Homework 6

Federico Veronesi

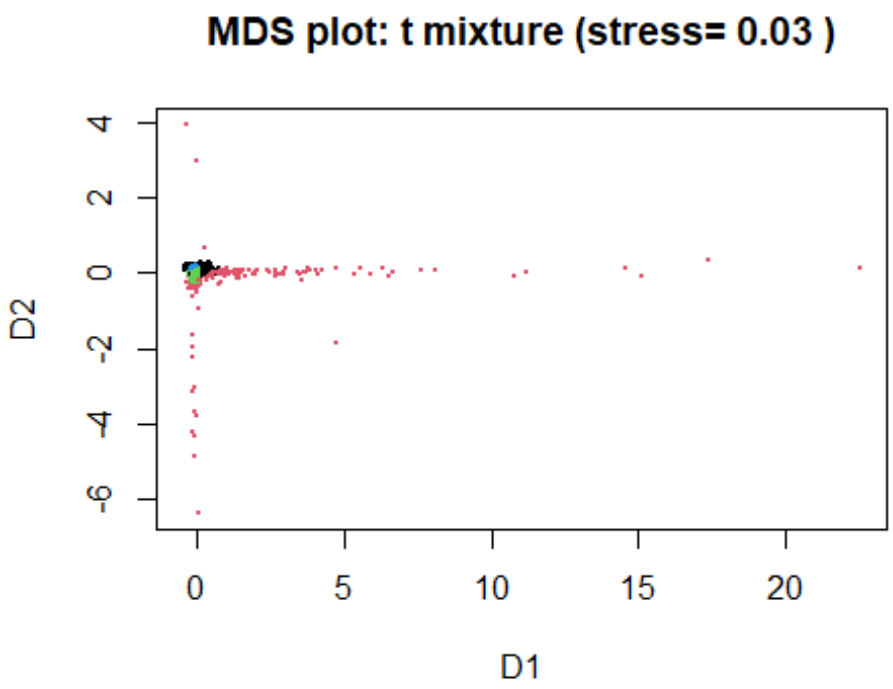
2023-11-18

Exercise 1

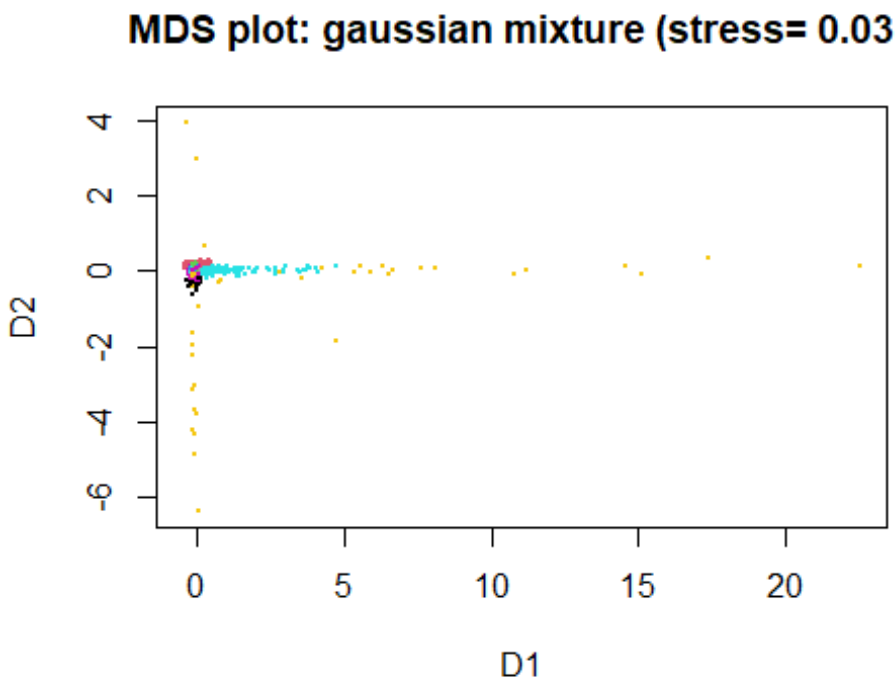
```
setwd("C:/Users/Veronesi/Desktop/uniBo/Magistrale/Modern Statistics and Big Data Analytics")
stars5000 <- read.table("stars5000.dat",header=TRUE)

##MDS
library(tidyverse)

median_standardization <- function(x) {
  median_val <- median(x)
  mad_val <- mad(x, constant = 1.4826)  # constant adjusts for normal distribution

  standardized_data <- (x - median_val) / mad_val

  return(standardized_data)
}

stars5000std <- stars5000 %>% mutate (casn = median_standardization(casn), cacont= median_standardization(cacont), kl1 = median_standardization(kl1), kl2 = median_standardization(kl2), xh1 = median_standardization(xh1), xh2 = median_standardization(xh2))

dist <- dist(stars5000std)
mdsstars <- mds(dist)

#Gaussian mixture

stars5000gaussmixt <- Mclust(stars5000std, G=2:15)

#t-mixture
library(teigen)
stars5000tmixture <- teigen(stars5000std, G=2:15)

summary(stars5000tmixture)

## ------------- Summary for teigen -------------
##              ------   RESULTS   ------
##              Loglik:          -6630.344
##              BIC:             -14240.16
##              ICL:             -15242.74
##              Model:           UUUU
##              # Groups:        4
##
```

```
##
## Clustering Table:
##
##    1    2    3    4
##  470  277 1787 2466
```

```
plot(mdsstars$conf,col=stars5000tmixture$classification, main=paste("MDS plot: t
mixture (stress=", round(mdsstars$stress, 2), ")"), pch=20,cex=0.1)
```




```
plot(mdsstars$conf,col=stars5000gaussmixt$classification, main=paste("MDS plot:
gaussian mixture (stress=", round(mdsstars$stress, 2), ")"), pch=20,cex=0.1)
```




```
adjustedRandIndex(stars5000gaussmixt$classification,
stars5000tmixture$classification)

## [1] 0.1712733
```

Gaussian mixture and t mixture are quite similar(ARI > 0), but it's difficult to see what happens near the origin of the MDS plot. This happens becuase, after the standardization, data are distributed in a strange way: at least one of the two dimension is near to 0 for all the MDS points. Also it is worth to specify that the MDS-plot is a reliable representation in two dimension since only 3% of information is lost. Maybe the presence of extreme ouliers (for example, look on the right side of the MDS plot) suggests that a mixture is t distribution should be more adapt to these data. Unfortunately, we aren't able to verify what happens when fitting skew-t or skew-normal mixtures because we experienced an error in the R code.

Exercise 2

```
library(mclust)
bigdata_method <- function (data, ns, G) {
  subs <- data[sample(x = 1:nrow(data), size = ns, replace = F),]
  subs_mixture <- Mclust(subs, G)
  data_mixture <- predict.Mclust(object = subs_mixture, newdata = data)
  return(data_mixture)
}

## no more than 15 components

stars5000_bdmethod <- bigdata_method(stars5000std, ns=2000, G=2:15)
system.time(bigdata_method(stars5000, ns=2000, G=2:15))
```

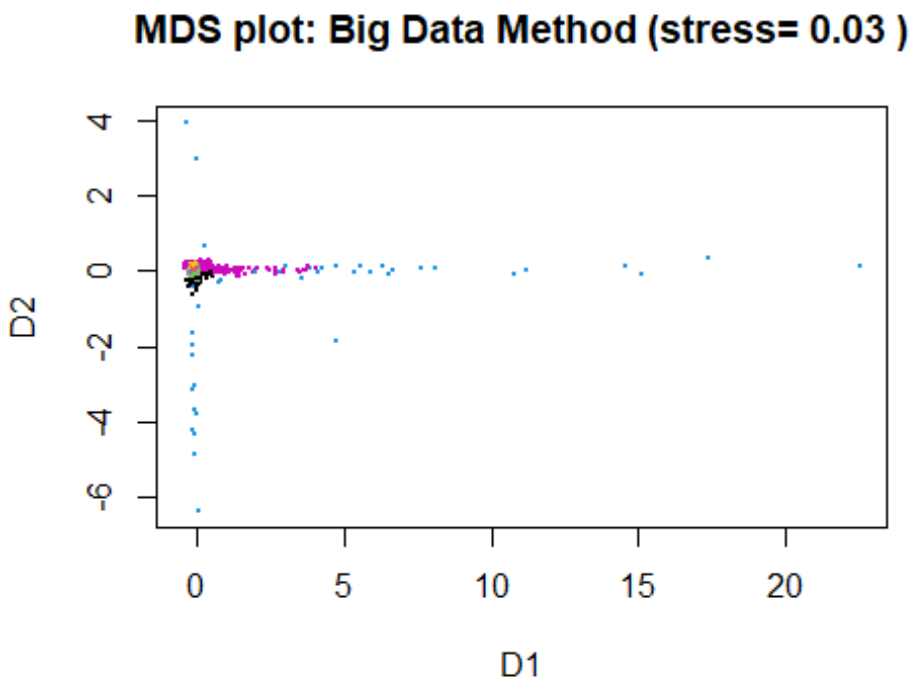
```
##    utente   sistema trascorso
##     33.92      0.02     34.02
```

```
stars5000gaussmixt <- Mclust(stars5000std, G=2:15)
system.time(Mclust(stars5000, G=2:15))
```

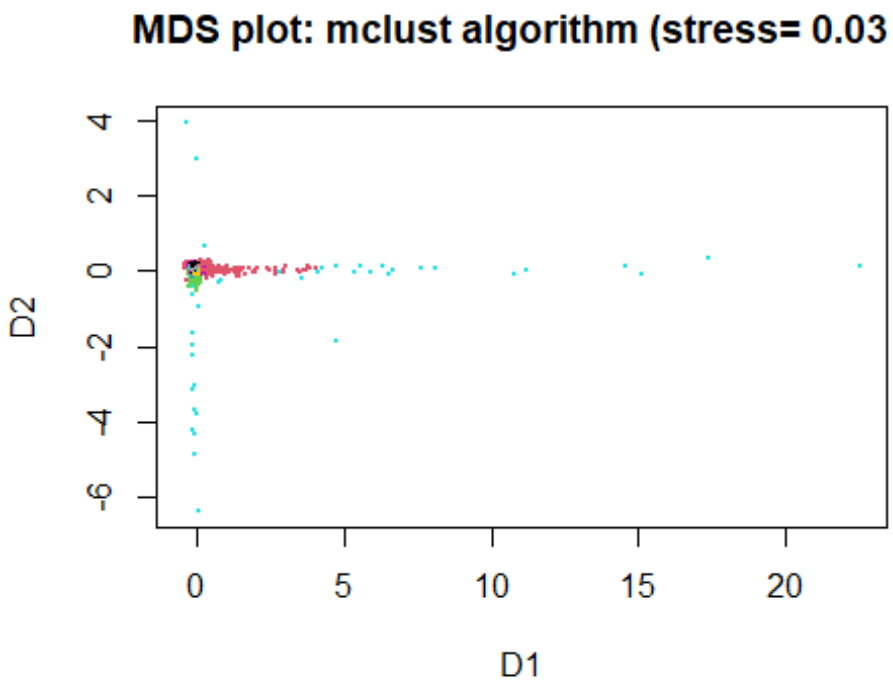
```
##    utente   sistema trascorso
##    103.41      0.38    138.00
```

Results comparison

```
library(fpc)
library(cluster)
library(smacof)
plot(mdsstars$conf,col=stars5000_bdmethod$classification, main=paste("MDS plot:
Big Data Method (stress=", round(mdsstars$stress, 2), ")"), pch=20,cex=0.1)
```




```
plot(mdsstars$conf,col=stars5000gaussmixt$classification,main=paste("MDS plot:
mclust algorithm (stress=", round(mdsstars$stress, 2), ")"), pch=20,cex=0.1)
```




```
adjustedRandIndex(stars5000_bdmethod$classification,
stars5000gaussmixt$classification)

## [1] 0.5893632
```

Results are very similar (ARI is quite high and > 0), but the big data method is faster. With n=5000 the speed of execution is higher (by a factor greater than 2) with respect to the traditional Mclust function applied to the entire dataset.

Exercise 3

Determine the n°of free parameters:

1) G components, p parameters for the mean vector, p(p-1)/2 for the covariance matrix and G-1 weights:
$$G * [p + p(p-1)/2] + G - 1 = 223$$

2) G components, p parameters for the mean, 1 free parameters for the cov.matrix (spherical), G-1 weights:
$$G[p+1] + G - 1 = 47$$

3) G components, p parameters for the mean, p(p-1)/2 for the cov.matrix, p parameters for the skewness, G-1 weights:
$$G[p + p(p+1)/2 + p] + G - 1 = 263$$

4) G components, p parameters for the locations vector, p(p-1)/2 for the cov.matrix, 1 parameters for the df of the t-distributions, G-1 weights: $G[p + p(p+1)/2 + 1] + G - 1 =$

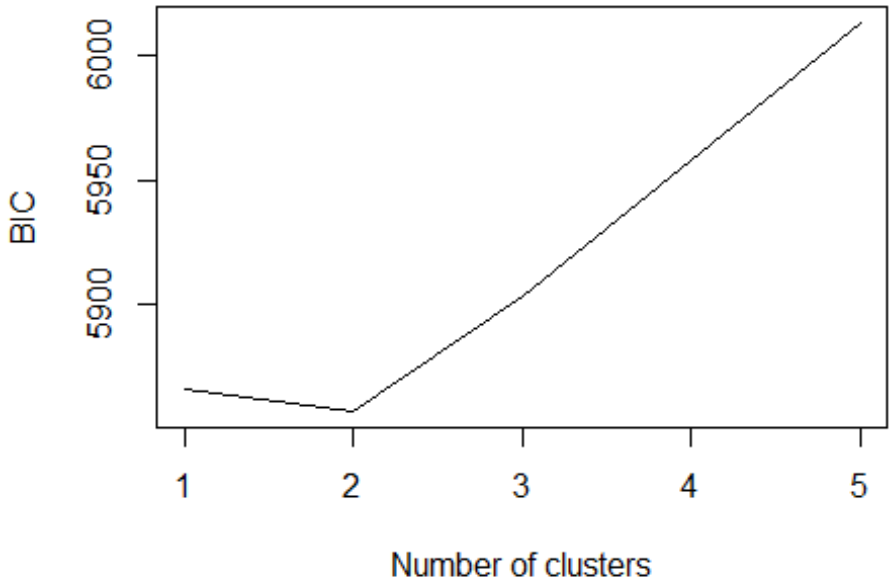
5) G*p parameters for the locations (can be different in each component), p(p-1)/2 parameters for the scale matrix, 1 for the df of the t-distributions, p for the skewnesses, G-1 weights: $G * p + p(p+1)/2 + p + 1 + G - 1 = 99$

In the first 4 points we multiplied for G because for each component we can have different parameters. In the last points only p(n°of free parameters for the location vector) is multiplied for G because it's the only vector that can change among the components.

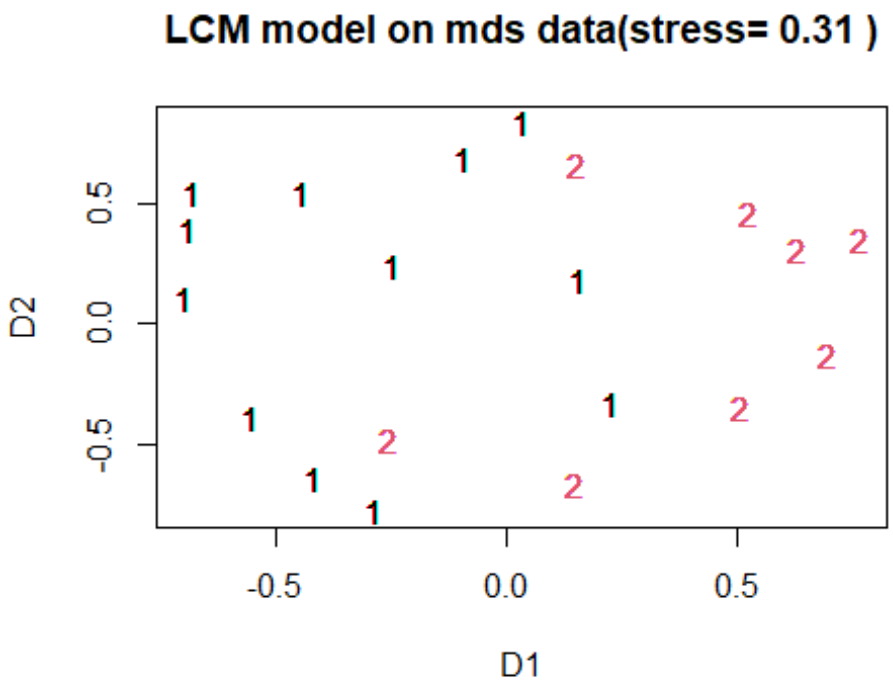
Exercise 4

```
prodcat <- c("PR","MB","AB","N") # product categories
eventcat <- c("S","M","C","P","N") # event categories
prodp <- eventp <- list()
# Category probabilities within the two mixture components (clusters)
prodp[[1]] <- c(0.4,0.4,0.1,0.1)
prodp[[2]] <- c(0.2,0.1,0.4,0.3)
eventp[[1]] <- c(0.5,0.2,0.1,0.1,0.1)
eventp[[2]] <- c(0.1,0.1,0.1,0.3,0.4)
# The first 400 observations from component 1, then 600 from component 2:
n1 <- 400
n2 <- 600
n <- n1+n2
# Initialisation (provide a data frame of the correct type and size
# without already having the data):
consumers <- data.frame(prod=factor(c(prodcat,rep(NA,n-4)),levels=prodcat),
event=factor(c(eventcat,rep(NA,n-5)),levels=eventcat))
# Generation of the data; you may want to set a seed here.
set.seed(555)
consumers$prod[1:n1] <- sample(prodcat,n1,replace=TRUE,prob=prodp[[1]])
consumers$event[1:n1] <- sample(eventcat,n1,replace=TRUE,prob=eventp[[1]])
consumers$prod[(n1+1):n] <- sample(prodcat,n2,replace=TRUE,prob=prodp[[2]])
consumers$event[(n1+1):n] <- sample(eventcat,n2,replace=TRUE,prob=eventp[[2]])
# You can run table(consumers) or str(consumers) to see what you got.

table(consumers)

##     event
## prod   S   M   C   P   N
##   PR  84  45  20  50  80
##   MB  81  44  19  41  37
##   AB  42  29  13  69 111
##   N   42  37  26  56  74

## latent class model
simple_matching_dist <- daisy(consumers, metric = "gower")
library(fpc)
library(smacof)
mdsconsumers <- mds(simple_matching_dist)
LCMconsumers <-flexmixedruns(consumers,continuous=0,discrete=2,n.cluster=1:5)
```

```
plot(1:5,LCMconsumers$bicvals,typ="l",
xlab="Number of clusters",ylab="BIC")
```



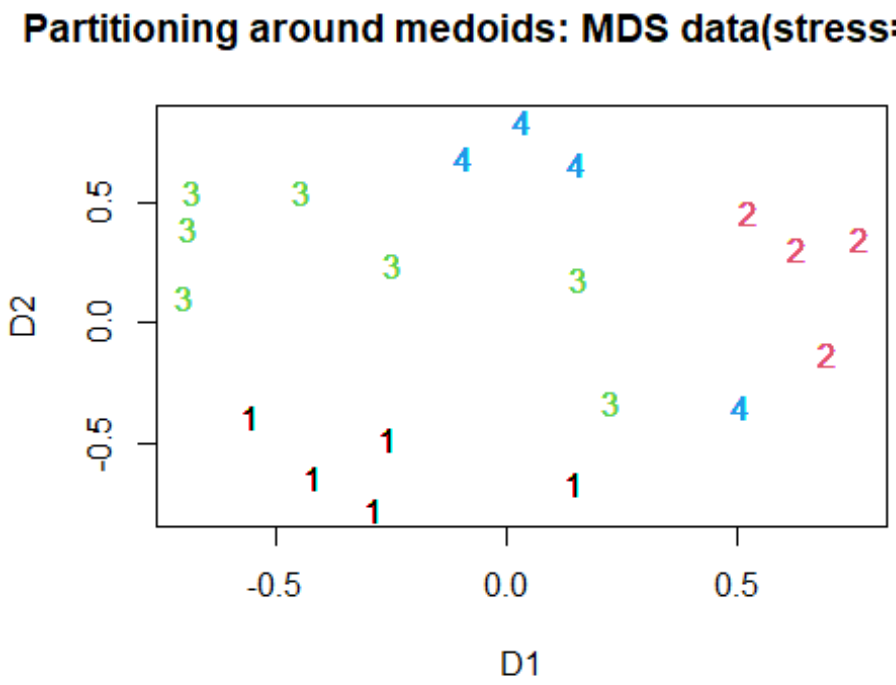

```
## choose k=2
plot(mdsconsumers$conf,col=LCMconsumers$flexout[[2]]@cluster,
pch=clusym[LCMconsumers$flexout[[2]]@cluster], main = paste("LCM model on mds
data(stress=", round(mdsconsumers$stress,2), ")"))
```



The two clusters are somehow meaningful, despite the multidimensional scaling loses a lot of information. There are three points that seem to be “misclassified”.

```
## PAM
library(cluster)
cost_pam <- list()
sil <- list()
asw <- c()
for (k in 2:5) {
  cost_pam[[k]] <- pam(simple_matching_dist, k)
  sil[[k]] <- silhouette(cost_pam[[k]],dist=simple_matching_dist)
  asw[k] <- summary(sil[[k]])$avg.width
  }
plot(1:5,asw,type="l",xlab="Number of clusters",ylab="ASW")
```


```
library(fpc)
plot(mdsconsumers$conf,col=cost_pam[[4]]$clustering,
pch=clusym[cost_pam[[4]]$clustering], main = paste("Partitioning around medoids:
MDS data(stress=", round(mdsconsumers$stress,2), ")"))
```



Also the PAM method performs quite well. There is a point assigned to cluster 4 while it seems to belong to cluster 2 or 3. Maybe the loss of information due to the MDS is responsible for this issue. Let's compare PAM with k=2 with the results of the Latent Class Model, although the ASW for PAM with k=2 is very low.

```
clust_pam2 <- cost_pam[[2]]$clustering
LCMclust2 <- LCMconsumers$flexout[[2]]@cluster
table(clust_pam2, LCMclust2)
```

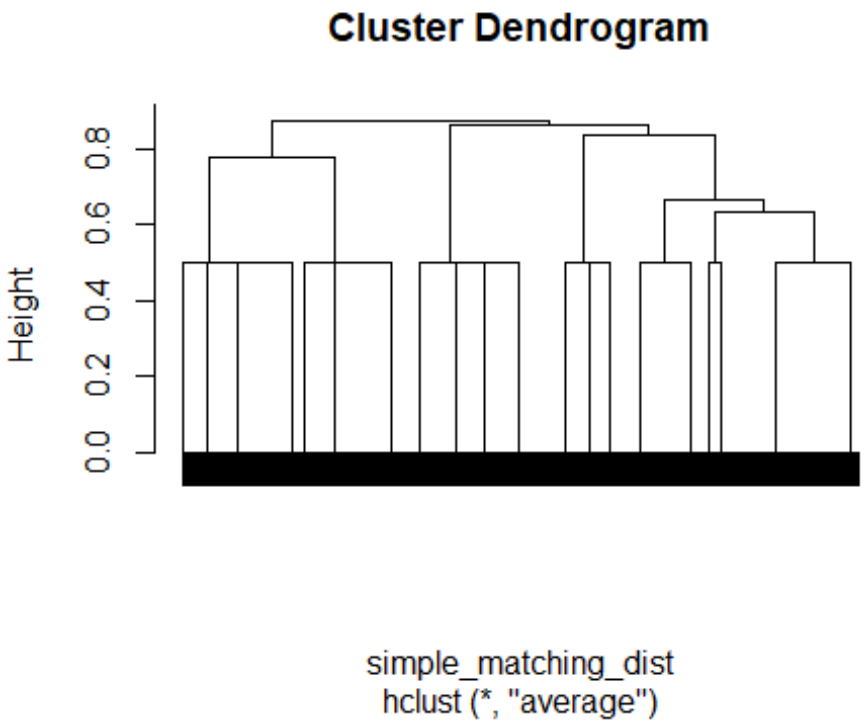
```
##           LCMclust2
## clust_pam2   1   2
##          1 333 357
##          2 185 125
```

```
#Adjusted Rand Index: similarity between the two clusterings
library(mclust)
adjustedRandIndex(clust_pam2, LCMclust2)
```

```
## [1] 0.006019787
```

The two clusterings are completely different. We already suspected that, since the average silhouette width is very low for PAM with k=2. This means that 2 is not a good choice for the number of clusters of Partitioning Around Medoids, and this method finds patterns in data that are different to what the Latent Class Model discovers.

```
### average linkage with simple matching (use gower with binary variables)
library(cluster)
simple_matching_dist <- daisy(consumers, metric = "gower")
dendrogram <- hclust(simple_matching_dist, method = "average")
plot(dendrogram, labels= F)
```



The problem with avg linkage, in a dataset like this, is that (having only two categorical variables) the distance between two units (=consumers) can assume only three values:
- 0, if both variables are equal
- 0.5, if the two units share the class for one variable out of two
- 1, if the units differ in both the variables.

The first aggregation happens at distance 0 (in which 20 groups of units are created). Here the units of the same group are all equal.
The following aggregations happen at a distance of 0.5. The problem is that the number of possible couples of clusters that share a class (and therefore have a distance of 0.5 between them) is very high, so among all these possible aggregations the algorithm chooses randomly. What we obtain here is a random partition, and the following steps of the avg linkage method are conditioned by what happens here.