Modern statistics

September 9, 2025

Contents

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.

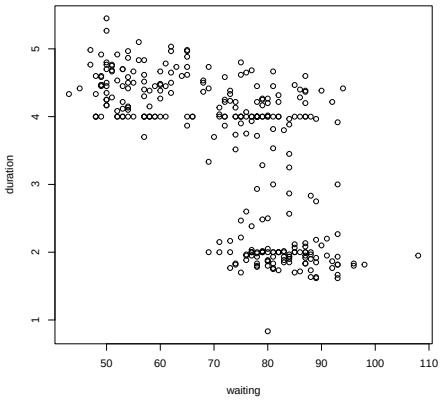


Figure 1.1: Old faithful geyser data

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.

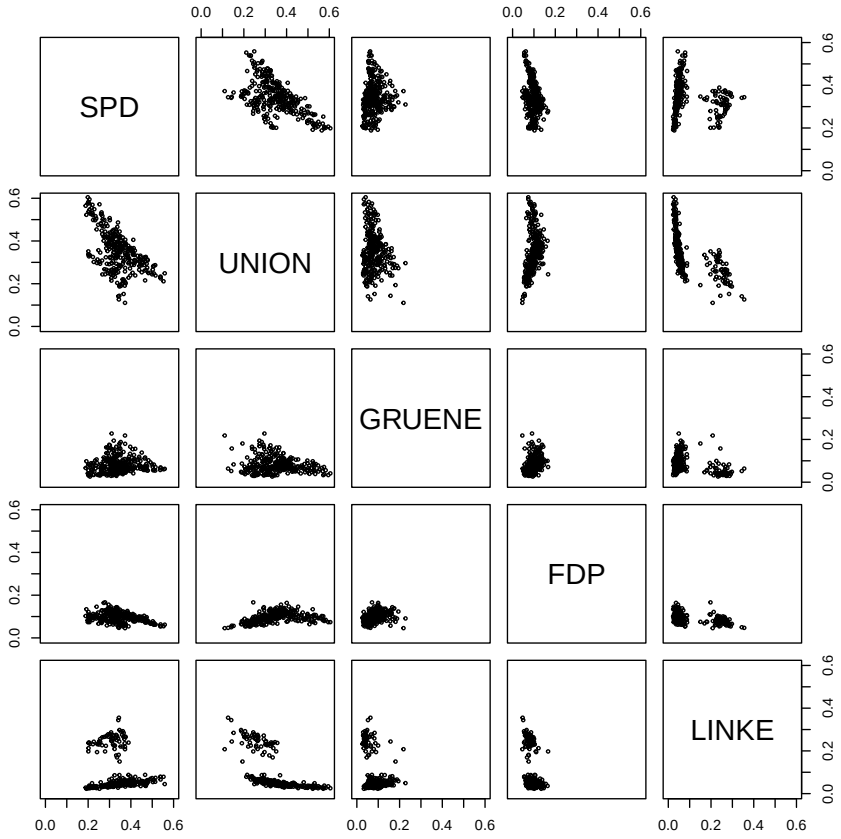
```
library(MASS)
data(geyser) # also available in read.table("data/geyser.dat", header = TRUE)
str(geyser)

## '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.
```

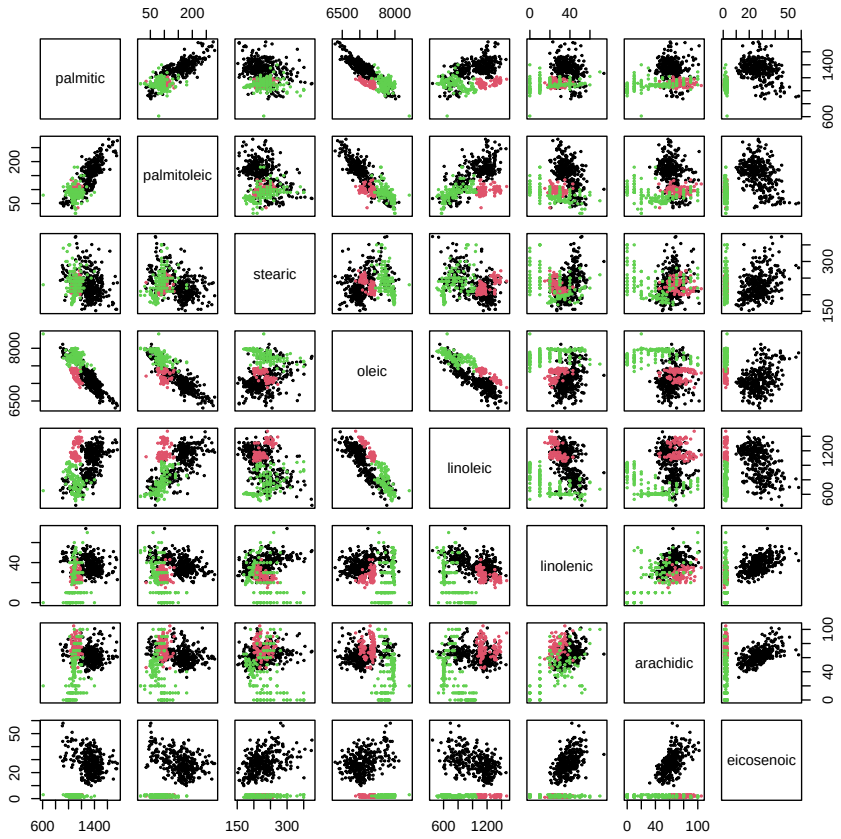
Il dataset è stato generato così

```
## ## 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"
```

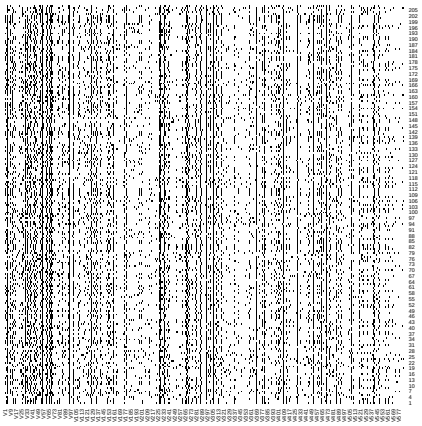


1.1.1.4 Veronica

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")
```



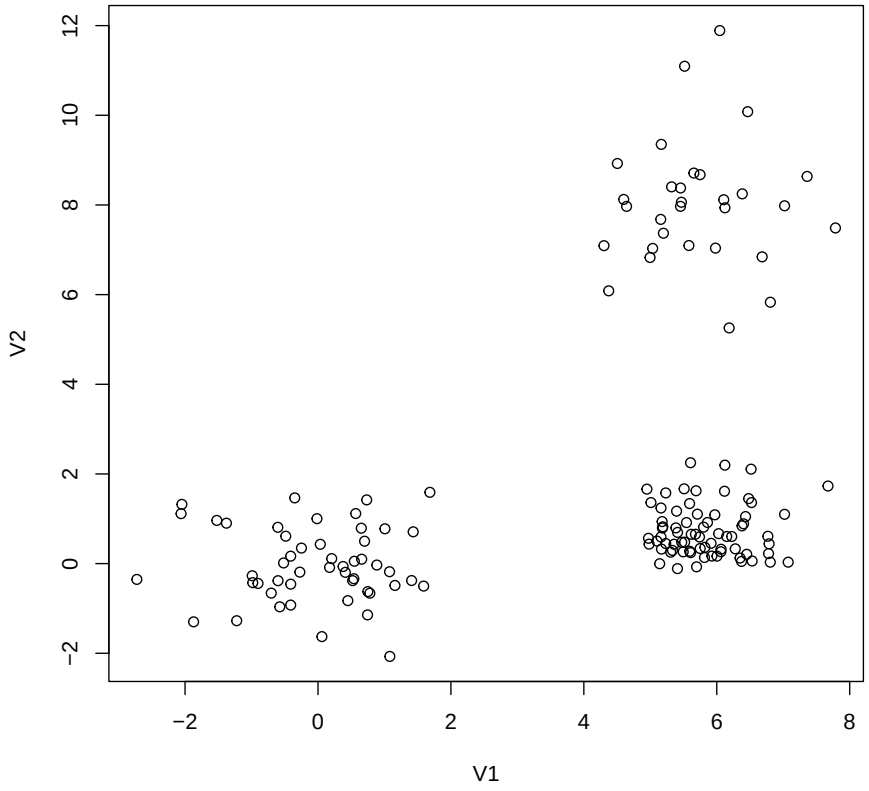
The 0/1 variable are not informative for creation of species/groups/cluster. This plot of all data is fairly uselesss (is like heatmap)

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.
```



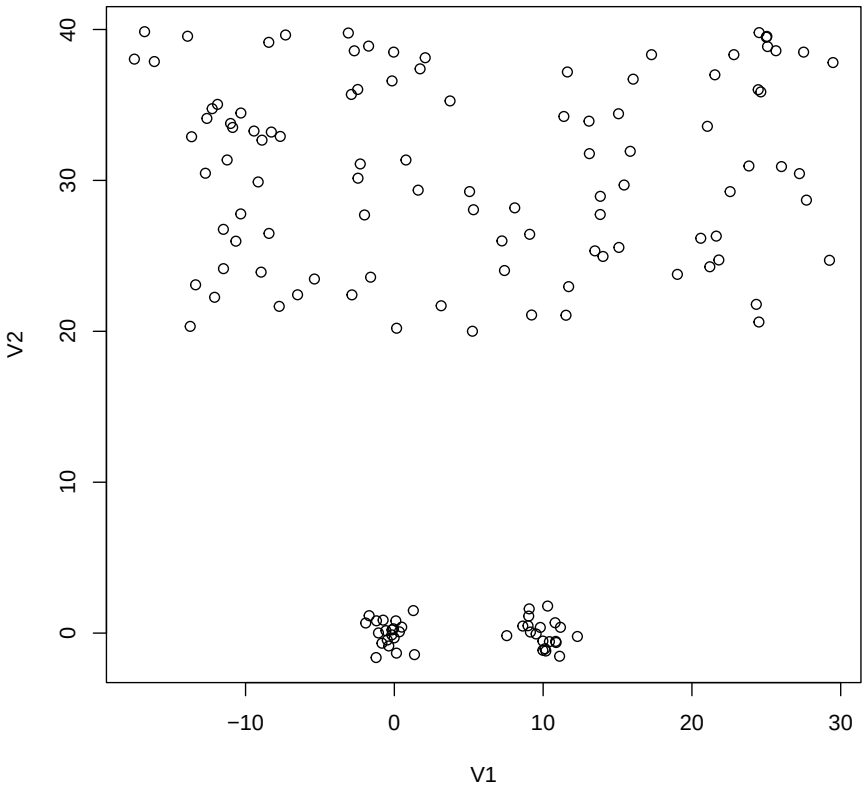

```
## This is how data set was generated.

## 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
```

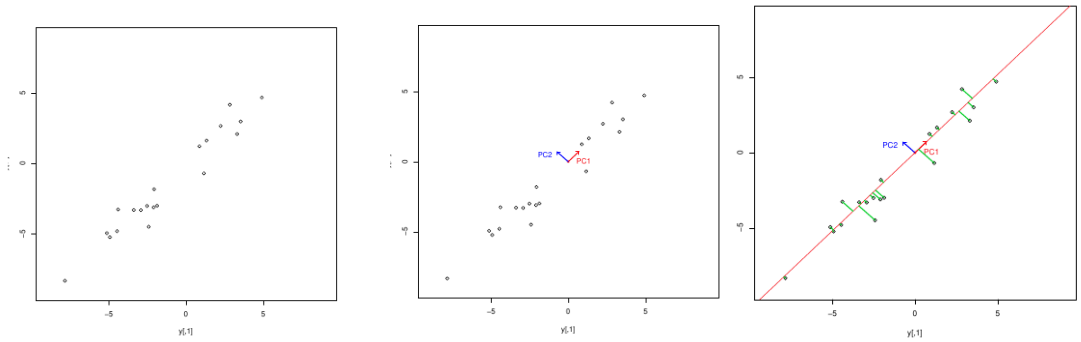


In this case the three cluster are characterized by different variability (two are strict, the above one has a lot of variability inside and some units are nearer to the cluster bolow than to units at the opposite side of the cluster).

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:

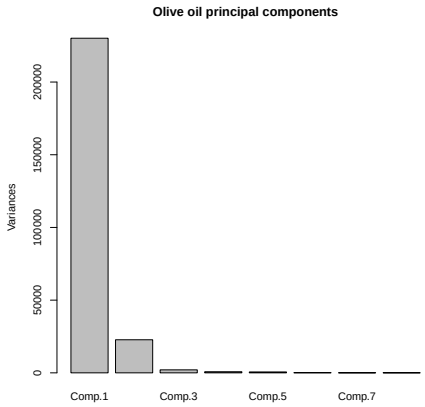
- D containing Principal Components (eigenvectors of Σ)
- Δ diagonal matrix of ordered eigenvalues (PC variances).

For standardised variables, Σ is the correlation matrix.

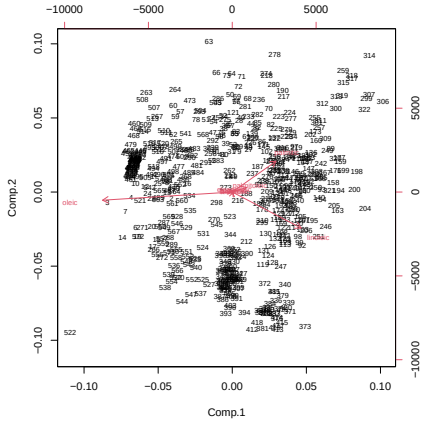
Important remark 4 (Problem with PCA). Are the Principal Components really the most informative dimensions? PCA identifies "information" with large variance. This is not necessarily appropriate; there is no guarantee that these dimensions are most informative for clustering or regression. However it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

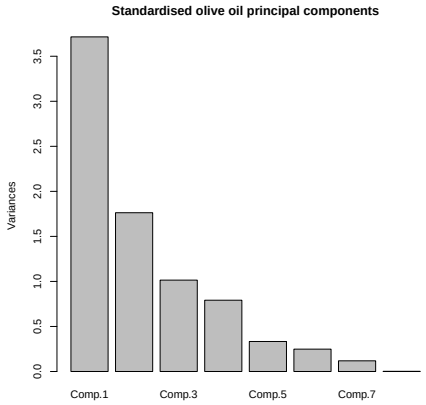
```
sapply(olive, sd)

##    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).

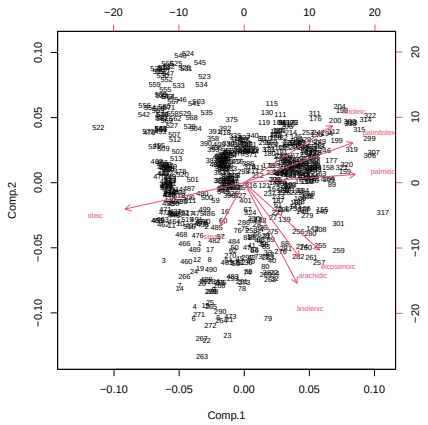
```
solive <- scale(olive) ## Standardise data
sprolive <- princomp(solive) ## PCA
summary(sprolive) # variance is more equally redistributed

## 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")
```



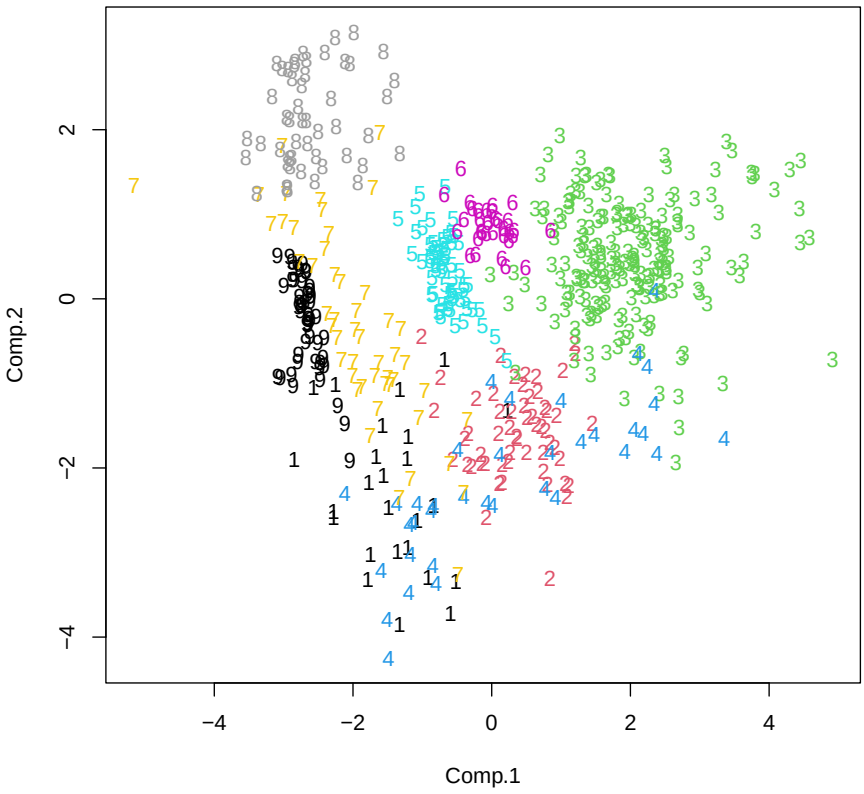

```
biplot(sprolive, cex = 0.7) # biplot
```



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

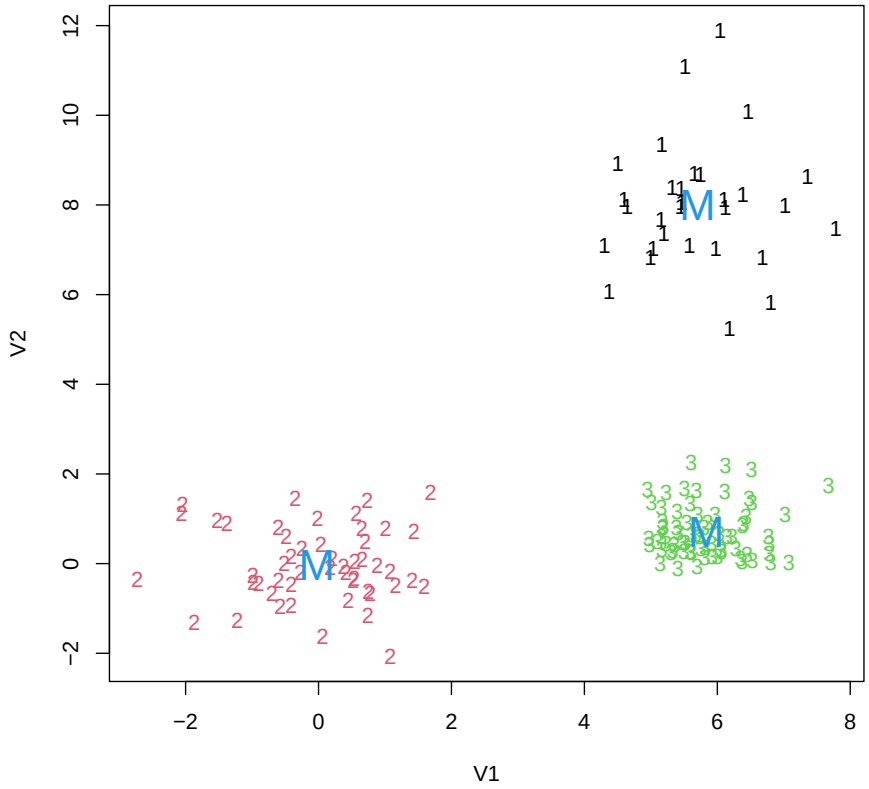
`kmeans` is used on the data.frame of observation:

- `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"
```

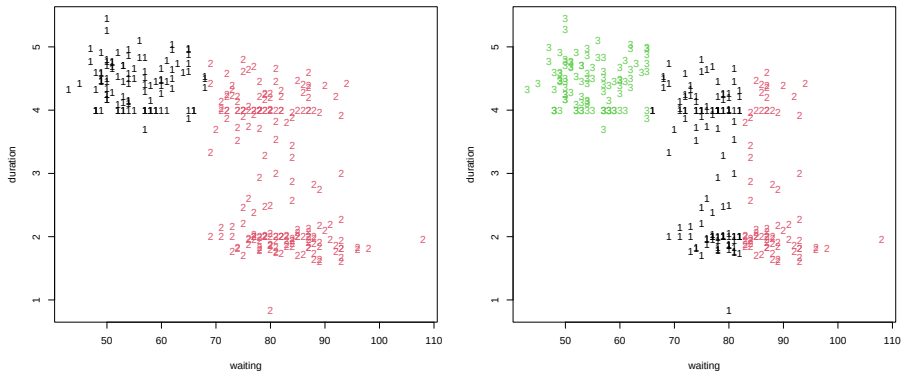
we have

- `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

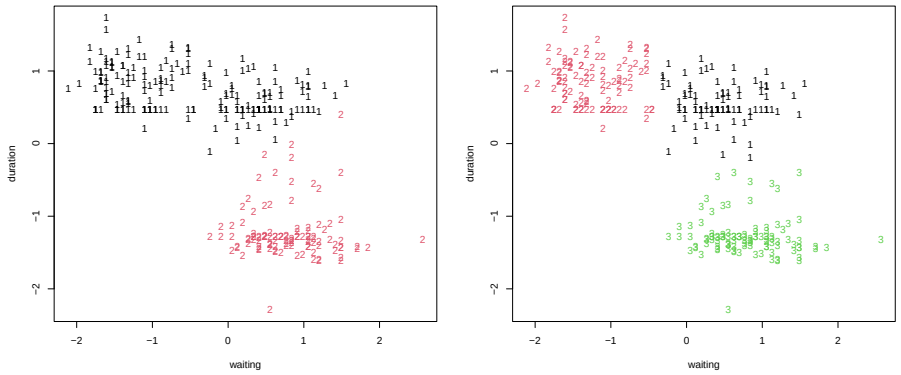
Example 1.3.2 (old faitful geyser)**.** Applying both 2-means and 3-means cluster to geyser data without scaling at first (very different)

```
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])
```

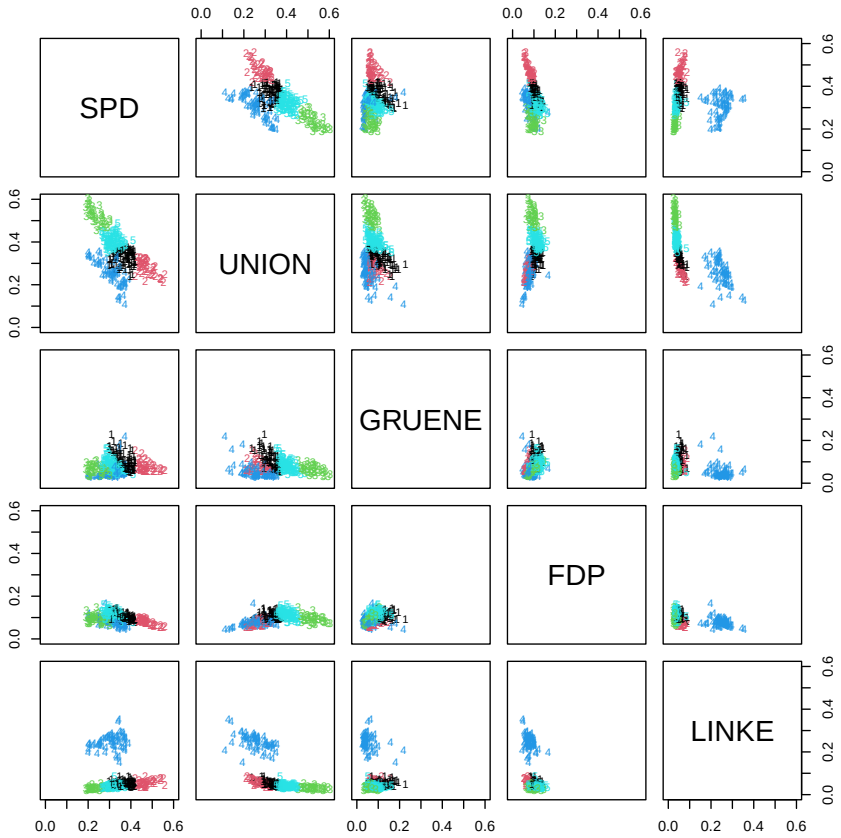


Important remark 8 (Scale equivariance and scaling). K-means is not *scale equivariant*: multiply variables by different constants and K-means result will change.
Scaling gives all variables same impact: scaling is not always good if all variables have same measurement unit and lower variation means 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:

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

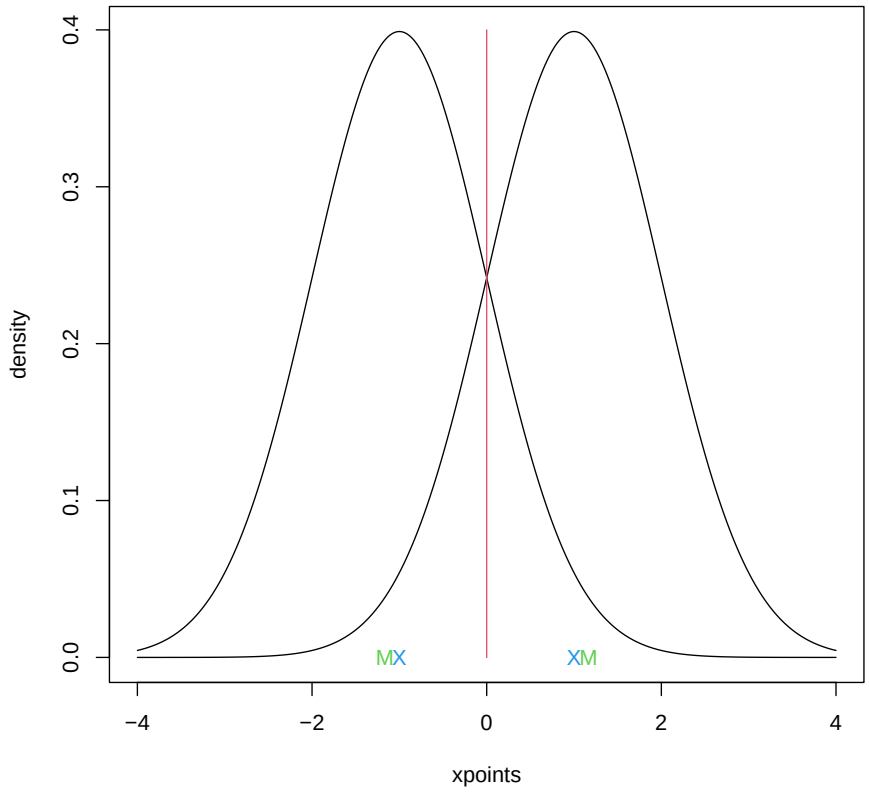


Figure 1.3: K-means is inconsistent

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.

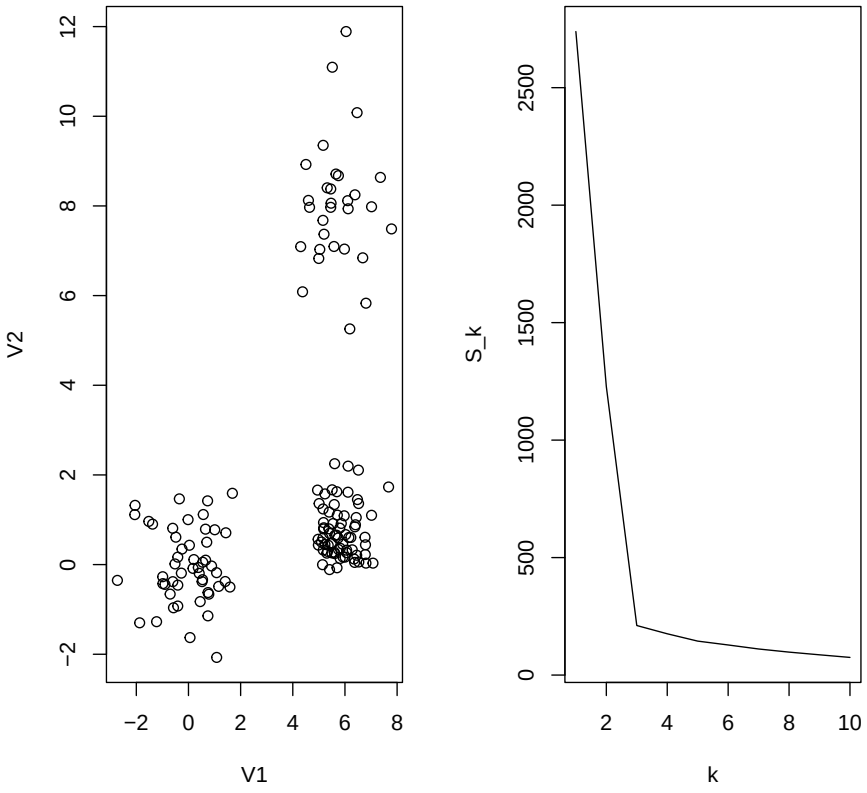


Figure 1.4: The elbow method

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:

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

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

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"

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

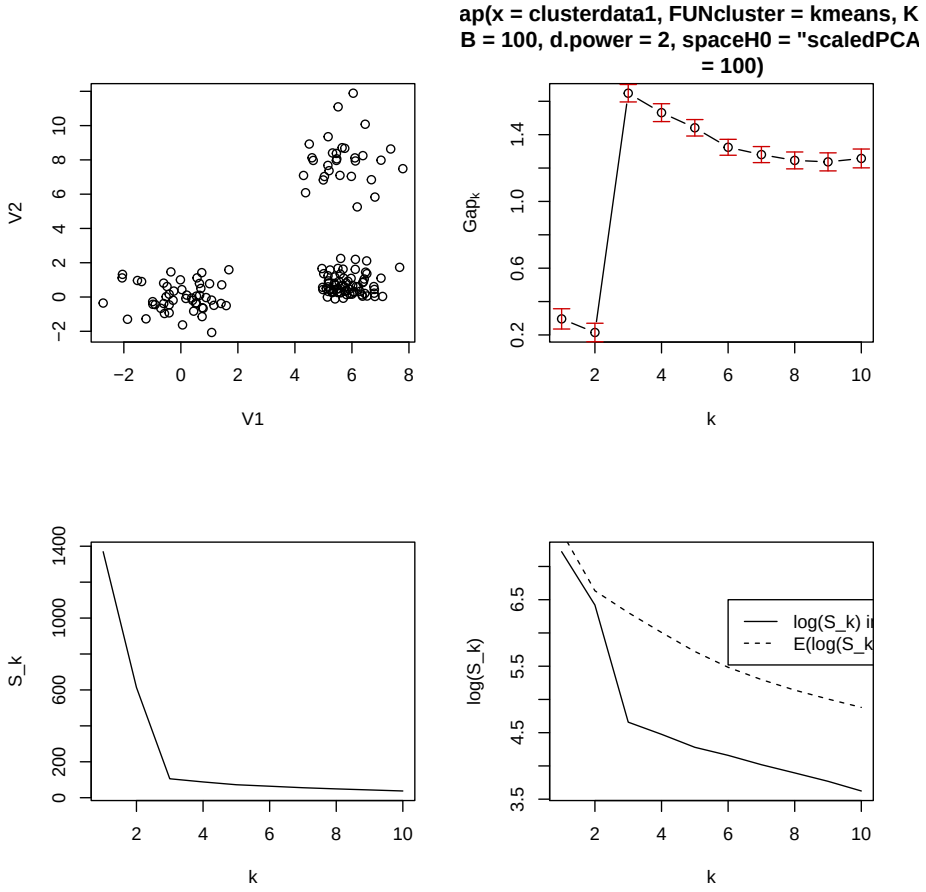
(in original paper with $q = 1$, but this can be set to 2). R-function `clusGap` in `cluster` offer all this.

[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)
```



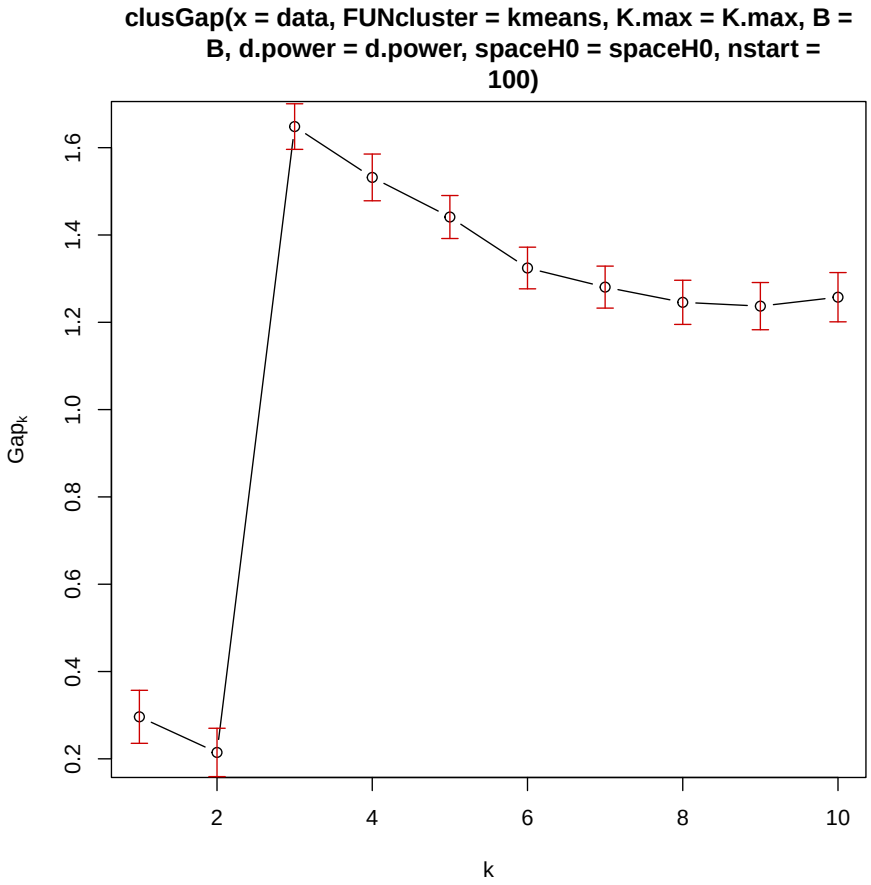
Remark 13. The `clusGap` function does not directly give out the optimal number of clusters K_0, and neither the resulting optimal clustering.
For extracting K_0 need to run `maxSE`, which operates on table entries of `clusGap` output. For the optimal clustering, need to re-run `kmeans` with optimal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

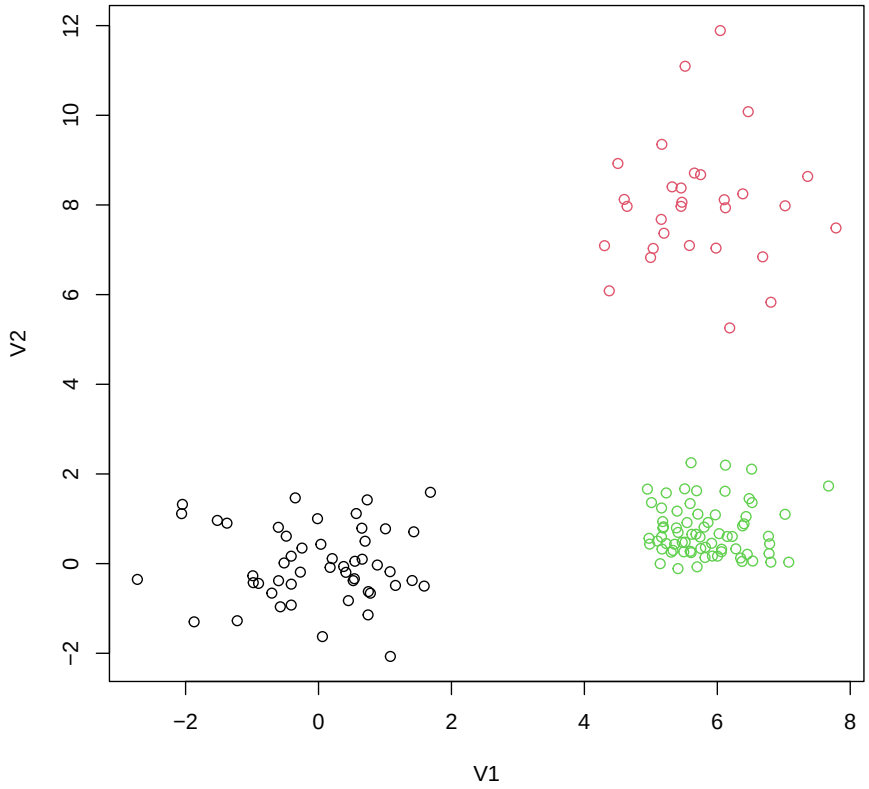
Example 1.3.5. The usage of gapnc with clusterdata

```
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:
```

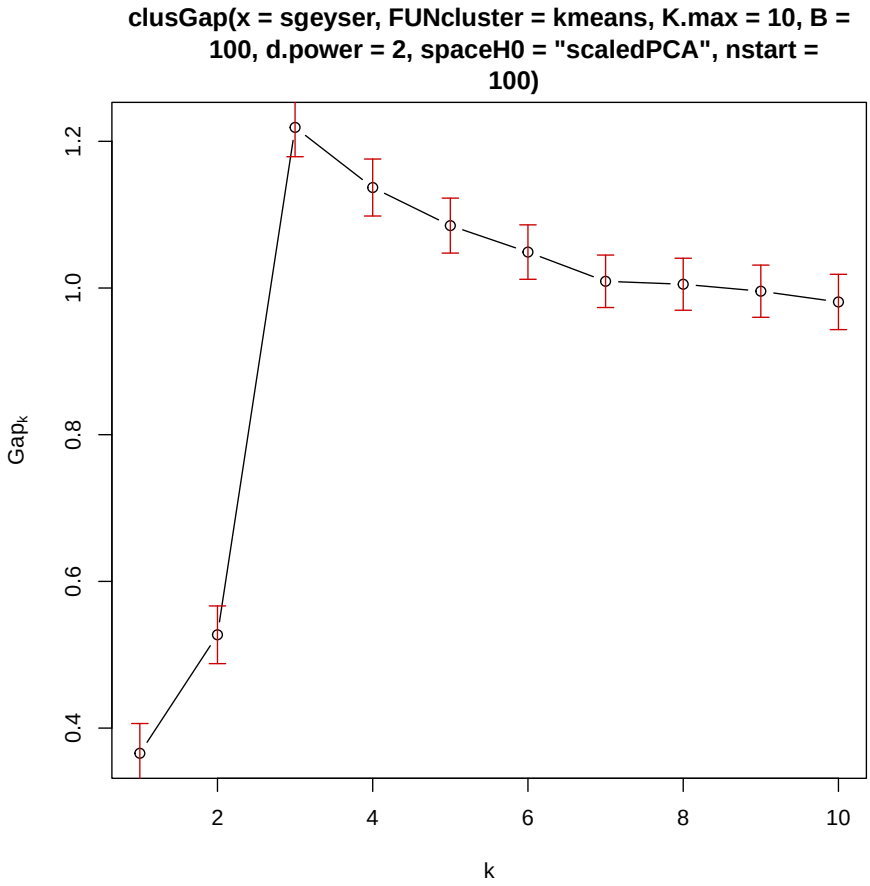
```
## clusGap(x = data, FUNcluster = kmeans, K.max = K.max, B = B, d.power = d.power, spaceH0 = sp
## B=100 simulated reference sets, k = 1..10; spaceH0="scaledPCA"
##  --> Number of clusters (method 'firstSEmax', SE.factor=1): 1
##          logW   E.logW       gap     SE.sim
##  [1,] 7.222172 7.518419 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
```



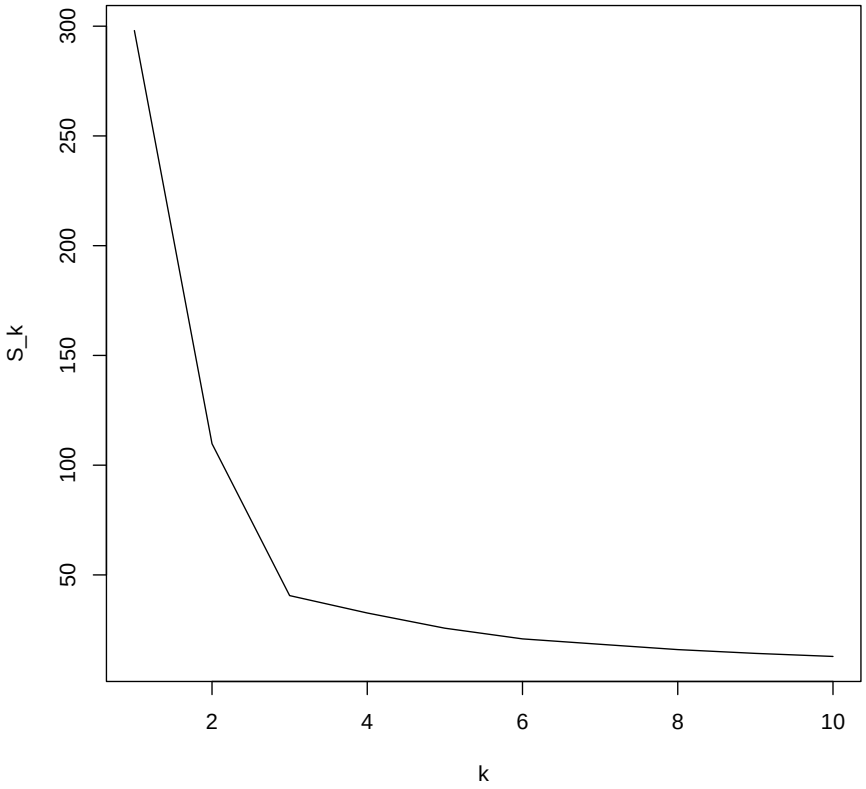
Example 1.3.6 (Geyser)**.** Other example of `gapnc` (non stampato l'output è la stessa roba di prima a vari pezzi, si sceglie chiaramente la soluzione a $k = 3$)

```
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
```

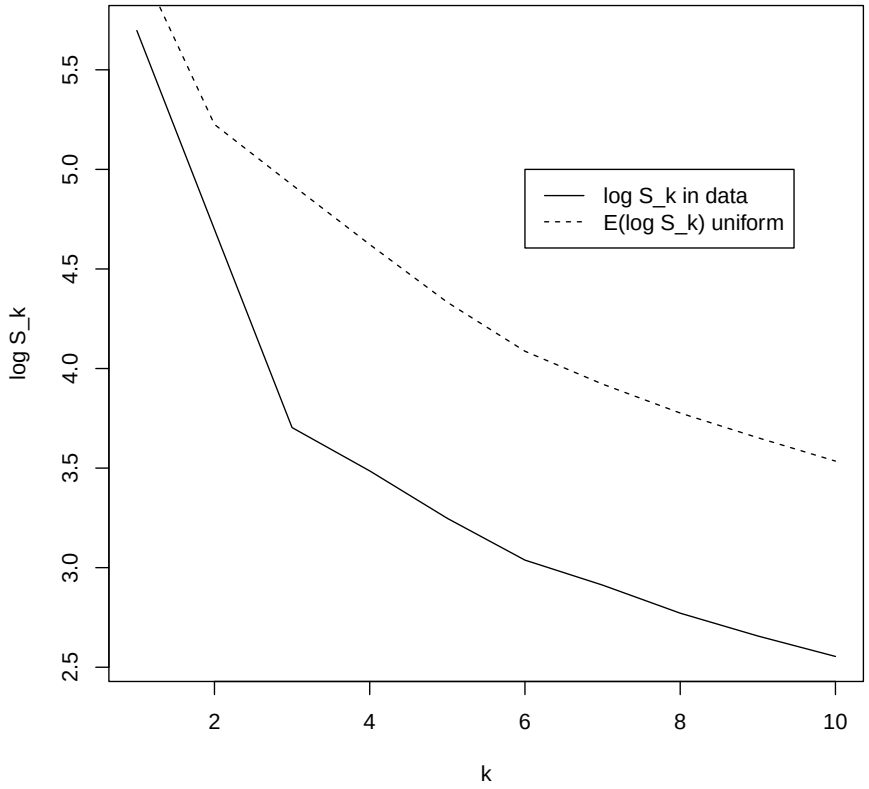
```
## --> Number of clusters (method <U+2019>globalSEmax<U+2019>, SE.factor=2): 3
plot(cgg)
```



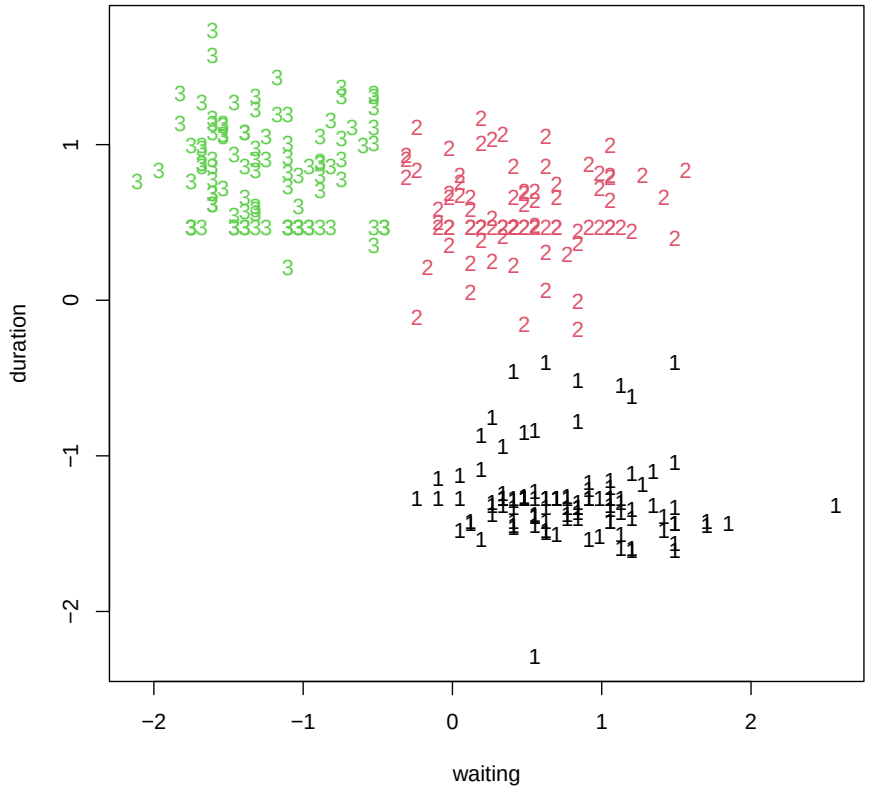

```
## Values of gap
plot(1:10,exp(cgg$Tab[,1]),xlab="k",ylab="S_k",type="l")
```




```
## 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])
```

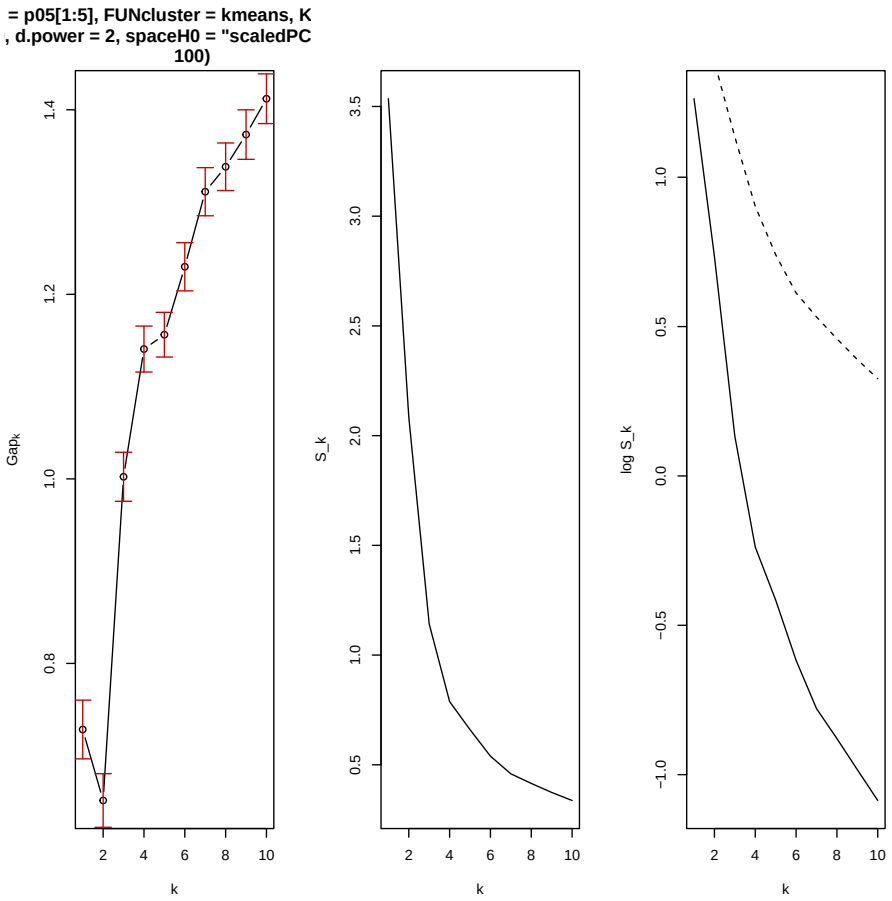



```
## Alternatively:
cgg2 <- gapnc(sgeyser,nstart=100)
```

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

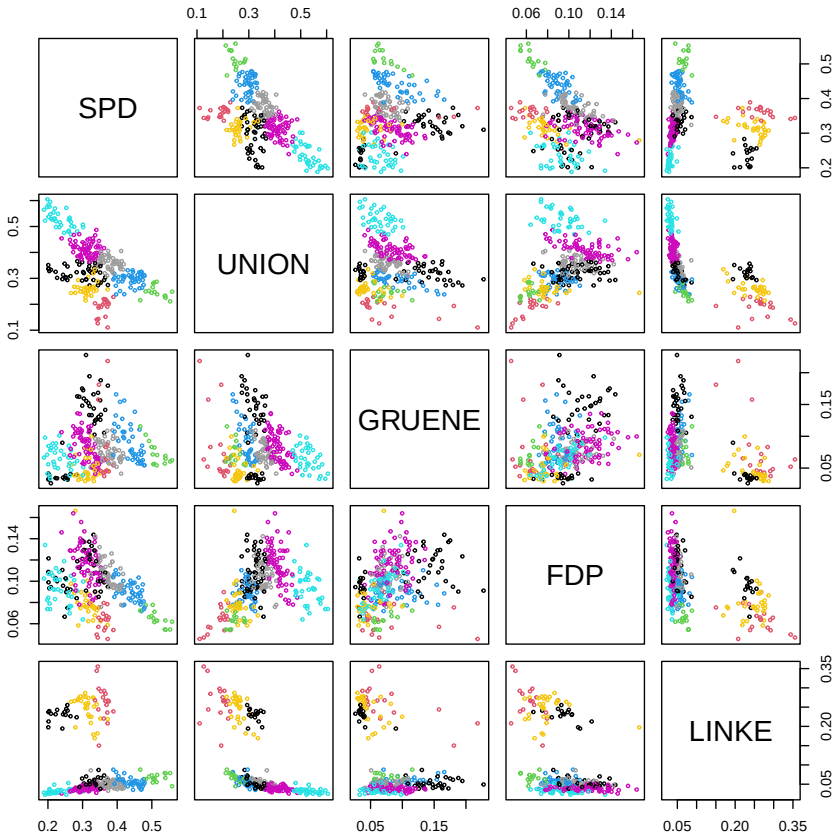
```
cgp05 <- clusGap(p05[1:5],kmeans,,K.max=10,B=100,d.power=2,spaceH0="scaledPCA",nstart=
print(cgp05,method="globalSEmax",SE.factor=2)

## 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*).

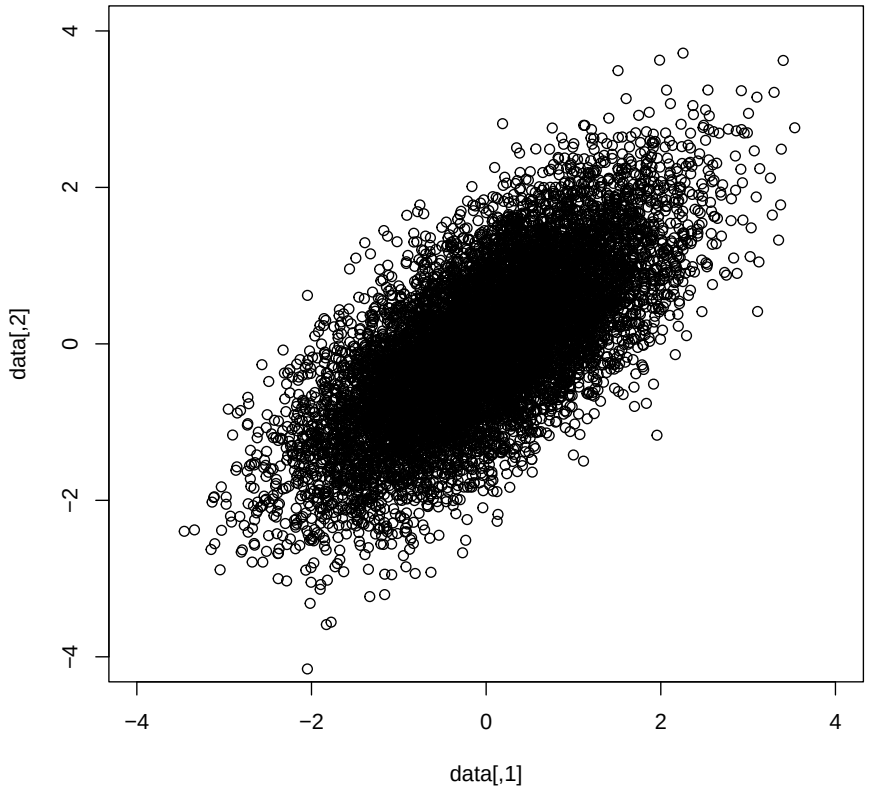
Important remark 20. Mahalanobis distance is **scale and rotation invariant**.

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
```

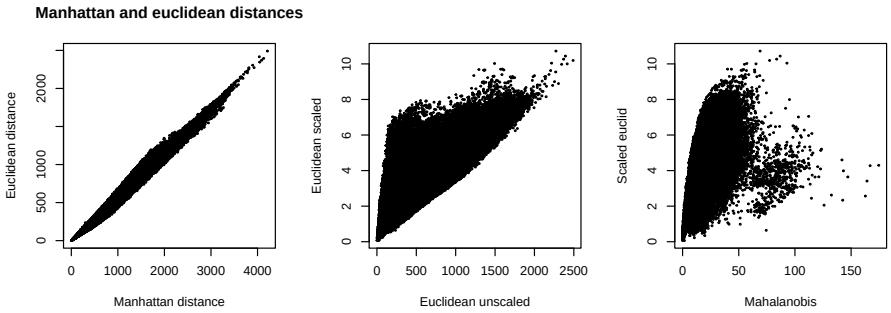
```r
library(pdfCluster)
data(oliveoil)
olive <- oliveoil[, 3:10]
solive <- scale(olive)
```

```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.

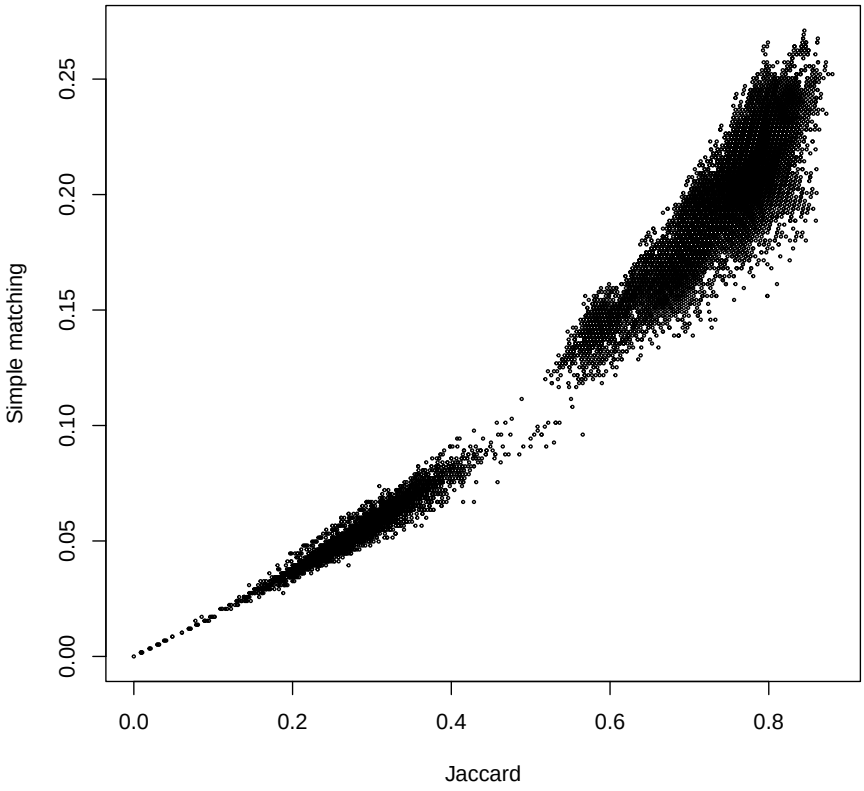
Example 2.2.2. If

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

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

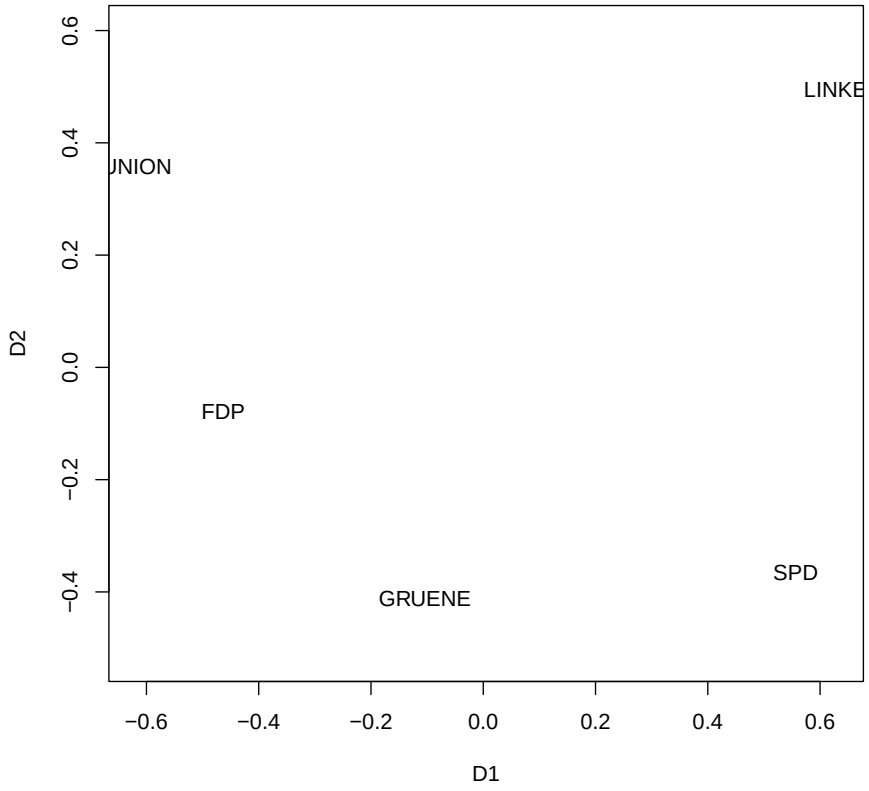
- choosing $\mathbf{b}$ and $\mathbf{y}_1, \dots, \mathbf{y}_n$: at numerator we have a least square principle, which is minimized (denominator is a normalizing constant, counting the number of unique i, j couples)
- minimization is conducted under the constraint $\sum_{i<j} b^2 d(\mathbf{x}_i, \mathbf{x}_j)^2 = \frac{n(n-1)}{2}$. 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?

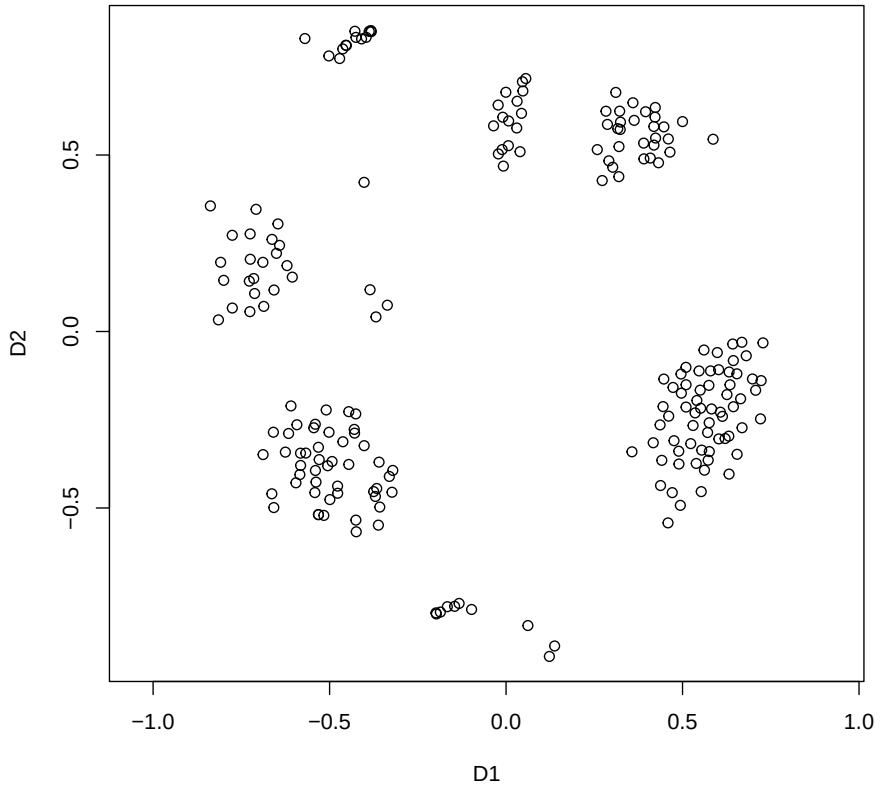
```
## this is our distance
p05 <- read.table("data/bundestag.dat", header = TRUE)
corp05 <- cor(p05[1:5])
(cordistp05 <- 0.5*(1 - corp05)) # matrix of distances

##              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 $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, $\mathbf{x}_i \in \mathcal{X}$, 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)
```

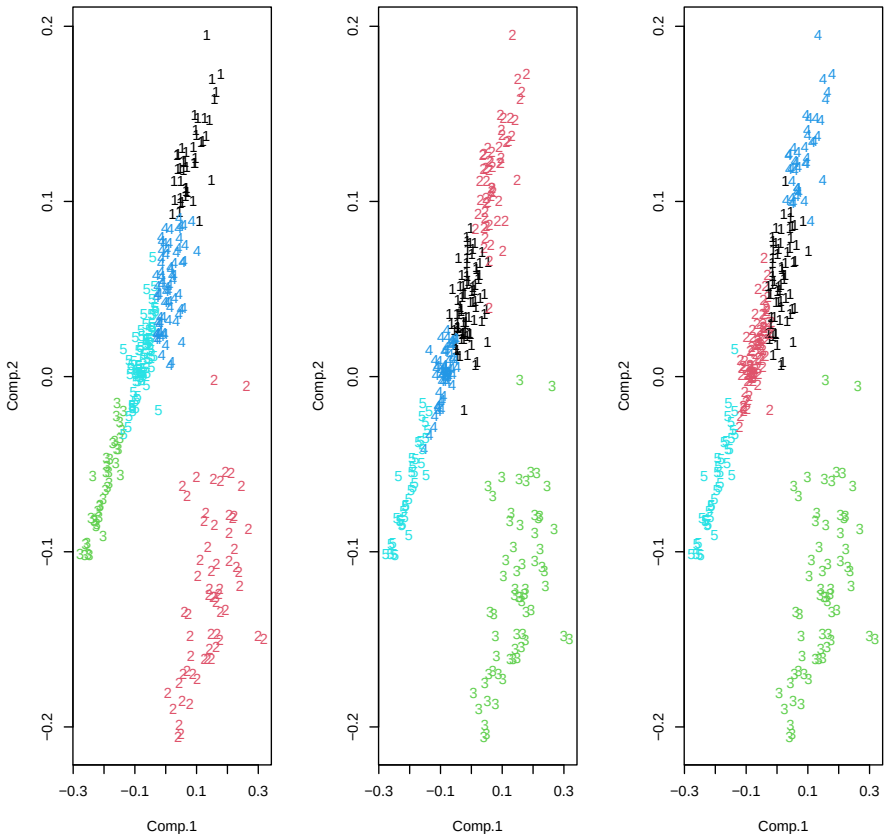


Figure 2.1: PCA-k-means, PAM/Euclidean, PAM/Manhattan

```
## [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])
```

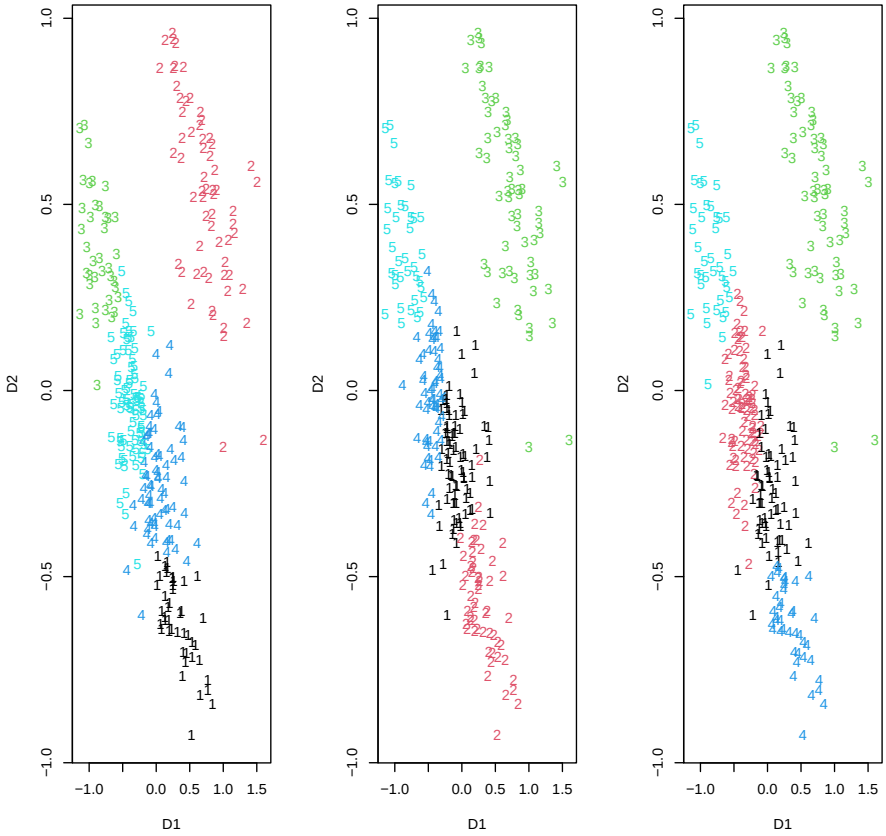


Figure 2.2: MDS manhattan-k-means, PAM/Euclidean, PAM/Manhattan

```
## 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:

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

Some disadvantage of ASW

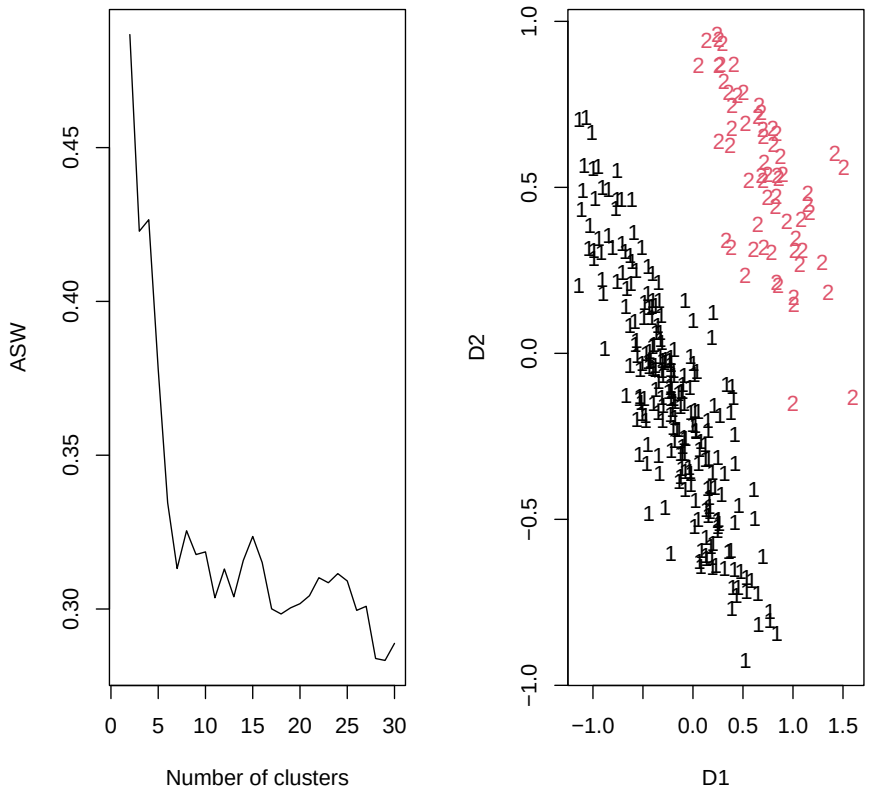
- particularly compared to gap statistics is that it cannot compute for $K = 1$ (we don't have cluster other than one so it's impossible to compute $b(i)$).
- In some applications there's tendency of ASW to favour $K = 2$; local optima for $K > 2$ may also be worth exploring (the idea is to use both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

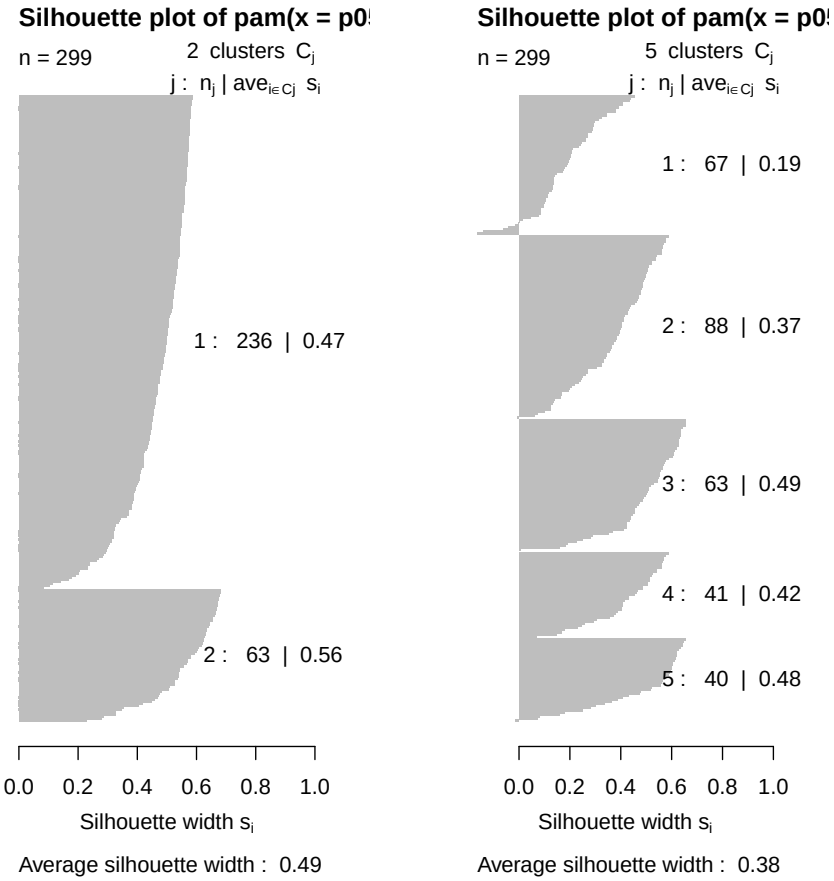
```
##  [1]        NA 0.4867459 0.4228157 0.4265560 0.3779154 0.3345530 0.3131574
##  [8] 0.3254449 0.3177023 0.3185734 0.3036558 0.3130183 0.3039691 0.3157744
## [15] 0.3235948 0.3151002 0.3000481 0.2983881 0.3003727 0.3017265 0.3043202
## [22] 0.3101771 0.3085357 0.3115082 0.3091192 0.2995540 0.3008825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

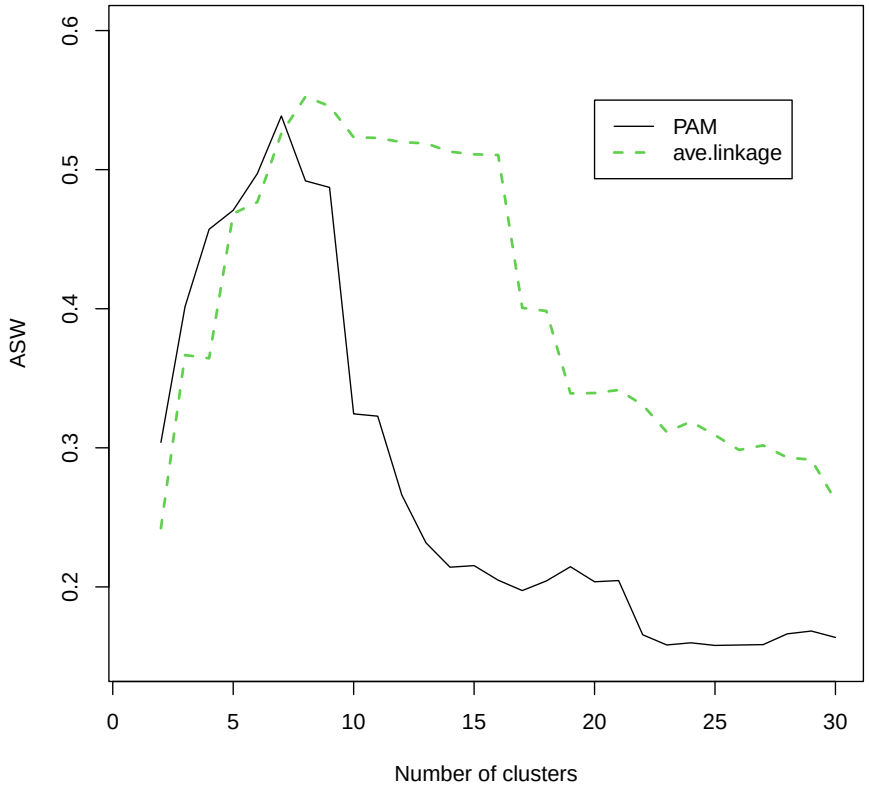


In the K=2 solution we can see we have 236 observation in the first cluster (average silhouette for its object is 0.47) and 63 in the second (average silhouette 0.56); objects are ordered with $s(i)$ in decreasing order. ASW is plotted as well 0.49 cluster.
If we split in $k = 5$ cluster the first one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

Example 2.7.3. Another example on veronica

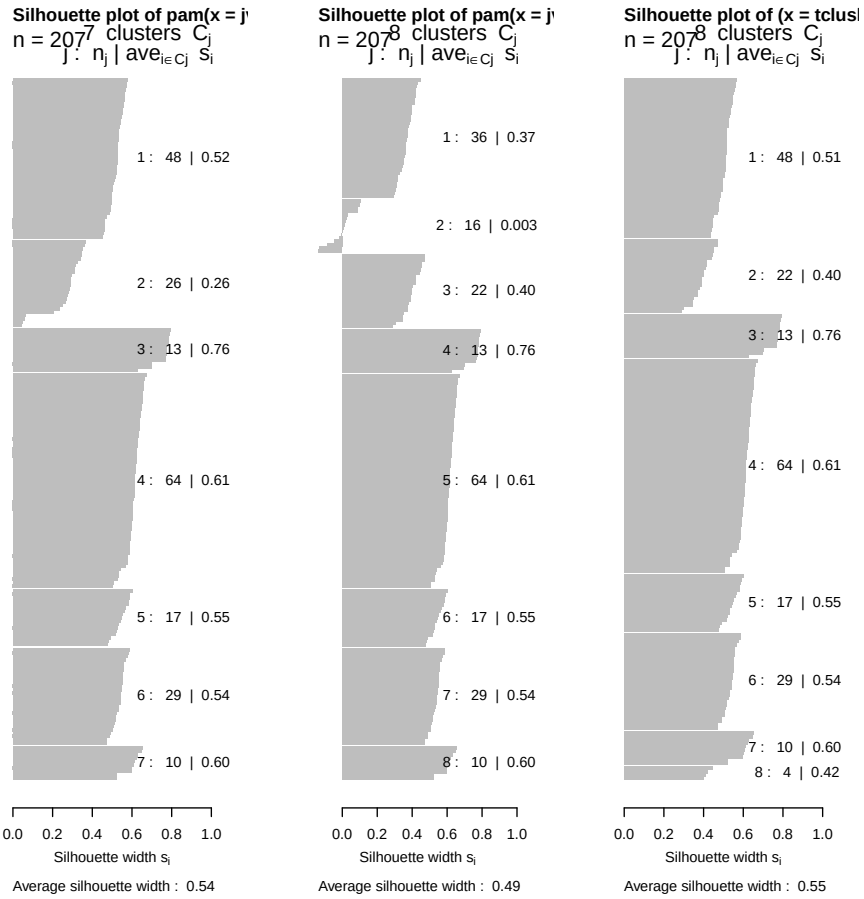
```
# PAM
pasw <- NA
pclusk <- list()
psil <- list()
for (k in 2:30){
  pclusk[[k]] <- pam(jveronica,k)
  psil[[k]] <- silhouette(pclusk[[k]])
  pasw[k] <- summary(psil[[k]])$avg.width
}
which.max(pasw) # Maximum at K=7, ASW=0.5386146

## [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

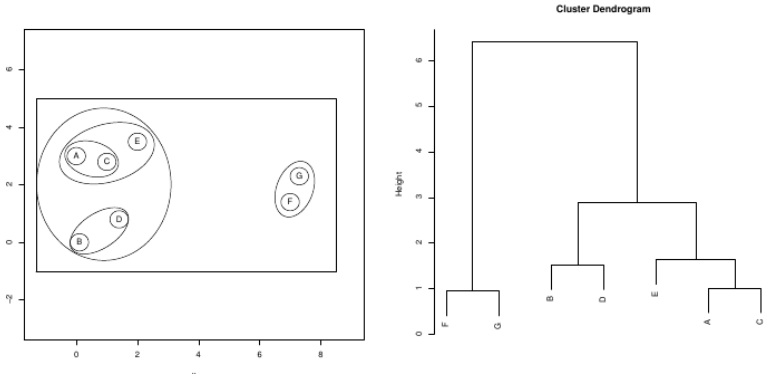


Figure 3.1: Hierarchical clustering

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.

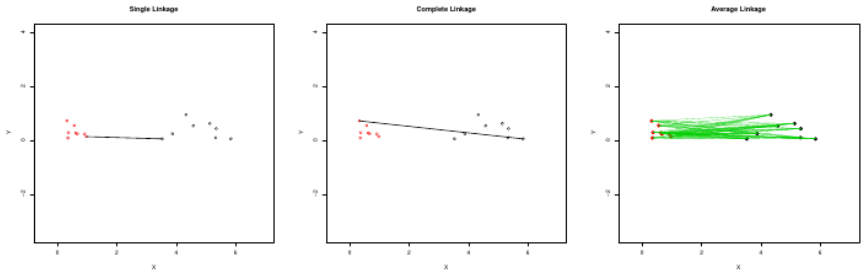


Figure 3.2: Linkage methods

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

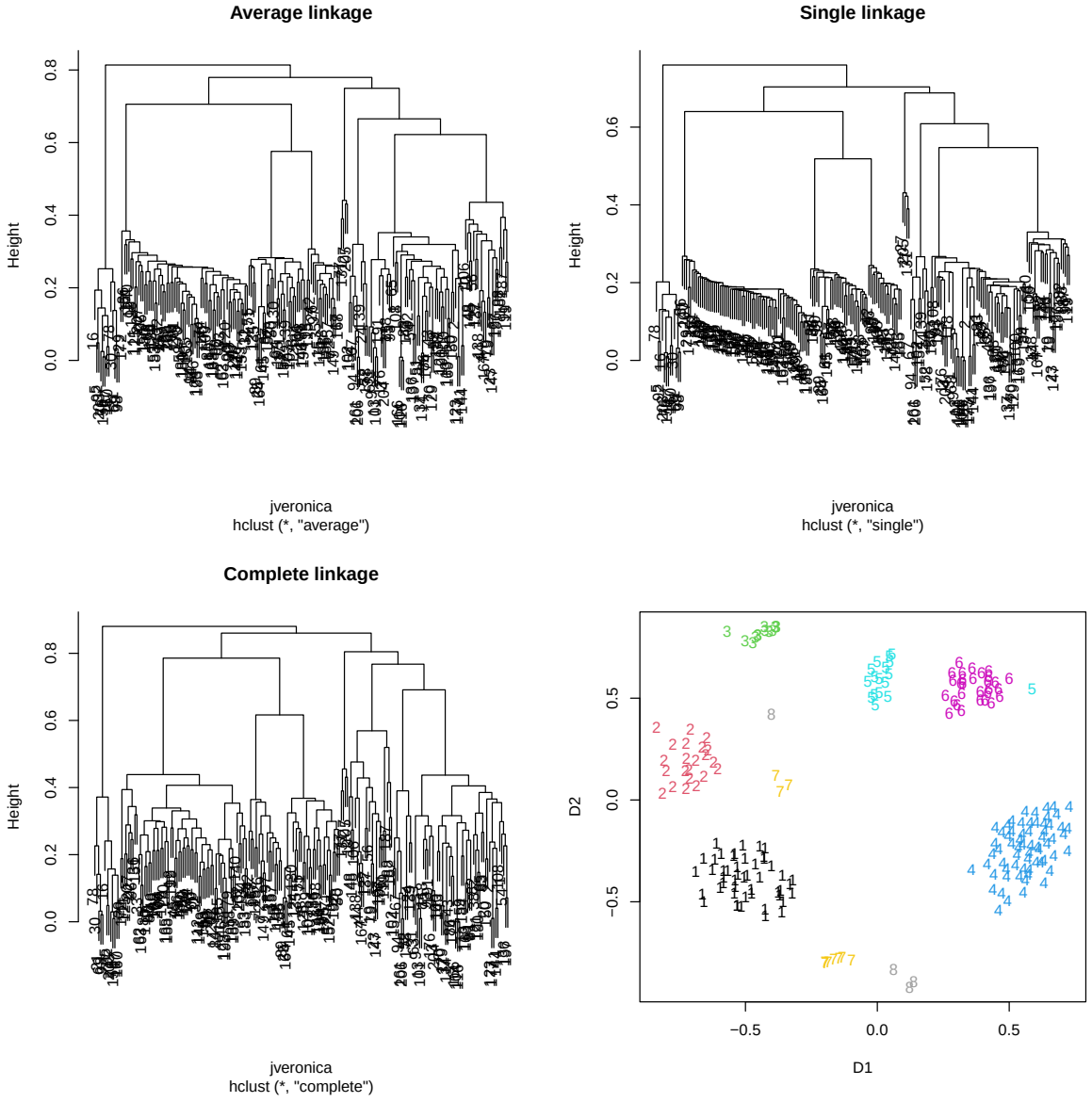


Figure 3.3: Veronica clusterings

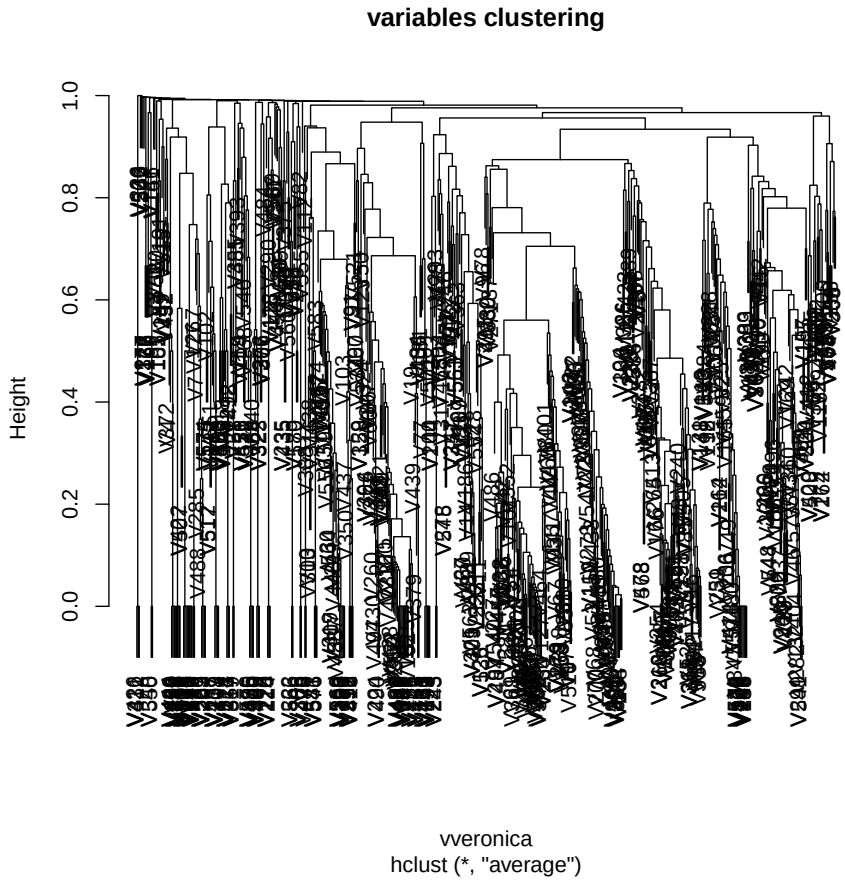


Figure 3.4: veronica heatmap

```
# 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

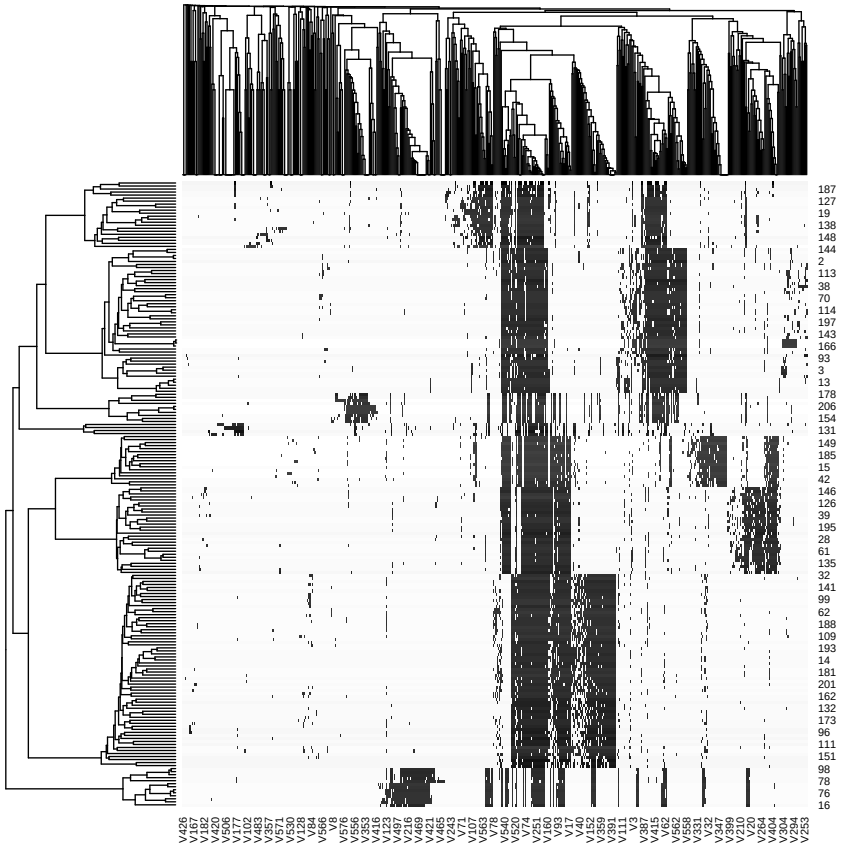


Figure 3.5: veronica heatmap

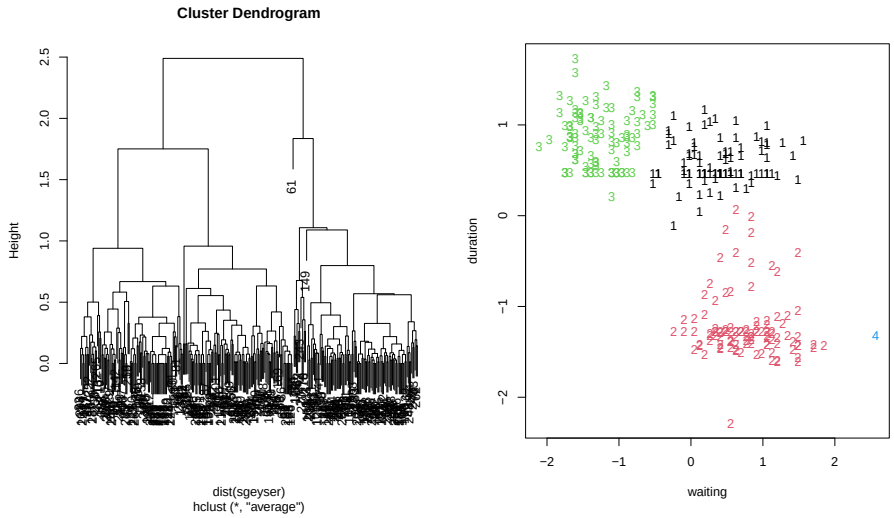


Figure 3.6: Geyser (average linkage)

- **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.

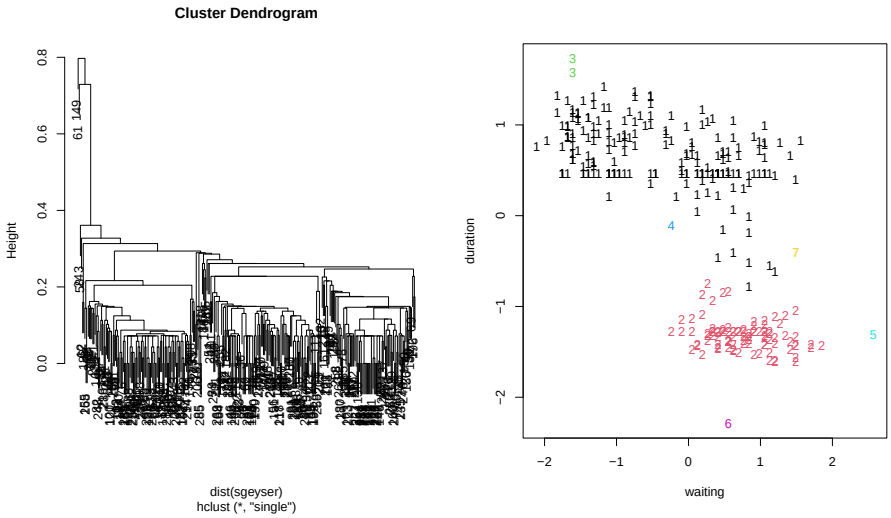


Figure 3.7: Geyser (single linkage)

```
# 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.

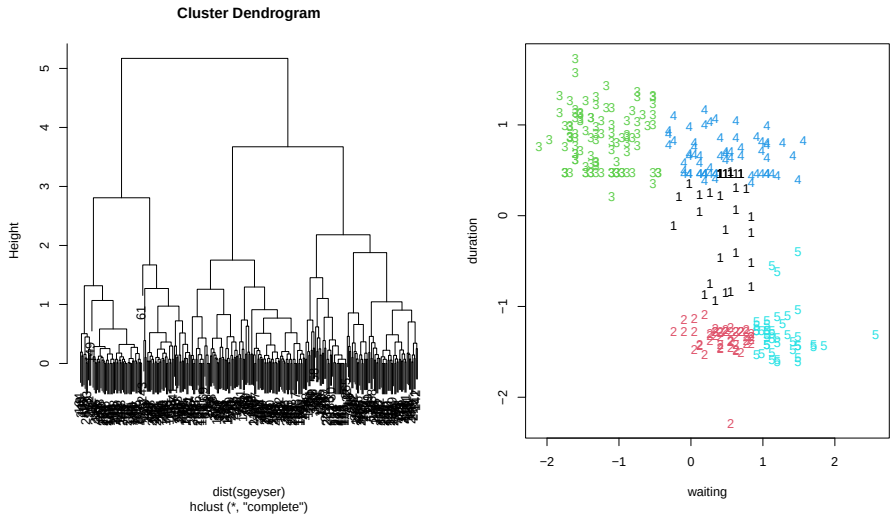


Figure 3.8: Geyser (complete linkage)

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
```

```r
## 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
```