Data Mining - TP clustering

Julien JACQUES

06/03/2019

Kmeans sur les données MNIST

Chargeons les images soit avec la fonction mnist.R si on a téléchargé les fichiers de données bruts depuis le site de Y. Lecun

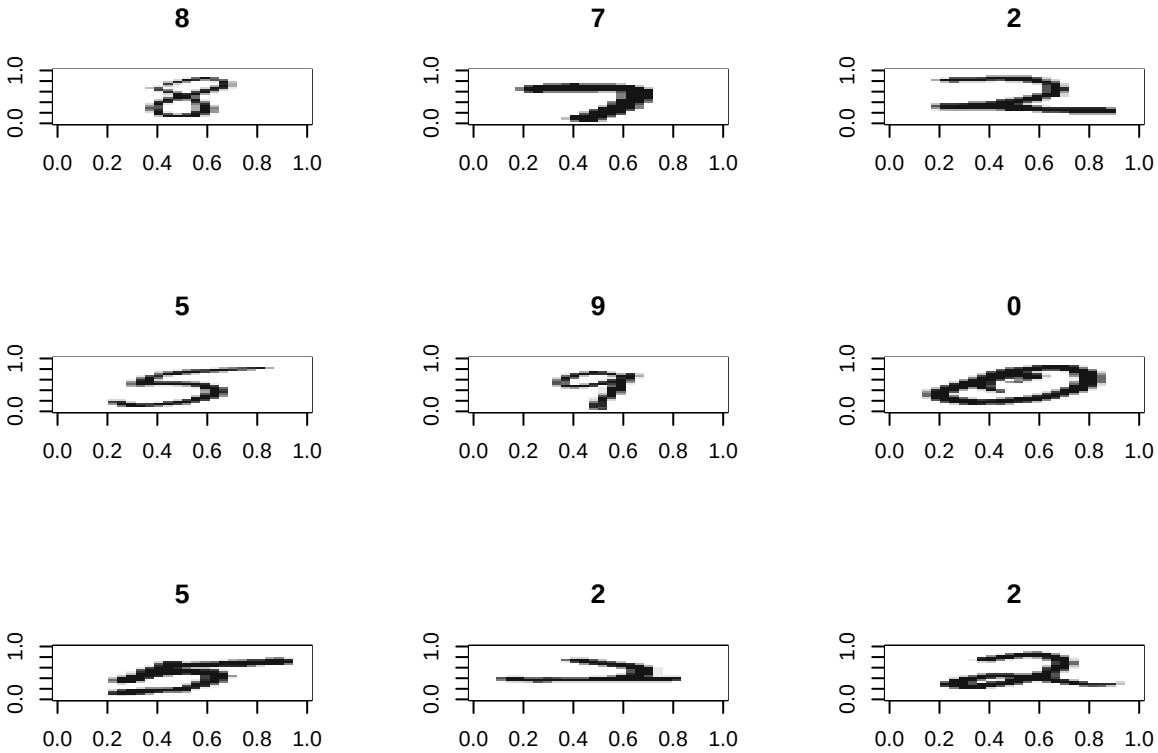
```
source("mnist.R")
```

soit directement à partir du fichier MNIST.Rdata :

```
load("MNIST.Rdata")
```

Affichons aléatoirement 9 images

```
numero=sample(1:60000,9)
par(mfrow=c(3,3))
for (i in 1:9) show_digit(train$x[numero[i],],title=train$y[numero[i]])
```



On extrait 1000 images aléatoirement

```
#index=sample(1:60000,1000)
index=1:1000
```

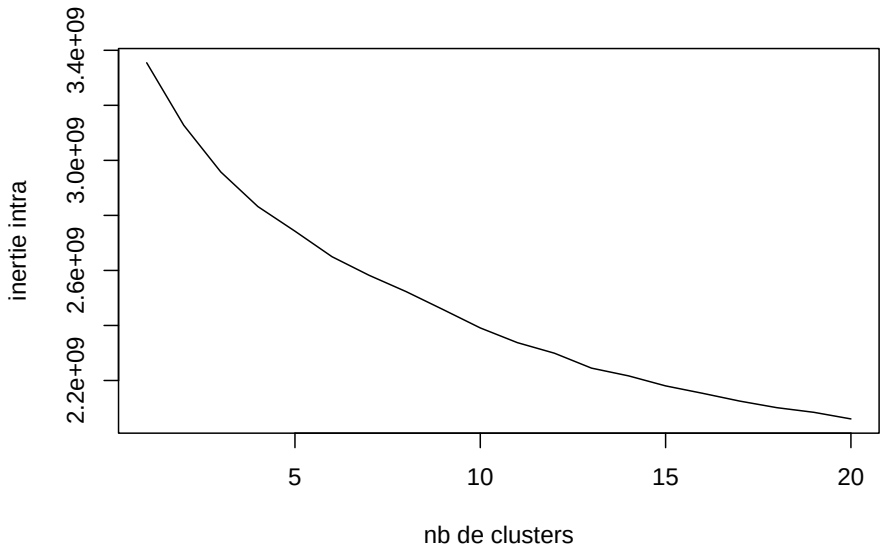
```
x=train$x[index,]
y=train$y[index]
```

On réalise un kmeans pour un nombre de classes entre 1 et 20

```
classes=1:20
inertie_w=NULL
for (K in classes){
  cat('k-means with ',K,' clusters \n')
  res=kmeans(x,centers = K,nstart = 5)
  inertie_w=c(inertie_w,res$tot.withinss)
}
```

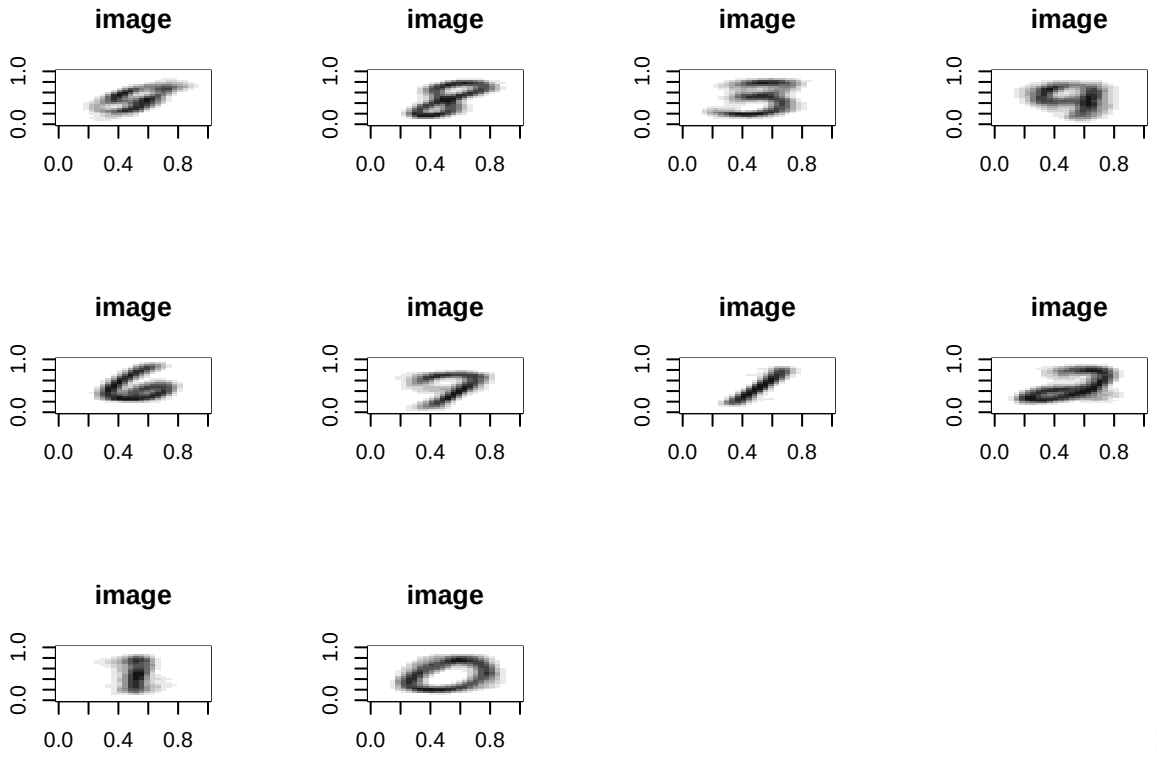
```
## k-means with  1  clusters
## k-means with  2  clusters
## k-means with  3  clusters
## k-means with  4  clusters
## k-means with  5  clusters
## k-means with  6  clusters
## k-means with  7  clusters
## k-means with  8  clusters
## k-means with  9  clusters
## k-means with  10  clusters
## k-means with  11  clusters
## k-means with  12  clusters
## k-means with  13  clusters
## k-means with  14  clusters
## k-means with  15  clusters
## k-means with  16  clusters
## k-means with  17  clusters
## k-means with  18  clusters
## k-means with  19  clusters
## k-means with  20  clusters
```

```
plot(classes,inertie_w,ylab='inertie intra',xlab='nb de clusters',type='l')
```



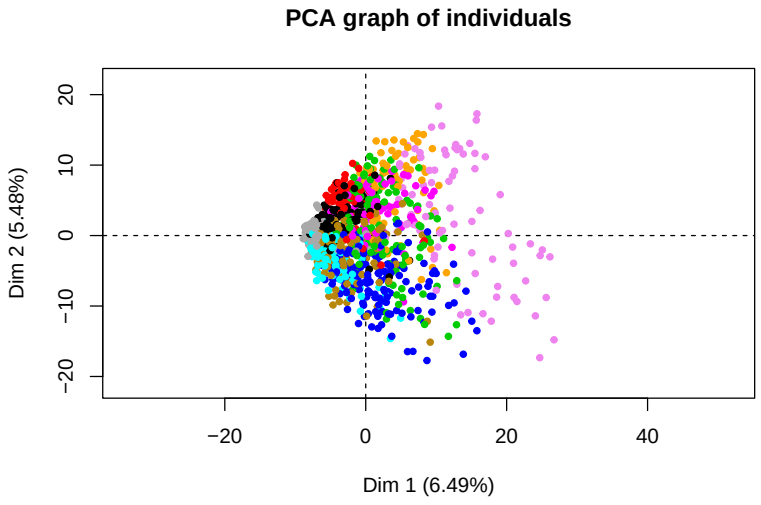
Pas de coude spécifique, on va choisir 10 clusters

```
K=10
res=kmeans(x,centers = K,nstart=10)
par(mfrow=c(3,4))
for (i in 1:K) show_digit(res$centers[i,])
```



Représentons les données clusterisées dans le plan de l'ACP

```
library("FactoMineR")
res.pca <- PCA(x,graph = F)
plot(res.pca,choix="ind",col.ind=res$cluster,
     graph.type = "classic",label='none')
```



Comparons les clusters obtenus aux chiffres présent sur les images

```
table(res$cluster,y)
```

```
##     y
##       0  1  2  3  4  5  6  7  8  9
##   1   5  0  2  1 38 33  5  8  3 17
##   2   1  1  1  4  0  1  0  0 53  0
##   3   1  0  4 69  0 42  0  0  8  2
##   4   5  1  7  6 50  7 11 21 10 37
##   5   4  0  1  0  2  2 68  0  0  0
##   6   0  0  3  2 10  0  0 70  0 34
##   7   0 62  9  0  0  0  1  5  1  1
##   8   5  0 56  1  0  1  1  0  0  0
##   9   0 52 16  9  5  4  7 13 12  8
##   10 76  0  0  1  0  2  1  0  0  1
```

On peut regarder grâce à cette matrice de confusion dans quelle classe sont répartis les différentes images.

On peut aussi calculer l'ARI

```
library(mclust)
```

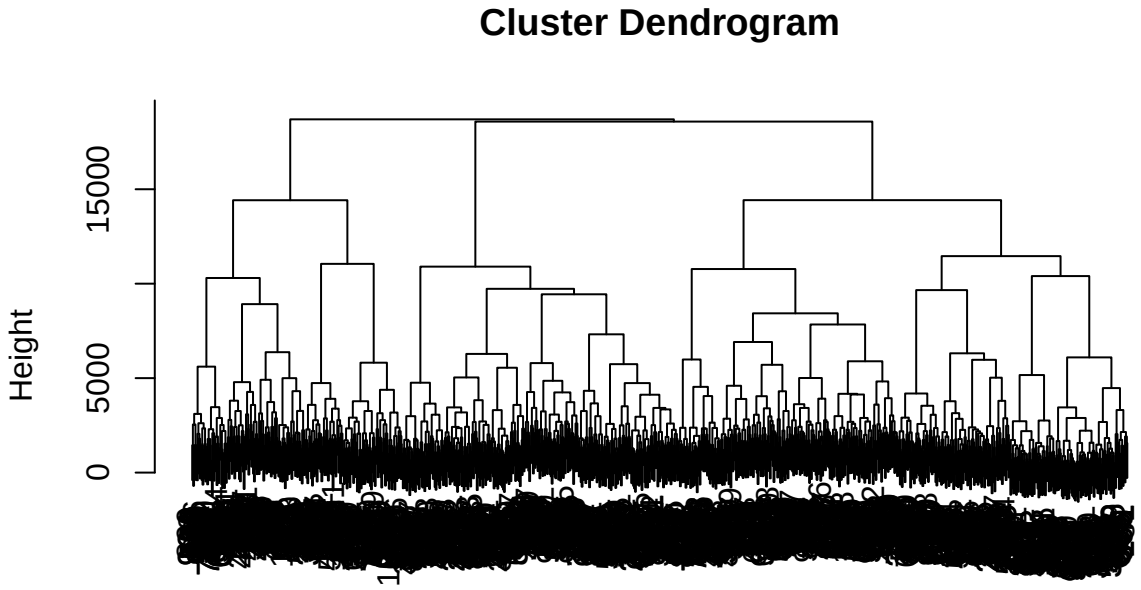
```
## Package 'mclust' version 6.1.1
## Type 'citation("mclust")' for citing this R package in publications.
```

```
adjustedRandIndex(res$cluster,y)
```

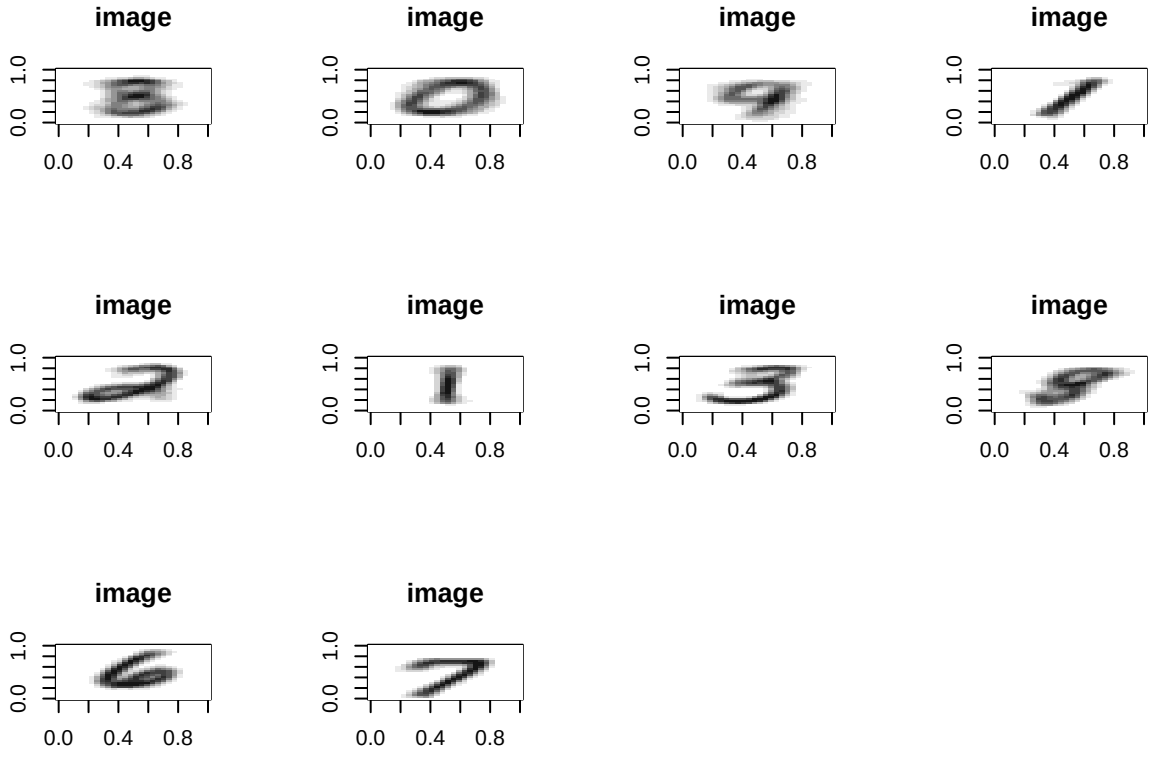
```
## [1] 0.3488921
```

avec la CAH

```
d=dist(x)
res2=hclust(d,method="ward.D2")
plot(res2)
```




```
classe=cutree(res2,k = 10)
par(mfrow=c(3,4))
for (i in 1:10) show_digit(colMeans(x[classe==i,]))
```



On peut comparer à la partition kmeans

```r
table(res$cluster,classe)
```

```
##     classe
##        1   2   3   4   5   6   7   8   9  10
##   1    3   3  39   0   2   0   0  59   5   1
##   2   10   1   1   1   0   0   0  48   0   0
##   3   68   7   2   0   0   0  45   4   0   0
##   4   27   7 119   0   0   0   0   1   1   0
##   5   15   4   2   0   0   0   0   0  56   0
##   6    2   5  65   1   0   0   0   2   0  44
##   7   11   0   0  64   0   1   0   3   0   0
##   8    9  13   0   0  39   0   0   2   1   0
##   9   45   0  22  11   0  48   0   0   0   0
##   10   0  81   0   0   0   0   0   0   0   0
```

```r
adjustedRandIndex(res$cluster,y)
```

```
## [1] 0.3488921
```

```r
adjustedRandIndex(classe,y)
```

```
## [1] 0.3420384
```

```r
adjustedRandIndex(res$cluster,classe)
```

```
## [1] 0.3577483
```

avec DBSCAN

```r
library(dbscan)
clus=dbscan(x, eps = .7, minPts = 5)
print(clus)
```

```
## DBSCAN clustering for 1000 objects.
## Parameters: eps = 0.7, minPts = 5
## Using euclidean distances and borderpoints = TRUE
## The clustering contains 0 cluster(s) and 1000 noise points.
##
##    0
## 1000
##
## Available fields: cluster, eps, minPts, metric, borderPoints
```