Partitionnement sur les données

Manon MAHEO - Valentin PENISSON

11/01/2022

```
# Installations nécessaires
# https://cran.r-project.org/web/packages/dbscan/readme/README.html

library(dbscan)
library(ggpubr)
library(factoextra)
library(ggplot2)
library(fpc)
library(cluster)
```

Exercice 1 (Jeux de données de petite dimension).

Importation des données

Nous allons travailler sur le jeu de données ***Lsun.lrn***. On utilise également le fichier ***Lsun.cls*** pour obtenir les labels.

```
# Importation des variables continues
lsun <-  read.csv(file = "Lsun.lrn", header = FALSE, sep = "\t", skip = 4)[,-1]
head(lsun)
```

```
##         V2       V3
## 1 3.277701 0.814082
## 2 0.387577 0.176780
## 3 0.268546 0.582963
## 4 2.031145 0.244597
## 5 0.188677 0.461280
## 6 3.525472 0.265579
```

```
# Ici, on enlève les 4 premières lignes du fichier avec skip = 4
# et on ne conserve pas l'index des lignes en retirant la première colonne
```

```
# Importation des labels
labels <-  as.factor(read.csv(file = "Lsun.cls", header = FALSE,
                              sep = "\t",skip = 1)[,-c(1,3)])
head(labels)
```

```
## [1] 1 1 1 1 1 1
## Levels: 1 2 3
```

```
# Fusion et création de notre jeu de données
data <-  data.frame(x = lsun[,1], y = lsun[,2], labels = labels)
head(data)
```

```
##          x        y labels
```

```
## 1 3.277701 0.814082      1
## 2 0.387577 0.176780      1
## 3 0.268546 0.582963      1
## 4 2.031145 0.244597      1
## 5 0.188677 0.461280      1
## 6 3.525472 0.265579      1
```

Notre jeu de données data contient ainsi des données 2D avec **trois classes distinctes** (i.e classes 1, 2 et 3). L'ensemble de données se compose donc de n = 400 points dans trois groupes distincts sur un plan.

```
# Visualisation des données
graph <- ggplot(data = data, aes(x=x, y =y)) + geom_point(aes(color = labels)) +
  ggtitle("Original data")
graph
```

Deux classes sont dessinées uniformément réparties à l'intérieur d'un rectangle de 1 × 4. Ces classes sont disposées sous la forme d'un « L » avec un espace entre les deux. Le troisième groupe est tiré d'une gaussienne standard bidimensionnelle indépendante et identiquement distribuée centrée en (3,2.5). Les classes sont bien séparées par une distance minimale de 0,6 et les classes sont linéairement séparables.

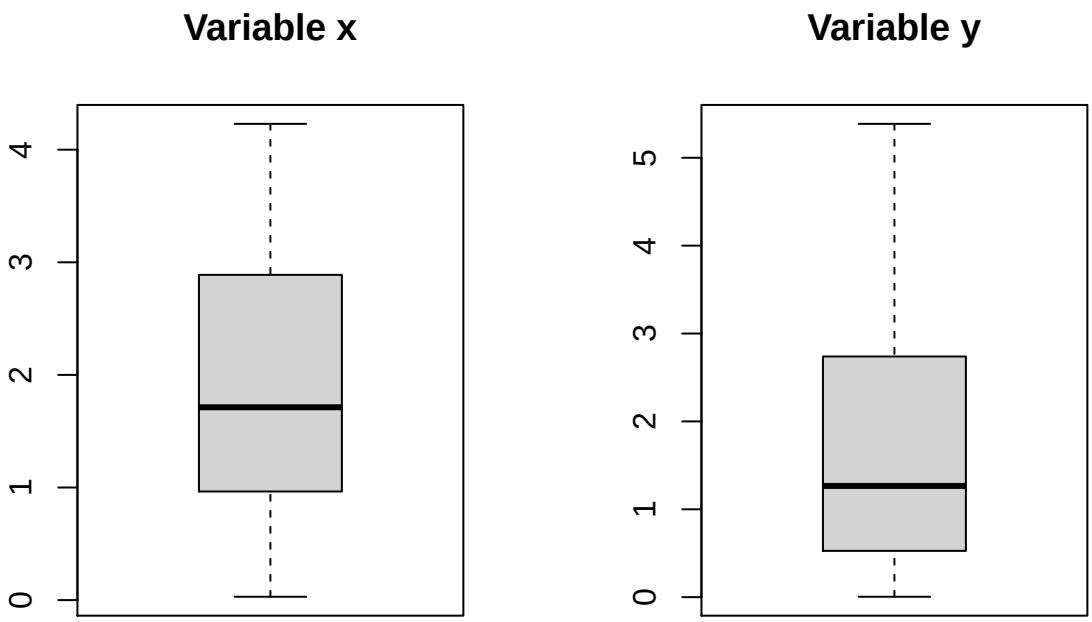
```
# Gestion des données manquantes

manquant <- is.na(data)
sum(manquant)
```

```
## [1] 0
```

Il n'y a pas de donnée manquante.

```r
# Echelle des variables
# On peut avoir des groupes très différents selon l'unité choisie.
# Il est important d'avoir des variables avec la même unité.
par(mfrow=c(1,2))
for (i in 1:2) boxplot(data[,i], main=paste("Variable ",names(data)[i],sep=""))
```



On peut faire des boxplots et voir si les échelles sont vraiment différentes. Ici il n'est pas vraiment nécessaire de standardiser les données avec la fonction scale().

Question 1 : Implémenter deux méthodes de partitionnement de votre choix (i.e. méthode de classification non supervisée vue en cours).

On se place ici dans une approche non-supervisée (i.e clustering). On dispose juste des individus mais pas des groupes. L'objectif est de rassembler des individus en classes créées de toute pièce. L'objectif est de constituer k groupes ou partitions homogènes à partir de ces individus. Les conditions sont les suivantes : les individus appartiennent à un seul groupe, on veut des groupes compacts (i.e variance faible au sein d'un groupe) et des groupes bien séparés (i.e variance forte entre les groupes). On implémente ici la méthode des k-means et la méthode dbscan.

Méthode 1 : K-means

```r
# On implémente ici la méthode k-means
# On choisit k = 3 car il y a trois labels différents
set.seed(123)
res.km <- kmeans(data[,1:2], 3 , nstart = 25)
res.km
```

```
## K-means clustering with 3 clusters of sizes 165, 155, 80
##
## Cluster means:
##          x        y
## 1 3.066889 1.710037
## 2 1.127876 0.715386
## 3 1.052019 3.979816
##
```

```
## Clustering vector:
##   [1] 1 2 2 2 2 1 2 1 1 2 2 1 1 2 2 1 2 2 2 2 2 1 2 2 2 2 1 1 1 1 1 2 1 2 1 2
##  [38] 2 1 2 1 2 2 1 2 2 2 2 1 2 2 2 1 2 2 1 2 2 2 2 2 2 1 2 1 2 2 2 2 2 2 2 1
##  [75] 2 1 1 2 2 2 2 2 2 2 1 1 1 2 1 2 2 1 2 2 2 1 2 2 1 1 1 2 2 1 2 1 2 2 1 1 2
## [112] 2 2 1 2 2 2 2 2 2 1 2 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 1 1 1 1 2 2 1 2 2 2 1
## [149] 1 2 2 1 2 2 2 2 2 1 2 2 2 2 2 1 1 2 1 2 1 2 2 1 2 2 1 2 2 1 1 2 1 1 2 1 2
## [186] 1 2 2 2 2 1 1 2 2 2 1 2 2 2 2 3 3 3 3 2 3 2 3 3 3 3 3 3 3 3 3 2 3 3 2 3
## [223] 3 3 3 3 3 3 2 3 3 2 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 2 3 3 3 3 3 2 3 2
## [260] 3 2 2 3 3 3 2 2 2 3 2 3 3 3 3 3 3 3 2 3 3 3 2 2 3 3 3 2 3 3 3 3 3 2 3 3
## [297] 3 2 2 3 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [334] 1 1 1 3 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [371] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##
## Within cluster sum of squares by cluster:
## [1] 208.09962 107.97031  65.57567
##  (between_SS / total_SS =  71.1 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

On observe ici dans quelle classe les individus ont été affectés et l'inertie intra-classes au sein de chaque des classe.

```
# On lance ici les prédictions
predict_label.km <- as.character(res.km$cluster)
```

Méthode 2 : DBSCAN (clustering basé sur la densité)

```
# On implémente ici la méthode dbscan
set.seed(123)
res.dbscan <- dbscan(data[,1:2], eps = 0.5, MinPts = 2)
res.dbscan
```

```
## dbscan Pts=400 MinPts=2 eps=0.5
##         1   2   3
## seed  200 100 100
## total 200 100 100
```

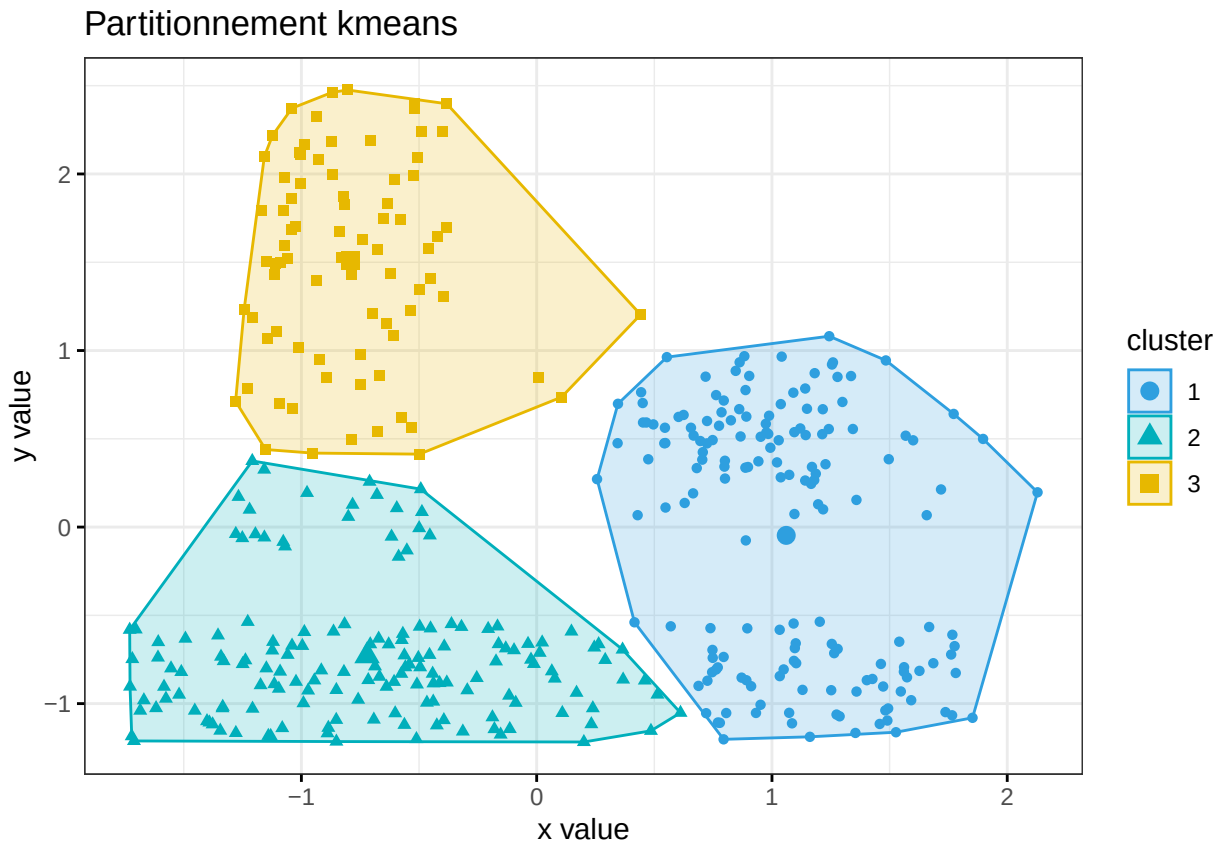
```
# Prédiction de la méthode dbscan
predict_label.dbscan <- as.factor(res.dbscan$cluster)
```

Question 2 : Bien présenter les résultats obtenus, les comparer visuellement aux vraies étiquettes lorsqu'elles sont connues.

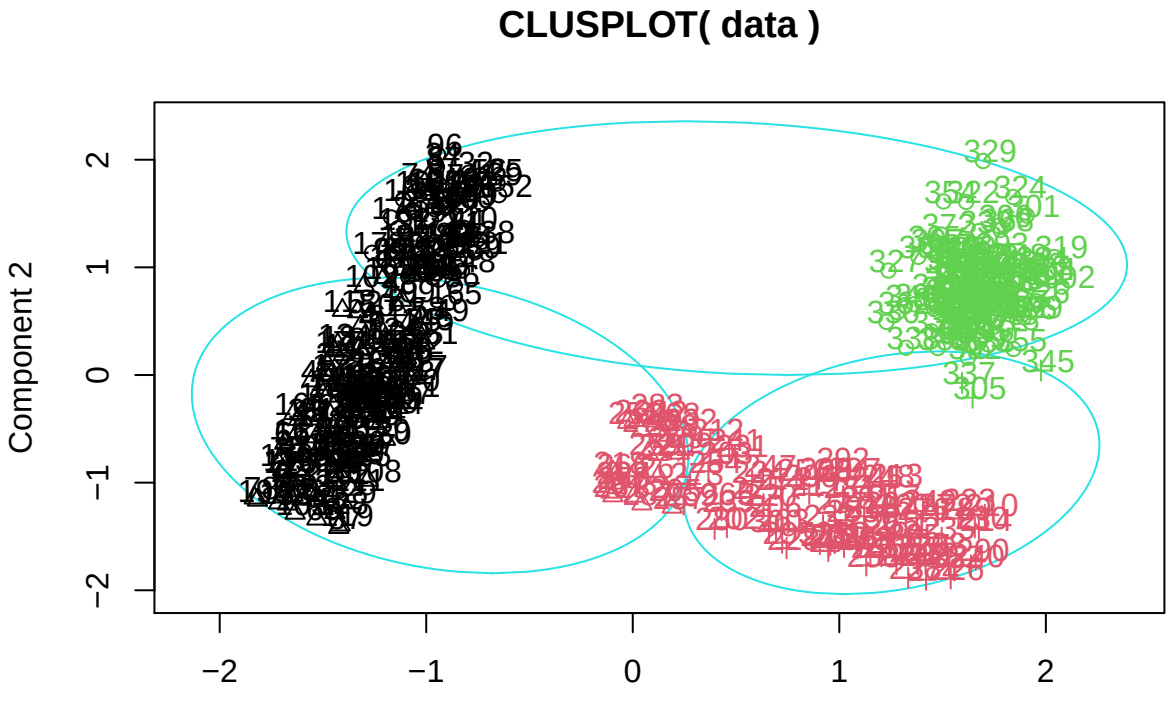
Méthode 1 : K-means

```
# Visualisation du partitionnement avec kmeans et comparaison
graph
```


```
fviz_cluster(res.km, data = data[,1:2],
             palette = c("#2E9FDF", "#00AFBB", "#E7B800"),
             geom = "point",
             ellipse.type = "convex",
             ggtheme = theme_bw(),
             main = "Partitionnement kmeans"
             )
```




```
clusplot(data,res.km$cluster,labels=3,col.p=as.numeric(data$labels))
```



These two components explain 93.11 % of the point variability.

```
table(res.km$cluster, data$labels)
```

```
##
##        1   2   3
##    1  68   0  97
##    2 132  23   0
##    3   0  77   3
```

Si on compare les deux graphiques, on remarque visuellement que certains individus du groupe 1 (en rouge) ont été classés dans le groupe bleu (groupe 1) avec le partitionnement de kmeans. Egalement, certains individus du groupe 2 (en vert) ont été classés dans le groupe vert (groupe 2) du partitionnement de kmeans. Plus particulièrement, on si on prend les individus prédits dans le groupe 1 avec l'algorithme de kmeans, 68 sont dans le bon groupe mais 97 sont dans le mauvais (et sont dans le groupe 3).

Nous allons tenter ici des modifications de l'algorithme des k-means avec des fonctions distance différentes que la norme Euclidienne. En ce sens, il est important aussi de savoir appliquer des méthodes de partitionnement de données plus évoluées que l'algorithme des K-means.

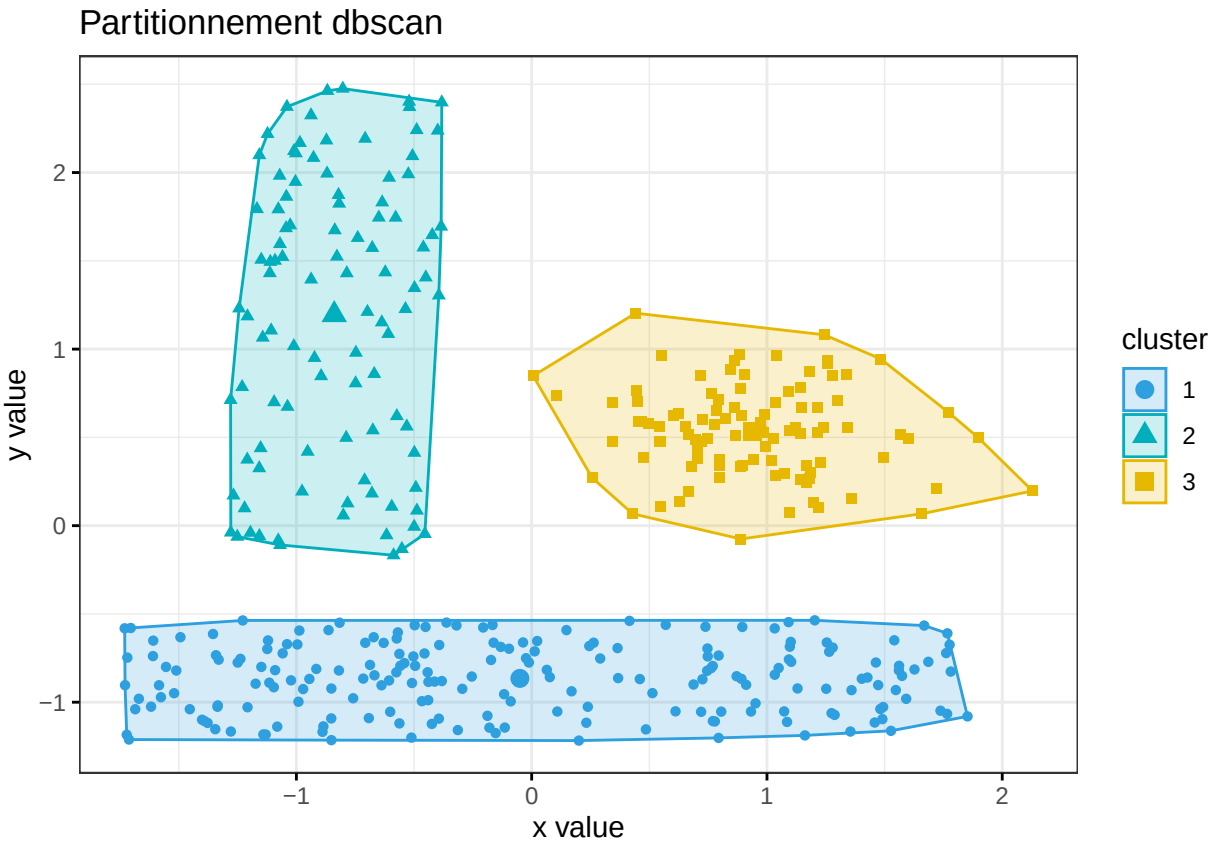
```
# Amélioration de l'algorithme des K-means
```

Méthode 2 : DBSCAN (clustering basé sur la densité)

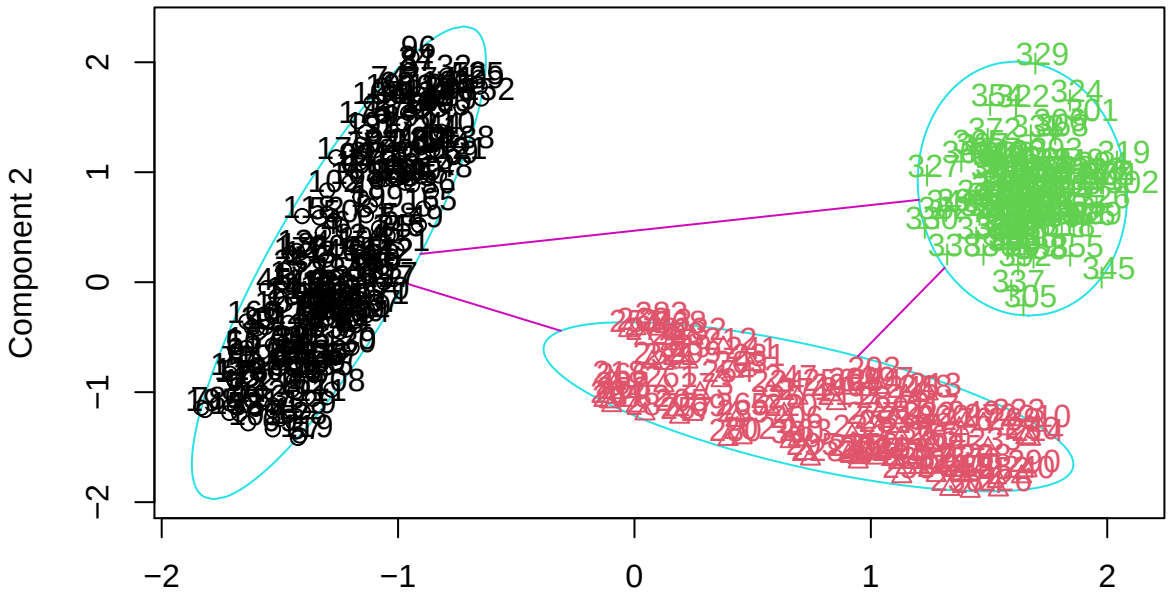
```
# Visualisation du partitionnement avec dbscan et comparaison
graph
```


```
fviz_cluster(res.dbscan, data = data[,1:2],
             palette = c("#2E9FDF", "#00AFBB", "#E7B800"),
             geom = "point",
             ellipse.type = "convex",
```

```
            ggtheme = theme_bw(),
            main = "Partitionnement dbscan"
            )
```




```
clusplot(data,res.dbscan$cluster,labels=3,col.p=as.numeric(data$labels))
```




```
table(res.dbscan$cluster, data$labels)
```

```
##
##        1   2   3
##   1  200   0   0
##   2    0 100   0
##   3    0   0 100
```

Avec l'algorithme dbscan, on observe un partitionnement parfait des données. Chaque individu est prédit dans le bon groupe.

Question 3 : Pour chacune des méthodes utilisées, expliquer son fonctionnement.

Méthode 1 : K-means

L'algorithme des k-means est un algorithme de classification non supervisée qui cherche à définir K classes $C_1, C_2, ..., C_K$ tels que $\cup_{k=1}^{K} C_k = \mathbb{X}$. Son objectif est de minimiser l'inertie intra-classes, ou de trouver les K centres $c_1^*, c_2^*, ..., c_K^*$ et les K classes $C_1^*, C_2^*, ..., C_K^*$ mnimisant $(c_1, c_2, ..., c_K) \mapsto \frac{1}{n} \sum_{i=1}^{N} min_{k=1,...,K} ||x - \overline{x}_{C_k}||^2$. L'algorithme fonctionne de la façon suivante : on chosit un nombre de classes k (ici k = 3). Puis, on choisit aléatoirement $c_1^*, c_2^*, ..., c_K^*$, nos K centroïdes. Pour chaque point x_i, on cherche le centroïde $c_1^*, c_2^*, ..., c_K^*$ le plus proche et on lui associe x_i. Pour chaque classe $j = 1, ...K$: on calcule les nouvelles coordonnées du centroïde en faisant la moyenne des points qui lui ont été associés. On répète les étapes deux dernières étapes un grand nombre de fois (par exemple 10 000) ou jusqu'à convergence.

Méthode 2 : DBSCAN (clustering basé sur la densité)

Tout comme l'algorithme des k-means, DBSCAN est un algorithme de classification non supervisée. DBSCAN est une méthode de partitionnement qui a été introduite dans Ester et al. (1996). Sa particularité est qu'il prend en compte la géométrie des données en se reposant sur des densités ou des fonctions distance. En effet, cette méthode s'appuie sur deux paramètres : la distance ϵ et $MinPts$, le nombre de points minimum

devant se trouver dans un rayon ϵ pour être considérés comme une classe. Cette méthode créee donc, en fonction de ces paramètres, un certain nombre de classes : ce n'est donc pas l'utilisateur qui le définit au préalable. L'idée de cet algorithme est de prendre un point, d'observer ses points voisins dans le rayon ϵ, et de regarder s'il vérifit bien $MinPts$. Dans ce cas, le point choisit fait parti d'un cluster. L'algorithme DBSCAN fonctionne de la façon suivante :

Pour chaque point p non visité du jeu de données, marquer comme vu le point p. Chercher tous les points voisins de p (= tous les points autour de p avec un rayon ϵ). Si p a moins de voisins que $MinPts$: on définit p comme un point aberrant. Sinon : on ajoute p à une classe C. Pour chaque point p' des points voisins de p : si p' n'a pas été visité : on marque p' comme visité. On cherche tous les points voisins de p'. Si p' a plus de voisins que $MinPts$: jointure des voisins de p et de p'. Si p' n'a pas encore de classe C on ajoute p' à C.

Exercice 2 (Partitionnement de données en plus grande dimension).

Question 1 : Appliquer une méthode de partitionnement de données sur un jeu de données réelles de votre choix, issues par exemple de : *https://www.cs.ucr.edu/~eamonn/time_series_data_2018/*