Projet_mad

FOISSEY Guillemette, BENBOUZID Zakaria

```
## Exercice 1

# 1)

distance = function(x,y){ #distance entre 2 points de l'espace
  return(sqrt(sum((x - y)**2)))
}

nearest = function(centers, point, C){ #points les plus proches d'un certain centre
  n = length(centers)
  tmp = list(point)
  s2 = 0
  for (i in 1:n){
    d = distance(point,unlist(centers[i]))
    if (d != 0){
      s1 = 0
      for (j in C){
        if ((distance(unlist(centers[i]),unlist(j)) <= d) & (distance(unlist(centers[i]),unlist(j)) !=
          s1 = s1 + 1
        }
        if (s1 == 0){
          s2 = s2 + 1
          tmp[[s2]] = unlist(centers[i])
        }
      }
    }
  }
  return(tmp)
}

somme = function(liste,length){ #somme de 2 listes
  s = rep(0,length)
  for (i in 1:length){
    for(j in 1:length(liste)){
      s[i] = s[i] + unlist(liste[j])[i]
    }
  }
  return(s)
}

kpp = function(k, X){
  C = list()
  X_tmp = X
  c1 = X_tmp[[sample(1:length(X_tmp),1)]]
```

```
index = match(c1,X_tmp)
X_tmp[index] = NULL
C[[1]] = c1
p = c()
sum = 0
for (i in 1:length(X_tmp)){
  sum = sum + distance(unlist(X_tmp[i]),unlist(c1))**2
}
for (i in 1:length(X_tmp)){
  p = c(p,(distance(unlist(X_tmp[i]),unlist(c1))**2)/sum)
}
c2 = X_tmp[[sample(1:length(X_tmp),1, prob = p)]]
index = match(c2,X_tmp)
X_tmp[index] = NULL
C[[2]] = c2
p = c()
sum = 0
ci = c1
for (i in 3:k){ #on choisit un centre avec une probabilité p, k fois
  for (j in 1:length(X_tmp)){
    sum = sum + distance(unlist(X_tmp[j]),unlist(ci))**2
  }
  for (j in 1:length(X_tmp)){
    p = c(p,(distance(unlist(X_tmp[j]),unlist(ci))**2)/sum)
  }
  ci = X_tmp[[sample(1:length(X_tmp),1, prob = p)]]
  index = match(ci,X_tmp)
  X_tmp[index] = NULL
  C[[i]] = ci
  sum = 0
  p = c()
}
C_bis = list()
for (i in 1:k){ #on pratique l'algorithme kmeans
  Ci = nearest(X,unlist(C[i]),C)
  ci = somme(Ci,length(Ci[[1]]))/length(Ci)
  #print(ci)
  C_bis[[i]] = ci
}
while(TRUE){
  a = 0
  for (i in 1:length(unlist(C))){
    if (unlist(C_bis)[i] != unlist(C)[i]){
      a = a+1
    }
  }
  if (a == 0){
    return(C_bis)
  }
  else{
    C = C_bis
    C_bis = list()
    for (i in 1:k){
```

```r
        Ci = nearest(X,unlist(C[i]),C)
        ci = somme(Ci,length(Ci[[1]]))/length(Ci)
        C_bis[[i]] = ci
      }
    }
  }
}

#test = list(c(1,0),c(5,2),c(1,9),c(5,7),c(9,6),c(0,8),c(1,5),c(6,2))


phi_kpp = function(k, X){ # fonction phi pour kmeans ++
  C = kpp(k, X)
  s = 0
  for (i in 1:length(X)){
    a = c()
    for (j in 1:length(C)){
      a = c(a,norm(unlist(X[i]) - unlist(C[j]), type = "2")**2)
    }
    s = s + min(a)
  }
  return(s)
}

phi_kmeans = function(X,k){ # fonction phi pour kmeans
  C = kmeans(X, k)
  s = 0
  for (i in 1:length(X)){
    a = c()
    for (j in 1:length(C)){
      a = c(a,norm(unlist(X[i]) - unlist(C[j]), type = "2")**2)
    }
    s = s + min(a)
  }
  return(s)
}
```

```r
library(MASS)
```

```
## Warning: le package 'MASS' a été compilé avec la version R 4.1.3
```

```r
#2 )

norm1025 = function(n, d){
  tmp_10 = list()
  tmp_25 = list()

  for (i in 1:10){
    tmp = c()
    for (j in 1:d){
      tmp = c(tmp,sample(0:500,size = 1))
    }
    tmp_10[[i]] = tmp
  }
  for (i in 1:25){
    tmp = c()
    for (j in 1:d){
      tmp = c(tmp,sample(0:500,size = 1))
    }
    tmp_25[[i]] = tmp
  }

  variance = diag(d)

  NORM_10 = list()
  NORM_25 = list()

  compt = 1
  for (i in 1:10){
    a = mvrnorm(n,unlist(tmp_10[i]),variance)
    for (j in 1:n){
      NORM_10[[compt]] =  a[j,]
      compt = compt + 1
    }
  }
  compt = 1
  for (i in 1:25){
    a = mvrnorm(n,unlist(tmp_25[i]),variance)
    for (j in 1:n){
      NORM_25[[compt]] =  a[j,]
      compt = compt + 1
    }
  }
  return(list(NORM_10,NORM_25))
}

#norm1025(10000,5)
```

```
# 3) ## test pour n=10, sinon l'algorithme prend trop de temps pour kmeans ++ (synonyme sûrement d'une

test_n_10_kpp = function(n,d){
  vect = c(10,20)

  resultat = list()

  average_phi = list()
  min_phi = list()
  average_T = list()

  NORM_10 = norm1025(n,d)[[1]]

  for (k in 1:2){
    a_phi = c()
    a_T = 0
    for (i in 1:10){
      a_phi = c(a_phi,phi_kpp(vect[k],NORM_10))
      a_T = system.time(phi_kpp(vect[k],NORM_10))[3]
    }
    average_phi[[k]] = sum(a_phi)/10
    min_phi[[k]] = min(a_phi)
    average_T[[k]] = sum(a_T)/10
  }
  resultat[[1]] = average_phi
  resultat[[2]] = min_phi
  resultat[[3]] = average_T
  return(resultat)
}

test_n_10_kmeans = function(n,d){
  vect = c(10,20)

  resultat = list()

  average_phi = list()
  min_phi = list()
  average_T = list()

  NORM_10 = norm1025(n,d)[[1]]

  for (k in 1:2){
    a_phi = c()
    a_T = 0
    for (i in 1:10){
      a_phi = c(a_phi,phi_kmeans(unlist(NORM_10),vect[k]))
      a_T = system.time(phi_kmeans(unlist(NORM_10),vect[k]))[3]
    }
    average_phi[[k]] = sum(a_phi)/10
    min_phi[[k]] = min(a_phi)
    average_T[[k]] = sum(a_T)/10
  }
  resultat[[1]] = average_phi
  resultat[[2]] = min_phi
  resultat[[3]] = average_T
  return(resultat)
}

test_n_10_mclust = function(n,d){

  resultat = list()

  average_phi = list()
  min_phi = list()
  average_T = list()

  NORM_10 = norm1025(n,d)[[1]]

  a_phi = c()
  a_T = 0
  for (i in 1:10){
    a_phi = c(a_phi,phi_mclust(unlist(NORM_10),))
    a_T = system.time(phi_mclust(unlist(NORM_10)))[3]
  }
  average_phi[[k]] = sum(a_phi)/10
  min_phi[[k]] = min(a_phi)
  average_T[[k]] = sum(a_T)/10

  resultat[[1]] = average_phi
  resultat[[2]] = min_phi
  resultat[[3]] = average_T
  return(resultat)
}


test_n_10_kpp(10,5)
```

```
## [[1]]
## [[1]][[1]]
## [1] 3346900
##
## [[1]][[2]]
## [1] 1652533
##
##
## [[2]]
## [[2]][[1]]
## [1] 2698316
##
## [[2]][[2]]
## [1] 849092.3
##
##
## [[3]]
## [[3]][[1]]
## [1] 0.222
##
## [[3]][[2]]
## [1] 0.261
```

```
test_n_10_kmeans(10,5)
```

```
## [[1]]
## [[1]][[1]]
## [1] 39066686
##
## [[1]][[2]]
## [1] 39090740
##
##
## [[2]]
## [[2]][[1]]
## [1] 38874255
```

```
##
## [[2]][[2]]
## [1] 38874255
##
##
## [[3]]
## [[3]][[1]]
## [1] 0.072
##
## [[3]][[2]]
## [1] 0.023
```

```
## Exercice 2


# 1)

library(mclust)
```

```
## Warning: le package 'mclust' a été compilé avec la version R 4.1.3
```

```
## Package 'mclust' version 6.0.0
## Type 'citation("mclust")' for citing this R package in publications.
```

```
library(dplyr)
```

```
## Warning: le package 'dplyr' a été compilé avec la version R 4.1.3
```

```
##
## Attachement du package : 'dplyr'
```

```
## L'objet suivant est masqué depuis 'package:MASS':
##
##     select
```

```
## Les objets suivants sont masqués depuis 'package:stats':
##
##     filter, lag
```

```
## Les objets suivants sont masqués depuis 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(FactoMineR)
```

```
## Warning: le package 'FactoMineR' a été compilé avec la version R 4.1.3
```

```
library(factoextra)
```

```
## Warning: le package 'factoextra' a été compilé avec la version R 4.1.3
## Le chargement a nécessité le package : ggplot2

## Warning: le package 'ggplot2' a été compilé avec la version R 4.1.3

## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
data = iris
X = data[,1:4]

fit_kpp = kpp(15,list(data[,1],data[,2],data[,3],data[,4]))
fit_kmeans = kmeans(X,15)

kmeansBIC = function(fit){

  m = ncol(fit$centers)
  n = length(fit$cluster)
  k = nrow(fit$centers)
  D = fit$tot.withinss
  return(D + log(n)*m*k)
}

BIC_kmeans = kmeansBIC(fit_kmeans)

hc1 <- hc(X, modelName = "VVV", use = "SVD")
BIC1 <- mclustBIC(X, initialization = list(hcPairs = hc1))
hc2 <- hc(X, modelName = "VVV", use = "VARS")
BIC2 <- mclustBIC(X, initialization = list(hcPairs = hc2))
hc3 <- hc(X, modelName = "EEE", use = "SVD")
BIC3 <- mclustBIC(X, initialization = list(hcPairs = hc3))

BIC <- mclustBICupdate(BIC1, BIC2, BIC3)

summary(BIC)
```

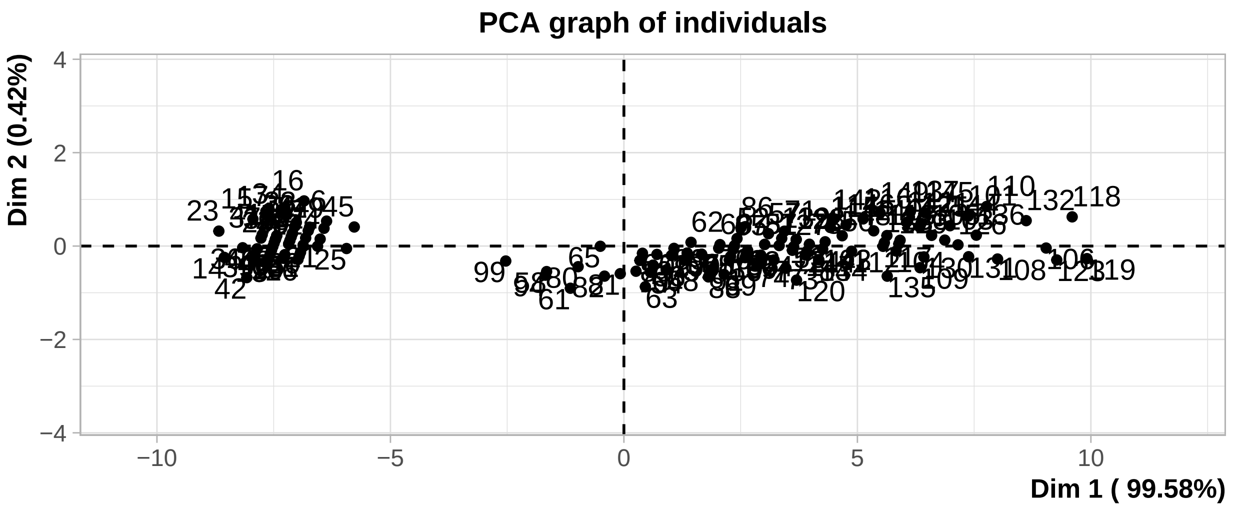
```
## Best BIC values:
##              VEV,2        VEV,3      VVV,2
## BIC      -561.7285 -562.5514380 -574.01783
## BIC diff    0.0000   -0.8229759  -12.28937
```

```
BIC_kmeans
```

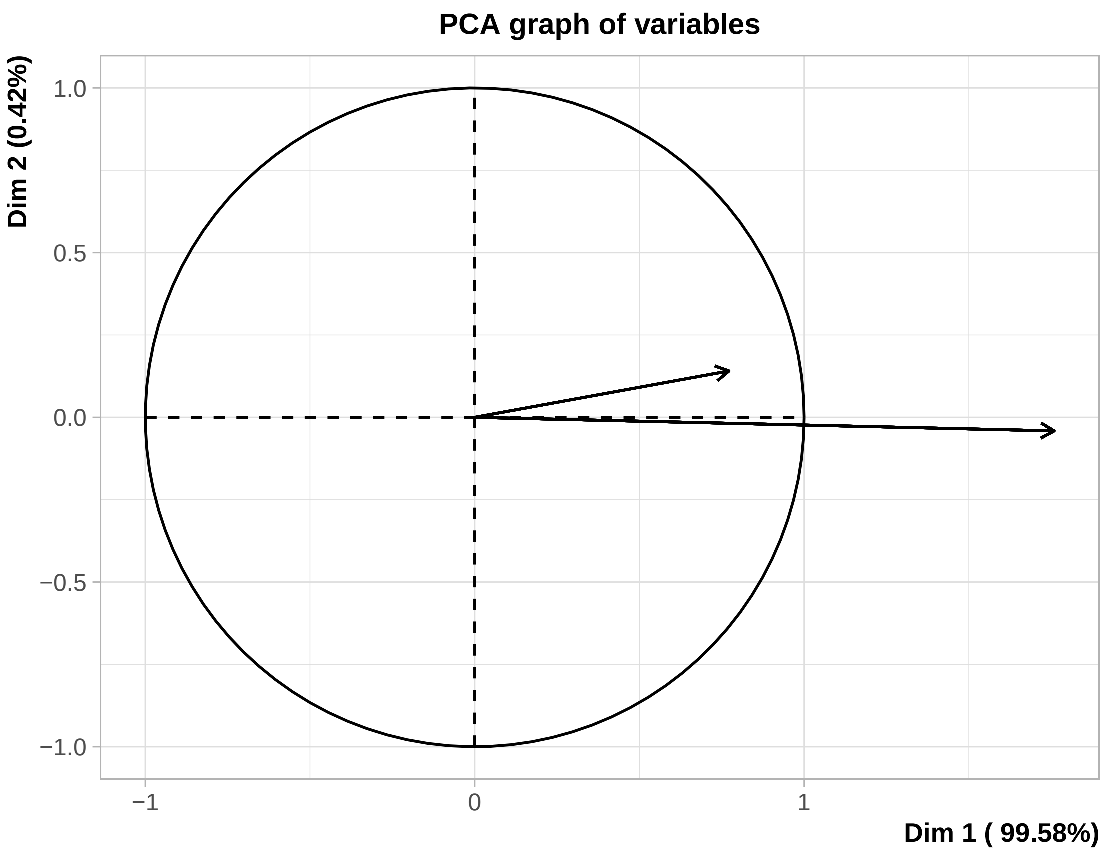
```
## [1] 321.124
```

```
# Plus le BIC est faible, plus le modèle est intéressant. Ainsi, Mclust reste bien le modèle le plus in
```

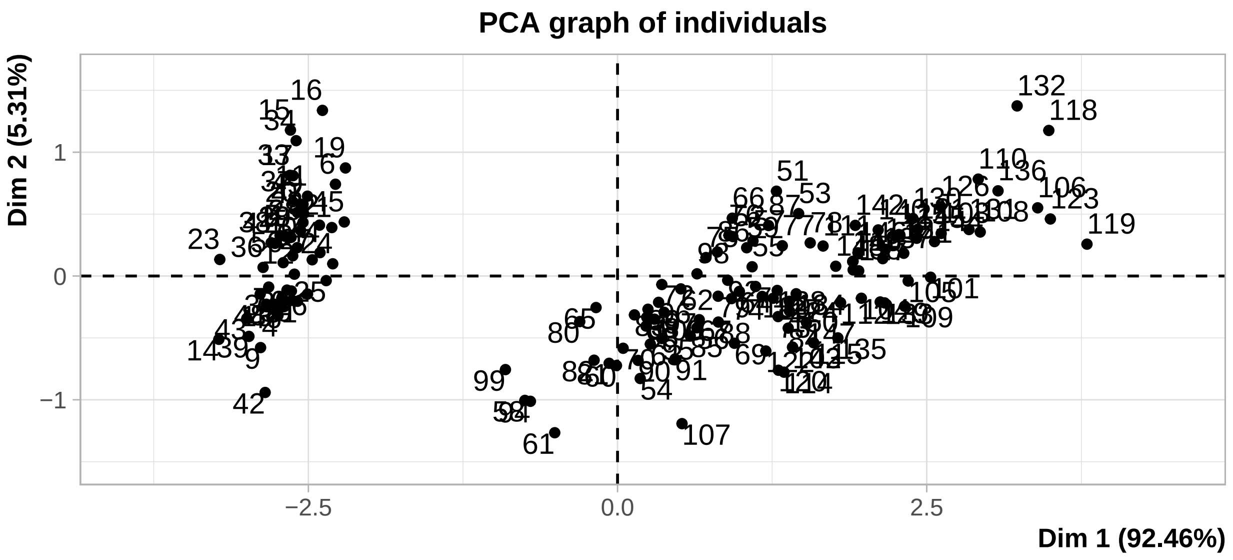
```
# 2)

res.pca1 = PCA(fit_kpp,scale.unit = FALSE,ncp = 5)
```

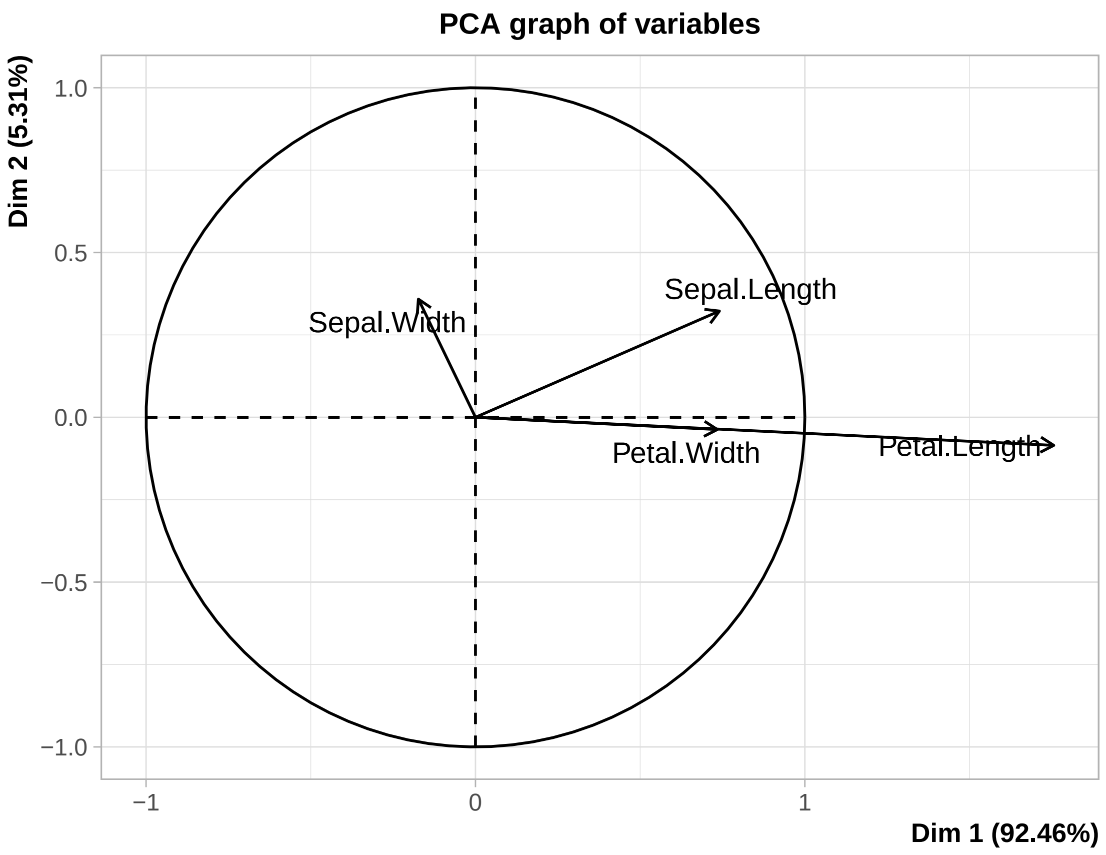


```
## Warning: ggrepel: 15 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



```
res.pca2 = PCA(fit_kmeans$cluster,scale.unit = FALSE,ncp = 5)
res.pca3 = PCA(X,scale.unit = FALSE,ncp = 5)
```


PCA graph of individuals
Dim 2 (5.31%)
Dim 1 (92.46%)
−2.5
0.0
2.5
1
0
−1



3) D'après le graphe des variables ACP, on comprend que comme la taille de pétale, la profondeur des pétales ainsi que la taille des sépales sont proches du cercle de corrélation, donc ces variables sont bien représentés. L'angle proche des 2 variables de pétale montrent que ces dernières sont fortement corrélées entre elles. Cependant, comme il y a un angle droit entre les 2 varibales de spéales, ces dernières sont indépendantes entre elles. Sur le graphe des individuels, les points qui sont très proches entre eux sont fortement corrélés.