TD5

Colin Coërchon

2023-12-11

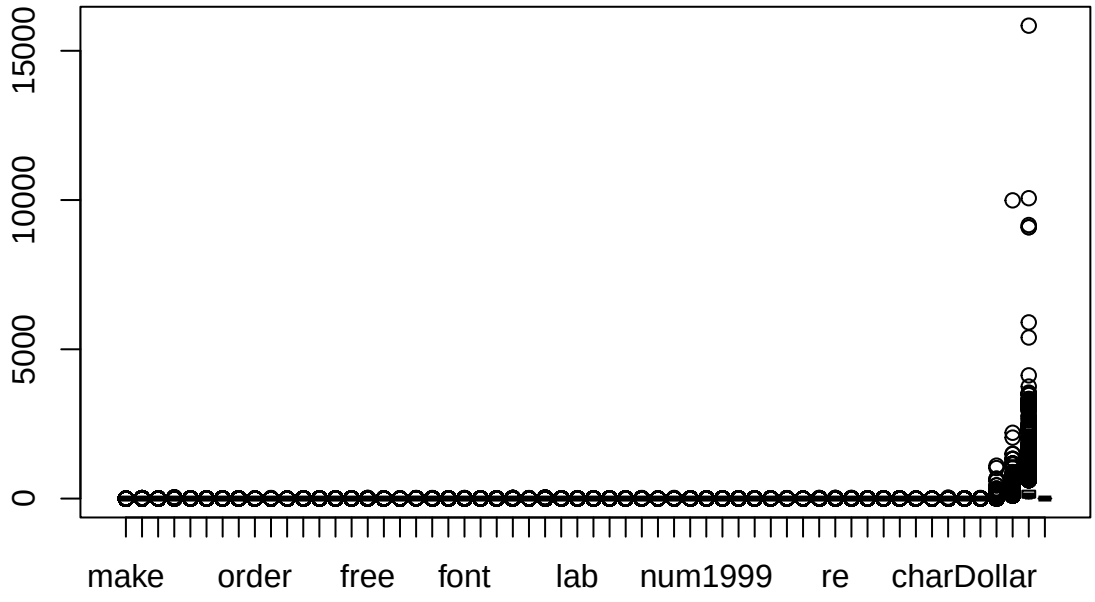
Contents

Exercice 1

Question 1

On import tout ce qui faut.

```
boxplot(spam)
```



Question 2

Petite ACP des familles.

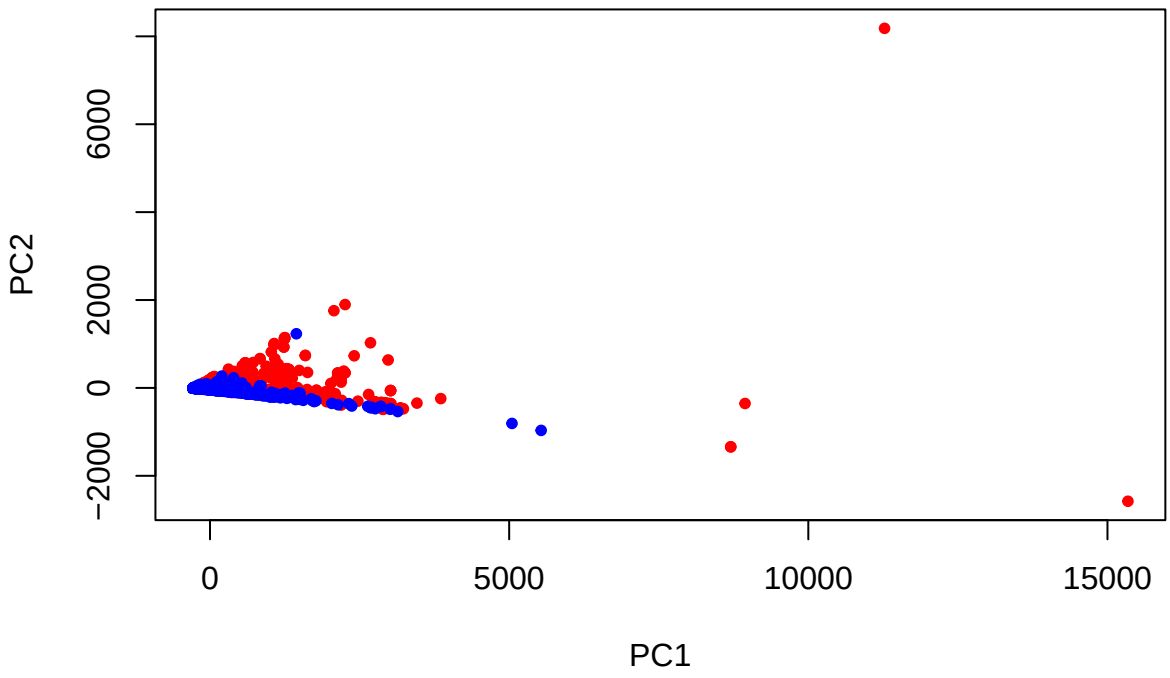
```
spam$type_numeric <- as.numeric(spam$type) - 1 # spam vaut 1 et "non-spam" vaut 0
data <- spam[, -58]

ACP <- prcomp(data)
```

Question 3

Petit résultat graphique de l'ACP.

```
colors <- ifelse(data$type_numeric == 1, "red", "blue")
plot(ACP$x, col=colors, pch = 20)
```



Question 4

Petite KPCA :

```
KPCA <- kpca(data)
```

Question 5

Notre algorithme qui réalise une KPCA :

```
X <- as.matrix(data)
K <- t(X) %*% X

n <- nrow(K)
Kc <- K + (1/n^2) * matrix(1,n,n) %*% K %*% matrix(1/n,n,n) - (2/n) * K %*% matrix(1,n,n)

DKc <- eigen(Kc)

# Tri des valeurs propres et sélection des vecteurs propres
values <- DKc$values
vectors <- DKc$vectors
ordered_indices <- order(values, decreasing = TRUE)
ordered_vectors <- vectors[, ordered_indices]
ordered_values <- values[ordered_indices]

# Choix du nombre de composantes à garder (par exemple, les 2 premières)
```

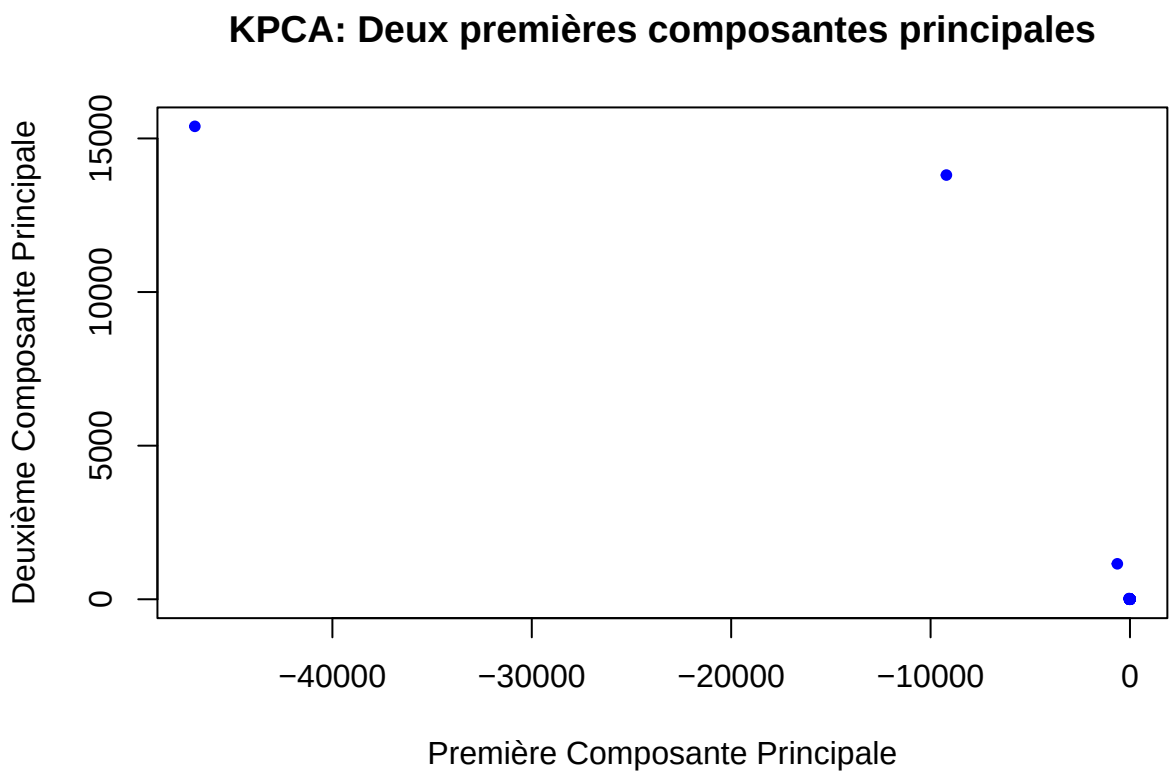
```r
num_components <- 2
selected_vectors <- ordered_vectors[, 1:num_components]

# Normalisation des vecteurs propres
normalized_vectors <- selected_vectors / sqrt(ordered_values[1:num_components])

# Projection des données
KPCA <- Kc %*% normalized_vectors

# KPCA contient maintenant les données projetées dans les composantes principales
```

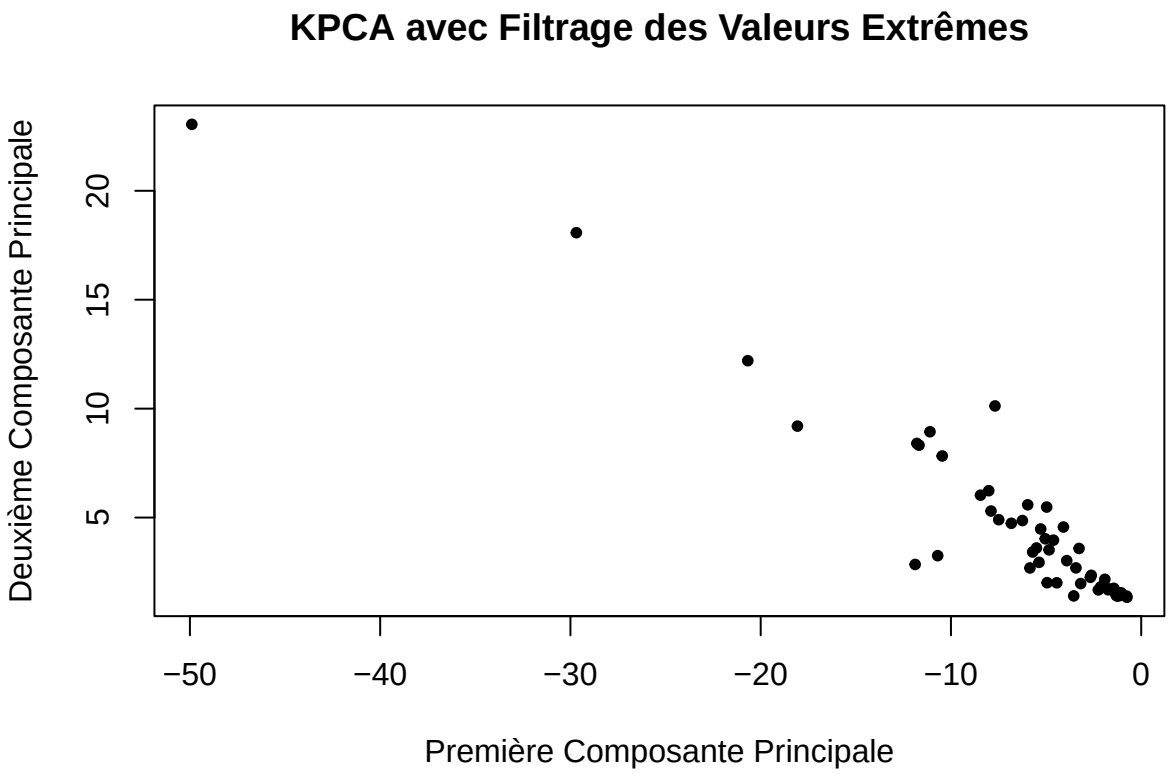
```r
# Plot des deux premières composantes principales
plot(KPCA[, 1], KPCA[, 2], xlab="Première Composante Principale", ylab="Deuxième Composante Principale"
```




```r
# Données KPCA
KPCA_data <- data.frame(V1 = KPCA[, 1], V2 = KPCA[, 2])

# Filtrer les valeurs extrêmes
seuil <- quantile(abs(c(KPCA_data$V1, KPCA_data$V2)), 0.95) # par exemple, garder 95% des données
KPCA_filtre <- KPCA_data[abs(KPCA_data$V1) < seuil & abs(KPCA_data$V2) < seuil, ]

# Plot des données filtrées
plot(KPCA_filtre$V1, KPCA_filtre$V2, xlab="Première Composante Principale", ylab="Deuxième Composante Pr
```



Algo des K-Médoïdes

```r
kmedoids <- function(X, K) {
    # Choix initial aléatoire des médoides
    medoids <- X[sample(nrow(X), K), ]
    # Initialisation de l'assignation des clusters
    clusters <- numeric(nrow(X))

    # Boucle jusqu'à ce que les médoides ne changent plus
    repeat {
        # Assignation de chaque point au médoid le plus proche
        for (i in 1:nrow(X)) {
            distances <- apply(medoids, 1, function(medoid) sum((X[i, ] - medoid)^2))
            clusters[i] <- which.min(distances)
        }

        # Mise à jour des médoides
        new_medoids <- array(dim = dim(medoids))
        for (j in 1:K) {
            cluster_points <- X[clusters == j, ]
            if (nrow(cluster_points) > 0) {
                medoid_distances <- apply(cluster_points, 1, function(point) sum(rowSums((t(cluster_poi
                new_medoids[j, ] <- cluster_points[which.min(medoid_distances), ]
            }
```

```
        }

        # Vérifier la convergence
        if (all(medoids == new_medoids)) {
            break
        }
        medoids <- new_medoids
    }

    list(clusters = clusters, medoids = medoids)
}
```

```
data("crabs")
crabsquant <- as.matrix(crabs[,4:8])
res <- kmedoids(crabsquant, 4)
```

```
# Combinaison des espèces et des sexes en une seule variable
species_sex <- paste(crabs[, 1], crabs[, 2])

# Créer la table de contingence
table(species_sex, res$clusters)
```

```
##
## species_sex  1  2  3  4
##         B F 10 18  1 21
##         B M 17 10  6 17
##         O F 23  4  7 16
##         O M 16  6  9 19
```

```
data("crabs")
crabsquant <- as.matrix(crabs[,4:8])
res <- pam(crabsquant, 4)
```

```
# Combinaison des espèces et des sexes en une seule variable
species_sex <- paste(crabs[, 1], crabs[, 2])

# Créer la table de contingence
table(species_sex, res$clustering)
```

```
##
## species_sex  1  2  3  4
##         B F 12 16 18  4
##         B M  7 13 15 15
##         O F  2  7 19 22
##         O M  5 14 11 20
```