Homework2- Casadio Sara

2022-10-05

(1a)Use the gap statistic method to estimate the number of clusters for the olive oil data, and for Artificial Data Set 2. Comment on the solutions.

```
library(pdfCluster)
```

```
## pdfCluster 1.0-3
```

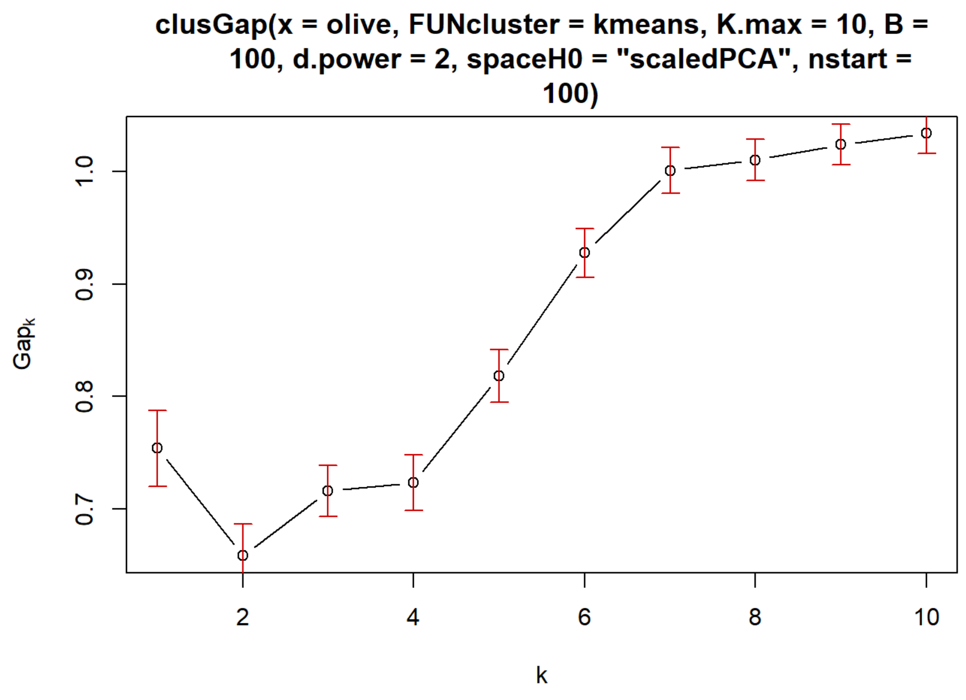
```
data("oliveoil")
olive<-oliveoil[,3:10]

library(cluster)
set.seed(12345)
gap.olive<- clusGap(olive,kmeans,K.max=10,B=100,d.power=2,spaceH0="scaledPCA",nstart=100)
```

```
## Warning: non converge in 10 iterazioni

## Warning: non converge in 10 iterazioni

## Warning: non converge in 10 iterazioni

## Warning: non converge in 10 iterazioni

## Warning: non converge in 10 iterazioni

## Warning: non converge in 10 iterazioni

## Warning: non converge in 10 iterazioni

## Warning: non converge in 10 iterazioni

## Warning: non converge in 10 iterazioni

## Warning: non converge in 10 iterazioni

## Warning: non converge in 10 iterazioni

## Warning: non converge in 10 iterazioni

## Warning: non converge in 10 iterazioni
```

```
print(gap.olive,method="globalSEmax",SE.factor=2)
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = olive, FUNcluster = kmeans, K.max = 10, B = 100, d.power = 2, spaceH0 = "scaledPCA", nstart = 100)
## B=100 simulated reference sets, k = 1..10; spaceH0="scaledPCA"
##  --> Number of clusters (method 'globalSEmax', SE.factor=2): 7
##           logW   E.logW       gap     SE.sim
##  [1,] 18.11113 18.86495 0.7538225 0.03386493
##  [2,] 17.07720 17.73547 0.6582719 0.02807365
##  [3,] 16.53988 17.25584 0.7159615 0.02301426
##  [4,] 16.28246 17.00596 0.7235044 0.02465233
##  [5,] 16.04280 16.86109 0.8182864 0.02352988
##  [6,] 15.80604 16.73375 0.9277099 0.02186258
##  [7,] 15.60848 16.60965 1.0011687 0.02045309
##  [8,] 15.48022 16.49087 1.0106461 0.01828674
##  [9,] 15.35631 16.38058 1.0242741 0.01805455
## [10,] 15.25315 16.28729 1.0341484 0.01809935
```

```
plot(gap.olive)
```



The plot of the gap for the dataset oliveoil shows that the best number of clusters is 7. Number of clusters (method 'globalSEmax', SE.factor=2): 7 Knowing that Gap(K) > Gap(K*) − 2*sk* where K* = arg maxGap(L), we see 1.0341484-2*0.01809935= 0.9979497. 1.0011687 is the value corresponding value of gap(K) for 7 clusters.

```
set.seed(12345)
gapnc.scal2 <- function(data,FUNcluster=kmeans,
                  K.max=10, B = 100, d.power = 2,
                  spaceH0 ="scaledPCA",
                  method ="globalSEmax", SE.factor = 2,...){
    gap1 <- clusGap(data,kmeans,K.max, B, d.power,spaceH0,...)
    nc <- maxSE(gap1$Tab[,3],gap1$Tab[,4],method, SE.factor)
    kmopt <- kmeans(data,nc,...)
    out <- list()
    out$gapout <- gap1
    out$nc <- nc
    out$kmopt <- kmopt
    out
}
gap.olive2<-gapnc.scal2(olive)
gap.olive2$nc
```

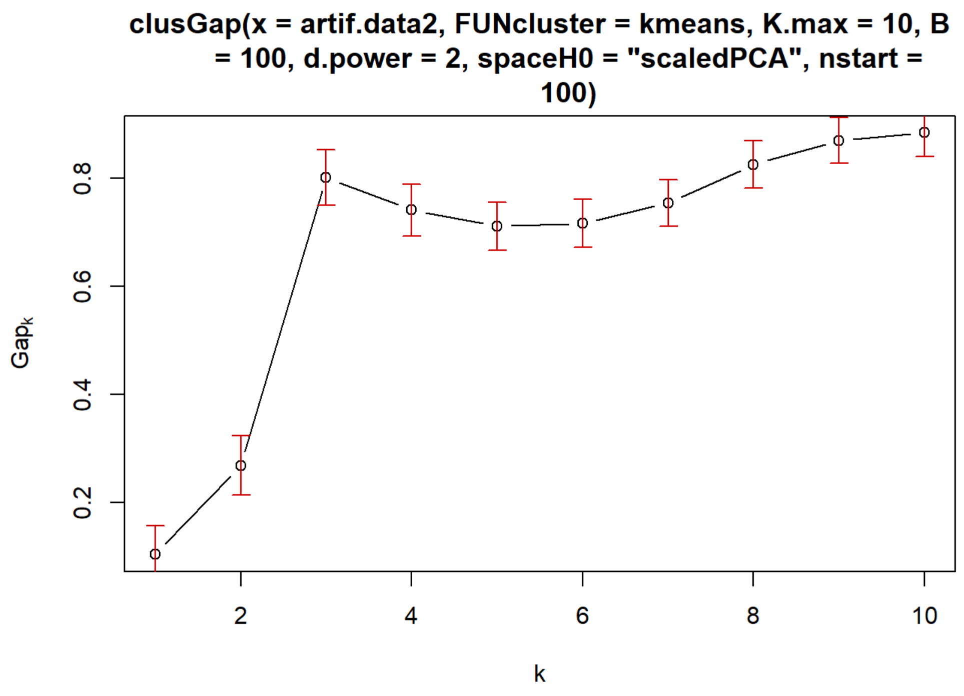
```
## [1] 9
```

Applying the gapnc function with spaceH0="scaledPCA" and SE.factor=2 results that the best number of cluster is 9.

```
set.seed(12345)
x1 <- rnorm(20)
y1 <- rnorm(20)
x2 <- rnorm(20,mean=10)
y2 <- rnorm(20)
x3 <- runif(100,-20,30)
y3 <- runif(100,20,40)
artif.data2 <- cbind(c(x1,x2,x3),c(y1,y2,y3))

gap.art<- clusGap(artif.data2,kmeans,K.max=10,B=100,d.power=2,spaceH0="scaledPCA",nstart=100)
print(gap.art,method="globalSEmax",SE.factor=2)
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = artif.data2, FUNcluster = kmeans, K.max = 10, B = 100, d.power = 2, spaceH0 = "scaledPCA", nstart = 100)
## 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,] 10.169166 10.273281 0.1041153 0.05306068
##  [2,]  9.472092  9.740655 0.2685626 0.05472599
##  [3,]  8.460702  9.262338 0.8016368 0.05113016
##  [4,]  8.097782  8.839089 0.7413061 0.04788341
##  [5,]  7.908722  8.619876 0.7111542 0.04423717
##  [6,]  7.704313  8.420603 0.7162905 0.04413090
##  [7,]  7.478584  8.232765 0.7541808 0.04268115
##  [8,]  7.238069  8.063388 0.8253190 0.04338744
##  [9,]  7.049870  7.919759 0.8698892 0.04237271
## [10,]  6.909756  7.793661 0.8839051 0.04423626
```

```
plot(gap.art)
```



The plot shows that the best number of cluster according to the gap interpretation is 3 since it is the highest peak. Number of clusters (method 'globalSEmax', SE.factor=2): 3 With k=9 and k=10 the peak are a little bit higher but the bowl between these values means that 3 is the best choice. Given Gap(K) > Gap(K) − *2sk* => 0.8839051-2*0.04423626 = 0.7954326. The corresponding minimum gap(k) that is 0.8016368, for 3 clusters.

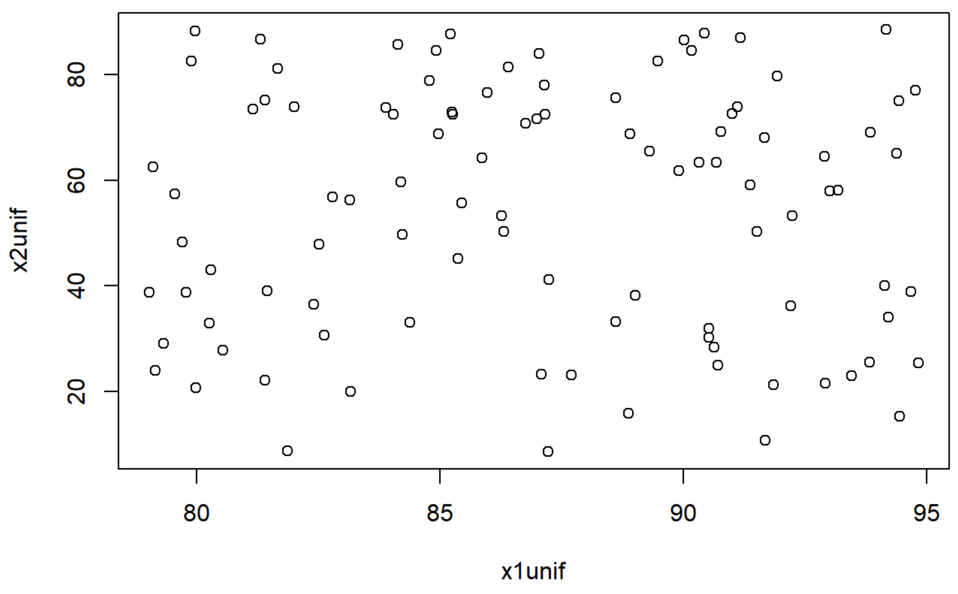
```
gap.art2<-gapnc.scal2(artif.data2)
gap.art2$nc
```

```
## [1] 10
```

In fact the function gapnc with the same parameter setting as before tell us the the best number of cluster if 10.

(1b)Generate a dataset from a two-dimensional uniform distribution on the rectangle [min x1; max x1]x[min x2; max x2], where x1; x2 are the values of the first and second variable of Artificial Data Set 2. Compute K-means clusterings for K from 1 to 10 (obviously for K = 1 it's just all points in the same cluster).

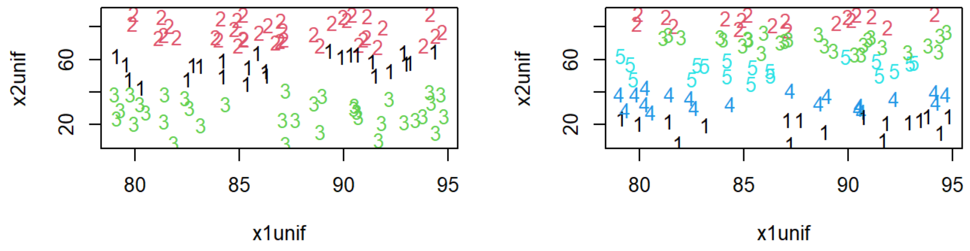
```
set.seed(12345)
clusterdata2 <- read.table("C:/Users/Utente/OneDrive/Desktop/clusterdata2.dat", quote="\"", comment.char="")

min.1<-which.min(clusterdata2$V1)
max.1<-which.max(clusterdata2$V1)
min.2<-which.min(clusterdata2$V2)
max.2<-which.max(clusterdata2$V2)

x1unif<-runif(100, min.1,max.1)
x2unif<-runif(100, min.2, max.2)

data<-cbind(x1unif, x2unif)
plot(data)
```

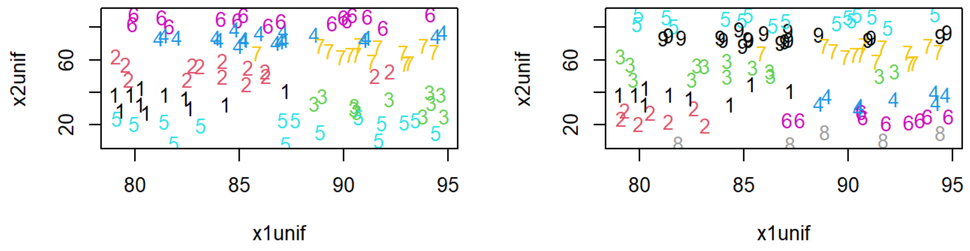


```
kvector<-list()
set.seed(12345)

kdata<-for (k in 1:10) {
    kvector[[k]]<-kmeans(data,centers = k, nstart=100)
}
k3<-kvector[[3]]
k5<-kvector[[5]]
k7<-kvector[[7]]
k9<-kvector[[9]]
```

Show at least three scatterplots of the dataset with clusterings with different K.

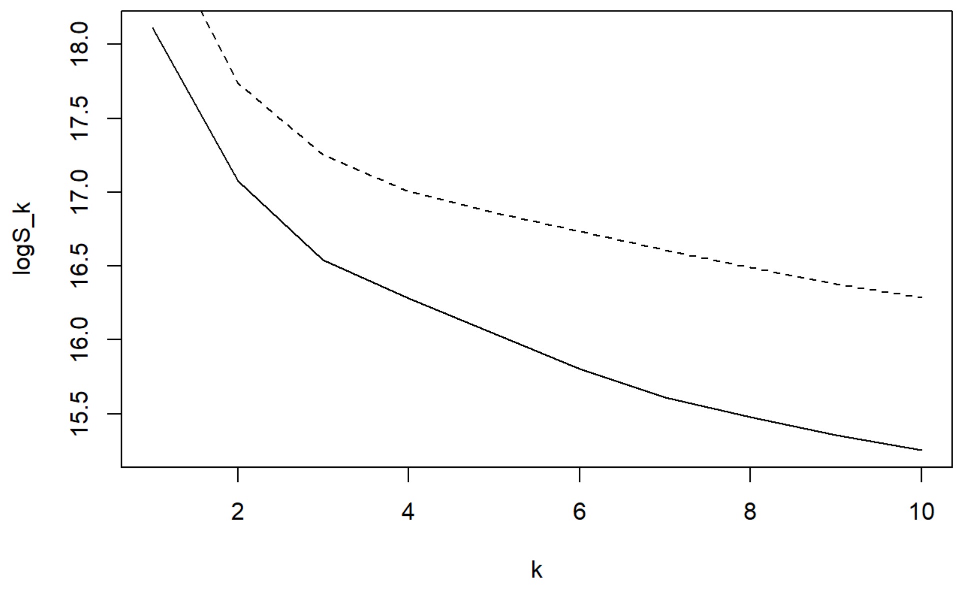
```
library(fpc)

par(mfrow=c(2,2))
plot(data,col=k3$cluster, pch=clusym[k3$cluster])
plot(data,col=k5$cluster, pch=clusym[k5$cluster])
plot(data,col=k7$cluster, pch=clusym[k7$cluster])
plot(data,col=k9$cluster, pch=clusym[k9$cluster])
```



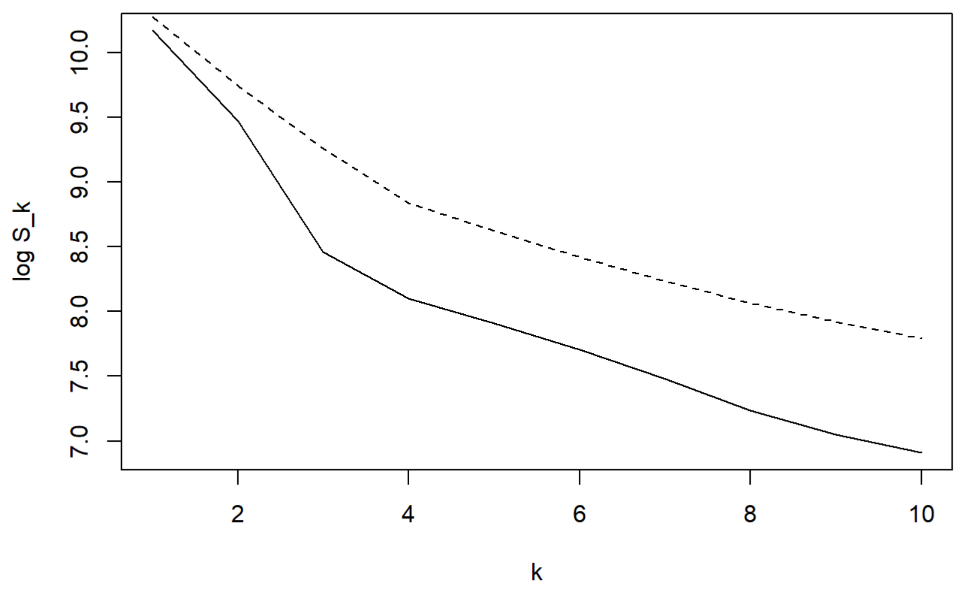


Compare the values of log Sk from clustering Artificial Data Set 2 in (a) with those from clustering the uniformly distributed dataset in a plot.

```
plot(1:10,gap.olive$Tab[,1],xlab="k", ylab="logS_k", type="l")
points(1:10,gap.olive$Tab[,2],xlab="k", ylab="logS_k",type="l",lty=2)
legend(8,9,c("log S_k in data","E(log S_k) uniform"),lty=1:2)
```



```
plot(1:10,gap.art$Tab[,1],xlab="k",ylab="log S_k",type="l")
points(1:10,gap.art$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)
```



(2)The R-function clusGap others various options, for example spaceH0=“original” (using a rectangle along the main axes for the uniform distribution) and SE.factor=1 (q = 1 for the factor q with which the standard error of Gap(K) is multiplied, as suggested in Tibshirani et al.’s original paper). We may be interested in whether these options could improve the results compared to the options spaceH0=“scaledPCA” and SE.factor=2 as used in the lecture. As every spaceH0 choice can be combined with every SE.factor choice, these options define four different ways to run the clusGap function.

```
require(cluster)

gapnc.scal2 <- function(data,FUNcluster=kmeans,
                  K.max=10, B = 100, d.power = 2,
                  spaceH0 ="scaledPCA",
                  method ="globalSEmax", SE.factor = 2,...){
    gap1 <- clusGap(data,kmeans,K.max, B, d.power,spaceH0,...)
    nc <- maxSE(gap1$Tab[,3],gap1$Tab[,4],method, SE.factor)
    kmopt <- kmeans(data,nc,...)
    out <- list()
    out$gapout <- gap1
    out$nc <- nc
    out$kmopt <- kmopt
    out
}
gapnc.or2 <- function(data,FUNcluster=kmeans,
                  K.max=10, B = 100, d.power = 2,
                  spaceH0 ="original",
                  method ="globalSEmax", SE.factor = 2,...){
    gap1 <- clusGap(data,kmeans,K.max, B, d.power,spaceH0,...)
    nc <- maxSE(gap1$Tab[,3],gap1$Tab[,4],method, SE.factor)
    kmopt <- kmeans(data,nc,...)
    out <- list()
    out$gapout <- gap1
    out$nc <- nc
    out$kmopt <- kmopt
    out
}
gapnc.scal1 <- function(data,FUNcluster=kmeans,
                  K.max=10, B = 100, d.power = 2,
                  spaceH0 ="scaledPCA",
                  method ="globalSEmax", SE.factor = 1,...){
    gap1 <- clusGap(data,kmeans,K.max, B, d.power,spaceH0,...)
    nc <- maxSE(gap1$Tab[,3],gap1$Tab[,4],method, SE.factor)
    kmopt <- kmeans(data,nc,...)
    out <- list()
    out$gapout <- gap1
    out$nc <- nc
    out$kmopt <- kmopt
    out
}
gapnc.or1 <- function(data,FUNcluster=kmeans,
                  K.max=10, B = 100, d.power = 2,
                  spaceH0 ="original",
                  method ="globalSEmax", SE.factor = 1,...){
    gap1 <- clusGap(data,kmeans,K.max, B, d.power,spaceH0,...)
    nc <- maxSE(gap1$Tab[,3],gap1$Tab[,4],method, SE.factor)
    kmopt <- kmeans(data,nc,...)
    out <- list()
    out$gapout <- gap1
    out$nc <- nc
    out$kmopt <- kmopt
    out
}
```

Using the gapnc-function that automatically estimates the number of clusters for a data set with clusGap with given choices for spaceH0 and SE.factor, generate 100 data sets according to the specifications for Artificial Data Set 1.

```
#install.packages("sn")
library(sn)
```

```
## Caricamento del pacchetto richiesto: stats4
```

```
##
## Caricamento pacchetto: 'sn'
```

```
## Il seguente oggetto è mascherato da 'package:stats':
##
##     sd
```

```r
#set.seed(665544)
simruns <- 100
nstart <- 100
n <- 1000

method.scal2 <- method.or2 <- method.scal1 <- method.or1 <- list()
dataset <- list()
results.scal2 <- results.or2 <- results.scal1 <- results.or1 <- numeric(0)

results.scal2 <- rep(NA, simruns)
results.or2 <- rep(NA, simruns)
results.scal1 <- rep(NA, simruns)
results.or1<-rep(NA, simruns)

set.seed(12345)
datagen<-for(i in 1:simruns){
    cat("Run ",i,"\n")

    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))
    dataset[[i]] <- cbind(v1,v2)

    method.scal2[[i]] <- gapnc.scal2(dataset[[i]])
    method.or2[[i]] <- gapnc.or2(dataset[[i]])
    method.scal1[[i]] <- gapnc.scal1(dataset[[i]])
    method.or1[[i]] <- gapnc.or1(dataset[[i]])

    results.scal2[i] = method.scal2[[i]]$nc
    results.or2[i] = method.or2[[i]]$nc
    results.scal1[i] = method.scal1[[i]]$nc
    results.or1[i] = method.or1[[i]]$nc
}
```

```
## Run  1
## Run  2
## Run  3
## Run  4
## Run  5
## Run  6
## Run  7
## Run  8
## Run  9
## Run  10
## Run  11
## Run  12
## Run  13
## Run  14
## Run  15
## Run  16
## Run  17
## Run  18
## Run  19
## Run  20
## Run  21
## Run  22
## Run  23
## Run  24
## Run  25
## Run  26
## Run  27
## Run  28
## Run  29
## Run  30
## Run  31
## Run  32
## Run  33
## Run  34
## Run  35
## Run  36
## Run  37
## Run  38
## Run  39
## Run  40
## Run  41
## Run  42
## Run  43
## Run  44
## Run  45
## Run  46
## Run  47
## Run  48
## Run  49
## Run  50
## Run  51
## Run  52
## Run  53
## Run  54
## Run  55
## Run  56
## Run  57
## Run  58
## Run  59
## Run  60
## Run  61
## Run  62
## Run  63
## Run  64
## Run  65
## Run  66
## Run  67
## Run  68
## Run  69
## Run  70
## Run  71
## Run  72
## Run  73
## Run  74
## Run  75
## Run  76
## Run  77
## Run  78
## Run  79
## Run  80
## Run  81
## Run  82
## Run  83
## Run  84
## Run  85
## Run  86
## Run  87
## Run  88
## Run  89
## Run  90
## Run  91
## Run  92
## Run  93
## Run  94
## Run  95
## Run  96
## Run  97
## Run  98
## Run  99
## Run  100
```

```
#these vectors contains the number of cluster for each on 100 dataset on which we are working
results.scal2
```

```
##   [1] 3 4 3 3 3 3 3 3 4 3 4 3 3 4 4 3 3 3 3 3 3 3 4 3 3 3 3 3 3 3 3 3 3 3 3 3 4
##  [38] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 3 3 3 3 3 3 3 3 3 4 3 3 4 3 3 3 3 3 3 3 3 3 3
##  [75] 4 3 3 5 3 3 3 3 3 3 4 3 3 4 3 4 3 3 3 3 3 4 3 3 3 3
```

```
results.or2
```

```
##   [1] 3 3 3 3 3 3 3 3 3 3 4 3 3 4 3 3 4 3 3 4 3 3 4 3 3 4 3 3 4 3 4 3 5 3 3 3 4
##  [38] 3 3 3 4 3 4 4 5 3 4 3 4 4 3 3 4 4 3 3 3 3 3 3 3 3 4 4 3 3 3 3 4 3 3 3 3 3
##  [75] 3 3 3 3 3 3 3 4 4 3 3 4 3 3 3 3 3 3 4 3 3 3 3 3 3 3
```

```
results.scal1
```

```
##   [1] 4 3 3 3 3 3 4 3 4 3 4 3 3 3 3 4 3 3 3 3 3 3 3 3 3 3 5 3 4 3 3 3 4 3 3 3 4
##  [38] 3 3 4 3 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 3 4 3 3 3 4 3 3 3 3 3 4 4 3 3 3 3
##  [75] 3 3 4 3 3 3 3 3 3 4 3 3 3 4 4 3 3 4 3 3 3 3 3 3 3 3
```

```
results.or1
```

```
##   [1] 4 3 3 3 4 3 3 3 3 3 4 3 4 4 3 4 3 3 3 4 4 4 3 3 3 4 4 3 3 3 4 3 3 3 3 3 4
##  [38] 3 3 4 3 3 4 4 4 3 3 3 3 3 4 3 3 3 4 3 3 4 3 3 4 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [75] 3 3 3 3 3 3 3 3 3 3 4 3 3 3 3 3 3 3 4 3 3 3 3 3 3 3
```

At the end look at the four distributions of estimated numbers of clusters using a suitable graphical display and comment on them. Are there better or worse results for some of the clusGap-versions?

```
table(results.scal2)
```

```
## results.scal2
##  3  4  5 
## 84 15  1
```

```
table(results.scal1)
```

```
## results.scal1
##  3  4  5 
## 79 20  1
```

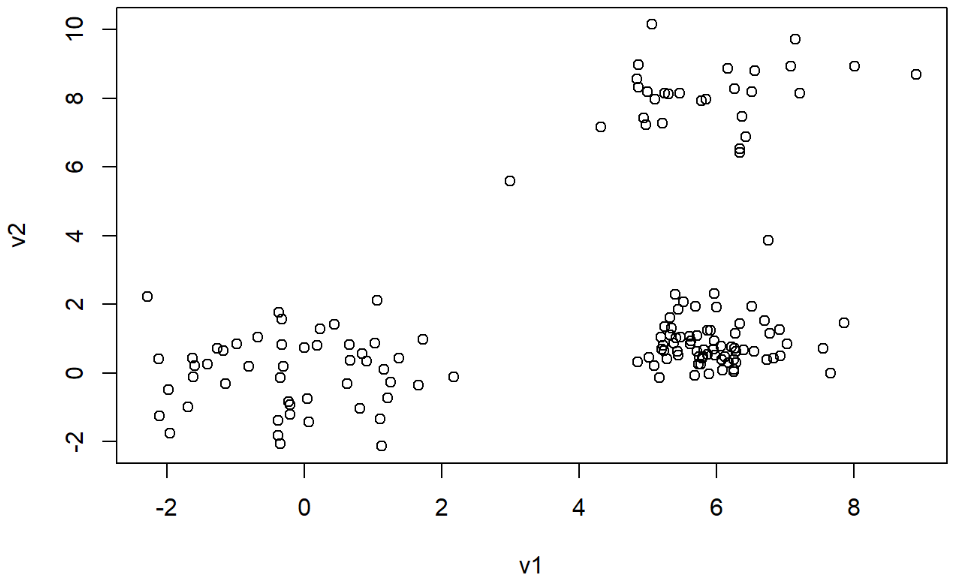
```
table(results.or1)
```

```
## results.or1
##  3  4
## 77 23
```

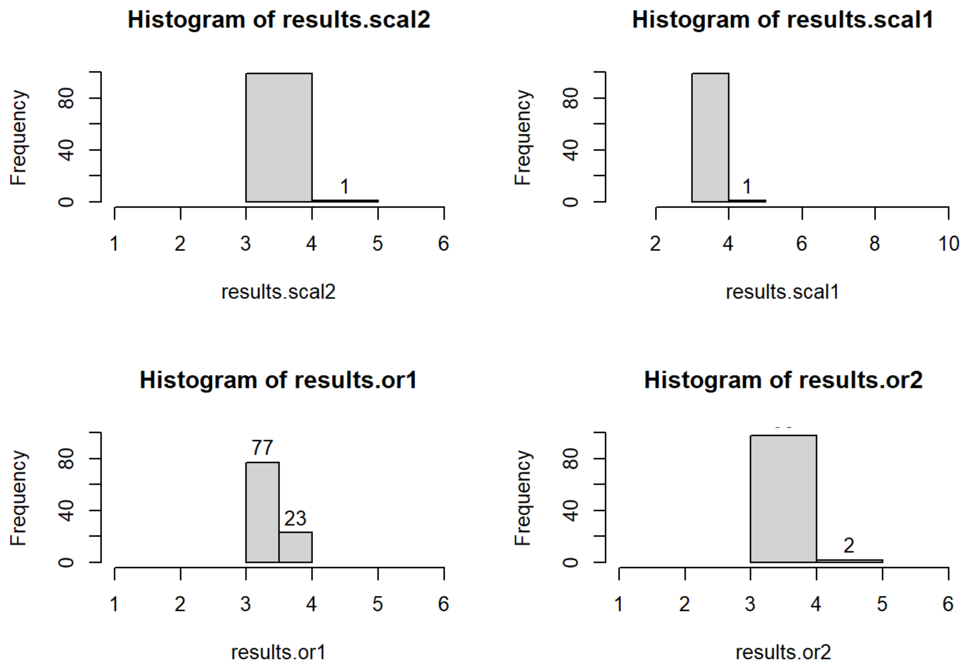
```
table(results.or2)
```

```
## results.or2
##  3  4  5
## 74 24  2
```

```
plot(dataset[[i]])
```



```
#the plot highlights that the best number of clusters is 3

par(mfrow=c(2,2))
hist.scal2<-hist(results.scal2,xlim=c(1,6), ylim=c(0,100), labels = TRUE, breaks = 2)
#for the clusGap function with spaceH0 ="scaledPCA" and SE.factor = 2 there are
#84 datasets over 100 for which correspond as best number of clusters k=3,
#14 with k=4,
#1 with k=5.
hist.scal1<-hist(results.scal1,xlim=c(1,10), ylim=c(0,100), labels = TRUE, breaks = 2)
#for the clusGap function with spaceHO= "scaledPCA" and SE.factor = 1 there are
#79 datasets with best k=3,
#20 with best k=4,
#1 with k=5.
hist.or1<-hist(results.or1,xlim=c(1,6), ylim=c(0,100), labels = TRUE, breaks = 2)
#for the clusGap function with spaceH0 ="original" and SE.factor = 1 there are
#77 datasets with best k=3,
#23 with k=4..
hist.or2<-hist(results.or2,xlim=c(1,6), ylim=c(0,100), labels = TRUE, breaks = 2)
```

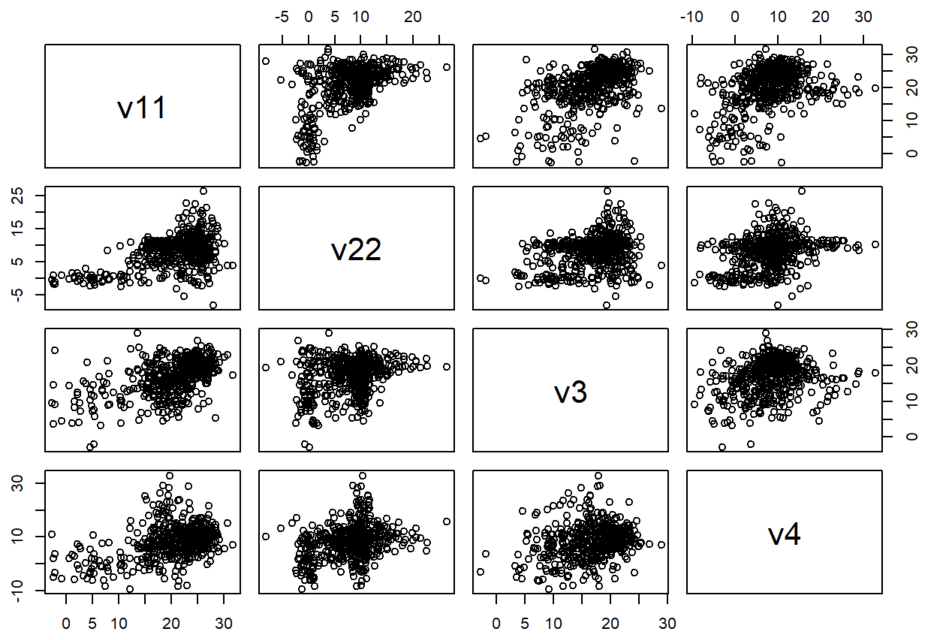


```
#for the clusGap function with spaceH0 ="original" and SE.factor = 2 there are
#74 datasets with best k=3,
#24 with best k=4 and
#2 with best k=5.
```

The clusGap function calculates a goodness of clustering measure. It looks for the optimal number of cluster with a comparison between the clustering of k datasets obtained with simulated replicas of the dataset. The version of cluGap with spaceH0= “original” and SE.factor=1 is the one the provides a more homogeneous results of clustering since form it derives only k=3 and k=4. From all versions we note that the best number of clusters is generally k=3. The version that provides the worse clustering the one with spaceH0=“original” and SE.factor=2.

Here you are asked to generate data from just one further model, with 4 clusters in 4 dimensions, and numbers of observations 50, 100, 150, and 200 for the four clusters. You can use normal distributions for all clusters. These could be generated independently in the four dimensions using rnorm, but using the function rmvnorm from package mtvnorm you could also specify full covariance matrices.

```
simruns <- 100
nstart <- 100
n <- 1000

method.scal2 <- method.or2 <- method.scal1 <- method.or1 <- list()
dataset <- list()
results.scal2 <- results.or2 <- results.scal1 <- results.or1 <- numeric(0)

results.scal2 <- rep(NA, simruns)
results.or2 <- rep(NA, simruns)
results.scal1 <- rep(NA, simruns)
results.or1<-rep(NA, simruns)

v11 <- c(rnorm(50,5,5), rnorm(100,20,5), rnorm(150,20,3), rnorm(200,25,2))
v22<- c(rnorm(50,0,1), rnorm(100,5,3), rnorm(150,10,1), rnorm(200,10,5))
v3 <- c(rnorm(50,10,5), rnorm(100,17,5), rnorm(150,15,4), rnorm(200,20,2))
v4 <- c(rnorm(50,0,5), rnorm(100,6,5), rnorm(150,10,8), rnorm(200,10,3))
dat<-cbind(v11,v22,v3,v4)
pairs(dat)
```



```
set.seed(12345)
datagen<-for(i in 1:simruns){
    cat("Run ",i,"\n")

    v11 <- c(rnorm(50,5,5), rnorm(100,20,5), rnorm(150,20,3), rnorm(200,25,2))
    v22 <- c(rnorm(50,0,1), rnorm(100,5,3), rnorm(150,10,1), rnorm(200,10,5))
    v3 <- c(rnorm(50,10,5), rnorm(100,17,5), rnorm(150,15,4), rnorm(200,20,2))
    v4 <- c(rnorm(50,0,5), rnorm(100,6,5), rnorm(150,10,8), rnorm(200,10,3))

    dataset[[i]] <- cbind(v11,v22,v3,v4)

    method.scal2[[i]] <- gapnc.scal2(dataset[[i]])
    method.or2[[i]] <- gapnc.or2(dataset[[i]])
    method.scal1[[i]] <- gapnc.scal1(dataset[[i]])
    method.or1[[i]] <- gapnc.or1(dataset[[i]])

    results.scal2[i] = method.scal2[[i]]$nc
    results.or2[i] = method.or2[[i]]$nc
    results.scal1[i] = method.scal1[[i]]$nc
    results.or1[i] = method.or1[[i]]$nc
}
```

```
## Run  1
## Run  2
## Run  3
## Run  4
## Run  5
## Run  6
## Run  7
## Run  8
## Run  9
## Run  10
```

```
## Warning: non converge in 10 iterazioni
## Run  11
## Run  12
## Run  13
## Run  14
## Run  15
## Run  16
## Run  17
## Run  18
## Run  19
## Run  20
## Run  21
## Run  22
## Run  23
```

```
## Warning: non converge in 10 iterazioni
```

```
## Run  24
## Run  25
## Run  26
```

```
## Warning: non converge in 10 iterazioni
```

```
## Run  27
## Run  28
## Run  29
## Run  30
## Run  31
```

```
## Warning: non converge in 10 iterazioni
```

```
## Run  32
## Run  33
## Run  34
## Run  35
## Run  36
## Run  37
## Run  38
## Run  39
## Run  40
```

```
## Warning: non converge in 10 iterazioni
```

```
## Run  41
## Run  42
## Run  43
## Run  44
## Run  45
```

```
## Warning: non converge in 10 iterazioni
```

```
## Run  46
## Run  47
## Run  48
## Run  49
## Run  50
## Run  51
## Run  52
```

```
## Warning: non converge in 10 iterazioni
## Run  53
## Run  54
## Run  55
## Run  56
## Run  57
## Run  58
## Run  59
```

```
## Warning: non converge in 10 iterazioni
```

```
## Run  60
## Run  61
## Run  62
## Run  63
## Run  64
## Run  65
## Run  66
## Run  67
## Run  68
## Run  69
## Run  70
## Run  71
## Run  72
## Run  73
```

```
## Warning: non converge in 10 iterazioni
```

```
## Run  74
## Run  75
## Run  76
## Run  77
## Run  78
## Run  79
## Run  80
## Run  81
```

```
## Warning: non converge in 10 iterazioni
```

```
## Run  82
## Run  83
## Run  84
## Run  85
## Run  86
## Run  87
## Run  88
## Run  89
## Run  90
## Run  91
## Run  92
```

```
## Warning: non converge in 10 iterazioni
```

```
## Run  93
## Run  94
## Run  95
## Run  96
## Run  97
## Run  98
## Run  99
```

```
## Warning: non converge in 10 iterazioni
```

```
## Run  100
```

```
results.scal2
```

```
##   [1] 5 2 5 5 5 5 3 5 6 5 2 5 5 4 7 5 6 7 5 6 5 5 5 6 5 5 6 2 5 4 2 2 7 5 4 2 2
##  [38] 7 2 2 5 6 1 5 5 5 2 7 4 2 5 6 5 6 5 5 4 6 4 5 5 5 5 5 5 6 2 5 5 5 6 6 5 6
##  [75] 5 6 5 5 6 5 5 6 5 3 2 7 5 5 5 5 5 5 6 2 5 5 5 5 5 5
```

```
results.or2
```

```
##   [1] 5 5 6 5 5 4 2 5 3 3 2 4 4 4 4 5 5 5 4 7 5 4 5 5 5 4 4 5 5 4 5 5 5 4 4 6 5
##  [38] 5 4 5 4 6 6 5 4 4 5 5 4 5 3 4 5 4 3 4 4 5 5 4 4 5 4 5 6 4 4 4 4 5 4 5 5 5
##  [75] 4 4 4 5 5 5 4 5 5 5 4 5 4 4 5 5 5 4 5 4 5 5 5 4 2 5
```

```
results.scal1
```

```
##   [1]  8  5  6  6  5  5  5  7  6  6  6  7  5  6  6  5  6  7  6  6  5  6  5  7  5
##  [26]  5  7  2  6  6  6  2  9  6  5  6  7  8  8  7  5  8  2  5  5  5  7  6  4  6
##  [51]  5  7  8  6  6  5  5  8  5  5  7  6  5  7  7  7  2  5  6  6  6  6  6 10  5
##  [76]  8  7  6  6  7  6  6  7  6  5  5  5  5  5  6  6  9  6  2  5  5  5  6  6  5
```

```
results.or1
```

```
##   [1] 6 6 6 7 5 5 5 5 2 4 5 5 4 4 5 6 7 6 5 6 5 5 5 5 5 4 6 5 5 5 6 7 6 5 4 5 7
##  [38] 7 4 5 4 7 6 5 5 5 7 7 4 6 5 6 5 4 5 5 5 7 5 4 5 5 5 6 8 5 4 5 5 5 4 6 6 6
##  [75] 5 5 4 5 6 5 4 5 6 5 5 7 4 5 5 5 5 6 5 5 6 5 7 5 2 5
```

```
table(results.scal2)
```

```
## results.scal2
##  1  2  3  4  5  6  7 
##  1 14  2  6 54 17  6
```

```
table(results.scal1)
```

```
## results.scal1
##  2  4  5  6  7  8  9 10 
##  5  1 32 36 16  7  2  1
```

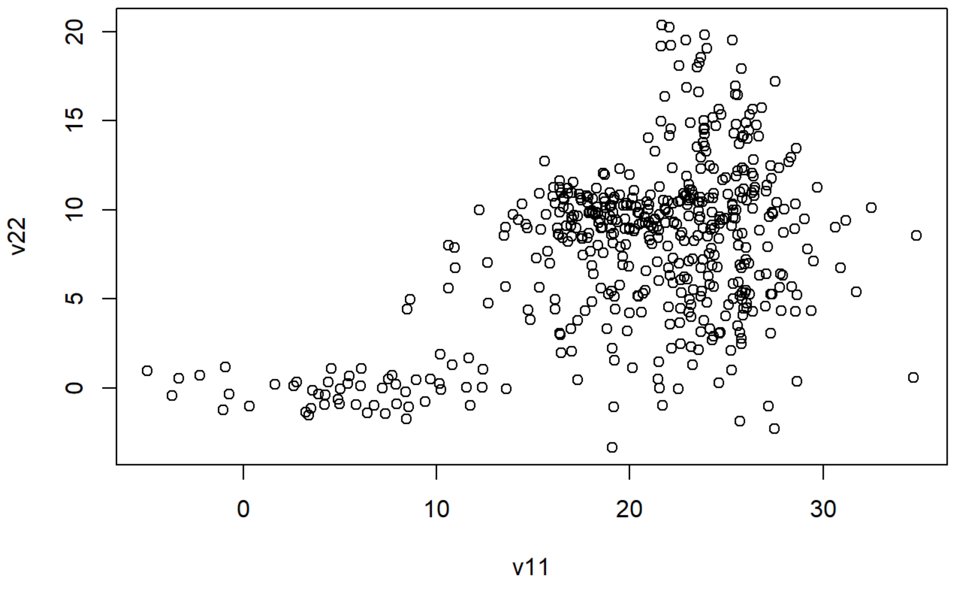
```
table(results.or1)
```

```
## results.or1
##  2  4  5  6  7  8 
##  2 15 51 20 11  1
```

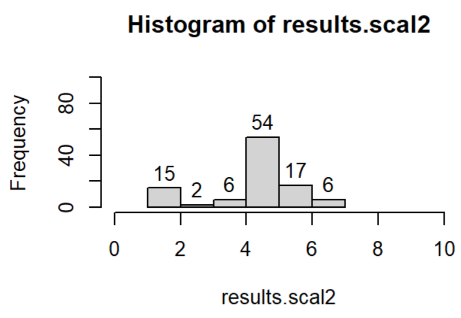
```
table(results.or2)
```

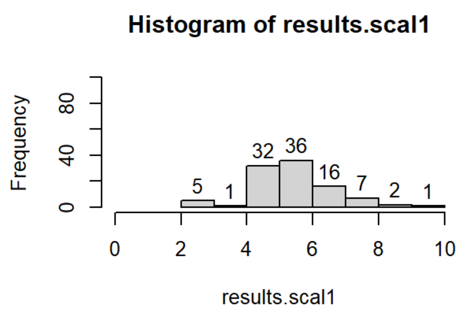
```
## results.or2
##  2  3  4  5  6  7 
##  3  4 39 48  5  1
```

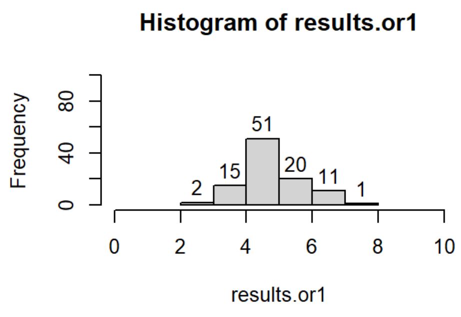
```
plot(dataset[[i]])
```

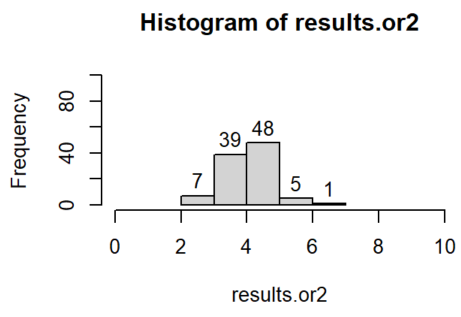


```
#the plot highlights that the best number of clusters is 3

par(mfrow=c(2,2))
hist.scal22<-hist(results.scal2,xlim=c(0,10), ylim=c(0,100), labels = TRUE, breaks = 7)
#for the clusGap function with spaceH0 ="scaledPCA" and SE.factor = 2 there are
#1 dataset for which the best number of cluster is 1,
#14, with k=2,
#2 with k=3,
#6 with k=4 and k=7,
#54 with k=5,
#17 with k=6.
hist.scal11<-hist(results.scal1,xlim=c(0,10), ylim=c(0,100), labels = TRUE, breaks = 7)
#for the clusGap function with spaceHO= "scaledPCA" and SE.factor = 1 there are
#5 with k=2,
#1 with k=4 and k=10,
#32 with k=5 ,
#36 with k=6,
#16 with k=7,
#7 with k=8 and
#2 with k=9.
hist.or11<-hist(results.or1,xlim=c(0,10), ylim=c(0,100), labels = TRUE, breaks = 7)
#for the clusGap function with spaceH0 ="original" and SE.factor = 1 there are
#2 datasets with k=2,,
#15 with k=4,
#51 with k=5,
#20 with k=6,
#11 with k=7,
#1 with k=8.
hist.or22<-hist(results.or2,xlim=c(0,10), ylim=c(0,100), labels = TRUE, breaks = 7)
```









```
#for the clusGap function with spaceH0 ="original" and SE.factor = 2 there are
#2 datasets with k=2,
#4 with k=3,
#39 with k=4,
#48 with k=5,
#5 with k=6,
#1 with k=7.
```

Generally we can say that k=5 is the best number of cluster for the majority of datasets for all versions of clusGap function excepted for the version with spaceHO="scaledPCA" and SE.factor=1. In this case we have a prevalence for k=6. The best result is given by the version with spaceHO="original" and SE.factor=2. Here the big majority of the datasets are caracterized by k=5 while the remaining are for the greater part divided in 4 clusters and in a negligible way, in some other possible values for k. The worse result is given by the version of the clusGap function with spaceH0="scalePCA" and SE.factor=1 since the datasets can be separated by the algorithm in more possible values of k with respect to other alogrithms.

5.

a. A political scientist wants to cluster the respondents of a questionnaire using a distance-based method. The questionnaire has preference questions with two response options of the type do you prefer the current level of taxes, or do you prefer higher taxes with all money from the higher taxes being invested in the health system?" Generally the option I prefer the current situation" is coded 0 and the option that suggests something else is coded 1. Would you prefer the simple matching distance or the Jaccard distance here? Why?

The simple matching and the Jaccard are distances computed for binary and categorical variables. The main difference is that in the second one the joint absence of a certain characteristic is meaningless. An example can be made considering the differences between a cat and a snake, if both haven't got wings, it can't be viewed as a factor of their similarity. In this case it is better if we not consider it in the computation of the index. So only the joint presence is meaningful. prefer current situation=0 higher taxes for health=1 In this case I suggest to use the Jaccard distance since the I think that the only specification of 'current situation' is not informative, From this we can only know that people are not agree to pay more taxes in order to use the difference with the previous one for the health system. If an other alternative was proposed maybe the simple matching distance would have been more adequate.

b.Geographers want to cluster areas in the Swiss alps according to danger from avalanches in order to produce a map with different colour codes for different danger levels, using a distance-based method. Their variables are, all for the year 2019: (i) the number of avalanches in the area, (ii) the average percentage of the area covered by an avalanche, (iii) number of persons injured or dead in incidents involving avalanches in the area, (iv) Swiss Francs investment in the security of the ski slopes in the area, (v) Swiss Francs budget for emergency rescue in the area. Would you prefer the Euclidean distance on raw data, the Euclidean distance on scaled data, the Manhattan distance on raw data, the Manhattan distance on scaled data, or the Mahalanobis distance for these data? Why? (Probably more than one option can convincingly be argued.)

Generally the Manhattan distance is preferred with respect to the euclidean distance when there is an high dimensionality of data. It is efficient when the majority of the data are discrete or binary. The euclidean distance give more weight on variables with higher distance with respect to the Manhattan distance, because the first one is the square root of the sum of differences between two data points. It is an easy way to measure the similarity between points when the data are low-dimensional, but it is not scale equivariant so it needs the variables are normalized, as the Manhattan distance. The Mahalanobis distance also is scale by definition, so it is not necessary to standardize data. The Mahalanobis distance is preferable to the Euclidean distance because it takes into account variances and covariances among the variables. In other word this kind of distance try to explain how variables are correlated. When you use Euclidean distance, you assume that the clusters have the identity covariance matrix, then using Mahalanobis over Euclidean will perform better modeling.

Since the variables are expressed in different unite of measures I suggest to use the scaled data in order to standardize variables and to avoid problems related with their scales. The dataset we deal with is low dimensional since it is composed by only 5 variables, of which only the average percentage of the area covered by an avalanche is very continuous. So I suggest to use the manhattan distance instead of the euclidean distance in order to avoid the problem specified before. Also the mahalanobis one can be a good choice because it doesn’t need the standardization of the data and it can perform well, returning results similar to the euclidean distance ones. Moreover the possibility of taking into account the correlation between variables will be so interesting for this type of analysis.