Projet2

Abdoulaye SAKHO, Naïm SOUNI

15/11/2021

Contents

Exercice 1: Un algorithme des k-médoïdes

Question 1:

Afin de programmer l'algorithme décrit dans la section 2 de l'article scientifique joint au sujet, nous allons au préalable implémenter plusieurs fonctions auxiliaires.

Fonctions auxiliaires

Fonction auxiliaire pour nous aider à l'initialisation des médoïdes:

```
V<-function(D){
  res=c()
  n = nrow(D)
  a=0

  for (j in 1:n ){
    res=c(res,0)
    for (i in 1:n ){
      a=sum(D[i,])
      res[j]=res[j]+(D[i,j]/a)
    }
  }
  return(sort(res,index.return=TRUE)$ix)#on va ressortir les indices triés
}
```

Fonction auxiliaire permettant d'assigner les données aux médoïdes (actuels):

```
Assignation<-function(X,M,k){# X est la matrice des données
  #M est la matrice des medoïdes actuels

  n=nrow(X)
  res<-matrix(ncol = n,nrow=2,data = Inf)#on affecte une distance distance infini,
  #comme ca lors de la comparaison, on ne peut que faire mieux

  for(i in 1:n){# on parcours les observations, les Xi
    for(j in 1:k){# on parcours les médoides

      if(dist(rbind(X[i,],M[j,]))[1] <= res[1,i]){
        res[1,i]=dist(rbind(X[i,],M[j,]))[1]
        res[2,i]= j
        }
    }
    for(l in 1:k){
      if(identical( M[l,],X[i,])){
        res[1,i]= dist(rbind(X[i,],M[l,]))[1]
        # ci dessus: pour voir les medoides plus facilement
        # sans tuer l'algo
        res[2,i]=l}
    }
  }
  return(res)
}
```

Fonction auxiliaire permettant de calculer le critère d'arrêt:

```
Etot<-function(X,C,M){
  res=0
  n=nrow(X)

  for(i in 1:n){
    a=C[2,i]
    res = res + dist(rbind(X[i,],M[a,]))[1]
  }
  return(res)
}
```

Fonction auxiliaire permettant de calculer/sélectionner les nouveaux médoïdes:

```
New1<-function(X,C,k){
  n=nrow(X)
  p=ncol(X)
  res<-matrix(ncol = p,nrow=k,data=0)

  for(l in 1:k){
    stock=c()# va contenir les indis ces Xi appartenant au l-ème cluster
    lesdist=c()
    for(i in 1:n){if(C[2,i]==l){stock=c(stock,i)}}
    D=as.matrix(dist(as.matrix(X[stock,]),diag=TRUE,upper=TRUE))

    for(j in 1:length(stock)){lesdist=c(lesdist,sum(D[j,]))}
    #au dessus:  va contenir la somme des  dij pour i fixé

    res[l,]=X[stock[ sort(lesdist,index.return=TRUE)$ix[1] ], ]
    # au dessus:
    #nous permet de recup l'indice du min, et on ^rends X[cet indice, ]

  }
  return(res)

}
```

Fonction finale

```
F_means<-function(X,k=3,eps=10E-8){
  X2<-as.matrix(X)
  rownames(X2)<-NULL
  colnames(X2)<-NULL
  n=nrow(X2)
  p=ncol(X2)
  M<-matrix(ncol = p,nrow=k,data = 0)
  C<-matrix(ncol = n,nrow=2,data = 0)


  D=as.matrix(dist(X2,diag=TRUE,upper=TRUE))
  Va=V(D)
```

```
  iM=Va[1:k]

  for( i in 1:k){
    M[i,]<-X2[as.numeric(Va[i]),]
  }
  C<-Assignation(X2,M,k)
  s1=Etot(X2,C,M)
  s2=-1

  #while( abs(s2-s1) >eps ){
  while(s2!=s1){
    M=New1(X2,C,k)
    C=Assignation(X2,M,k)

    s2<-s1
    s1<-Etot(X2,C,M)
  }
  return(C)
}
```

Question 2:

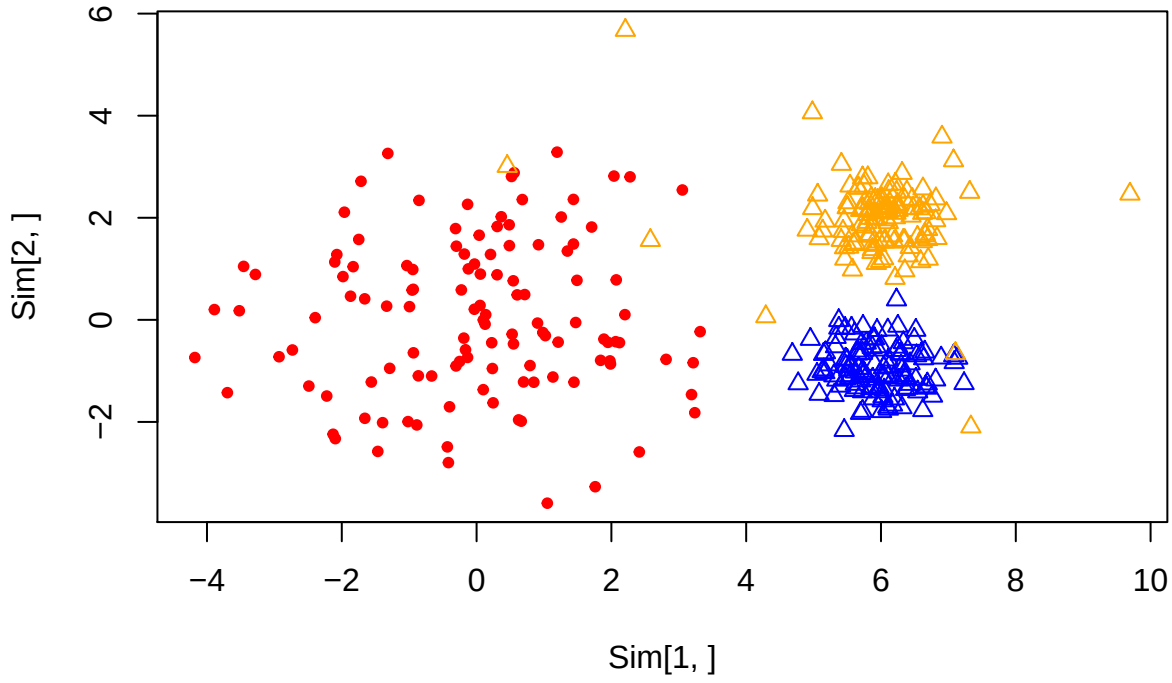
On implémente notre fonction de simulation des données:

```
simul<-function(n,mu,muAx,muAy,muBx,muBy,sA,sB,muCx,muCy,sC,sCL){
  #mu proportion bruit
  Ax<-matrix(ncol=n,nrow=1,rnorm(n,muAx,sA))
  Ay<-matrix(ncol=n,nrow=1,rnorm(n,muAy,sA))
  Bx<-matrix(ncol=n,nrow=1,rnorm(n,muBx,sB))
  By<-matrix(ncol=n,nrow=1,rnorm(n,muBy,sB))

  Cx<-matrix(nrow=1,ncol=n,data=0)
  Cy<-matrix(nrow=1,ncol=n,data=0)
  for(i in 1:n){
    z=rbinom(1,1,mu)# prb d'etre du bruit ou non
    if(z[1]==1){
      ax=rnorm(1,muCx,sCL)
      ay=rnorm(1,muCy,sCL)
      Cx[1,i]=as.numeric(ax)
      Cy[1,i]=as.numeric(ay)
    }
    else{
      ax=rnorm(1,muCx,sC)
      ay=rnorm(1,muCy,sC)
      Cx[1,i]=as.numeric(ax)
      Cy[1,i]=as.numeric(ay)
    }
  }
  resx=cbind(Ax,Bx,Cx)
  resy=cbind(Ay,By,Cy)
  res=rbind(resx,resy)
  return(res)
```

```
}


Sim<-simul(120,0.1,muAx=0,muAy = 0,muBx = 6,muBy = -1,sA=1.5, sB=0.5,
           muCx =6 ,muCy =2 ,sC=0.5,sCL=2)
#Sim
vrai<-c(rep(1,120),rep(2,120),rep(3,120))
plot(Sim[1,],Sim[2,],col=c("red","blue","orange")[vrai],pch=c(20,24,24)[vrai])
```




```
Sim2<-cbind(Sim[1,],Sim[2,])
```

Question 3:

Essai de la méthode proposée

On essaie notre algorithme sur nos données simulées:

```
clus1<-F_means(Sim2,k=3,eps=10E-20)
clus2<-F_means(scale(Sim2),k=3,eps=10E-20)

table(valeur=vrai,prédit=clus1[2,])# données non scaled
```

```
##       prédit
```

```
## valeur   1   2   3
##      1   1 115   4
##      2   0   0 120
##      3 115   2   3
```

```
table(valeur=vrai,prédit=clus2[2,])# données scaled
```

```
##       prédit
## valeur   1   2   3
##      1   4 108   8
##      2   0   0 120
##      3 116   1   3
```

Outils

On prépare les fonctions auxiliaires que nous allons utiliser pour comparer les algorithmes.

Dans ce chunk, nous allons implémenter la fonction qui nous permettra de simuler 100 échantillons différents(à chaque en suivant le protocole de la *section 3.3*). De plus, nous allons préparer les fonctions auxiliaires qui nous permettront de calculer l'AdjustedrandIndex de chaque méthode:

```
##########################################################
#############Creation Vecteur avec % bruit ##############
create_vect<-function(n,mu1){#n :taille échantillon / mu1: % bruit
  myList=list()
  for(i in 1:n){
    Sim<-simul(120,mu1,muAx=0,muAy = 0,muBx = 6,muBy = -1,sA=1.5, sB=0.5,
               muCx =6 ,muCy =2,sC=0.5,sCL=2)
    Sim2<-cbind(Sim[1,],Sim[2,])
    myList[[i]]<-Sim2
    #myList[[i]]<-scale(Sim2)   # voir la partie sur les résultats
    #pour compendre ce choix
  }
  return(myList)
}

#############################################################################################
###Calcul AdjRandIndex pour notre algo (un élmt particulier en sortie de notre algo)####
library(mclust)
```

```
## Warning: package 'mclust' was built under R version 4.0.5
```

```
## Package 'mclust' version 5.4.7
## Type 'citation("mclust")' for citing this R package in publications.
```

```
meanclus<-function(L,vrai){
  som=0
  for(i in 1:length(L)){
    som = som + adjustedRandIndex((L[[i]])[2,],vrai)
  }
  return( (1/length(L))*som )
}
```

```
##############################################################################
##############Calcul adjRanIndex pour algo PAM et kmeans et mclust##############
meanclus2<-function(L,vrai){
  som=0
  for(i in 1:length(L)){
    som = som + adjustedRandIndex(L[[i]],vrai)
  }
  return( (1/length(L))*som )
}


#### fonction avec des arguments par défaut .Permet d'utiliser lapply ensuite:

kmeans2<-function(X){return( (kmeans(X,3,30))$cluster ) }

mclust2<-function(X){
  a=mclustBIC(X,verbose = FALSE)
  return((Mclust(X,x=a))$classification )}

library(cluster)
```

```
## Warning: package 'cluster' was built under R version 4.0.4
```

```
pam2<-function(X){
  return( (pam(X, 3, metric = "euclidean", stand = FALSE))$cluster )
}
```

On compare les algorithmes. Nous allons comparer les algorithmes sur les données scaled et sur les données non scaled à la fois:

```
x<-c(0,0.05,0.1,0.15,0.20,0.40)

a<-c("% noisy objects", "K-means", "M-clust", "PAM", "Proposed method")
b<-c(0, 5,10,15,20,40)
Final<-matrix(data=0,ncol=5,nrow=6)
colnames(Final)<-a
Final[,1]<-b

Final2<-matrix(data=0,ncol=5,nrow=6)
colnames(Final2)<-a
Final2[,1]<-b

for(i in 1:(length(x))){# indice de mu: % de bruit

  myList1<-create_vect(100,x[i])# creéation des élémts
  MyList1scaled<-lapply(myList1,scale) # nos données scaled

  #k-means:
  kmeansclus<-lapply(myList1,kmeans2)
  kmeansclus2<-lapply(MyList1scaled,kmeans2)
  Final[i,2]<-meanclus2(kmeansclus,vrai)
  Final2[i,2]<-meanclus2(kmeansclus2,vrai)
```

```
  #M-clust
  mcl<-lapply(myList1,mclust2)
  mcl2<-lapply(MyList1scaled,mclust2)
  Final[i,3]<-meanclus2(mcl,vrai)
  Final2[i,3]<-meanclus2(mcl2,vrai)

  #PAM:
  pamclust<-lapply(myList1,pam2)
  pamclust2<-lapply(MyList1scaled,pam2)
  Final[i,4]<-meanclus2(pamclust,vrai)
  Final2[i,4]<-meanclus2(pamclust2,vrai)

  #Notre algo:
  resList<-lapply(myList1,F_means)
  resList2<-lapply(MyList1scaled,F_means)
  Final[i,5]<-meanclus(resList,vrai)
  Final2[i,5]<-meanclus(resList2,vrai)

}
```

Résultats

On affiche les résultats:

```
as.data.frame(Final)#données NON-scaled
```

```
##   % noisy objects   K-means   M-clust       PAM Proposed method
## 1               0 0.8669328 0.9942493 0.9641016       0.9592912
## 2               5 0.8276464 0.9699255 0.9546944       0.9492815
## 3              10 0.8318487 0.9412941 0.9412066       0.9367647
## 4              15 0.8112327 0.9132530 0.9276187       0.9082560
## 5              20 0.7955653 0.8955668 0.9208590       0.8950219
## 6              40 0.7813937 0.8213538 0.8693842       0.8567776
```

```
as.data.frame(Final2)#données scaled
```

```
##   % noisy objects   K-means   M-clust       PAM Proposed method
## 1               0 0.8566347 0.9942490 0.9236382       0.9007339
## 2               5 0.8677546 0.9685055 0.9203945       0.9204795
## 3              10 0.8523071 0.9418635 0.9057277       0.9045470
## 4              15 0.8453856 0.9162365 0.8961480       0.8911093
## 5              20 0.8498592 0.8960285 0.8903288       0.8801059
## 6              40 0.8236617 0.8214655 0.8414953       0.8319576
```

Nous avons utilisé l' "Adjusted Rand Index" pour différentes données simulés avec un pourcentage de bruits différents.

Dans un premeier temps on observe que **seul l'algorithme des k-means** est plus performant sur les données scaled que sur celles non scaled. Cela prouve que les autres méthodes (y compris la méthode proposées) sont plus robustes faces aux données aberrantes. On remarque alors que les données n'ont pas été scale dans l'article. En effet, on retrouve les résultats de l'article si on observe les résultats pour les données non scaled.

Nous allons nous concentrer sur le tableau des données non scaled puisque c'est celui qui corresponds à la démarche suivie dans l'article. On retrouve des résultats très similaires à ceux de l'article. notre implémentation de l'algorithme décrit dans la section 2 de l'article est un succès. On en tire les mêmes conclusions que celles de l'article: La méthode proposé est supérieur aux k-means est aussi performante que PAM. Notons qu'ici, nous avons aussi comparé la méthode proposé à M-clust. Ce dernier possède de meilleurs performances que les autres, ce qui est prévisible puisque M-clust est construit pour modéliser des distributions sous la forme de mélanges de plusieurs lois gaussiennes et qu'en l"occurence nous avons un mélange de plusieurs lois gaussiennes ici.

Exercice 2: Données Iris

Question 1:

On récupère les données iris:

```
library(MASS)
data("iris")
irisdata<-iris[,1:4]
#irisdata
```

On applique les différens algoritthmes à présent:

K-means:

```
table(valeur=iris$Species,prédit=(kmeans(scale(irisdata),3,30))$cluster)
```

```
##             prédit
## valeur        1  2  3
##   setosa     50  0  0
##   versicolor  0 39 11
##   virginica   0 14 36
```

M-clust:

```
table(valeur=iris$Species,prédit=mclust2(scale(irisdata)))
```

```
##             prédit
## valeur        1  2
##   setosa     50  0
##   versicolor  0 50
##   virginica   0 50
```

(On notera que Mclust a décidé de simuler les données par deux clusters)

PAM:

```
table(valeur=iris$Species,prédit=pam2(scale(irisdata)))
```

```
##             prédit
## valeur        1  2  3
##   setosa     50  0  0
##   versicolor  0  9 41
##   virginica   0 36 14
```

Proposed method:

```
clus<-F_means(scale(irisdata),k=3,eps=10E-8)

table(valeur=iris$Species,prédit=clus[2,])
```

```
##             prédit
## valeur        1  2  3
##   setosa      0 50  0
##   versicolor 41  0  9
##   virginica  15  0 35
```

On observe que notre algorithme est presque aussi efficace que l'algorithme PAM sur les données iris. De plus, la méthode proposée est plus efficace que les K-means.

Question 2:

```
library(factoextra)
```

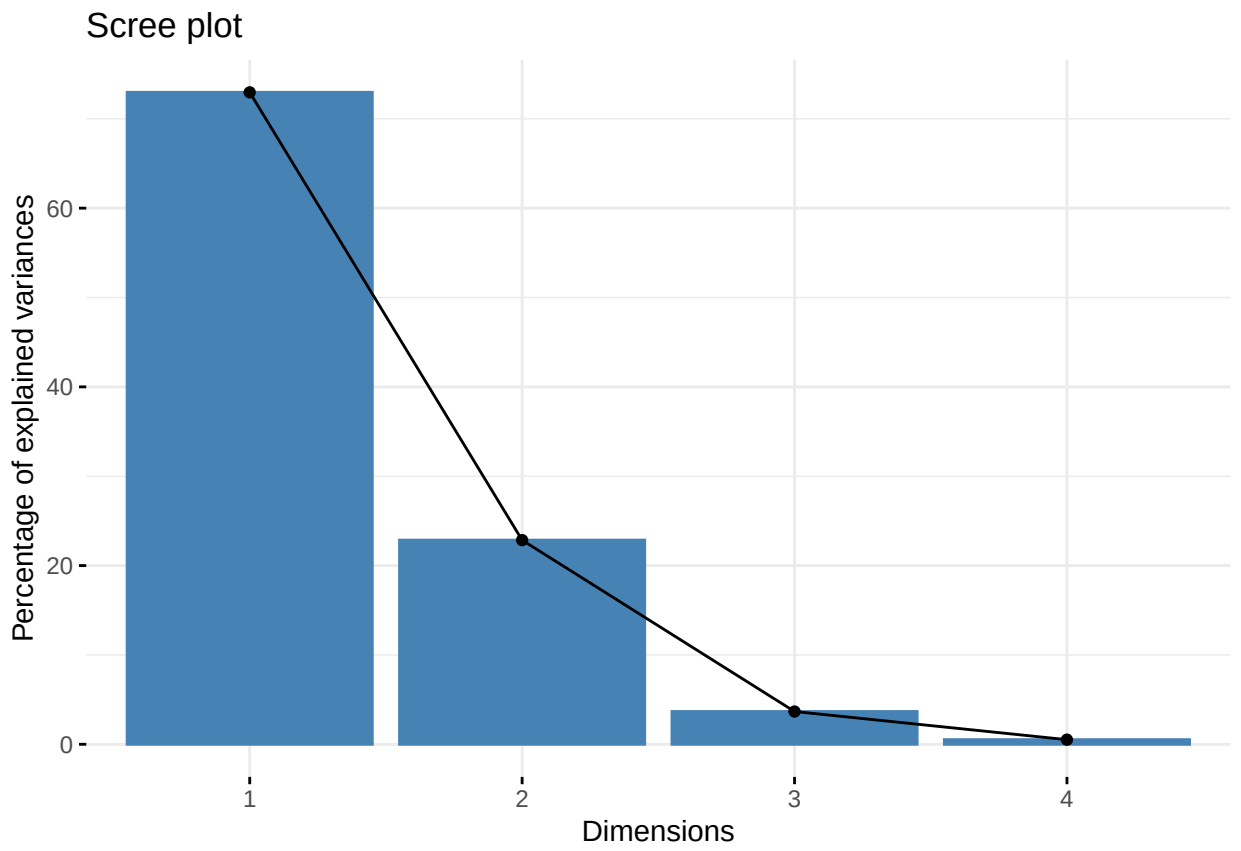
```
## Warning: package 'factoextra' was built under R version 4.0.5
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.0.5
```

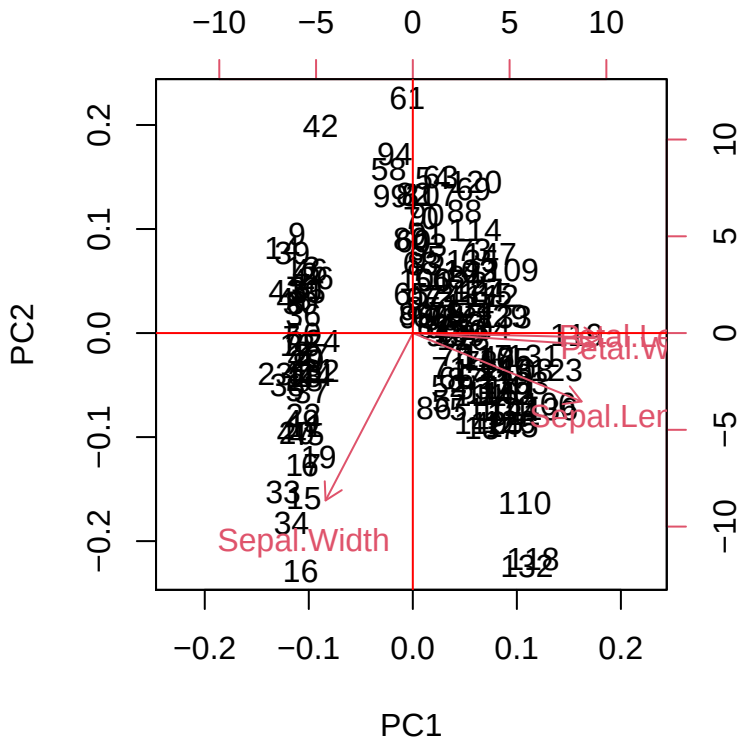
```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

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

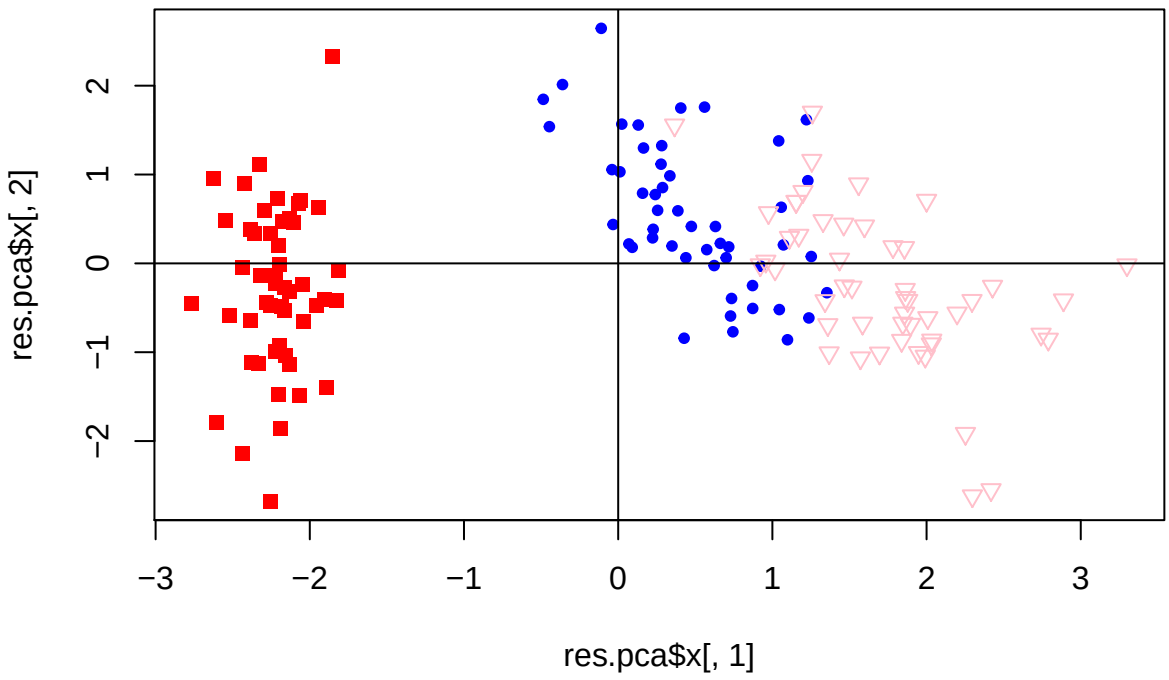


```r
#barplot(PCA,main="pourcentage de variance par axe")
```

```r
#par(mfrow=c(1,2))
biplot(res.pca,c(1,2))
abline(h=0,col="red")
abline(v=0,col="red")
```




```r
irisspecies<-c(rep(1,50), rep(2,50), rep(3,50))
plot(res.pca$x[,1],res.pca$x[,2],
     col=c("red","blue","pink")[irisspecies],pch=c(15,20,25)[irisspecies])
abline(h=0,col="black")
abline(v=0,col="black")
```



Question 3:

Pour commencer plus de 95% de la variance totale est exprimé par les deux premières composantes. On peut en déduire que les deux premiers axes retranscrivent suiffisament l'information pour qu'on n'effectue une ACP en 2 composantes seulement.

Attention: nous faison cela car ce qui nous interresse ici est de savoir si l'ACP retrouve la forme de "clusterisation" de nos données. Si nous voulions observer d'autres éléments plus fins/cachés dans nos données, il faudrait pluôt s'intéresser aux autres composantes principales (pour déjouer l'effet de taille).

Pour en revenir à notre objectif principal, on observe qu'avec une représentation en sur les deux premières compisantes principales de l'ACP permet de retrouver une forme de "clusterisation/séparation" de nos données en 3 clusters.

FIN