Projet MAD 2022 - KMeans++

Teddy ALEXANDRE

Décembre 2022

L'algorithme de clustering K-moyennes, ou K-Means, est souvent utilisé en apprentissage non supervisé afin d'effectuer des regroupements et de rassembler des données par "similitude". L'objectif de ce projet est d'améliorer l'algorithme de base, détaillé en section 2.1 de l'article scientifique étudié. Pour cela, on propose dans l'exercice 1 une implémentation à la main de l'algorithme appelé "K-Means++", qui est une optimisation de la phase d'initialisation de l'algorithme de base. On se proposera ensuite de comparer les performances entre les deux algorithmes de clustering sur deux datasets, avant de l'étudier avec le célèbre dataset Iris dans le cadre du deuxième exercice, avec de plus une analyse en composantes principales (ACP) pour étudier comment l'information est retranscrite en plus petite dimension.

Exercice 1

Dans le cadre de l'exercice, X est l'ensemble des points de notre dataset, tous dans l'ensemble $\mathbb{R}^d, d \geq 1$. Notre dataset est donc une matrice avec n lignes (si n est le nombre de points) et d le nombre de coordonnées / la dimension de l'espace.

Question 1

On définit quelques fonctions utiles à l'implémentation de l'algorithme K-Means :

```
# Distance euclidienne au carré (norme 2) entre deux points
# de R^d
dist_eucli_sq <- function(p1, p2) {
    return(sum((p2 - p1)^2))
}

# Fonction qui calcule le barycentre / centre de masse
# associé à un cluster
barycenter_cluster <- function(cluster) {
    n = nrow(cluster)  # Nombre de points du cluster
    d = ncol(cluster)  # Nombre de coordonnées des points
    barycenter = rep(0, d)  # Vecteur des coordonnées du barycentre
    for (i in 1:d) {
        for (j in 1:n) {
            barycenter[i] = barycenter[i] + cluster[j, i]
        }
        barycenter[i] = (1/n) * barycenter[i]
    }
    return(barycenter)
}
```

On implémente d'abord la partie où l'on optimise l'initialisation des centroïdes initiaux comme indiqué dans l'article :

```r
# Initialisation de l'algo KMeans++ (étape 1) X est la
# matrice des points (n est le nombre de points, d est la
# dimension) K est un entier quelconque (le nombre de
# clusters) On renvoie la matrice des K centres initiaux
# avec leurs coordonnées
KMeansplusplus_init <- function(X, K) {
    n = nrow(X)
    d = ncol(X)
    squared_dist = rep(NA, n)  # vector of the squared distances of every point to the closest center
    centers = matrix(NA, nrow = K, ncol = d)  # matrix of centers, K rows and d columns

    ### Step 1a
    centers[1, ] = X[sample(1:n, size = 1), ]  # first center, chosen uniformly in the set of the point

    k = 1
    while (k < K) {
        ### Step 1b and 1c
        k = k + 1
        # Initialization of the K-1 centers left We compute
        # all the squared distances to build a probability
        # distribution
        for (i in 1:n) {
            x = X[i, ]
            dist_min_x_sq = Inf
            for (j in 1:(k - 1)) {
                c = centers[j, ]
                dist_cur_sq = dist_eucli_sq(x, c)  # Distance entre x et tous les centres calculés
                if (dist_cur_sq < dist_min_x_sq)
                  dist_min_x_sq = dist_cur_sq
            }
            squared_dist[i] = dist_min_x_sq
        }

        # We build our probability distribution
        probs = rep(NA, n)
        for (i in 1:n) probs[i] = squared_dist[i]/sum(squared_dist)

        # The cumulative probability is computed to help us
        # choose our next center
        cumulative_probs = cumsum(probs)

        # We choose our next center : we generate a random
        # value between 0 and 1
        prob = runif(1, min = 0, max = 1)
        j = 1
        # We then consider the smallest value of j such
        # that prob is greater than the cumulative
        # probability calculated
        while (j < n && cumulative_probs[j] < prob) j = j + 1
        # We update our list of centers
        centers[k, ] = X[j, ]
    }
    return(centers)
```

```
}
```

On peut donc implémenter le reste de l'algorithme K-Means à la main :

```
# Fonction qui réalise l'algo KMeans à partir de
# l'initialisation
KMeansplusplus <- function(X, K) {
    ### Steps 2 to 4
    n = nrow(X)
    d = ncol(X)
    centers = KMeansplusplus_init(X, K)
    partition_old = matrix(1, n, K)  # Partition matrix associated with the clusters
    partition_new = matrix(0, n, K)

    i = 1
    # Stopping criteria : the centers are the same after an
    # iteration
    while (i < 10 && !identical(partition_old, partition_new)) {
        partiton_old = partition_new
        partition_new = matrix(0, n, K)
        ### We update the clusters : step 2
        for (i in 1:n) {
            x = X[i, ]
            dist_min = Inf
            index_closest_center = 1
            for (c in 1:K) {
                # We compare the distance to every center
                # with the minimum found
                dist_cur = dist_eucli_sq(x, centers[c, ])
                if (dist_cur < dist_min) {
                  dist_min = dist_cur
                  index_closest_center = c
                }
            }
            # We assign the i-th point to the closest
            # center found
            partition_new[i, index_closest_center] = 1
        }
        ### Then update the barycenters : step 3
        for (k in 1:K) {
            cluster_k = X[which(partition_new[, k] == 1), ]
            centers[k, ] = barycenter_cluster(cluster_k)
        }
        i = i + 1
    }
    return(centers)
}
```

L'algorithme KMeans++ a donc une complexité plus importante que l'algorithme de base en ce qui concerne **l'initialisation**, mais **converge plus rapidement** que l'algorithme de base. On va donc comparer les performances entre les deux algorithmes avec deux datasets construits à la main.

Question 2

On construit le dataset NORM-10 comme décrit dans l'énoncé : on choisit de représenter 1000 points au total (pour plus de rapidité d'exécution), dans $\mathbb{R}^d, (d = 5)$, avec $m = 10$ centres dont les coordonnées sont générées chacune dans $[0, 500]$. On s'aide alors d'une loi normale multivariée, de moyenne les centres initiaux et de variance l'identité de taille d, pour créer le "bruit" autour de chacun des m centres.

```
N = 1000
d = 5
m = 10

# 10 centres dans R^5
real_centers10 = matrix(data = NA, nrow = m, ncol = d)
for (i in 1:m) {
    # Pour chaque centre, on simule d lois uniformes dans
    # [0, 500]
    real_centers10[i, ] = runif(d, min = 0, max = 500)
}
real_centers10
```

```
##            [,1]      [,2]     [,3]        [,4]      [,5]
##  [1,]  27.07431 479.16656 353.1744 217.0984061 463.27242
##  [2,] 307.56608 306.02751 365.1144 318.5496479 328.42062
##  [3,] 403.63511 106.84572 147.8174   0.8786136 432.52448
##  [4,] 118.89813 294.33786 412.8800 210.9662731  54.80531
##  [5,] 270.60081 272.08475 446.4717 113.0597834 138.93112
##  [6,] 370.27305 191.51319 242.2904 271.3460092  90.66608
##  [7,] 359.72354 272.82302 360.6963  59.4661271 364.35064
##  [8,] 313.49342  32.49932 430.8263 161.7665455 443.26172
##  [9,]  31.13077 270.80171 134.9016 496.2705683 245.17600
## [10,] 410.86673 463.08359 421.5143  13.4828450 364.57067
```

```
norm10points = c()
for (i in 1:m) {
    # Pour chaque centre, on génère N/m points autour de ce
    # centre
    norm10points = rbind(norm10points, mvrnorm(N/m, real_centers10[i,
        ], Sigma = diag(d)))
}

head(norm10points, 10)
```

```
##           [,1]     [,2]     [,3]     [,4]     [,5]
##  [1,] 25.44467 479.5059 354.1432 217.0892 464.0893
##  [2,] 28.21005 479.6117 353.6959 218.0153 463.2453
##  [3,] 27.92873 478.3804 353.0303 218.4211 461.8299
##  [4,] 27.36608 477.6849 353.6131 216.2506 462.8123
##  [5,] 27.24255 479.1402 353.5376 215.6723 465.1381
##  [6,] 26.32938 479.4159 351.6620 217.9180 463.1028
##  [7,] 27.10792 480.2352 353.5018 217.9181 462.3976
##  [8,] 25.32001 479.0473 352.3909 217.6975 461.6691
##  [9,] 27.46938 478.1864 354.7264 216.5931 463.0329
## [10,] 28.32870 478.4657 353.2476 219.7714 463.3963
```

De même pour NORM-25, on prend cette fois $d = 15$ et $m = 25$ centres.

```r
N = 1000
d = 15
m = 25

# 25 centres dans R^15
real_centers25 = matrix(data = NA, nrow = m, ncol = d)
for (i in 1:m) {
    # Pour chaque centre, on fait d lois uniformes dans [0,
    # 500]
    real_centers25[i, ] = runif(d, min = 0, max = 500)
}

norm25points = c()
for (i in 1:m) {
    norm25points = rbind(norm25points, mvrnorm(N/m, real_centers25[i,
        ], Sigma = diag(d)))
}

real_centers25
```

```
##            [,1]      [,2]      [,3]      [,4]       [,5]      [,6]       [,7]
##  [1,] 467.11915 126.64109 256.34197 208.68953 396.414703 321.50011 149.217859
##  [2,]  65.78883 131.88577 291.34938  98.56798 226.051901 429.64785 429.849285
##  [3,] 451.03000  40.69404 329.31113 209.43754  68.754907 341.94152   6.860308
##  [4,] 192.10390 382.30567  71.38154 200.35074 140.518214 160.26079 490.211246
##  [5,] 324.46836 499.72507 436.30224 101.14189 484.664345  57.87152 183.456292
##  [6,]  46.09474 202.73333 346.94800  84.86358  62.612308 282.82438 137.657313
##  [7,] 295.14836 305.07176 281.06890 331.91421 373.878228 390.83334 255.105666
##  [8,] 392.27304 128.96207 331.21930 471.55619 247.020107 357.95269  31.049842
##  [9,] 256.41225 103.92783 335.73249  84.93726 400.541418 456.20587  17.667584
## [10,] 399.57196  51.08198 184.39130 393.20527  85.816162  53.62985 388.799597
## [11,] 464.85603 305.79909 234.29986 252.97674  26.128057 456.81461  94.121297
## [12,] 189.75608 153.82173 419.03252  49.00617 496.309579 274.62799 306.862130
## [13,] 447.19405 293.71669 332.15614 138.67793 432.714813  70.57130 325.901052
## [14,] 256.75129  17.11060 364.96572 366.99536   2.173568 393.34854 384.572395
## [15,] 162.50718 319.51668 187.55299 243.90540  35.911843 272.83518   9.394668
## [16,]  39.77537 337.48074  66.97851 431.54100 372.661141 395.17814 169.986067
## [17,] 278.46108   3.98093 267.34478 394.27436 141.629903 220.13521 262.035209
## [18,] 251.65610 114.21915 236.35666 227.15496 108.443587 437.34040 466.954045
## [19,]  29.01905 388.94480 178.88553 162.32023 222.738528 213.95845  16.799850
## [20,] 124.09743 254.50899 296.04267 331.65968 382.523272 454.55034 103.268815
## [21,] 159.91572  59.10183 279.80450 253.31004   9.932871 120.83005 388.675557
## [22,] 432.69080 320.47170 311.50401 207.40910 317.476854 171.21416 245.021645
## [23,] 213.21171 156.15028  69.61454 164.34566  87.577225 271.10566 391.495698
## [24,] 450.32507 483.67045 124.23976 422.27114 266.952193 161.73647 246.894983
## [25,] 168.69551 452.48213  40.20995 306.54685  49.670228 262.87535 210.645469
##            [,8]       [,9]     [,10]     [,11]     [,12]      [,13]     [,14]
##  [1,]  71.79221 487.059653 280.82250 179.12333 359.97834 262.363239 319.08849
##  [2,] 328.94825   5.276249 436.35400 209.97263 482.32599 470.486062 108.64358
##  [3,] 180.56763 466.296809 320.11859 200.83323 378.11484 270.148271 370.39412
##  [4,] 349.40513 177.469532  74.71595 261.82532 239.21100 407.512106 173.14978
##  [5,] 415.92579  89.936255 489.05352 315.05376 285.04557 437.334804 376.01355
##  [6,]  91.57491 379.952824 203.77169 261.58957 175.95959 121.963915 124.24101
##  [7,] 363.26907 455.181519  81.54779 294.56469 393.88431  73.302988 430.21340
```

```
##  [8,] 303.36457 144.485840 280.47751 119.43224  96.97062 306.214764 423.96501
##  [9,] 399.04707 248.722945  75.97832 497.36458 327.90708 253.441686 152.28332
## [10,] 460.25127 443.736249 379.86251  74.14250 129.27895 436.795571  90.61709
## [11,] 264.32921 449.209478  48.43585 413.52694  15.28676 248.760184  77.06857
## [12,]  72.30838  45.092729 455.81792  65.29811 271.13453 333.729744 301.28604
## [13,] 310.12037 102.146640  99.38904 171.65012 365.90274  12.633071 245.09882
## [14,] 498.21313  80.945117 447.04307 184.80308 175.88421 470.263687 353.74818
## [15,]  77.22184 241.146830 201.69598 246.17792 158.90559 171.593900 286.31800
## [16,] 437.18041 376.985084  85.37638 203.95506 421.45079 447.007169 430.43021
## [17,] 469.51800 137.134302  45.50402 331.36323 332.30022 308.705488 149.72064
## [18,] 485.22297 460.185230 360.42384 260.87599 430.88092 287.707906 293.19843
## [19,] 378.98466 129.491762 340.25549 216.72376 154.33079  81.417255 158.01564
## [20,] 262.43192 363.857506 398.45833 175.49049 396.63928 127.515035 183.26202
## [21,]  94.04888 404.709163 466.16400 253.64571 150.94282 192.345733  42.93426
## [22,] 348.62936 181.655923 480.41540  34.03116 425.00641   7.293755 187.72343
## [23,] 448.01214 203.943050 476.39096 336.41864 194.76872 169.119098 329.25469
## [24,] 327.73942  47.107246 391.81479 387.96885 301.18147  20.930601 264.80296
## [25,] 163.44669 221.857768 222.97155 442.97282 386.45167  90.726563 286.63896
##           [,15]
##  [1,] 276.71895
##  [2,] 270.76901
##  [3,] 435.84407
##  [4,]  64.11488
##  [5,] 355.74120
##  [6,] 277.29573
##  [7,] 360.18105
##  [8,] 231.31209
##  [9,] 496.51856
## [10,]  11.22142
## [11,] 368.85366
## [12,] 139.39542
## [13,]  31.69745
## [14,] 420.50324
## [15,] 495.20334
## [16,] 398.98150
## [17,] 421.89866
## [18,] 163.86932
## [19,] 187.69410
## [20,] 333.88895
## [21,] 184.59676
## [22,] 351.13654
## [23,]  62.27176
## [24,] 492.92624
## [25,] 236.15661
```

```
head(norm25points, 10)
```

```
##           [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]     [,8]
##  [1,] 465.8644 127.0097 256.0423 208.9445 395.4811 320.1041 150.3298 72.67292
##  [2,] 466.5418 125.2047 255.3719 209.2474 395.1792 319.7058 149.8673 72.19712
##  [3,] 466.5983 126.7924 256.0972 208.5439 395.0377 322.7554 148.6242 72.50066
##  [4,] 466.9435 125.2983 256.3367 207.0830 396.8411 321.0003 148.9927 71.87978
##  [5,] 468.9408 125.8091 255.2034 209.4724 396.2887 320.5567 149.9709 69.41612
##  [6,] 467.7052 126.2531 257.7580 207.5030 397.5681 320.6686 150.7355 73.85893
##  [7,] 468.5349 127.3917 257.6727 209.7378 397.0913 322.5730 150.3606 71.57720
```

```
##  [8,] 466.8810 128.0996 255.6436 207.5941 395.5562 321.2942 150.2199 70.61679
##  [9,] 467.6677 125.5135 257.8314 209.2705 398.1837 321.0936 150.2042 70.79051
## [10,] 467.4909 127.7652 256.1694 209.4408 397.0525 322.7213 149.2254 71.09384
##           [,9]    [,10]    [,11]    [,12]    [,13]    [,14]    [,15]
##  [1,] 486.8563 280.9162 181.4877 361.4353 263.0110 318.7797 275.7563
##  [2,] 487.1279 282.6146 180.3933 359.4837 262.1753 319.0840 278.5229
##  [3,] 486.9122 281.8345 178.9608 357.8392 262.4705 318.2806 277.0463
##  [4,] 487.7576 281.1992 178.2451 358.6366 261.6193 320.8395 276.1242
##  [5,] 487.1196 281.5778 178.9021 360.2878 264.9883 318.5765 276.6633
##  [6,] 486.5735 281.9425 179.1692 360.8972 262.0546 323.6276 276.0721
##  [7,] 488.0526 279.2479 179.1887 360.4449 262.5897 320.5559 276.2796
##  [8,] 488.8926 281.4032 180.0979 358.7239 261.4351 320.5912 276.0647
##  [9,] 487.9681 281.2982 179.9469 358.8704 261.6253 317.2529 276.9584
## [10,] 487.9874 280.4903 178.9670 359.5524 259.8697 319.4950 276.2889
```

Question 3

On utilise cette fois la méthode KMeans fournie de base dans R :

```
# Determines and returns the closest center
closest_center <- function(pt, centers) {
    center = rep(0, ncol(centers))
    dist_min = Inf
    for (i in 1:nrow(centers)) {
        dist_cur = dist_eucli_sq(pt, centers[i, ])
        if (dist_cur < dist_min) {
            dist_min = dist_cur
            center = centers[i, ]
        }
    }
    return(center)
}

# Calculates the potential associated to the data set when
# clusters are done
potential <- function(X, centers) {
    res = 0
    for (i in 1:nrow(X)) {
        res = res + dist_eucli_sq(X[i, ], closest_center(X[i,
            ], centers))
    }
    return(res)
}
```

```
nbpoints1 = nrow(norm10points)
# Comparison between KMeans and KMeans++ with Norm-10

time_kmeans_begin = Sys.time()
res = kmeans(x = norm10points, iter.max = 10, centers = 10)
time_kmeans_end = Sys.time()

print("Temps d'exécution KMeans avec Norm-10, 10 centres :")
```

```
## [1] "Temps d'exécution KMeans avec Norm-10, 10 centres :"
```

```
print(time_kmeans_end - time_kmeans_begin)
```

```
## Time difference of 0.005419016 secs
```

```
time_kmeanspp_begin = Sys.time()
centers10 = KMeansplusplus(norm10points, 10)
time_kmeanspp_end = Sys.time()

print("Temps d'exécution KMeans++ avec Norm-10, 10 centres :")
```

```
## [1] "Temps d'exécution KMeans++ avec Norm-10, 10 centres :"
```

```
print(time_kmeanspp_end - time_kmeanspp_begin)
```

```
## Time difference of 0.3310621 secs
```

```
print("Potentiel moyen KMeans :")
```

```
## [1] "Potentiel moyen KMeans :"
```

```
res$tot.withinss/nbpoints1
```

```
## [1] 13675.77
```

```
print("Potentiel moyen KMeans++ :")
```

```
## [1] "Potentiel moyen KMeans++ :"
```

```
potential(norm10points, centers10)/nbpoints1
```

```
## [1] 5.116046
```

```
nbpoints2 = nrow(norm25points)
# Same with Norm-25

time_kmeans_begin = Sys.time()
res2 = kmeans(x = norm25points, iter.max = 10, centers = 25)
time_kmeans_end = Sys.time()

print("Temps d'exécution KMeans avec Norm-25, 25 centres :")
```

```
## [1] "Temps d'exécution KMeans avec Norm-25, 25 centres :"
```

```
print(time_kmeans_end - time_kmeans_begin)
```

```
## Time difference of 0.008713722 secs
```

```
time_kmeanspp_begin = Sys.time()
centers25 = KMeansplusplus(norm25points, 25)
time_kmeanspp_end = Sys.time()

print("Temps d'exécution KMeans++ avec Norm-25, 25 centres :")
```

```
## [1] "Temps d'exécution KMeans++ avec Norm-25, 25 centres :"
```

```
print(time_kmeanspp_end - time_kmeanspp_begin)
```

```
## Time difference of 1.160514 secs
```

```
print("Potentiel moyen KMeans :")
```

```
## [1] "Potentiel moyen KMeans :"
```

```
res2$tot.withinss/nbpoints2
```

```
## [1] 40555.04
```

```
print("Potentiel moyen KMeans++ :")
```

```
## [1] "Potentiel moyen KMeans++ :"
```

```
potential(norm25points, centers25)/nbpoints2
```

```
## [1] 14.81567
```

Le potentiel associé au KMeans++ est donc significativement amélioré par rapport à celui de base. Néanmoins on observe dans les deux cas un temps d'exécution un peu plus important mais toujours très honnête? L'algorithme KMeans++ constitue donc une bonne amélioration de l'algorithme de base, grâce à une phase d'initialisation améliorée par rapport à l'algorithme de base.

Exercice 2

On importe le data set *Iris*, et on met en place les méthodes KMeans, KMeans++ et MClust :

Question 1

```
# Data visualization and preparation
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

```
iris_quantitative = as.matrix(iris[, -5])  # We remove the last column, not quantitative
K = 3  # On choisit K = 3

# KMeans++
```

```
time_kmeanspp_begin = Sys.time()
centers_iris_kmpp = KMeansplusplus(iris_quantitative, K)
time_kmeanspp_end = Sys.time()
print("Temps d'exécution KMeans++")
```

```
## [1] "Temps d'exécution KMeans++"
```

```
print(time_kmeanspp_end - time_kmeanspp_begin)
```

```
## Time difference of 0.01064038 secs
```

```
print("KMeans++ : Centres")
```

```
## [1] "KMeans++ : Centres"
```

```
centers_iris_kmpp
```

```
##          [,1]     [,2]     [,3]     [,4]
## [1,] 5.005660 3.369811 1.560377 0.290566
## [2,] 7.475000 3.125000 6.300000 2.050000
## [3,] 6.135294 2.852941 4.769412 1.645882
```

```
print("Potentiel KMeans++ sur les données iris : K = 3")
```

```
## [1] "Potentiel KMeans++ sur les données iris : K = 3"
```

```
print(potential(iris_quantitative, centers_iris_kmpp))
```

```
## [1] 99.4267
```

```
# KMeans
time_kmeans_begin = Sys.time()
res_km = kmeans(x = iris_quantitative, centers = K)
time_kmeans_end = Sys.time()
print("Temps d'exécution KMeans")
```

```
## [1] "Temps d'exécution KMeans"
```

```
print(time_kmeans_end - time_kmeans_begin)
```

```
## Time difference of 0.003837347 secs
```

```
print("KMeans : centres")
```

```
## [1] "KMeans : centres"
```

```
res_km$centers
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     5.006000    3.428000     1.462000    0.246000
## 2     5.901613    2.748387     4.393548    1.433871
## 3     6.850000    3.073684     5.742105    2.071053
```

```
print("Potentiel KMeans sur les données iris : K = 3")
```

```
## [1] "Potentiel KMeans sur les données iris : K = 3"
```

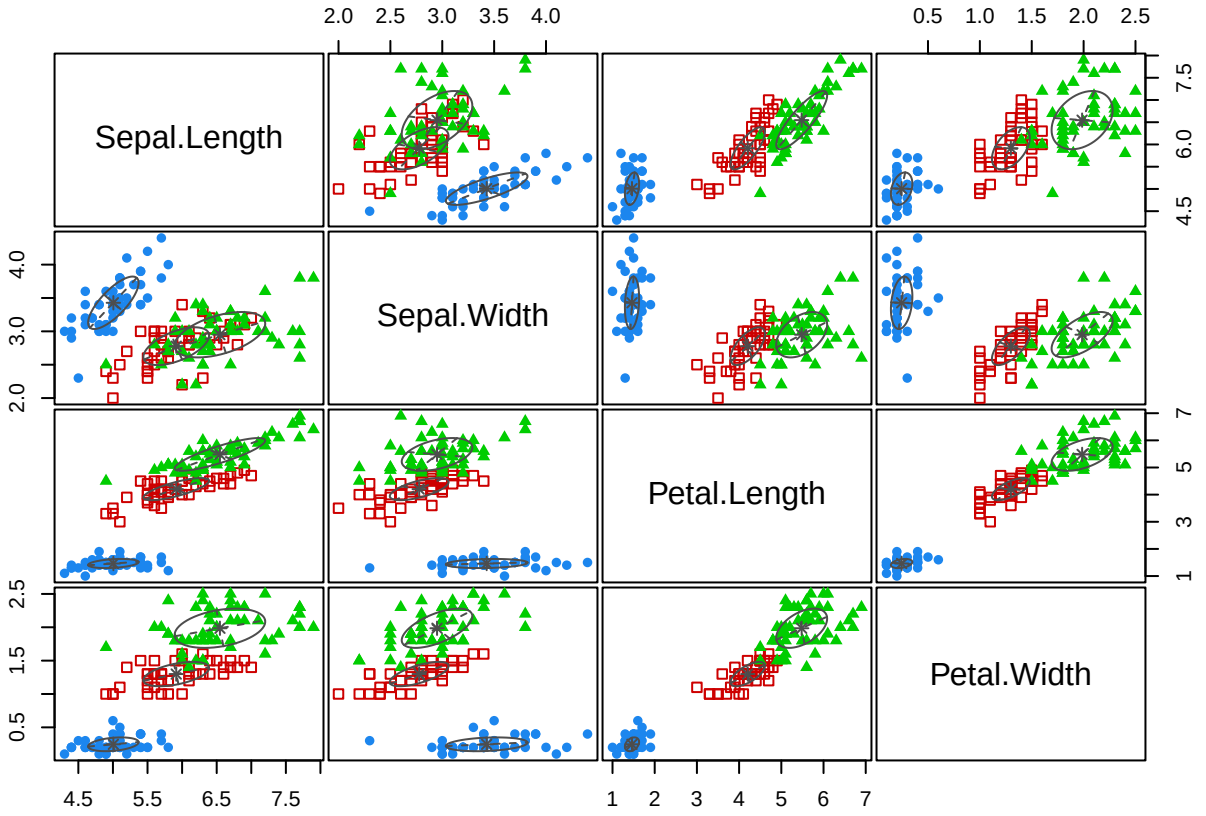
```
res_km$tot.withinss
```

```
## [1] 78.85144
```

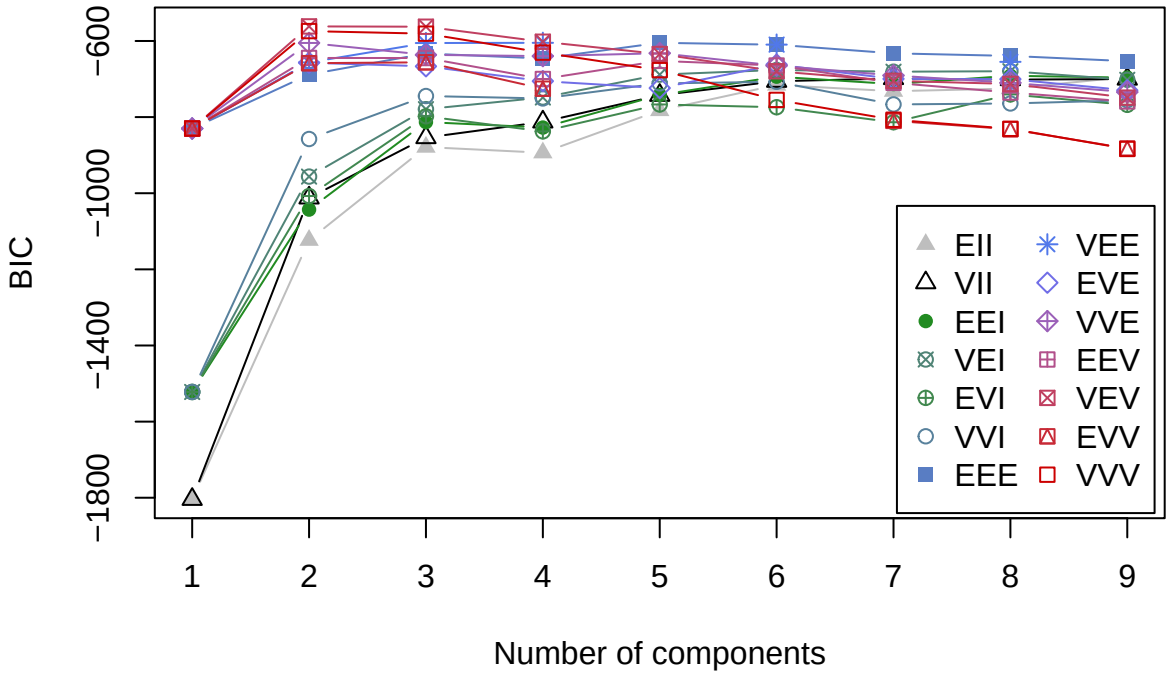
```r
# MClust
centers_iris_mclust = Mclust(data = iris_quantitative, G = K)
summary(centers_iris_mclust, parameters = TRUE)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VEV (ellipsoidal, equal shape) model with 3 components:
##
##  log-likelihood   n df       BIC       ICL
##        -186.074 150 38 -562.5522 -566.4673
##
## Clustering table:
##  1  2  3
## 50 45 55
##
## Mixing probabilities:
##         1         2         3
## 0.3333333 0.3005423 0.3661243
##
## Means:
##                [,1]     [,2]     [,3]
## Sepal.Length 5.006 5.915044 6.546807
## Sepal.Width  3.428 2.777451 2.949613
## Petal.Length 1.462 4.204002 5.482252
## Petal.Width  0.246 1.298935 1.985523
##
## Variances:
## [,,1]
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length   0.13320850  0.10938369  0.019191764 0.011585649
## Sepal.Width    0.10938369  0.15495369  0.012096999 0.010010130
## Petal.Length   0.01919176  0.01209700  0.028275400 0.005818274
## Petal.Width    0.01158565  0.01001013  0.005818274 0.010695632
## [,,2]
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length   0.22572159  0.07613348   0.14689934  0.04335826
## Sepal.Width    0.07613348  0.08024338   0.07372331  0.03435893
## Petal.Length   0.14689934  0.07372331   0.16613979  0.04953078
## Petal.Width    0.04335826  0.03435893   0.04953078  0.03338619
## [,,3]
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length   0.42943106  0.10784274   0.33452389  0.06538369
## Sepal.Width    0.10784274  0.11596343   0.08905176  0.06134034
## Petal.Length   0.33452389  0.08905176   0.36422115  0.08706895
## Petal.Width    0.06538369  0.06134034   0.08706895  0.08663823
```

```r
plot(centers_iris_mclust, what = "classification")
```




```
bic_iris = mclustBIC(iris_quantitative)
plot(bic_iris)
```



Les valeurs obtenues pour ϕ sont relativement proches les unes des autres. Le graphe du BIC en fonction du nombre de composantes montre que le choix optimal du nombre de composantes (donc de clusters) se situe entre 2 et 3 (on veut maximiser le BIC). On a pris ici la valeur 3 pour y visualiser 3 clusters dans le cadre d'une analyse en composantes principales en deux dimensions.

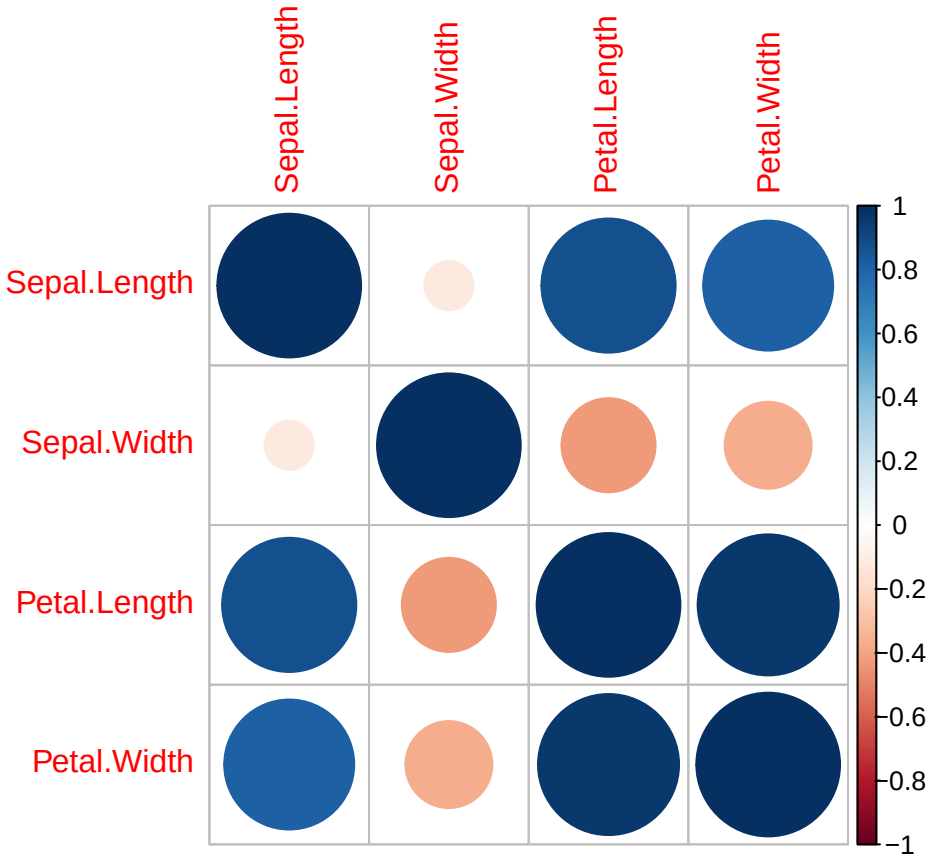
Question 2

La matrice de corrélation :

```
print(cor(iris_quantitative))
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
## Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
## Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
## Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

```
corrplot(cor(iris_quantitative))
```



La mise en place de l'ACP :

```
iris.pca = PCA(iris_quantitative, graph = FALSE)
iris.pca$eig
```

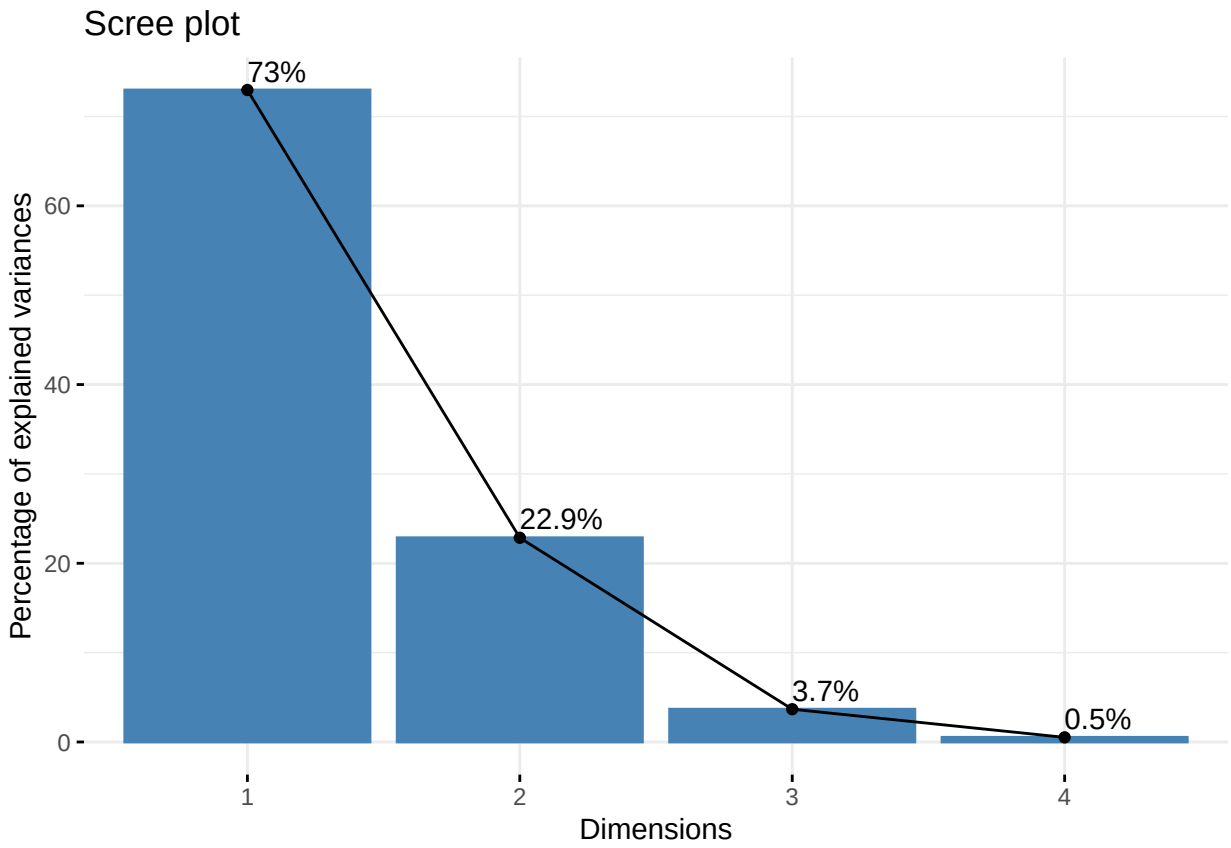
```
##        eigenvalue percentage of variance cumulative percentage of variance
## comp 1 2.91849782             72.9624454                           72.96245
## comp 2 0.91403047             22.8507618                           95.81321
## comp 3 0.14675688              3.6689219                           99.48213
## comp 4 0.02071484              0.5178709                          100.00000
```

```
summary(iris.pca)
```

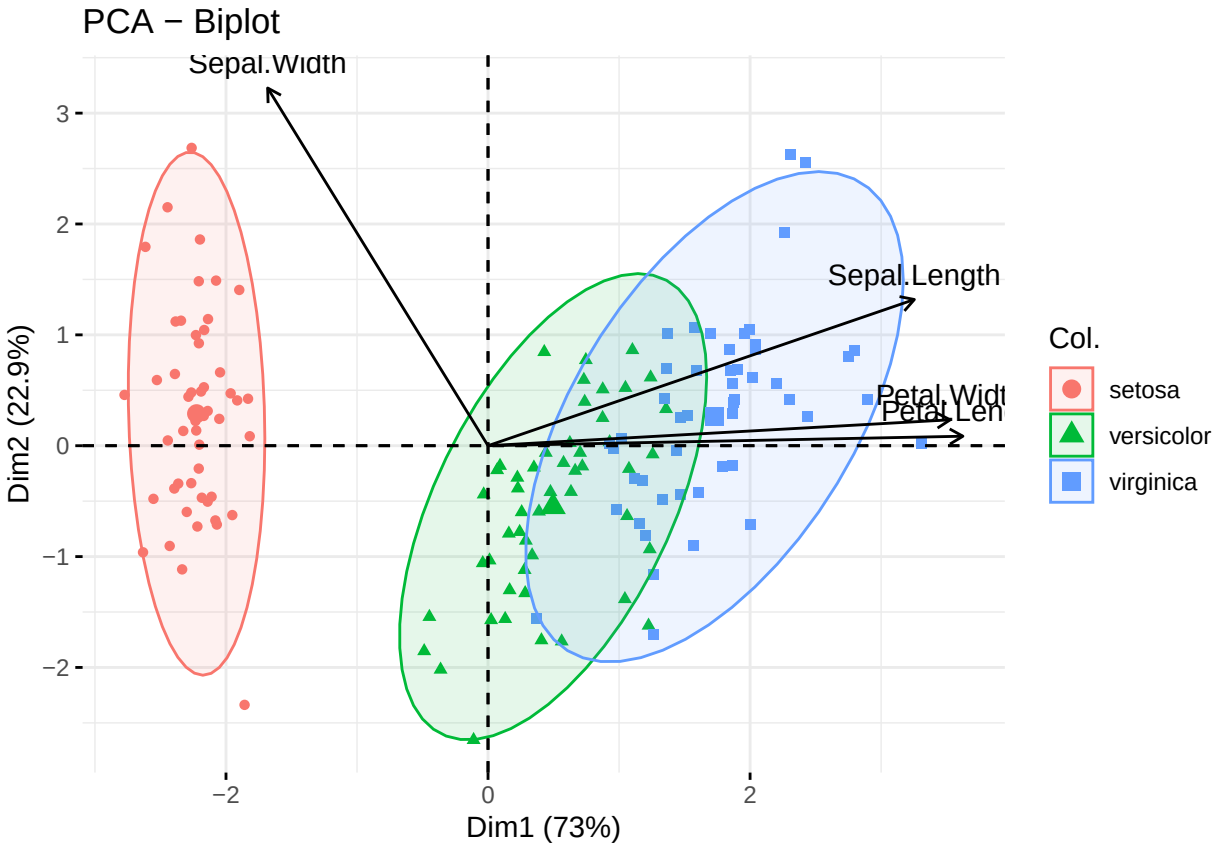
```
##
## Call:
## PCA(X = iris_quantitative, graph = FALSE)
```

```
##
##
## Eigenvalues
##                        Dim.1   Dim.2   Dim.3   Dim.4
## Variance               2.918   0.914   0.147   0.021
## % of var.             72.962  22.851   3.669   0.518
## Cumulative % of var.  72.962  95.813  99.482 100.000
##
## Individuals (the 10 first)
##                   Dist    Dim.1    ctr   cos2    Dim.2    ctr   cos2    Dim.3
## 1              |  2.319 | -2.265  1.172  0.954 |  0.480  0.168  0.043 | -0.128
## 2              |  2.202 | -2.081  0.989  0.893 | -0.674  0.331  0.094 | -0.235
## 3              |  2.389 | -2.364  1.277  0.979 | -0.342  0.085  0.020 |  0.044
## 4              |  2.378 | -2.299  1.208  0.935 | -0.597  0.260  0.063 |  0.091
## 5              |  2.476 | -2.390  1.305  0.932 |  0.647  0.305  0.068 |  0.016
## 6              |  2.555 | -2.076  0.984  0.660 |  1.489  1.617  0.340 |  0.027
## 7              |  2.468 | -2.444  1.364  0.981 |  0.048  0.002  0.000 |  0.335
## 8              |  2.246 | -2.233  1.139  0.988 |  0.223  0.036  0.010 | -0.089
## 9              |  2.592 | -2.335  1.245  0.812 | -1.115  0.907  0.185 |  0.145
## 10             |  2.249 | -2.184  1.090  0.943 | -0.469  0.160  0.043 | -0.254
##                   ctr   cos2
## 1              0.074  0.003 |
## 2              0.250  0.011 |
## 3              0.009  0.000 |
## 4              0.038  0.001 |
## 5              0.001  0.000 |
## 6              0.003  0.000 |
## 7              0.511  0.018 |
## 8              0.036  0.002 |
## 9              0.096  0.003 |
## 10             0.293  0.013 |
##
## Variables
##                  Dim.1    ctr   cos2    Dim.2    ctr   cos2    Dim.3    ctr
## Sepal.Length |  0.890 27.151  0.792 |  0.361 14.244  0.130 | -0.276 51.778
## Sepal.Width  | -0.460  7.255  0.212 |  0.883 85.247  0.779 |  0.094  5.972
## Petal.Length |  0.992 33.688  0.983 |  0.023  0.060  0.001 |  0.054  2.020
## Petal.Width  |  0.965 31.906  0.931 |  0.064  0.448  0.004 |  0.243 40.230
##                 cos2
## Sepal.Length  0.076 |
## Sepal.Width   0.009 |
## Petal.Length  0.003 |
## Petal.Width   0.059 |
```

```
fviz_eig(iris.pca, addlabels = TRUE)
```




```
fviz_pca_biplot(iris.pca, col.ind = iris$Species, addEllipses = TRUE,
    label = "var", col.var = "black")
```



Question 3

En analysant les résultats obtenus, on s'aperçoit que la première composante résume une bonne partie de l'information de base sur le data set *Iris* (pourcentage de variance à 73%). On remarque également avec le graphe que les variables *Petal.Length* et *Petal.Width* sont très fortement corrélées sur la première composante, ce qui est cohérent avec ce qui est résumé dans la matrice de corrélation (corrélation égale à environ 0.96, très proche de 1), et que de même, *Sepal.Length* est fortement corrélée aux deux dernières variables (corrélation entre 0.8 et 0.9). La deuxième composante résume globalement l'information sur la variable restante, *Sepal.Width* (23% de la variance totale), qui est moins corrélée aux trois autres (information qui se retrouve dans le corrplot effectué ci-dessus et dans la matrice de corrélation). Les deux composantes à elles seules résument la quasi-totalité de l'information (pourcentage de variance cumulée proche de 96% !).

La première composante est positivement corrélée avec *Sepal.Length*, *Petal.Length* et *Petal.Width*, et négativement corrélée avec *Sepal.Width*. La deuxième composante est positivement corrélée avec cette dernière.