MIXCLUSTERING_Nour Dass HAMMADI-Farida BENCHALAL

Nour Dass HAMMADI & Farida BENCHALAL

14/03/2021

```
#' Mixclustering

#' @param X
#' @param clust
#' @param iterations
#' @param initialisation

#' @return
#' @export


library(dplyr)

## Warning: package 'dplyr' was built under R version 3.6.3

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(tidyverse)

## Warning: package 'tidyverse' was built under R version 3.6.3

## -- Attaching packages --------------------------------------- tidyverse
1.3.0 --

## v ggplot2 3.3.3     v purrr   0.3.3
## v tibble  3.0.3     v stringr 1.4.0
## v tidyr   1.0.2     v forcats 0.4.0
## v readr   1.3.1

## Warning: package 'ggplot2' was built under R version 3.6.3

## Warning: package 'tibble' was built under R version 3.6.3

## Warning: package 'readr' was built under R version 3.6.3
## -- Conflicts ------------------------------------------
tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(gtools)
```

```
## Warning: package 'gtools' was built under R version 3.6.3
```

```r
library(FactoMineR)
```

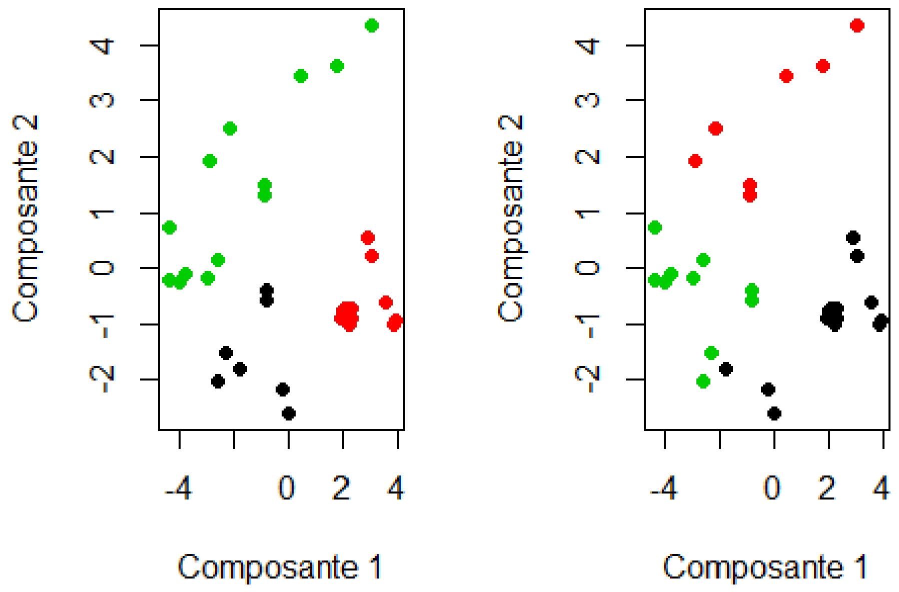
```
## Warning: package 'FactoMineR' was built under R version 3.6.3
```

```r
library(bayess)
```

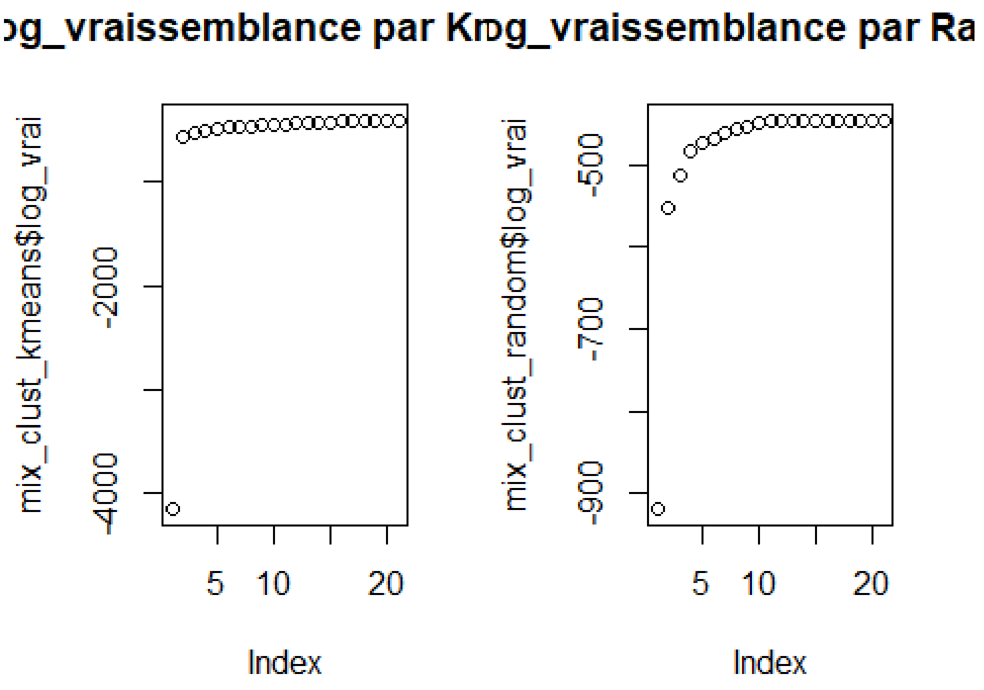
```
## Warning: package 'bayess' was built under R version 3.6.3

## Loading required package: MASS

## Warning: package 'MASS' was built under R version 3.6.3

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

## Loading required package: mnormt

## Loading required package: gplots

##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess

## Loading required package: combinat

##
## Attaching package: 'combinat'

## The following object is masked from 'package:utils':
##
##     combn

##
## Attaching package: 'bayess'

## The following object is masked from 'package:gtools':
##
##     rdirichlet
```

```r
library(mvtnorm)

## Warning: package 'mvtnorm' was built under R version 3.6.3

library(devtools)

## Warning: package 'devtools' was built under R version 3.6.3

## Loading required package: usethis

## Warning: package 'usethis' was built under R version 3.6.3

## Error in get(genname, envir = envir) : objet 'testthat_print' introuvable

My_Mix_clustering <-
  function(X, clust, iterations, initialisation) {
    set.seed(123)
    n <- nrow(X)    # nombre de lignes
    col <- ncol(X)  # nombre de colonnes

    ## S?paration des variables qualitatives et variables quantitatives:

    ## Varibales quantitatives
    donnees_quati <- as.matrix(X %>% select_if(is.numeric))
    col <- ncol(donnees_quati)
    ## Variables qualitatives
    donnees_quali <- X %>% select_if(is.factor)
    col_quali <- ncol(donnees_quali)

    mod <- sapply(seq(col_quali), function(i) {
      length(levels(donnees_quali[, i]))
    })
    # initialisation des objets
    prop <- matrix(NA, iterations + 1, clust)
    mu <- array(NA, dim = c(iterations + 1, clust, col))
    sigma <- array(NA, dim = c(iterations + 1, clust, col, col))
    alpha <- array(NA, dim = c(iterations + 1, clust, col_quali))
    mode(alpha) <- "list"
    log_vrai <- rep(0, iterations + 1)
    # initialisation de l'algorithme => random/kmeans
    if (initialisation == 'random') {
      prop[1,] <- rdirichlet(1, par = rep(1, clust))
      mu[1, ,] <- donnees_quati[sample(1:n, clust),]
      for (k in 1:clust)
        sigma[1, k, ,] <- rWishart(1, 8, var(donnees_quati))
    }
    if (initialisation == 'kmeans') {
      z <- kmeans(donnees_quati, clust)$clust
      for (k in 1:clust) {
        prop[1, k] <- mean(z == k)
        mu[1, k,] <- colMeans(donnees_quati[which(z == k),])
        sigma[1, k, ,] <- var(donnees_quati[which(z == k),])
      }
    }
    # initialisation des parametres
    for (k in 1:clust) {
      for (i in 1:col_quali) {
        alpha[1, k, i] <- list(rdirichlet(1, rep(1, mod[i])))
        names(alpha[1, k, i][[1]]) <- levels(donnees_quali[, i])
      }
    }
    # calcul de log de vraisemblance
    for (i in 1:n) {
      tmp <- 0
      for (k in 1:clust) {
        fk <- 1
        for (j in 1:col_quali) {
          fk <-fk * alpha[1, k, j][[1]][donnees_quali[i, j]]
        }
        tmp <-
          tmp + prop[1, k] * (fk * dmvnorm(donnees_quati[i,], mu[1, k,],
sigma[1, k, ,]))
      }
      log_vrai[1] <- log_vrai[1] + log(tmp)
    }
    # algorithme EM
    for (iter in 1:iterations) {
      #E-step
      tik <- matrix(NA, n, clust)
      for (k in 1:clust) {
        fk <- 1
        for (i in 1:col_quali) {
          fk <- fk * alpha[iter, k, i][[1]][donnees_quali[, i]]
        }
        tik[, k] <-
          prop[iter, k] * (fk + dmvnorm(donnees_quati, mu[iter, k,],
sigma[iter, k, ,]))
      }
      tik <- tik / rowSums(tik)
      #M-step
      for (k in 1:clust) {
        nk <- sum(tik[, k])
        prop[iter + 1, k] <- nk / n
        mu[iter + 1, k,] <- colSums(tik[, k] * donnees_quati) / nk
        sigma[iter + 1, k, ,] <- Reduce('+', lapply(1:n, function(m) {
          tik[m, k] * (donnees_quati[m,] - mu[iter + 1, k,]) %*%
t(donnees_quati[m,] - mu[iter + 1, k,]) / nk
        }))
        for (i in 1:col_quali) {
          alpha[iter + 1, k, i] <- list(sapply(1:mod[i], function(a) {
            sum(tik[, k] * (donnees_quali[, i] == levels(donnees_quali[,
i])[a])) / nk
          }))
          names(alpha[iter + 1, k, i][[1]]) <- levels(donnees_quali[, i])
        }
      }
      #calcul de log vraisemblance
      for (i in 1:n) {
        tmp <- 0
        for (k in 1:clust) {
          fk <- 1
          for (j in 1:col_quali) {
            fk <-fk * alpha[iter + 1, k, j][[1]][donnees_quali[i, j]]
          }
          tmp <-
            tmp + prop[iter + 1, k] * (fk * dmvnorm(donnees_quati[i,],
mu[iter + 1, k,], sigma[iter + 1, k, ,]))
        }
        log_vrai[iter + 1] <- log_vrai[iter + 1] + log(tmp)
      }
    }
    z <- max.col(tik)
    BIC <- log_vrai[iterations + 1] - clust / 2 * log(n)
    ICL <-  BIC - sum(tik * log(tik), na.rm = TRUE)

    return(
      list(
        prop = prop,
        mu = mu,
        sigma = sigma,
        clust = clust,
        log_vrai = log_vrai,
        z = z,
        BIC = BIC,
        ICL = ICL
      )
    )
  }


###############################


x <- mtcars
x$vs = as.factor(x$vs)
x$am = as.factor(x$am)
x$gear = as.factor(x$gear)

mix_clust_kmeans <- My_Mix_clustering(x, 3, 20, 'kmeans')
mix_clust_random <- My_Mix_clustering(x, 3, 20, 'random')
```

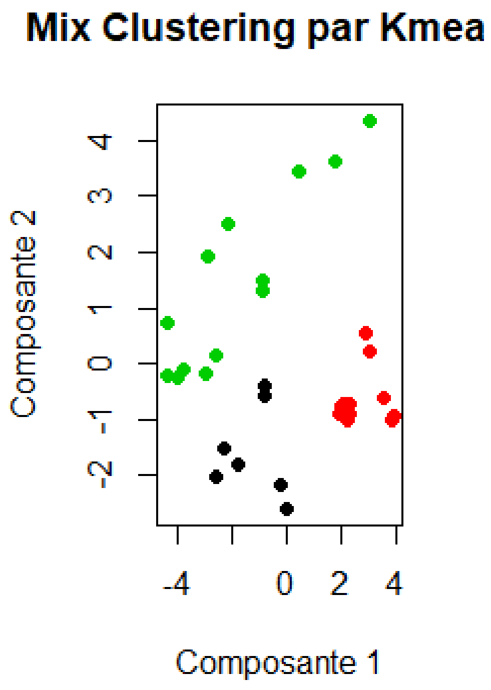
```r
##############################


affiche_graphe <- function(resultat_mixclust){
  res.famd <- FAMD (x, ncp = 5, graph = FALSE)
  plot(res.famd$ind$coord[,1], res.famd$ind$coord[,2],
       # col = res$z, pch=19, xlab = "Composante 1", ylab = "Composante 2")
       col =  resultat_mixclust$z, pch=19, xlab = "Composante 1", ylab =
"Composante 2")
}





set.seed(123)
par(mfrow = c(1,2))
affiche_graphe(mix_clust_kmeans)
affiche_graphe(mix_clust_random)
```



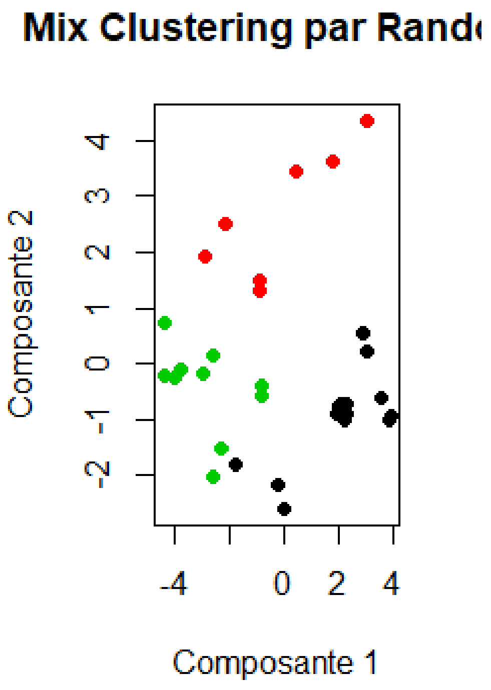
```r
par(mfrow = c(1,2))
plot(mix_clust_kmeans$log_vrai, main = "Log_vraissemblance par Kmeans")
plot(mix_clust_random$log_vrai, main = "Log_vraissemblance par Random"
     )
```



```
res.famd <- FAMD (x, ncp = 5, graph = FALSE)
par(mfrow = c(1,2))
plot(res.famd$ind$coord[,1], res.famd$ind$coord[,2],
     col = mix_clust_kmeans$z, pch=19, xlab = "Composante 1", ylab =
"Composante 2", main = "Mix Clustering par Kmeans")
plot(res.famd$ind$coord[,1], res.famd$ind$coord[,2],
     col = mix_clust_random$z, pch=19, xlab = "Composante 1", ylab =
"Composante 2", main = "Mix Clustering par Random")
```



Mix Clustering par Kmea
Composante 2
Composante 1
-4
0
2
4
-2
-1
0
1
2
3
4



Mix Clustering par Rando
Composante 2
Composante 1
-4
0
2
4
-2
-1
0
1
2
3
4