prima della pausa

So in a sense here we end with a weighted relative frequency within cluster k of category l on variable j.

So this define the EM algorithm (which still needs to start with parameters and then)

Remark 41. One can define mixtures for *mixed-type* continuous and categorical data by assuming local independence between categorical and continuous variables, modelled by multinomial and Gaussian distributions.

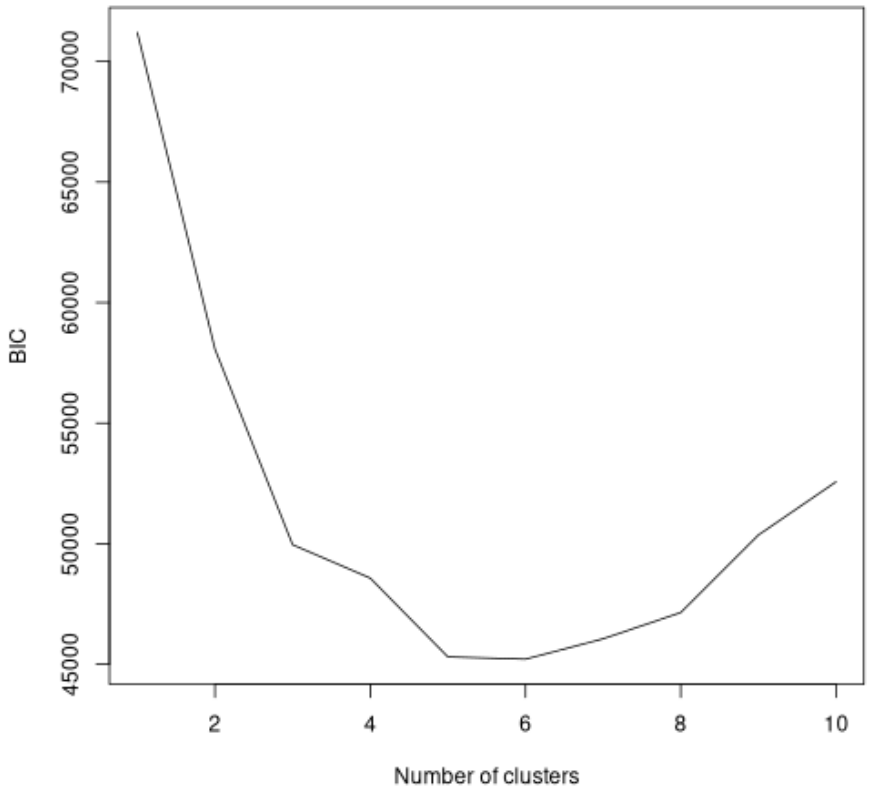
Remark 42. Could have even more parsimonious models if the number of df $(K-1)+K\sum_{j=1}^p(m_j-1)$ is still too many (and identifiability needs p not too small), see Celeux and Govaert (2015).

4.8.0.3 Example on veronica

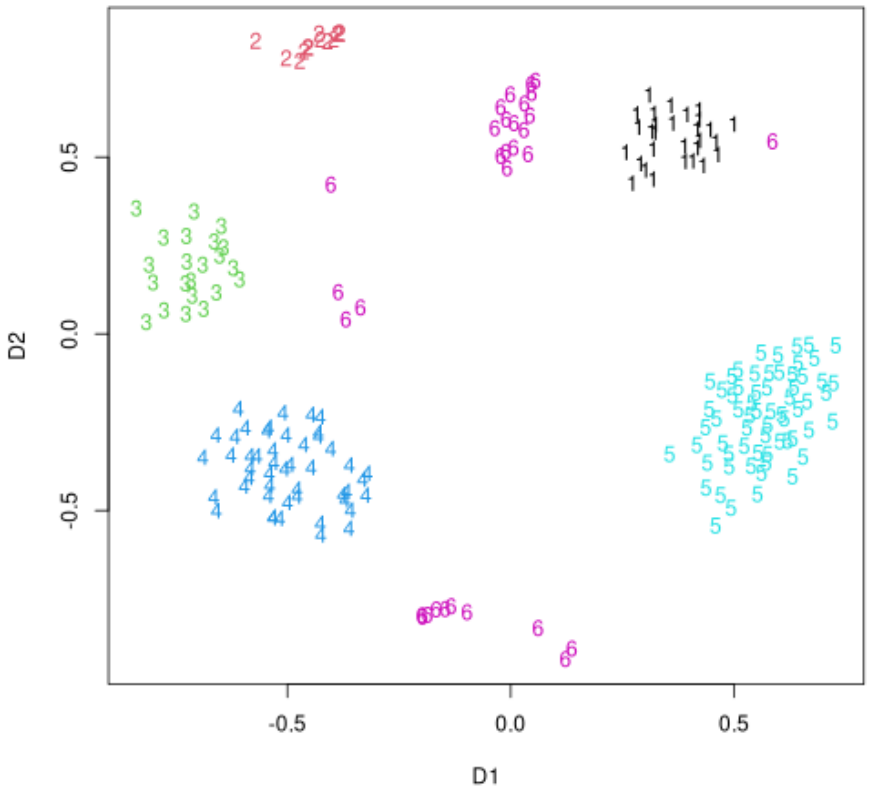
Example 4.8.1 (Veronica)**.** `flexmix::flexmix` offers a flexible EM-algorithm for mixture model; users can write methods for specific models/specific EM algorithm. For doing this it's needed to implement how to compute the component estimator for single distribution (which is complicated).

We use `fpc::flexmixedruns` (che se non ho capito male è un wrapper a `flexmix`) is a method for latent class mixture that can also fit mixed type continuous/categorical data. If one has only categorical variable it has to specify

```
set.seed(887766)
veronica <- read.table("data/veronica.dat")
veronicam <- as.matrix(veronica)
tmp <- capture.output(
  veronicabernm <- fpc::flexmixedruns(veronica,
                                      continuous = 0, # there are 0 continuous variable
                                      discrete = 583, # and all 583 var are discrete
                                      n.cluster = 1:10) #n of cluster fitted
  # - Default assumption: continuous variables come first, therefore
  #   continuous=0, discrete=583 - all variables are categorical.
  # - with xvarsorted=FALSE (variable type not sorted) one can specify what
  #   kind of variable is where
  # - Uses by default 20 random initialisations (similar to k-means)
)

# bicvals output component are the BIC values. we see that k=5, 6 component
# are the best: ``Small is good'' version of BIC
plot(1:10, veronicabernm$bicvals, typ="l", xlab="Number of clusters", ylab="BIC")
```




```
# mds for plotting in 2d
jveronica <- dist(veronica, method = "binary") # jaccard
mdsveronica <- smacof::mds(jveronica)

## if we know number of cluster are 6 (loking or which.min-ing at bic)
## we take veronicabernm$flexout[[6]]@cluster for cluster label
clust <- veronicabernm$flexout[[6]]@cluster # S4
head(clust)

## [1] 4 4 4 3 2 5

plot(mdsveronica$conf, col = clust, pch = fpc::clusym[clust])
```

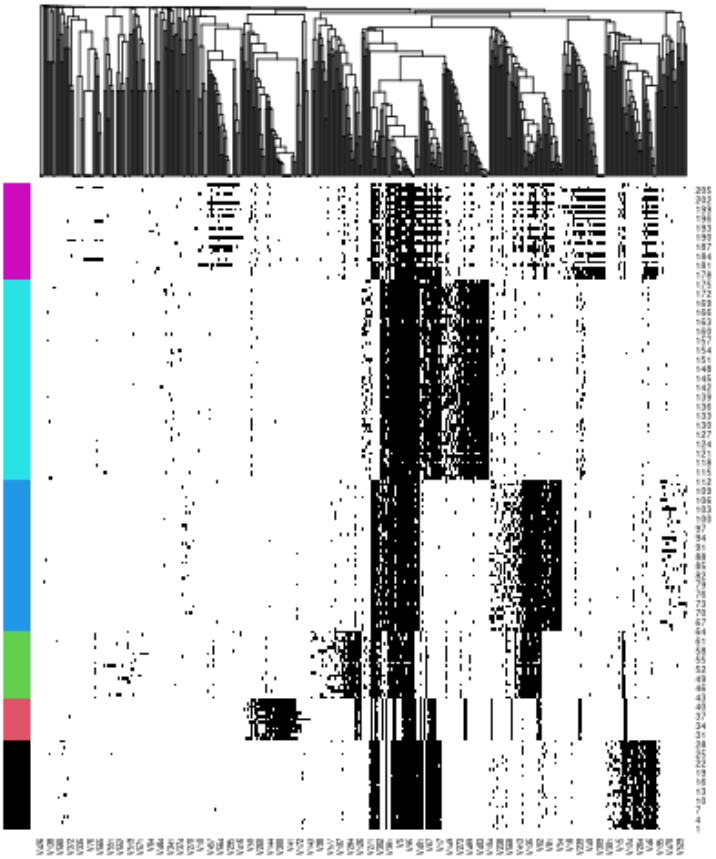



```
## cluster from 1 to 5 seems ok, while cluster 6 appear to be strange
## toghether while in lower dimension plot is separated. hard to tell if 6 is
## a good clustering

## access parameter estimators in the 6-th element of flexout
str(veronicabernm$flexout[[6]], max.level = 2) # all the output from flexmix

## Formal class 'flexmix' [package "flexmix"] with 18 slots
##   ..@ posterior  :List of 2
##   ..@ weights    : NULL
##   ..@ iter       : int 4
##   ..@ cluster    : int [1:207] 4 4 4 3 2 5 6 1 4 3 ...
##   ..@ logLik     : num -13361
##   ..@ df         : num 3467
##   ..@ control    :Formal class 'FLXcontrol' [package "flexmix"] with 6 slots
##   ..@ group      : Factor w/ 0 levels:
##   ..@ size       : Named int [1:6] 29 13 22 48 64 31
##   .. ..- attr(*, "names")= chr [1:6] "1" "2" "3" "4" ...
##   ..@ converged  : logi TRUE
##   ..@ k0         : int 6
```

```
##   ..@ model      :List of 1
##   ..@ prior      : num [1:6] 0.1401 0.0628 0.1063 0.2319 0.3092 ...
##   ..@ components :List of 6
##   ..@ concomitant:Formal class 'FLXP' [package "flexmix"] with 7 slots
##   ..@ formula    :Class 'formula'  language x ~ 1
##   .. .. ..- attr(*, ".Environment")=<environment: 0x5649a632f778>
##   ..@ call       : language flexmix(formula = x ~ 1, k = k, cluster = initial.clus
##   ..@ k          : int 6

# - posterior should be probability of all the observation to belong to all the
#   cluster
# - iter number of iteration before optimum was found
# - cluster is the clustering vector
# - df are actually the number of parameters estimated. in veronica we have 207
#   observation and 583 so we can fit up to 207*583=120681 so no problem if df>n.
#   Difference with regression (where n has to be > p) we have a model that
#   say that Y is random and X are fixed: we only have 1 information about
#   random process, we don't model the distribution of X which are
#   fixed. Here OTOH there's more information about random process than regression
# - size are the sizes of the cluster
# - prior should be component proportion
# - components contain parameters estimates
# - k0 is the number of component we started with k is the final number of
#   components (can be the case that are different)

## Marginal proportions within cluster of categories are in the slots components
## [[5]] is for looking at the 5-th component  (of the 6 cluster solution)
## [[1]] s'à da fare per scegliere la struttura dati dove sono i parameter

head(veronicabernm$flexout[[6]]@components[[5]][[1]]@parameters$pp)

## [[1]]
## [1] 1 0
##
## [[2]]
## [1] 0.03125 0.96875
##
## [[3]]
## [1] 1 0
##
## [[4]]
## [1] 1 0
##
## [[5]]
## [1] 0 1
##
## [[6]]
## [1] 0.984375 0.015625

## very long list: it has 583 elements (one per variable), each with two
```

```
## probability
## we consider the take the first for example
## it gives for every variable within component 5 the parameter values
## are actually the probabilities for 0 and for 1 (levels of variable)
## the first element is first variable of the fifth component the
## probability of absence and probability of presence
## in most variable observation are either on one side (so lot of 0 and 1)
## but in some variable observation are divided
## is required to find the parameters, but otherwise doesn't mean anything)
```

```
# heatmap with rows ordered by clusters found
# and columns by earlier variable clustering

# variable clustering as done before
vveronica <- dist(t(veronicam), method = "binary")
varclust <- hclust(vveronica, method = "average")

# clust is the clustering solution appena trovata
rowcol <- palette()[clust][order(clust)] # color for each row
heatmap(veronicam[order(clust),], # ordering of rows according to clustering
        Rowv = NA, # not a dendrogram for rows
        RowSideColors = rowcol, # but in rows i want colors
        Colv = as.dendrogram(varclust),
        col = c(0, 1),
        scale = "none")
```



Cluster are pretty good there are some clear patterns, presences of variables within cluste. This can be used to interpret the cluster
The cluster n 6, which were caotic in the plot before (is the first plotted) is sparse compared to other cluster which make sense.

How does local independence (Variable within cluster are independent) look like? Is it fulfilled here?

problem with cluster 6 is also problem with local independence

if the variable are all independent within cluster 6, what happens in one column is not informative (within the cluster) to what happens in other column here otoh variables seems to be linked (presence/absence) within cluster and there should be a problem of local independence

NB: chiarissimo come al solito, lezione prima del 21 novembre qui

Method	Data	Cluster concept	clustering type	number of clusters?
k-means	Euclidean	homogeneous, spherical	partition	gap, ASW
Single Linkage	dissimilarities	separated	hierarchy	ASW
Average Linkage	dissimilarities	homogeneous, separated	hierarchy	ASW
Complete Linkage	dissimilarities	homogeneous	hierarchy	ASW
Ward	Euclidean	homogeneous, spherical	hierarchy	gap, ASW
pam	dissimilarities	homogeneous	partition	ASW
Gaussian mixture	Euclidean	elliptical	probabilistic	BIC
t mixture	Euclidean	elliptical, heavy tails	probabilistic	BIC
skew Gaussian mixture	Euclidean	skew	probabilistic	BIC
skew t mixture	Euclidean	skew, heavy tails	probabilistic	BIC
Latent class mixture	categorical	local independence	probabilistic	BIC

Table 4.1: Overview Tables on Clustering methods

Dissimilarities	Data	Distance?	Characteristic
Euclidean L_2	Euclidean	yes	Euclidean geometry, rotation invariant
Manhattan L_1	Euclidean	yes	all variables same weight
Maximum L_∞	Euclidean	yes	maximum distance variable
Mahalanobis	Euclidean	yes	affine invariant (incl. scaling)
Simple Matching	categorical	yes	all categories treated equally
Jaccard	binary	yes	dominated by presences not absences
Correlation (1)	variables	no	max. dissimilarity $\rho = -1$
Correlation (2)	variables	no	max. dissimilarity $\rho = 0$
Gower	mixed type	no (with missings)	Manhattan aggregation, missing values

Table 4.2: overview distances

Method	Data	Task
Principal Components	Euclidean	dimension reduction, visualisation
Data scaling	Euclidean	making variables comparable
Multidimensional Scaling	dissimilarities	maps to Euclidean, visualisation
Adjusted Rand Index	clusterings	similarity between clusterings
Pairs plot	Euclidean	visualisation
Heatplot	general (shown for binary)	visualisation
Parallel coordinates plot	mostly Euclidean, time series	visualisation
gap, ASW, BIC results plots	clustering indexes	visualisation
Dendrogram	clustering hierarchy	visualisation
Silhouette plot	pointwise silhouette widths	visualisation

Table 4.3

Part II

Assignments