Projet kmeans

Manzione Amaury,Tireau Antoine

Algorithme kmeans++

Question 1

On code dans cette question l'algorithme kmeans++ comme décrit page 6 de l'article. Les fonctions les plus importantes que j'ai codé dans cette question sont :

- La fonction centers_init_bis qui calcule les centres initiaux de l'algorithme kmeans comme décrit dans l'article.
- La fonction set_clusters_bis qui assigne à chaque point un cluster en fonction du centre le plus proche.
- La fonction update_centers_bis qui recalcule les centres après avoir reassigné à des clusters les points.
- La fonction fit_plus_bis qui implémente l'algorithme kmeans.

On code au préalable une fonction distance_min qui renvoie :

$$\min_{c \in C} ||x - c||^2$$

avec C l'ensemble des centres et l'indice du c minimum.

```
distance_min <- function(x,centers){
  K = nrow(centers) # nombre de clusters
  distance = c(sum((x-centers[1,])**2),1) #distance avec le premier centre et sa position
  if( K != 1){
    for(j in 2:K){
      new_dist = sum((x-centers[j,])**2)
      if( new_dist < distance[1] ){
        # on update la position si on trouve un nouveau minimum
        distance = c(new_dist,j)
      }
    }
  }
  return(distance)
}
```

```
# retourne les centres initialises selon kmeans++
centers_init_bis <- function(X,K){
  X = as.matrix(X) # transforme X en une matrice pour une manipulation plus facile
  n = nrow(X) # nombre d'observations
  p = ncol(X) # dimension

  centers = matrix(0,nrow = K,ncol = p) # on initialise la matrice des centres
  centers[1,] = X[sample(n,size=1),] # on choisit le premier centre au hasard
```

```
  for( k in 2:K){
    distance_min = c()

    for(i in 1:n){
      # pour chaque x on calcule sa distance minimum par rapport aux centres actuels
      # on stocke la distance dans une liste
      x = X[i,]
      centre = centers[1:k,]
      dist = distance_min(x,centre)[1]
      distance_min = append(distance_min,dist)
    }

    # liste des probas
    prob = distance_min / sum(distance_min)
    new_center = sample(nrow(X),prob = prob,size = 1)
    centers[k,] = X[new_center,]
  }

  return(centers)
}
```

Note : On n'a pas besoin de retirer x, si x est choisi comme centre, car la probabilité de le choisir à la prochaine itération sera nulle.

```
# assigne chaque points à des clusters
set_clusters_bis <- function(X,centers){
  n = nrow(X)
  p = ncol(X)

  # data frame contenant les observations et un vecteur colonne dont les éléments
  # sont les index des centres
  clusters = data.frame(X, label = rep(0,n))


  for(i in 1:n){
    x = X[i,]
    # on assigne x au cluster le plus proche
    clusters$label[i] = distance_min(x,centers)[2]
  }


  return(clusters)
}
```

```
#update la position des centres
update_centers_bis <- function(X,df,centers){
  X = as.matrix(X)
  p = ncol(X)

  K = nrow(centers)

  for(i in 1:K){
```

```
    # pour chaque centre on récupère tout les x qui ont été assigné à ce centre
    ind = which(df$label == i,arr.ind = TRUE)
    for(j in 1:p){
      # on calcule le nouveau centre en moyennant avec les x assignés.
      centers[i,j] = mean(X[ind,j])
    }
  }

  return(centers)
}
```

```
# retourne des clusters optimaux calculés selon l'algorithme kmeans++
fit_plus_bis <- function(X,K,max_iterations){

  old_centers = centers_init_bis(X,K) # initalisation des centres


  old_clusters = set_clusters_bis(X,old_centers)


  for(iteration in 1:max_iterations){
    #on assigne chaque x à des clusters
    centers = update_centers_bis(X,old_clusters,old_centers)
    #on update les valeurs des centres
    clusters = set_clusters_bis(X,centers)

    #on arrete si le nouveau cluster est identique au dernier
    if(identical(clusters$label,old_clusters$label) ){
      return(centers)
    }

    else{
      # on garde en mémoire la valeur actuelle des centres et des clusters
      # pour la comparer à la prochaine
      old_clusters = clusters
      old_centers = centers
    }

  }

  return(centers)

}
```

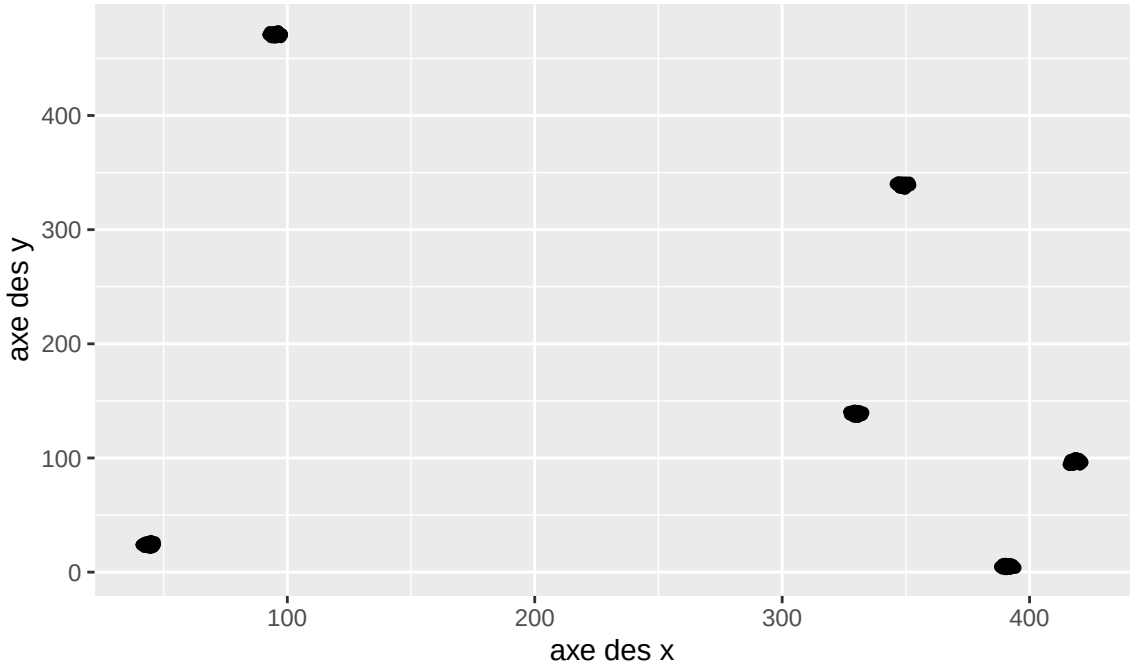
Question 2

On implémente dans cette question les échantillons RNORM-N comme décrit dans l'article. On sélectionne N centres dans un hypercube de dimension d. Ensuite on construit un échantillon de taille n dans $\mathbb{R}^d$ tel que chaque élément de cet échantillon suit une loi normale multivariée centrée en un des centres et de matrice de variance-covariance I_d.

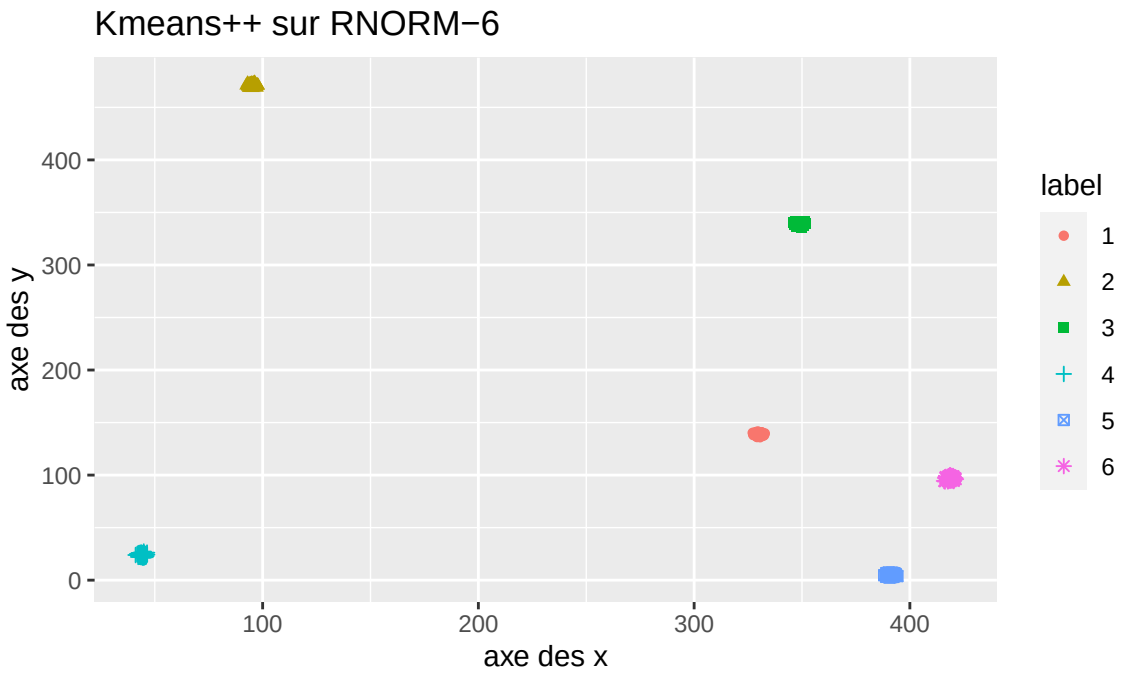
```r
library(MASS)
# genre un RNORM en dimension d avec K centres et de taille n
gen_norm <- function(K,d,n){
  centres = matrix(0,nrow = K,ncol = d) #initialisation de la matrice des centres
  for(i in 1:K){
    #chaque centre est un vecteur de taille d avec des composantes pouvant
    # variees entre 0 et 500
    centres[i,] = sample(0:500,size=d)
  }

  # on construit l'échantillon RNORM
  X = matrix(0,nrow = n,ncol = d)
  for(i in 1:n){
    # chacun des éléments de X suit une loi normale multivariee centre en un des centres
    # et de matrice de variance covariance l'identité.
    X[i,] = mvrnorm(n = 1, mu = centres[sample(1:K,size=1),],Sigma = diag(1,d,d))
  }

  return(X)
}
```

On plot un RNORM-6 en dimension 2 pour voir si gen_norm génère bien des clusters distints.

```r
library(ggplot2)

X = gen_norm(6,2,1000)

ggplot(data = NULL,aes(x = X[,1], y = X[,2]))+
  geom_point()+
  xlab("axe des x")+
  ylab("axe des y")
```



```r
centers_test = fit_plus_bis(X,6,100)

X_fit = set_clusters_bis(X,centers_test)


X_fit$label = as.factor(X_fit$label)

ggplot(X_fit, aes(x=X_fit[,1], y=X_fit[,2], color=label,shape=label)) +
  geom_point()+
  xlab("axe des x")+
  ylab("axe des y")+
  ggtitle(paste("Kmeans++","sur","RNORM-6"))
```



On voit qu'on a bien 6 clusters distints et que l'algorithme kmeans++ a bien identifié ces clusters.

Question 3

On compare notre algorithme kmeans++ au kmeans standard de R, dans un format similaire à celui de l'article. On calcule au préalable une fonction coût (inertie totale du nuage de points moyennée par la taille de l'échantillon).

```r
cost <- function(obs,cen){

  p = nrow(obs)
  count = 0
  for( i in 1:p ){
    count = count + distance_min(obs[i,],cen)[1]
  }
  return(count/p)
}
```

Table 1: Résultats expérimentaux sur NORM-10 n=1000, d=5

	Average.cost.kmean_plus	Average.cost.kmean	minimum.cost.kmean_plus	minimum.cost.kmean
K- 10	4.96	9074.21	4.83	4.88
K- 25	3.88	541.65	3.71	3.85

Pour RNORM-10

```
list_mean_plus = c()
list_mean = c()
list_min_plus = c()
list_min = c()

list_cost = c()
list_cost2 = c()
for( i in 1:20 ){
  data_10 = gen_norm(10,5,1000)
  centers_i = fit_plus_bis(data_10,10,100)
  kmeans.centers = kmeans(x = as.matrix(data_10),centers=10)$centers
  list_cost = append(list_cost,cost(data_10,centers_i))
  list_cost2 = append(list_cost2,cost(data_10,kmeans.centers))
}
list_mean_plus = append(list_mean_plus,mean(list_cost))
list_mean = append(list_mean,mean(list_cost2))
list_min_plus = append(list_min_plus,min(list_cost))
list_min = append(list_min,min(list_cost2))


list_cost = c()
list_cost2 = c()
for( i in 1:20 ){
  data_10 = gen_norm(10,5,1000)
  centers_i = fit_plus_bis(data_10,25,100)
  kmeans.centers = kmeans(x = as.matrix(data_10),centers=25)$centers
  list_cost = append(list_cost,cost(data_10,centers_i))
  list_cost2 = append(list_cost2,cost(data_10,kmeans.centers))
}
list_mean_plus = append(list_mean_plus,mean(list_cost))
list_mean = append(list_mean,mean(list_cost2))
list_min_plus = append(list_min_plus,min(list_cost))
list_min = append(list_min,min(list_cost2))

df = data.frame("Average cost kmean_plus"=list_mean_plus,"Average cost kmean" =list_mean,"minimum cost k
rownames(df) = c(paste("K-","10"),paste("K-","25"))
```

```
knitr::kable(df,format = "latex", caption = "Résultats expérimentaux sur NORM-10 n=1000, d=5",digits = 2
```

Pour RNORM-25

```
list_mean_plus = c()
list_mean = c()
list_min_plus = c()
```

Table 2: Résultats expérimentaux sur NORM-25 n=1000,d=5

	Average.cost.kmean_plus	Average.cost.kmean	minimum.cost.kmean_plus	minimum.cost.kmean
K- 10	21609.86	25133.03	16663.22	16008.66
K- 25	4.87	6025.57	4.67	2559.73

```
list_min = c()

list_cost = c()
list_cost2 = c()
for( i in 1:20 ){
  data_10 = gen_norm(25,5,1000)
  centers_i = fit_plus_bis(data_10,10,100)
  kmeans.centers = kmeans(x = as.matrix(data_10),centers=10)$centers
  list_cost = append(list_cost,cost(data_10,centers_i))
  list_cost2 = append(list_cost2,cost(data_10,kmeans.centers))
}
list_mean_plus = append(list_mean_plus,mean(list_cost))
list_mean = append(list_mean,mean(list_cost2))
list_min_plus = append(list_min_plus,min(list_cost))
list_min = append(list_min,min(list_cost2))


list_cost = c()
list_cost2 = c()
for( i in 1:20 ){
  data_10 = gen_norm(25,5,1000)
  centers_i = fit_plus_bis(data_10,25,100)
  kmeans.centers = kmeans(x = as.matrix(data_10),centers=25)$centers
  list_cost = append(list_cost,cost(data_10,centers_i))
  list_cost2 = append(list_cost2,cost(data_10,kmeans.centers))
}
list_mean_plus = append(list_mean_plus,mean(list_cost))
list_mean = append(list_mean,mean(list_cost2))
list_min_plus = append(list_min_plus,min(list_cost))
list_min = append(list_min,min(list_cost2))

df = data.frame("Average cost kmean_plus"=list_mean_plus,"Average cost kmean" =list_mean,"minimum cost k
rownames(df) = c(paste("K-","10"),paste("K-","25"))
```

```
knitr::kable(df,format = "latex", caption = "Résultats expérimentaux sur NORM-25 n=1000,d=5",digits = 2)
```

```
mean(list_cost)
```

```
## [1] 4.865848
```

```
dim(data_10)
```

```
## [1] 1000    5
```

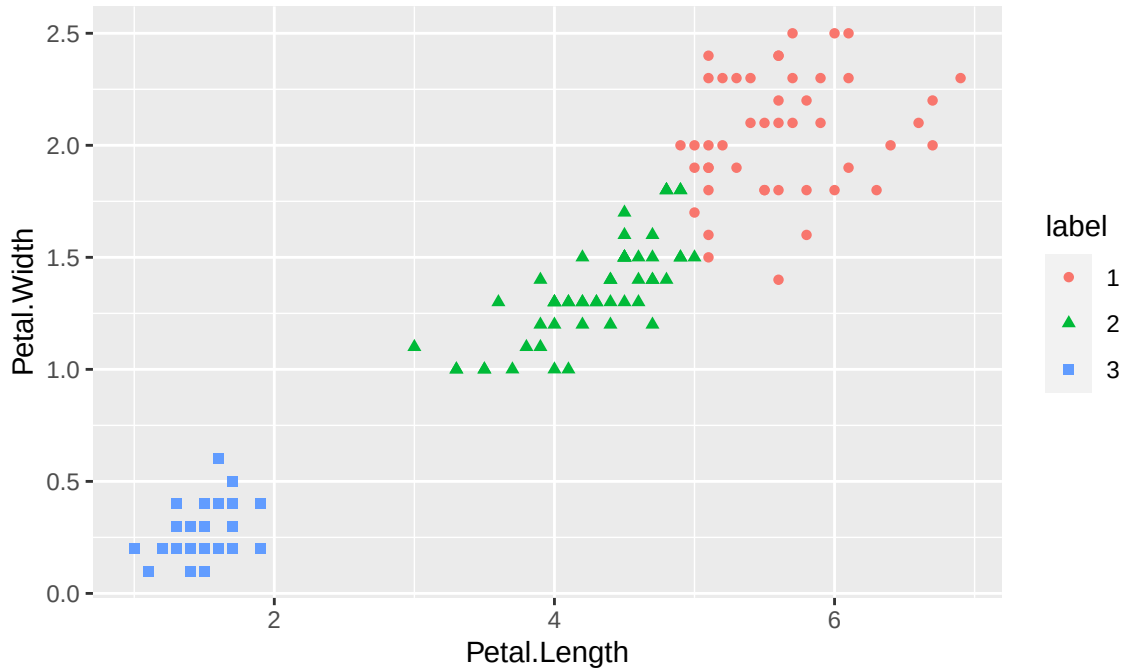
on voit que kmeans++ surpasse largement kmeans standard sur ces datasets. En effet plus le nombre de centres est grand, plus l'initialisation de kmeans standard va être imprécise. On ne teste pas l'algortithme pour n=50, pour des raisons de performances.

Partie 2

Question 1

On visualise tout d'abord le jeu de données iris avec les algorithmes kmeans++, kmeans standard, et mclust, par rapport à Petal.Width et Petal.Length. En appliquant notre algorithme kmeans++ au data iris, on trouve :

```
library(ggplot2)

centres = fit_plus_bis(iris[3:4],3,100)


df2 = set_clusters_bis(iris[3:4],centres)


df2$label = as.factor(df2$label)

ggplot(df2, aes(x=Petal.Length, y=Petal.Width, color=label,shape=label)) +
  geom_point()
```

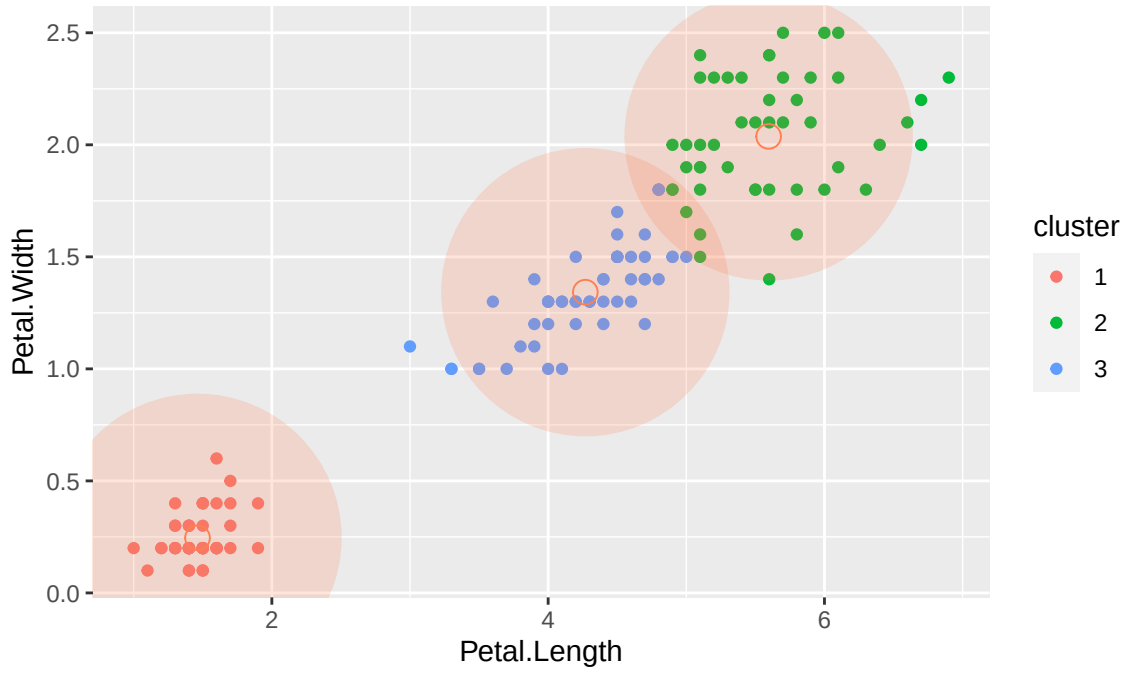


On compare ensuite les résultats obtenus avec d'autres algorithmes.
Commençons par l'algorithme k-means classique.

```
library(dbplyr)
library(tibble)
library(ggplot2)
data(iris)

k.res <- kmeans(iris[3:4], 3, nstart = 10)
cluster<-as.factor(k.res$cluster)
centers <-as.tibble(k.res$centers)
```

```
## Warning: ‘as.tibble()‘ was deprecated in tibble 2.0.0.
## i Please use ‘as_tibble()‘ instead.
## i The signature and semantics have changed, see ‘?as_tibble‘.
```

```
ggplot(iris, aes(x=Petal.Length, y=Petal.Width, color=cluster)) +
geom_point() +
geom_point(data=centers, color='coral',size=4,pch=21)+
geom_point(data=centers, color='coral',size=50,alpha=0.2)
```



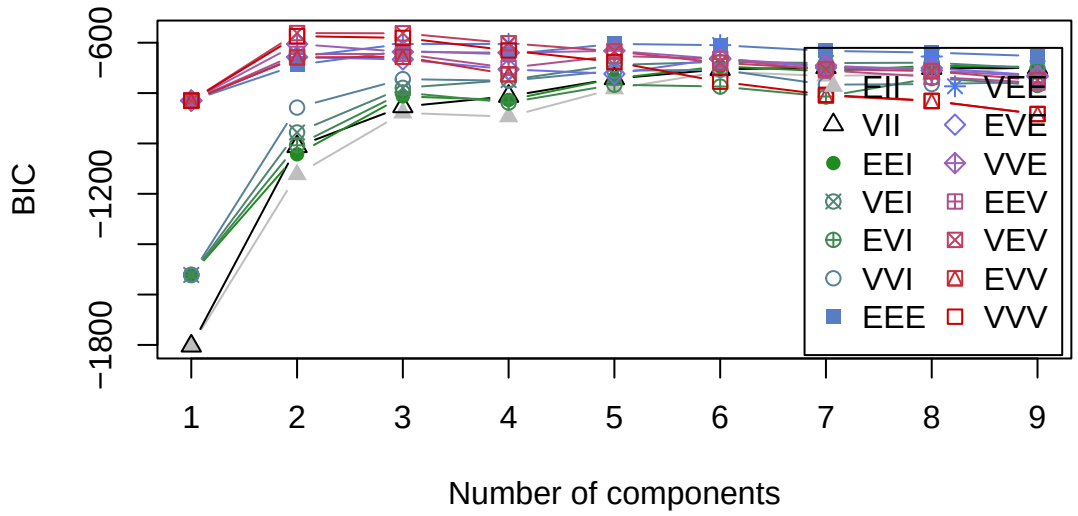

```
X = iris[,-5]
class <- iris$Species
```

On regarde tous les plots, différents selon chaque modèle proposés par la librairie mclust.

```
library(mclust)
```

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

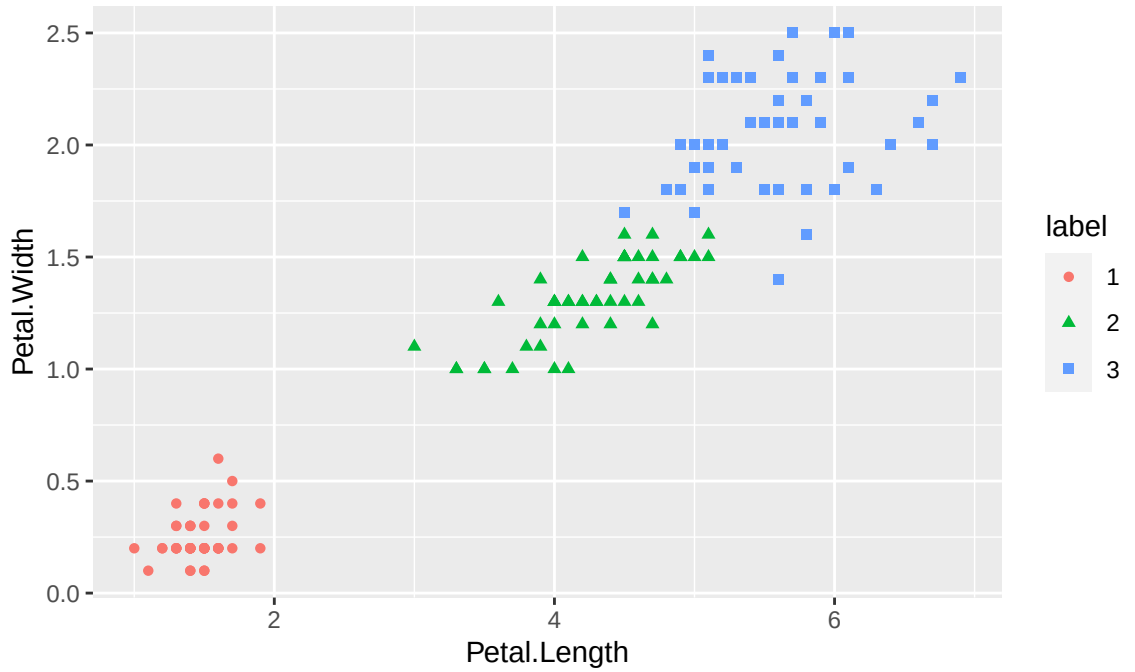
```
BIC <- mclustBIC(X)
plot(BIC)
```



On choisit donc le modèle qui maximise le BIC pour une répartition en 3 composantes.
On prend le modèle “VVE”.

Regardons le nombre d’imprécisions commises.

```
library(mclust)

model1 <- Mclust(iris[3:4], ModelNames = "VVE")

label = as.factor(model1$classification)
ggplot(iris, aes(x=Petal.Length, y=Petal.Width, color=label,shape=label)) +
  geom_point()
```



On voit que les graphes semblent similaires avec une légère différence sur l'algorithme Mclust. On calcule la matrice de confusion de chacun des clusters obtenues avec chacunes des méthodes sur l'ensemble des données iris. Ainsi on peut voir plus présicement les performances des différents algorithmes.

```
#kmeans++
centres = fit_plus_bis(X,3,100)


df2 = set_clusters_bis(X,centres)

table(df2$label,iris$Species)
```

```
##
##     setosa versicolor virginica
##   1      0          2        36
##   2     50          0         0
##   3      0         48        14
```

```
#kmean standard
k.res <- kmeans(X, 3, nstart = 10)

table(k.res$cluster,iris$Species)
```

```
##
##     setosa versicolor virginica
##   1     50          0         0
##   2      0          2        36
##   3      0         48        14
```

```r
#mclust
mclust.res <- Mclust(X,G=3)

table(mclust.res$classification,iris$Species)
```

```
##
##     setosa versicolor virginica
##   1     50          0         0
##   2      0         45         0
##   3      0          5        50
```

On voit finalement que l'algorithme Mclust a une bien meilleure précision que les algorithmes kmeans. Néanmoins les algorithmes kmeans et kmeans++ ont une précision similaire.

Question 2

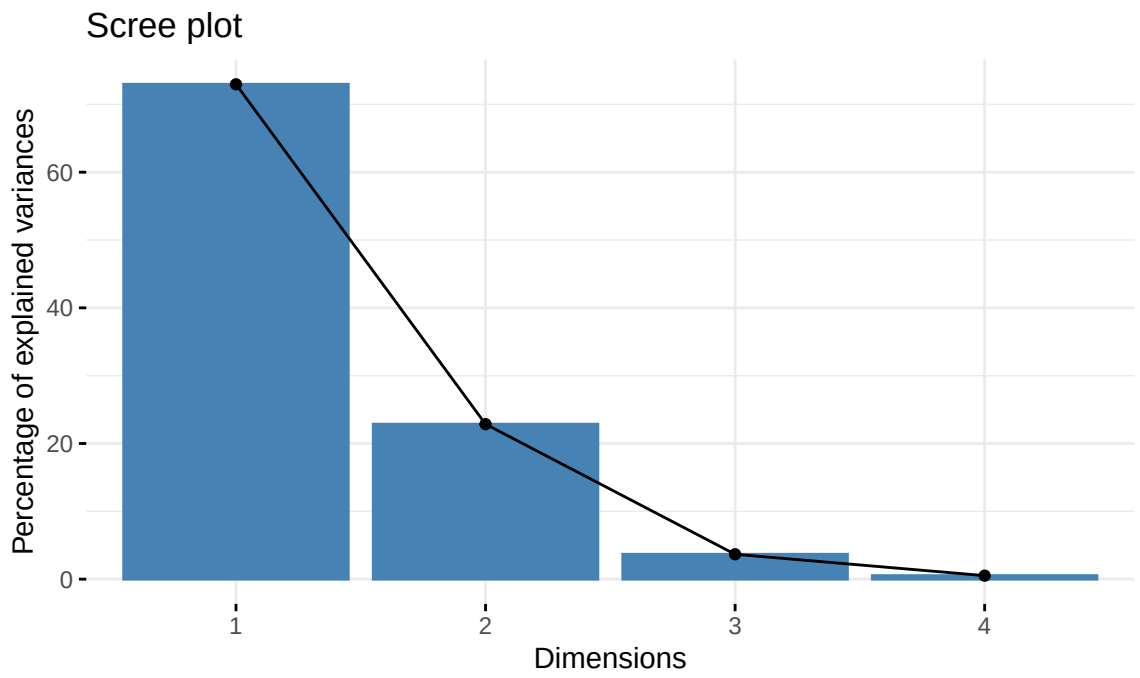
On visualiser maintenant les différentes partitions sur les premiers plans d'une analyse en composantes principales.

```r
data = iris[,-5]
data <- scale(data)
```

```r
library(factoextra)
```

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

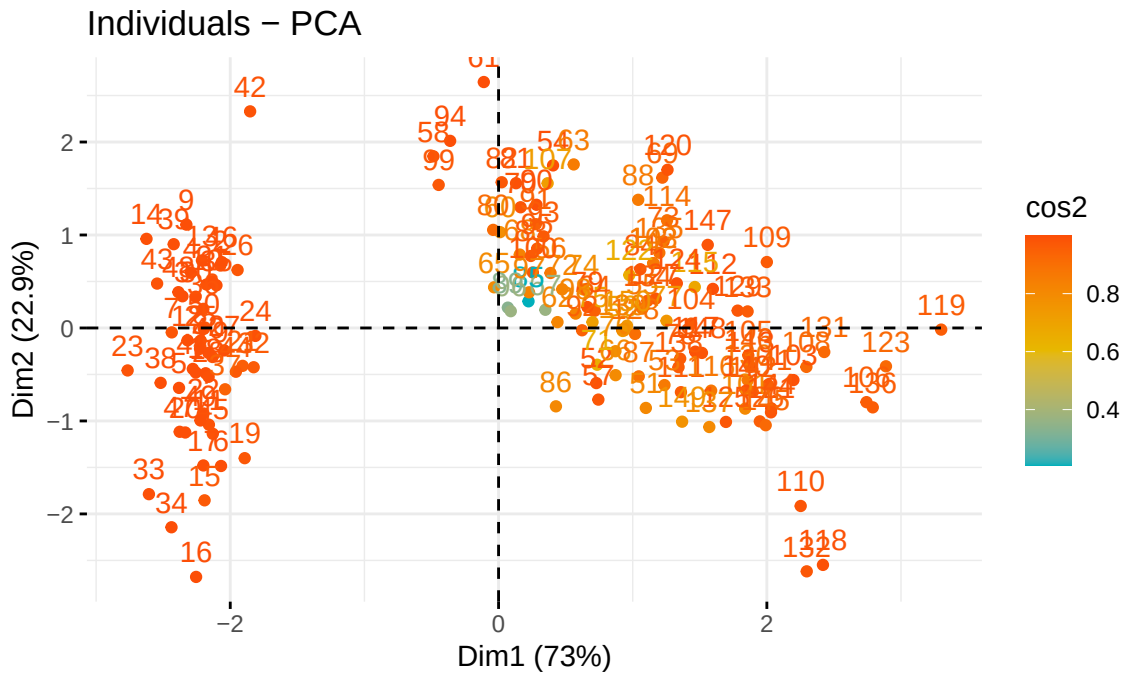
```r
res.pca <- prcomp(data, scale = TRUE)
fviz_eig(res.pca)
```



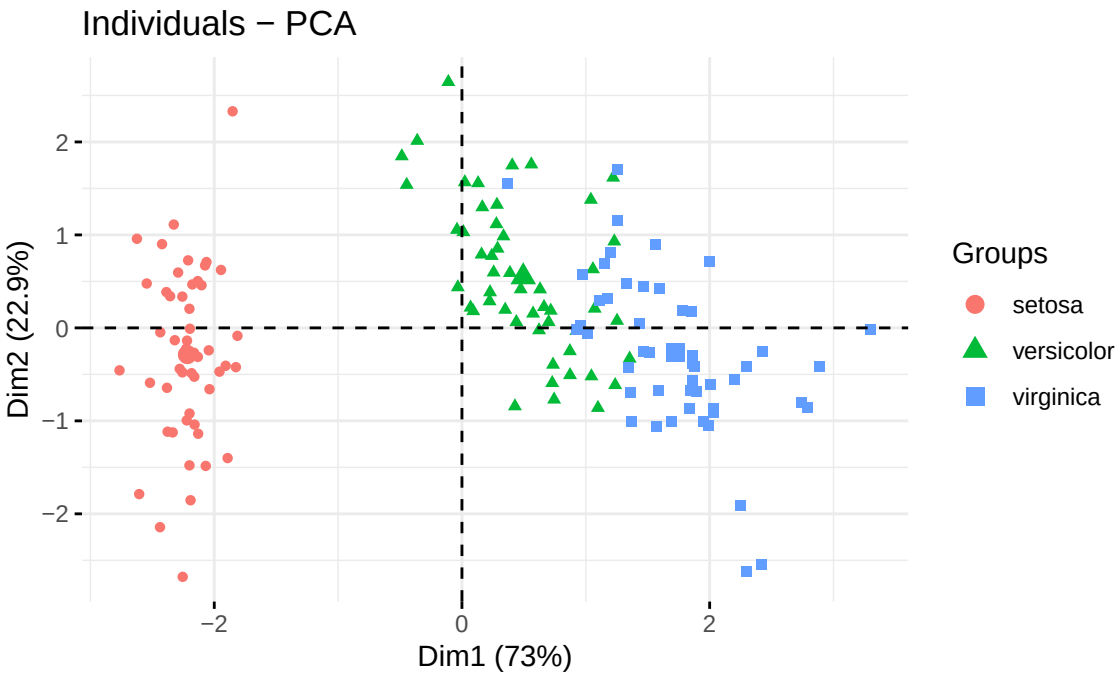
On voit que les 2 premières composantes expliquent à elles seules 95% de la variance. Il semble donc suffisant de s'intéresser à ces 2 composantes.

On regarde maintenant comment sont répartis, sur ces 2 composants, les différents individus du data, premièrement en fonction de leur valeur, puis ensuite en fonction du groupe auquel ils appartiennent.

```
library(factoextra)

fviz_pca_ind(res.pca,
             col.ind = "cos2",
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             )
```

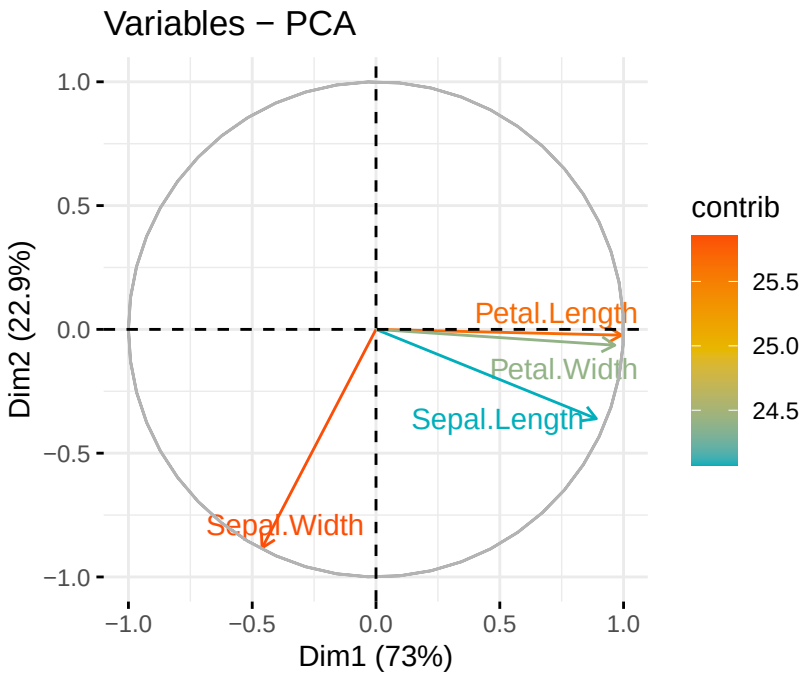



```
library(factoextra)

fviz_pca_ind(res.pca,
             label="none",
             habillage=iris$Species)
```



On voit sur ce plot qu'on ne peut pas vraiment distinguer les espèces versicolor et virginica. On pourrait visualiser les plans des composantes principales en dimesnion 3, mais cela ne changerait pas grand chose car 2 composantes expliques déjà 95% de la variance totale.

```
library(factoextra)

fviz_pca_var(res.pca,
             col.var = "contrib",
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE
             )
```



Question 3

On a obtenu des résultats similaires pour les 3 méthodes proposés. On remarque que notre méthode kmeans++ codé à la main obtient des résultats convaincants,au vu des résultats sur les RNORM notamment..

Pour la partition en composantes principales, on remarque que seules 2 composantes nous suffisent pour expliquer la majorité de la variance. Une fois le data projeté sur ces 2 composantes, on remarque que l'on a toujours nos 3 groupes différents, mais qu'après analyse du graphique, 2 groupes semblent quelque peu se mélanger (les velsicolor et verginica, respectivement en vert et bleu sur le schéma) ce qui peut nous laisser penser que l'on pourrait les réunir en un seul et même groupe. C'est peut-être pour cela que les algortihmes kmeans avaient des scores mauvais sur le datatest iris. En effet les algortithmes kmeans assignent des individus à des clusters en fonction de leur distance. Dans le datatest iris les espèces versicolor et verginica ne sont pas éloignés ce qui fausse donc les algorithmes kmeans.