

# Méthodes de type K-means et DBSCAN

Cathy Maugis-Rabusseau

4modIA / INSA Toulouse & ENSEEIHT

2022-2023

# Introduction

- Données : On observe  $n$  individus décrits par  $p$  variables

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix} \text{ avec } x_i = (x_{i1}, \dots, x_{ip}) \in \mathcal{X}$$

- Soit  $d$  une distance euclidienne
- But : trouver une partition de l'ensemble des individus telle que l'inertie intra-classe soit minimale

# Plan

- 1 Méthodes de type K-means
- 2 Méthodes de type K-médoïdes
- 3 Choix des noyaux initiaux
- 4 Choix du nombre de classes
- 5 DBSCAN

# Principe général des méthodes de type K-means

- Initialisation :
  - ▶ choix du nombre de classes  $K$
  - ▶ choix de  $K$  noyaux initiaux  $c_1^{(0)}, \dots, c_K^{(0)}$
- On itère les deux étapes suivantes : à l'itération  $t$ 
  - ▶ Affectation des individus à la classe la plus proche :
$$i \in \mathcal{C}_k^{(t)} \text{ si } d\left(x_i, c_k^{(t-1)}\right) = \min_{1 \leq k' \leq K} d\left(x_i, c_{k'}^{(t-1)}\right)$$
  - ▶ Mise à jour des noyaux :  $c_1^{(t)}, \dots, c_K^{(t)}$
- Arrêt de l'algorithme quand la classification n'est plus modifiée

## Méthode des centres mobiles [3]

- $c_1^{(0)}, \dots, c_K^{(0)}$  sont choisis au hasard (sans remise) dans  $\mathcal{X}$
- A l'itération  $t$  :
  - ▶  $i \in \mathcal{C}_k^{(t)}$  si  $d\left(x_i, c_k^{(t-1)}\right) = \min_{1 \leq k' \leq K} d\left(x_i, c_{k'}^{(t-1)}\right)$
  - ▶ Mise à jour des noyaux :
- Arrêt de l'algorithme quand 2 itérations successives fournissent la même classification

$$\forall k \in \{1, \dots, K\}, \quad c_k^{(t)} = \frac{1}{|\mathcal{C}_k^{(t)}|} \sum_{i \in \mathcal{C}_k^{(t)}} x_i$$

## Méthode des centres mobiles [3]

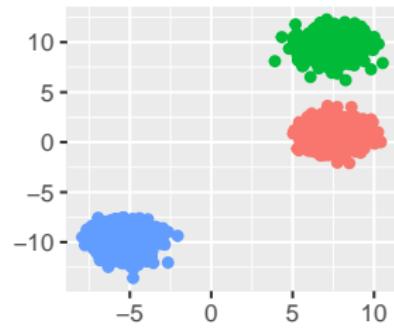
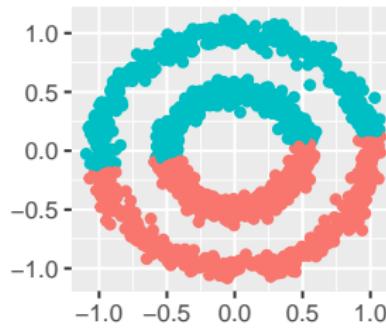
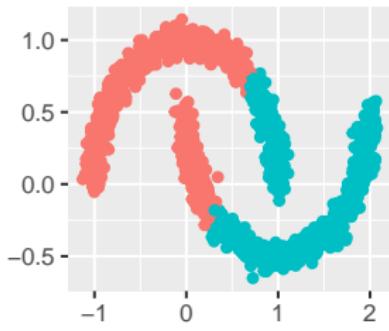
# Points forts / faibles

- Avantages :

- ▶ Relativement efficace (rapide)
- ▶ Tend à réduire l'inertie intra-classe à chaque itération
- ▶ Forme des classes compactes, bien séparées

- Inconvénients :

- ▶ Influence du choix des noyaux initiaux
- ▶ Convergence vers un minimum local
- ▶ Nécessite la notion de centre de gravité
- ▶ Influence des outliers
- ▶ Non adapté pour des classes non convexes



# Propriétés

## Proposition:

L'inertie intraclasse  $I_{\text{intra}}(\mathcal{P}_K^{(t)})$  décroît à chaque étape.

⇒ convergence vers un minimum local de l'inertie intra-classe.

## Preuve:

Les deux points clés sont :

- si  $i$  passe de  $\mathcal{C}_k^{(t-1)}$  à  $\mathcal{C}_{k'}^{(t)}$  alors

$$d(x_i, c_{k'}^{(t-1)})^2 \leq d(x_i, c_k^{(t-1)})^2$$

- $c_k^{(t)}$  étant le centre de  $\mathcal{C}_k^{(t)}$

$$\sum_{i \in \mathcal{C}_k^{(t)}} d(x_i, c_k^{(t)})^2 \leq \sum_{i \in \mathcal{C}_k^{(t-1)}} d(x_i, c_k^{(t-1)})^2$$

## Variante : K-means [5]

- Initialisation :
  - ▶ choix de  $K$
  - ▶  $c_1^{(0)}, \dots, c_K^{(0)}$  sont tirés aléatoirement parmi les  $n$  points observés
- A chaque itération  $t$ 
  - ▶ tirage au hasard d'un nouveau point  $x_i \in \mathcal{X}$
  - ▶ détermination du noyau  $c_k^{(t-1)}$  le plus proche de  $x_i$
  - ▶ mise à jour du noyau :

$$c_k^{(t)} = \frac{x_i + |\mathcal{C}_k^{(t-1)}| c_k^{(t-1)}}{|\mathcal{C}_k^{(t-1)}| + 1}$$

# Nuées dynamiques [1]

- Initialisation : choix de  $K$  sous-ensembles disjoints  $N_1^{(0)}, \dots, N_K^{(0)}$  de cardinal  $q$  chacun ( $q$  fixé)
- A l'itération  $t$  :
  - On affecte les points de  $\mathcal{X}$

$$\mathcal{C}_k^{(t)} = \left\{ x \in \mathcal{X}; D\left(x, N_k^{(t-1)}\right) \leq D\left(x, N_r^{(t-1)}\right) \forall r \neq k \right\}$$

avec  $D(A, B) = \sum_{x \in A} \sum_{y \in B} d(x, y)$  pour toutes parties  $A, B$  de  $\mathcal{X}$

- Mise à jour des noyaux  $(N_k^{(t)})_{1 \leq k \leq K}$ :  $N_k^{(t)}$  est le sous-ensemble de cardinal  $q$  qui minimise

$$\sum_{x \in N_k^{(t)}} D\left(x, \mathcal{C}_k^{(t)}\right).$$

# Quelques commandes



- Avec R

```
km=kmeans(x, centers=, iter.max=, algorithm=)
```

- *centers*: vecteur des noyaux initiaux ou nombre de classes
- *iter.max*: nombre d'itérations maximale
- *algorithm*: "Forgy" (centres mobiles), "MacQueen" ( $K$ -means), nuées dynamiques par défaut

```
km$cluster, km$centers, km$withinss, km$size
```



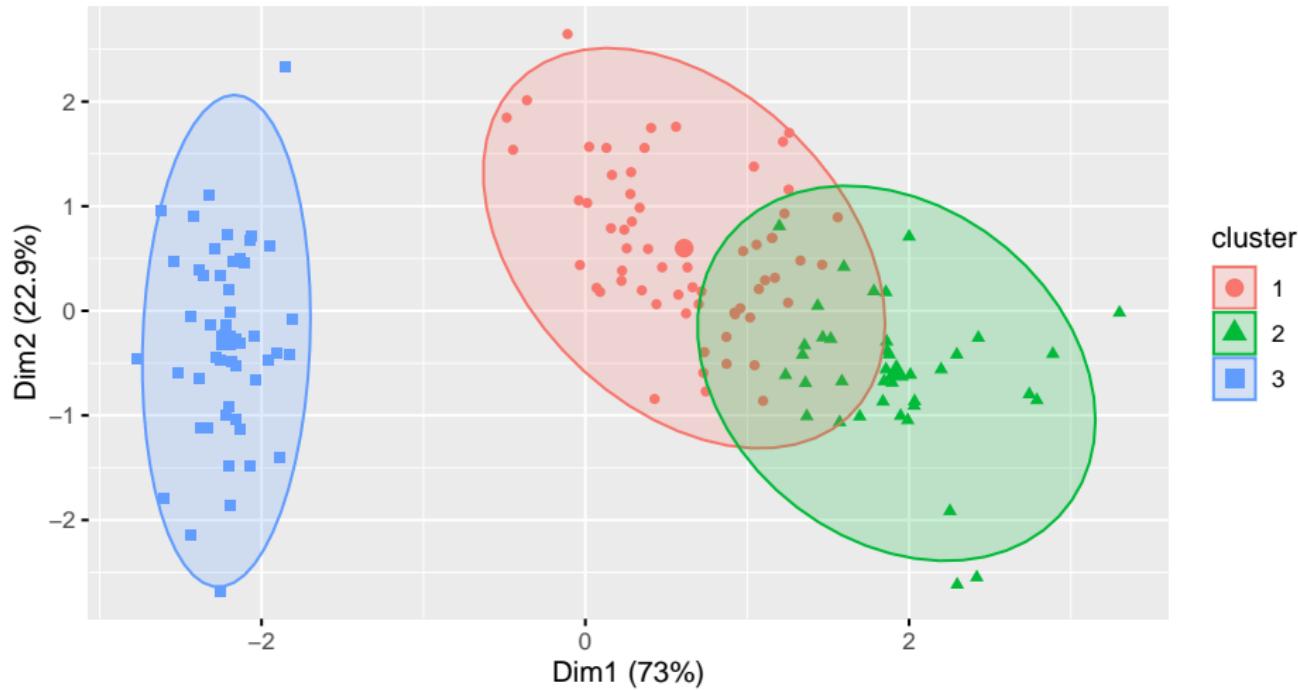
- Avec Python

```
kmeans=sklearn.cluster.KMeans(n_clusters=...); kmeans.fit(data)  
kmeans.labels_, kmeans.cluster_centers_, ...
```

# Exemple - Données iris



```
data(iris)
kmiris=kmeans(iris[,1:4],centers=3)
fviz_cluster(kmiris,data=iris[,1:4],ellipse.type="norm",labelsize=8,geom=c("point"))+ggtitle("")
```

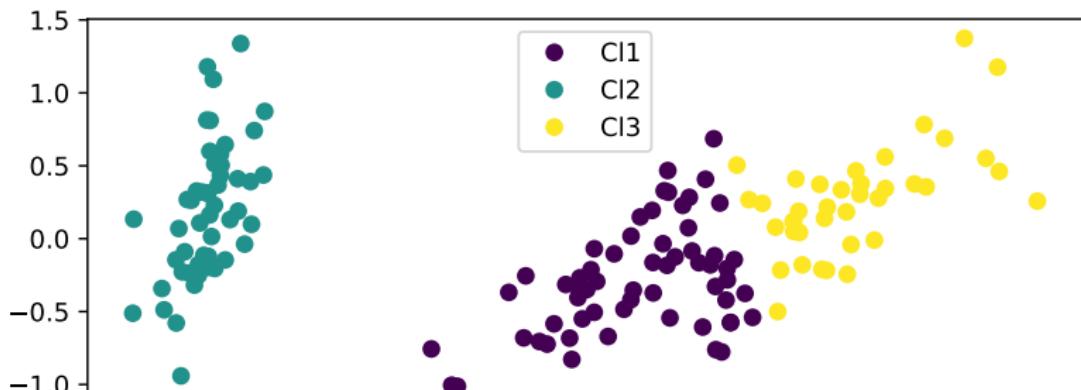




## Exemple - Données iris

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import plotly.express as px
pyiris=r.iris
X=pyiris.iloc[:, [0, 1, 2, 3]].values
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(X)
pca = PCA(n_components=2);
components = pca.fit_transform(X)

scatter = plt.scatter(components[:, 0], components[:, 1], c = y_kmeans)
plt.legend(handles=scatter.legend_elements()[0], labels=['Cl1','Cl2','Cl3'], loc=9)
plt.show()
```



# Extensions des K-means

- Pour variables **qualitatives** :
  - ▶ Dans les méthodes de type *K*-means, chaque classe est représentée par son centre de gravité, non adapté pour des variables qualitatives
  - ▶ Algorithme ***K*-modes** [Huang, 98] : même principe que l'algorithme des centres mobiles en modifiant la dissimilarité

$$d(x_i, x_\ell) = \sum_{j=1}^p \frac{n_{ij} + n_{\ell j}}{n_{ij} \times n_{\ell j}} \mathbb{1}_{x_{ij} \neq x_{\ell j}}$$

où  $n_{ij} = \#\{v \in \{1, \dots, n\}; x_{vj} = x_{ij}\}$ . Utilisation des modes à la place des centres de gravité

- Pour variables **mixtes** : Algorithme ***K*-prototype** [Huang, 97]
- Fuzzy c-means

# Exemple des races de chiens

- Avec l'algorithme des K-modes

```
library(klaR)
clkmodes<-kmodes(chiens[,-7],3,iter.max=100,weight=FALSE)
adjustedRandIndex(chiens[,7],clkmodes$cluster)
```

```
[1] 0.3066504
```

- Avec l'algorithme des Kmeans sur les coordonnées d'ACM

```
afcm<-MCA(chiens,quali.sup=7,graph=F)
coeff<-afcm$ind$coord
clkmeans<-kmeans(coeff,3,nstart=50)
adjustedRandIndex(chiens[,7],clkmeans$cluster)
```

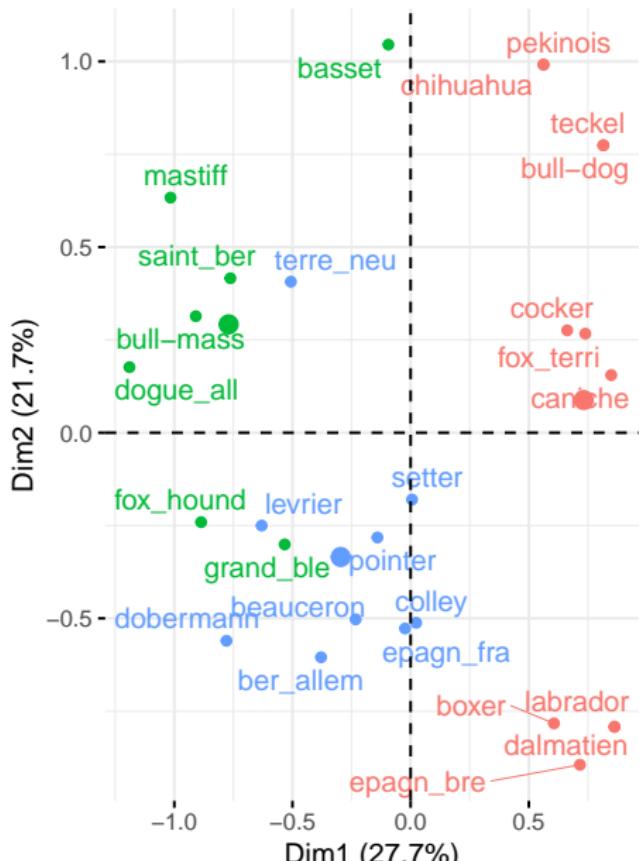
```
[1] 0.2824869
```

```
adjustedRandIndex(clkmodes$cluster,clkmeans$cluster)
```

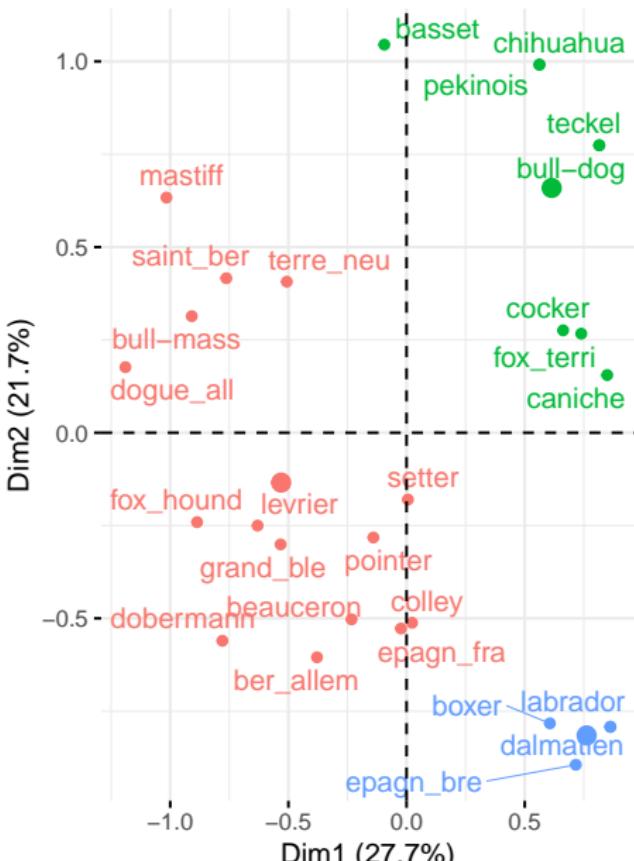
```
[1] 0.4146406
```

# Exemple des races de chiens

Kmodes



MCA+Kmeans



# Plan

- 1 Méthodes de type K-means
- 2 Méthodes de type K-médoïdes
- 3 Choix des noyaux initiaux
- 4 Choix du nombre de classes
- 5 DBSCAN

# Méthode des K-médoïdes

- Idée : On remplace les centres de gravité par des médoïdes (des points de  $\mathbf{X}$ )
- Initialisation : choix aléatoire de  $K$  points (médoïdes) parmi les  $n$
- A chaque itération  $t$ 
  - ▶ on associe chaque point  $x_i$  à son plus proche médoïde
  - ▶ pour chaque médoïde  $m_k^{(t-1)}$  : pour chaque point  $x_i$  qui n'est pas un médoïde, on échange  $x_i$  avec  $m_k^{(t-1)}$  et on calcule le coût total de la configuration
  - ▶ on sélectionne la configuration la moins coûteuse

## PAM (Partitioning Around Medoids)

Cet algorithme a été proposé par Kaufman and Rousseeuw [4].

- Choix de  $K$  médoïdes  $c_1, \dots, c_K$  parmi les  $n$  individus
- Sélectionner au hasard un médoïde  $c_k$  et un autre objet (non médoïde)  $x_i$
- Calculer la qualité de la nouvelle partition si les rôles de  $c_k$  et  $x_i$  sont inversés,
- Échanger  $c_k$  et  $x_i$  si la qualité est supérieure
- Et retourner en (2) jusqu'à stabilité de la qualité de la partition.

# Méthode des K-médoïdes

- Chaque classe est représentée par un individu de la classe donc pas de limitation sur le type de variables prises en compte
- Algorithme efficace pour de petits jeux de données
- PAM est plus robuste que *K-means* en présence de bruit ou d'outliers (un médoïde est moins influencé par un outlier que la moyenne)

# Quelques commandes

- Avec R 

- ▶ `pam(x, k, diss = , metric = , ...)` [library(cluster)]  
metric = “euclidian” ou “manhattan”, sinon x matrice de dissimilarité
- ▶ `clara(x, k, metric = , ...)` [library(cluster)]  
metric = “euclidian” ou “manhattan”, sinon x matrice de dissimilarité  
fonction à privilégier si données de grande dimension

- Avec python



- ▶ `sklearn_extra.cluster.KMedoids()`

# Exemple des iris avec

```
A<-pam(iris[,1:4], 3, metric = "euclidean")
```

```
A$medoids
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
[1,]	5.0	3.4	1.5	0.2
[2,]	6.0	2.9	4.5	1.5
[3,]	6.8	3.0	5.5	2.1

```
A$id.med
```

```
[1] 8 79 113
```

```
adjustedRandIndex(A$clustering,iris$Species)
```

```
[1] 0.7302383
```

```
table(iris$Species,A$clustering)
```

	1	2	3
setosa	50	0	0
versicolor	0	48	2
virginica	0	14	36

# Exemple des iris avec



```
import pandas as pd
from sklearn_extra.cluster import KMedoids
from sklearn.metrics.cluster import adjusted_rand_score
pyiris=r.iris
X=pyiris.iloc[:, [0, 1, 2, 3]].values
resKMed = KMedoids(n_clusters=3).fit(X)
resKMed.medoid_indices_
array([147,  99,   7])
adjusted_rand_score(pyiris["Species"],resKMed.labels_)
```

```
0.7583384522539416
pd.crosstab(resKMed.labels_,pyiris['Species'])

Species    setosa    versicolor    virginica
row_0
0            0           13          49
1            0           37           1
2           50            0           0
```

# Exemple des races de chien

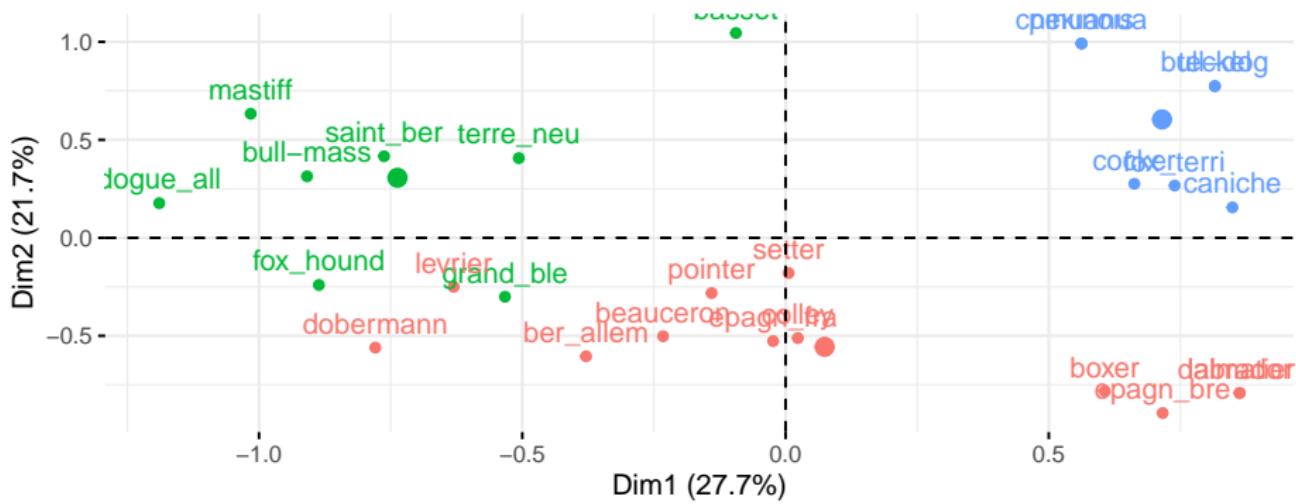
```
Clpam<-pam(daisy(chiens[,-7],metric="gower"),3)  
adjustedRandIndex(Clpm$clustering,chiens[,7])
```

```
[1] 0.2528289
```

```
adjustedRandIndex(Clpm$clustering,clkmeans$cluster)
```

```
[1] 0.3739158
```

PAM



# Plan

1 Méthodes de type K-means

2 Méthodes de type K-médoïdes

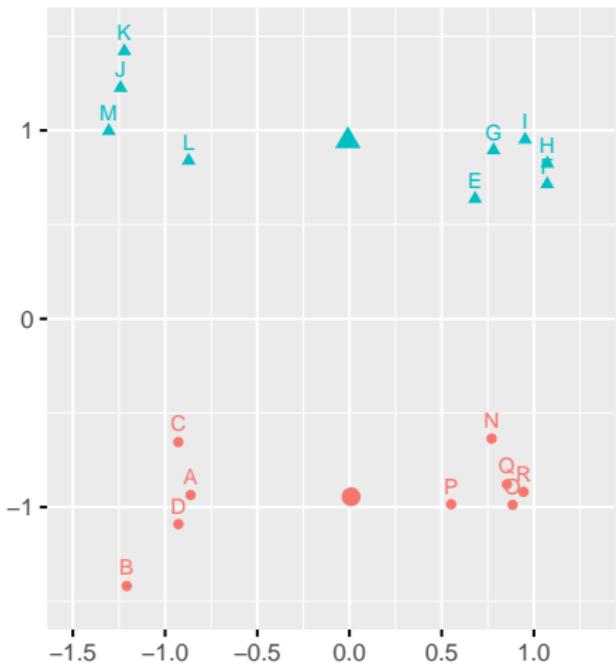
3 Choix des noyaux initiaux

4 Choix du nombre de classes

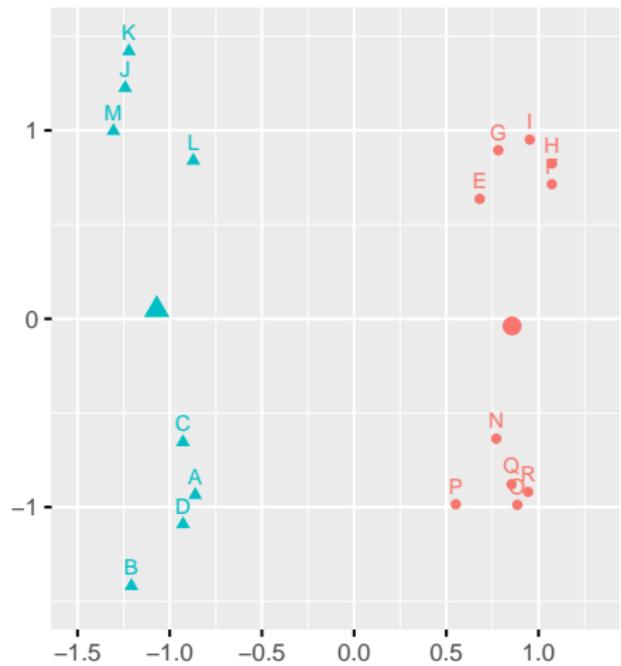
5 DBSCAN

# Initialisation des noyaux

Init. avec D et L



Init. avec E et L



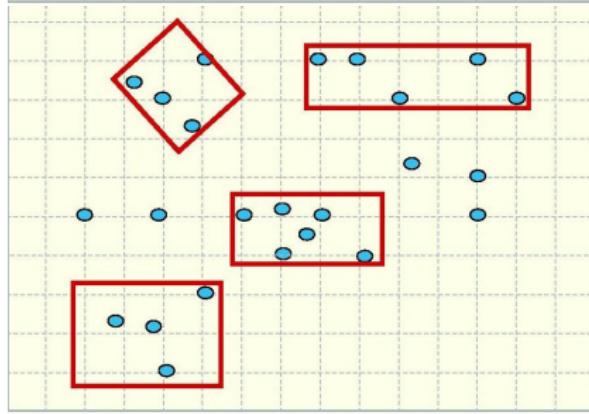
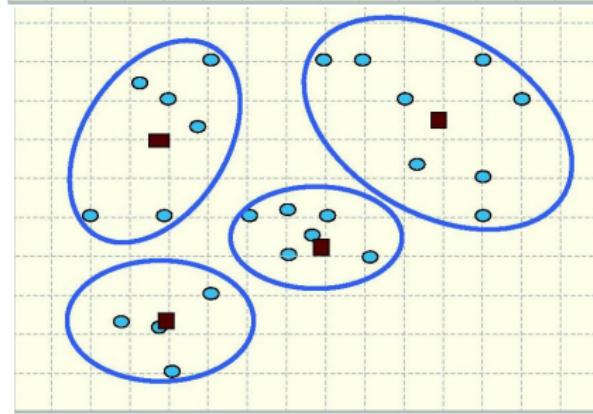
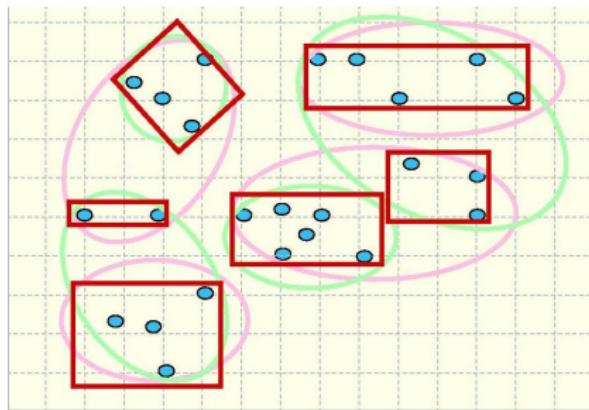
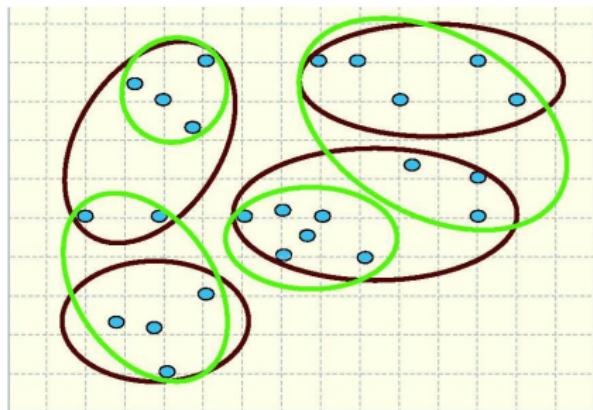
## Choix des noyaux initiaux

- Sélection fondée sur des connaissances complémentaires
- Etude préliminaire des données: histogrammes, ...
- Répétition de la méthode  $N$  fois  $\Rightarrow N$  partitions

Sélection de la partition ayant la plus faible inertie intra-classe

- Avec les formes fortes :
  - ▶ Répétition de la méthode  $N$  fois
  - ▶ Détermination des formes fortes par superposition (ou intersection) des différentes partitions
  - ▶ Initialisation des noyaux = centres de gravité des meilleures formes fortes

# Formes fortes



# Plan

- 1 Méthodes de type K-means
- 2 Méthodes de type K-médoïdes
- 3 Choix des noyaux initiaux
- 4 Choix du nombre de classes
- 5 DBSCAN

## Choix de $K$

- Pour chaque valeur de  $K \in \{2, \dots, K_{\max}\}$ , on obtient une classification et on sélectionne finalement celle où on observe un saut important de l'inertie intra-classe ("coude")
- Choix par d'autres indices basés sur les inerties intra- et inter-
- Choix par maximisation d'un indice  
Ex: silhouette, Gap statistique, ...
- Autres critères de type sélection de modèles (voir modèles de mélange)
- ...

## Critères fondés sur les inerties

- **R-Square :**

$$K \mapsto RSQ(K) = 1 - \frac{I_{intra}(\mathcal{P}_K)}{I_{totale}} = \frac{I_{inter}(\mathcal{P}_K)}{I_{totale}}$$

On retient l'endroit où la courbe  $K \mapsto RSQ(K)$  forme un coude.

- **Semi-Partial R-Square :**

$$K \mapsto SPRSQ(K) = \frac{I_{inter}(\mathcal{P}_K) - I_{inter}(\mathcal{P}_{K-1})}{I_{totale}}$$

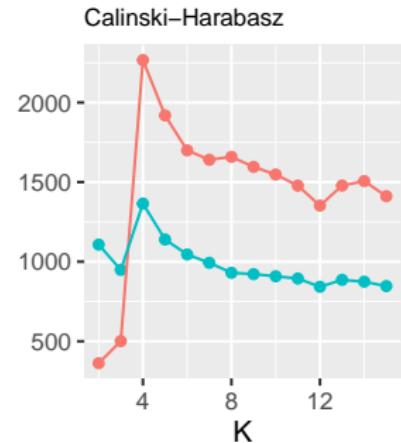
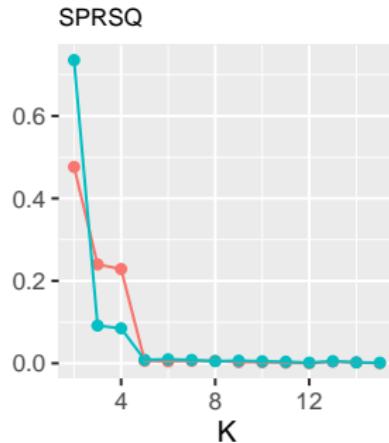
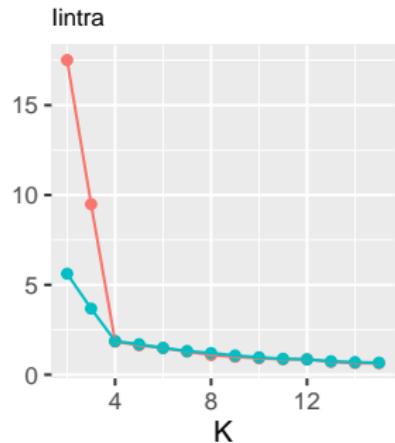
On retient l'endroit où on a la plus forte réduction du SPRSQ.

- **Calinski-Harabasz (CH) :**

$$K \mapsto PseudoF(K) = \frac{I_{inter}(\mathcal{P}_K)/(K-1)}{I_{intra}(\mathcal{P}_K)/(n-K)}$$

On cherche un pic sur cette courbe

# Exemple sur les données simulées



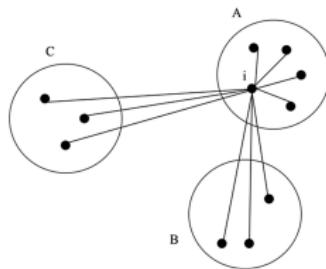
type ● Data1 ● Data2

type ● Data1 ● Data2

type ● Data1 ● Data2

# Silhouette

- Pour toute valeur  $K \in \{1, \dots, K_{\max}\}$  :
  - ▶  $\{\mathcal{C}_1, \dots, \mathcal{C}_K\}$  classification de  $\{1, \dots, n\}$
  - ▶  $\forall i \in \{1, \dots, n\}, \exists! k \in \{1, \dots, K\}; i \in \mathcal{C}_k$ 
    - ★  $a(i) = \frac{1}{|\mathcal{C}_k|-1} \sum_{\substack{\ell \in \mathcal{C}_k \\ \ell \neq i}} d(x_i, x_\ell)$  et  $b(i) = \min_{k' \neq k} \frac{1}{|\mathcal{C}_{k'}|} \sum_{\ell \in \mathcal{C}_{k'}} d(x_i, x_\ell)$
    - ★  $s(i) = \frac{b(i)-a(i)}{\max(b(i), a(i))} \in [-1, 1]$
  - ▶  $S(K) = \frac{1}{n} \sum_{i=1}^n s(i)$
- Le nombre de classes retenu :  $\hat{K} = \underset{1 \leq K \leq K_{\max}}{\operatorname{argmax}} S(K)$



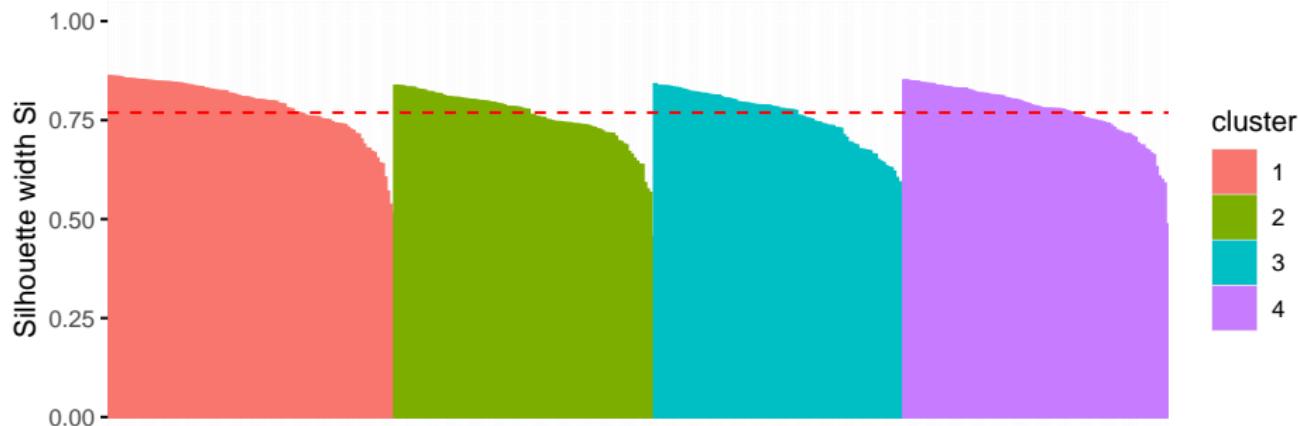
- avec `silhouette(.)` [library(cluster)]
- avec `silhouette_samples(.), silhouette_score(.)` [scikit-learn]

# Exemple - Silhouette

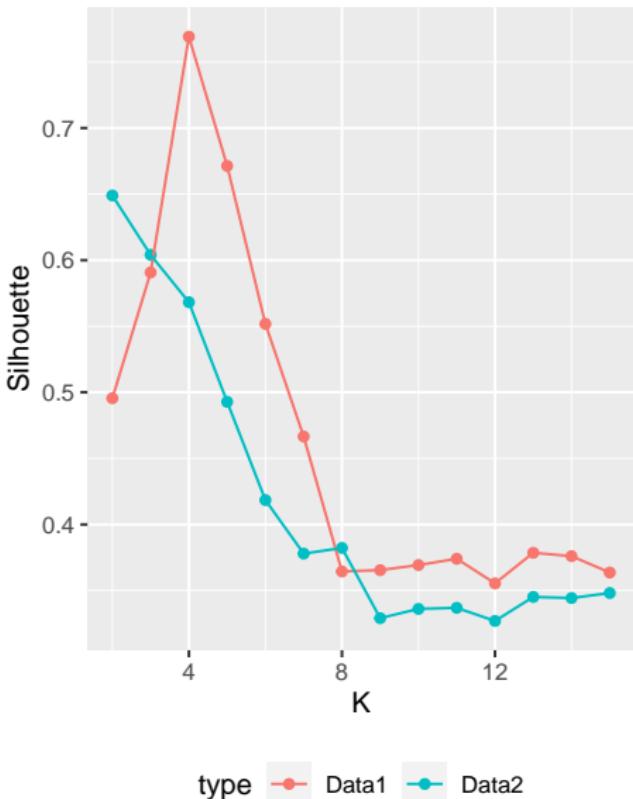
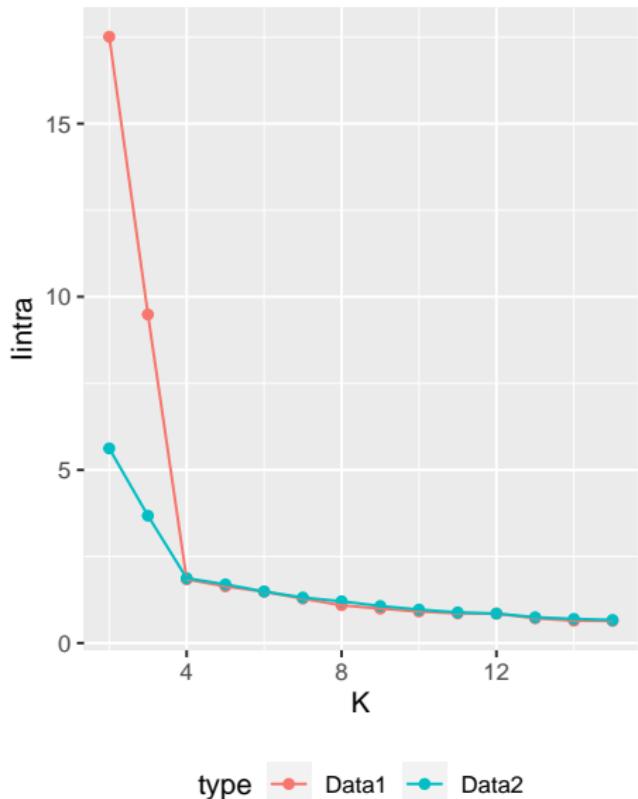
```
Classif<-kmeans(datasimu1[,1:2],4,nstart=10)
aux<-silhouette(Classif$cl, daisy(datasimu1[,1:2]))
fviz_silhouette(aux)+theme(plot.title = element_text(size =9))
```

	cluster	size	ave.sil.width
1	1	108	0.79
2	2	98	0.76
3	3	94	0.76
4	4	100	0.77

Clusters silhouette plot  
Average silhouette width: 0.77



## Exemple - Silhouette

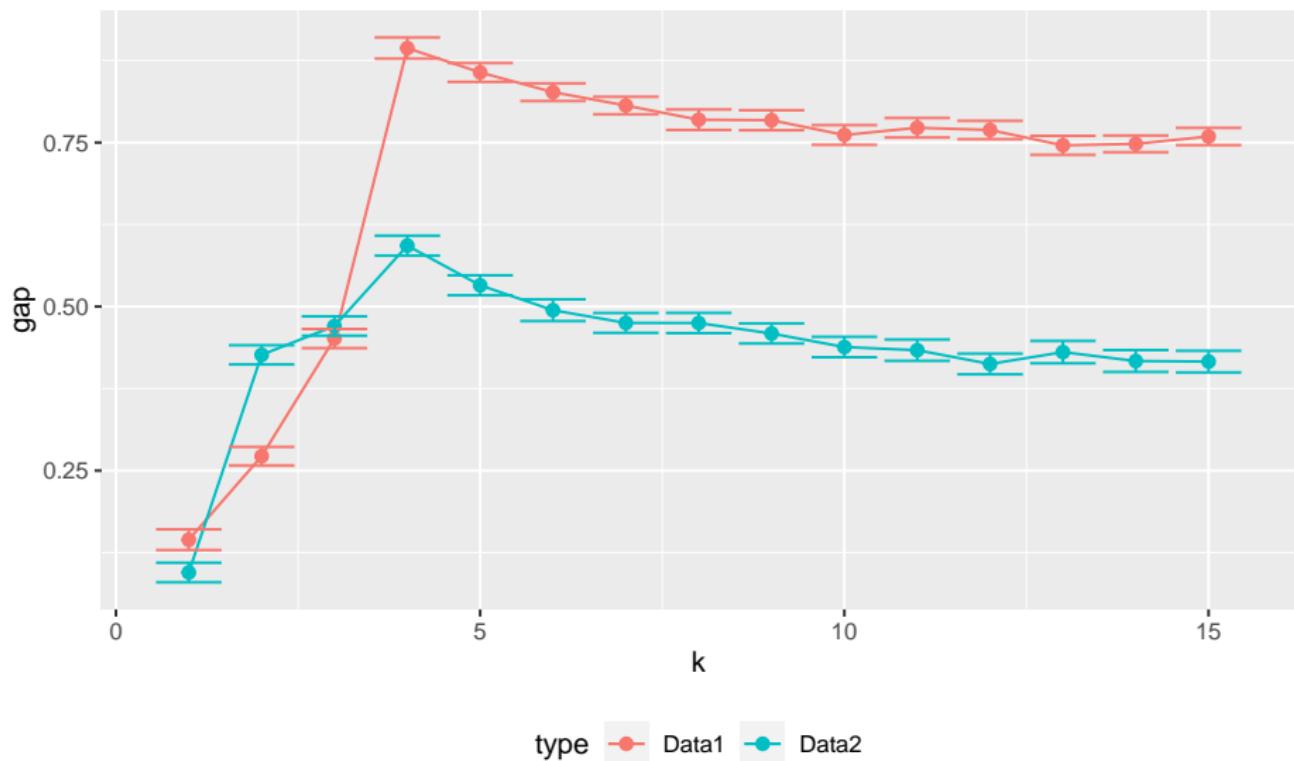


## Gap statistique [6]

- Pour chaque  $K$  dans  $\{1, \dots, K_{\max}\}$ :
  - ▶ Calculer la matrice de dispersion intra-classe  
$$W_K = \frac{1}{n} \sum_{k=1}^K \sum_{i \in C_k} (x_i - m_k)'(x_i - m_k)$$
 et son déterminant  $\det(W_K)$
  - ▶ Construire la courbe des  $\ln(\det(W_K))$  en fonction de  $K$
  - ▶ Comparer cette courbe à celle obtenue à partir des données uniformément réparties
- L'estimation du nombre de classes à retenir est la valeur  $K$  correspondant au plus grand écart entre les deux courbes
- Avec , `clusGap()` [`library(cluster)`]
- Avec , `github milesgranger/gap_statistic`

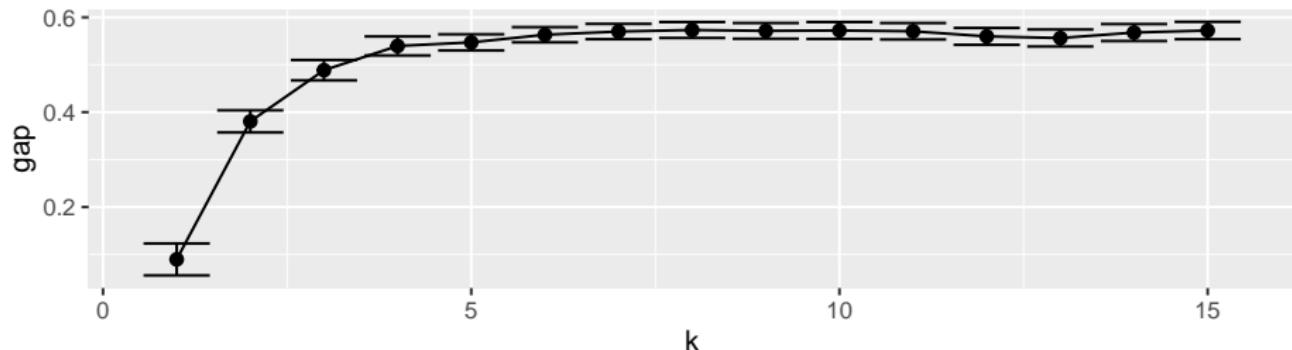
# Exemple - Gap Statistic

Gap Statistic results

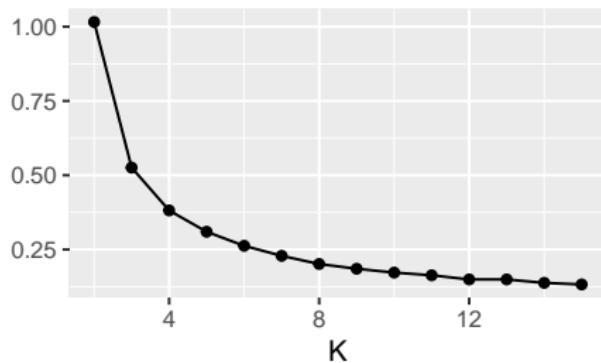


# Exemple des Iris

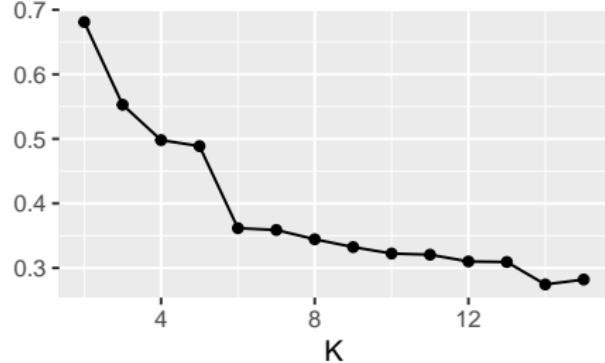
Gap Statistic results



Inertie Intra



Silhouette

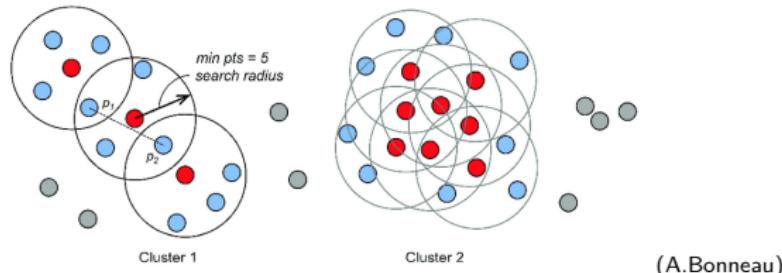


# Plan

- 1 Méthodes de type K-means
- 2 Méthodes de type K-médoïdes
- 3 Choix des noyaux initiaux
- 4 Choix du nombre de classes
- 5 DBSCAN

## Principe [2]

- DBSCAN = Density-Based Spatial Clustering of Applications
- 2 paramètres :
  - ▶ la distance de voisinage  $\varepsilon$
  - ▶ le nombre minimum de points minPts
- Idée : pour un indiv. donné, vérifier si son  $\varepsilon$ -voisinage contient au moins minPts points. Si oui, ce point fait partie d'une classe. On parcourt ensuite le  $\varepsilon$ -voisinage de proche en proche pour trouver l'ensemble des points de la classe.
- Usuellement, la distance euclidienne est utilisée



# Animation

[https://aaronscotthq.com/2020-05-28-scott\\_dbSCAN/](https://aaronscotthq.com/2020-05-28-scott_dbSCAN/)

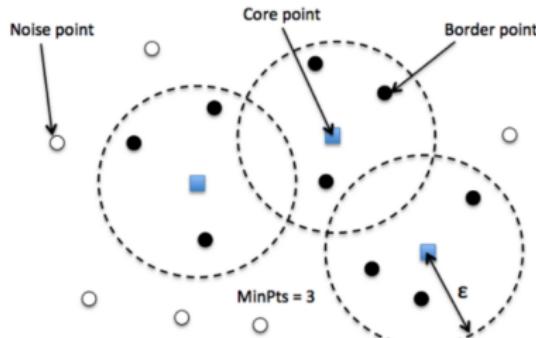
## Catégories d'individus

A la fin de l'algorithme, les individus appartiennent à l'une de ces 3 catégories :

- Point central : son voisinage est dense

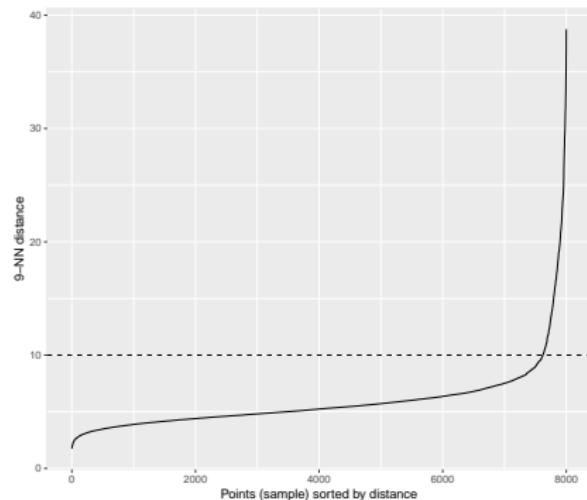
$$\#\{\ell \in \{1, \dots, n\}; d(x_i, x_\ell) < \varepsilon\} \geq \text{minPts}$$

- Point frontière : appartient au  $\varepsilon$ -voisinage d'un point central mais n'est pas un point central
- Point aberrant : n'est ni un point central, ni un point frontière (donc non classé)



## Choix des paramètres

- Il est préconisé de choisir au moins  $\text{minPts} \geq p + 1$ . Pour des données avec du bruit,  $\text{minPts} = 2p$  peut être utilisé mais il peut être nécessaire de choisir des valeurs plus grandes pour des données de grande dimension. Dans d'autres sources,  $\ln(n)$  est préconisé.
- Ensuite, pour le choix de  $\varepsilon$ , on trace le graphe de distance kNN:
  - Pour chaque point, on calcule sa distance moyenne à ses  $k = \text{minPts} - 1$  plus proches voisins
  - On trace les distances obtenues par ordre croissant
  - On choisit  $\varepsilon$  comme la valeur de la distance kNN où on observe un "coude"



# Quelques commandes

- Avec 

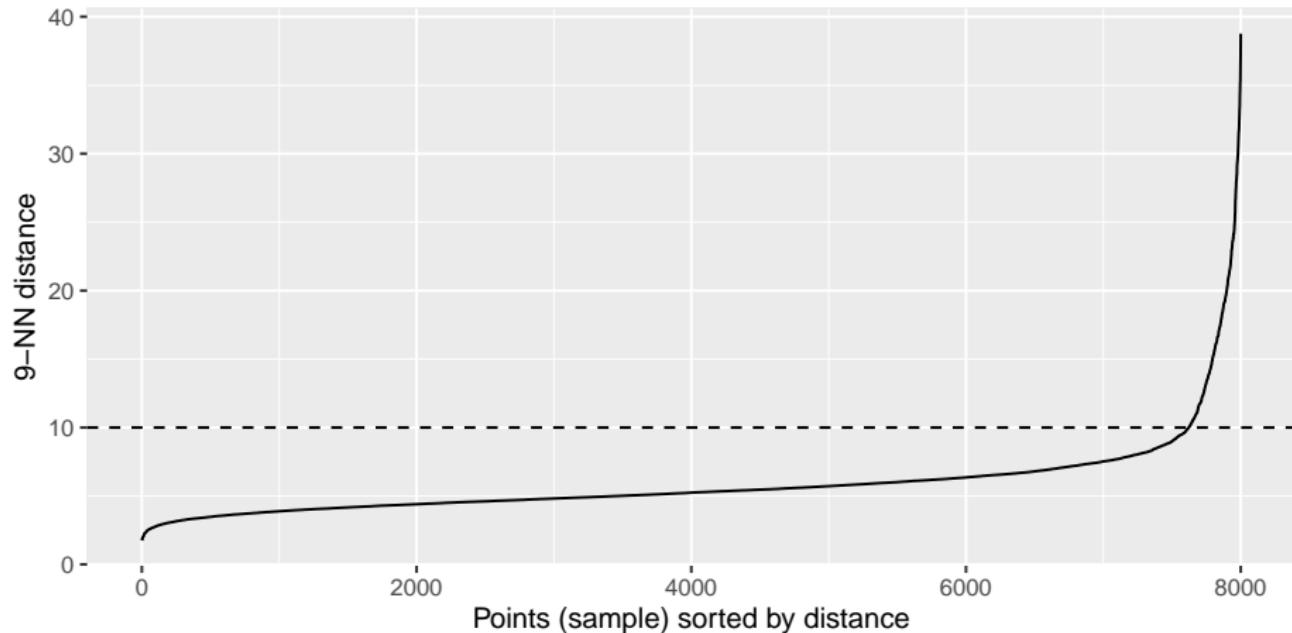
- ▶ Fonction `dbSCAN()` de `library(fpc)`
  - ▶ Dans la `library(dbSCAN)`, les fonctions `dbSCAN()` et `kNNdistplot()`

- Avec 

- ▶ Fonction `sklearn.cluster.DBSCAN()` de scikit-learn

# Exemple avec R

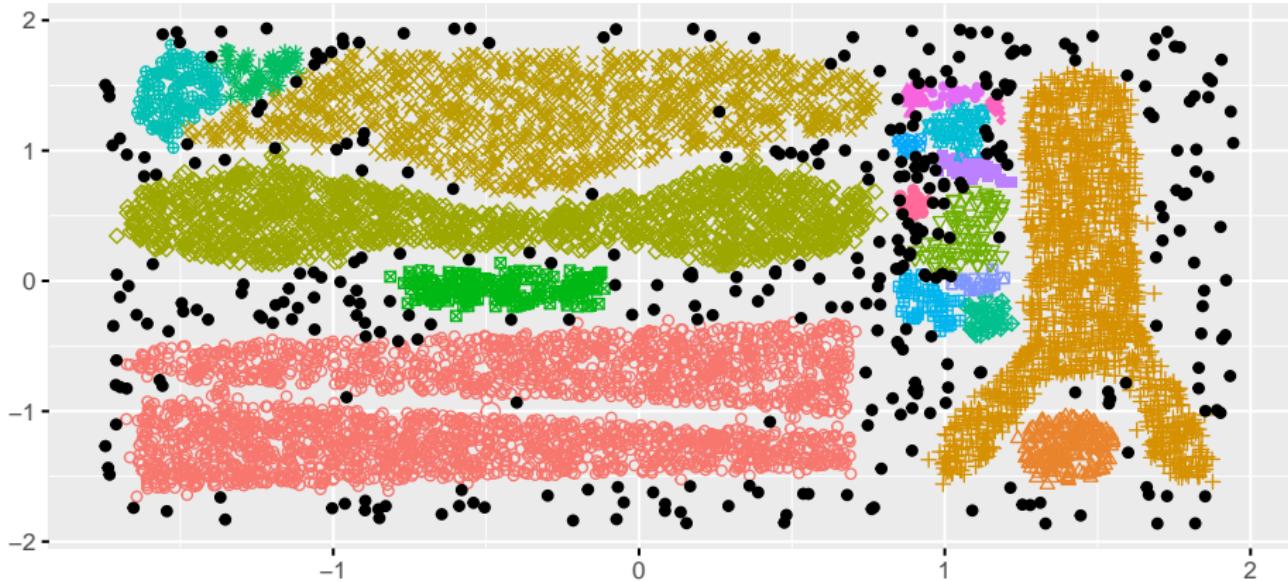
```
library(seriation)
data(Chameleon)
DS8<-chameleon_ds8
library(dbscan)
dbscan::kNNdistplot(DS8, k =round(log(nrow(DS8))))
abline(h = 10, lty = 2)
```



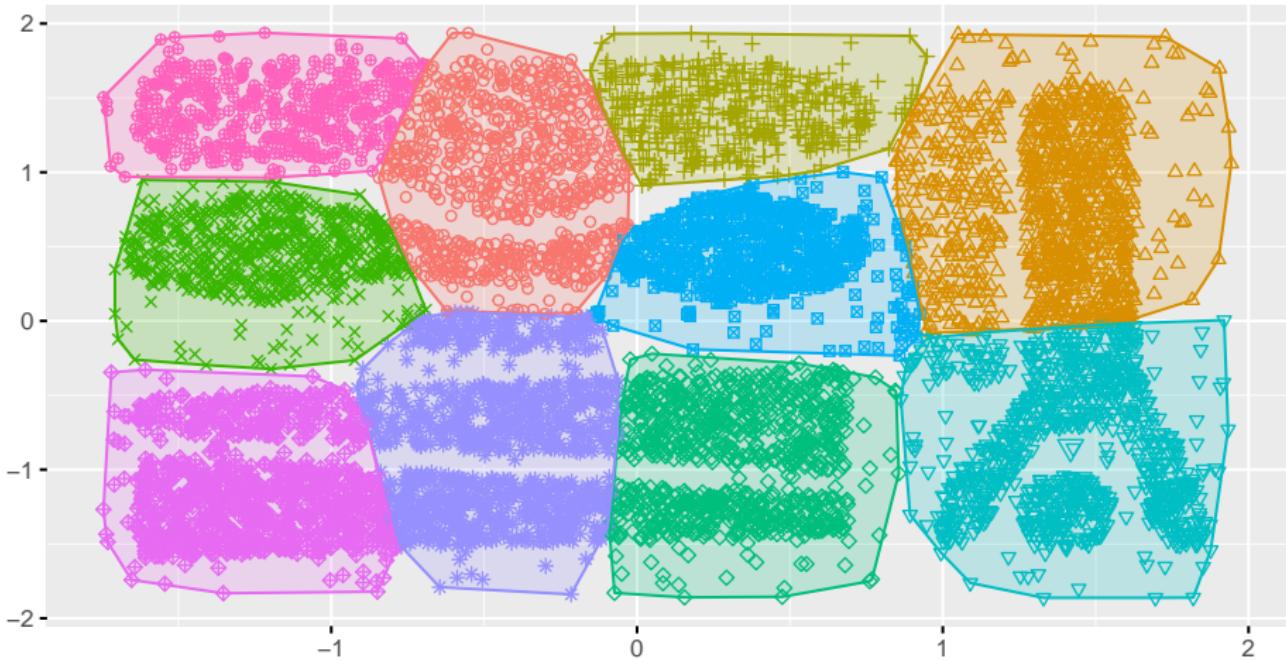
# Exemple avec R

```
res.db <- dbscan::dbscan(DS8, 10, round(log(nrow(DS8))))  
table(res.db$cluster)
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
357	2588	182	1391	1268	1576	79	177	55	31	120	31	42	9	16	30
16	17	18	19												
15	13	7	13												



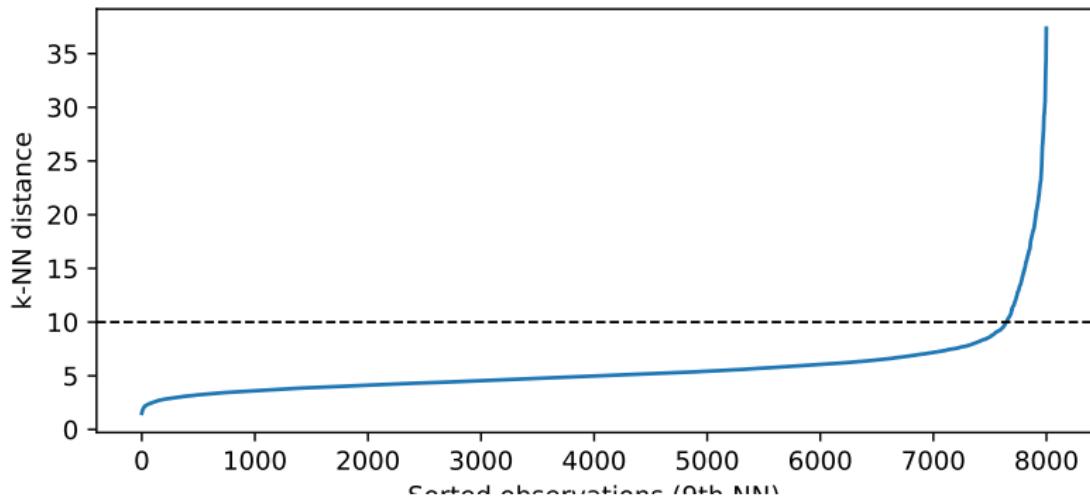
## Avec les Kmeans



# Exemple avec



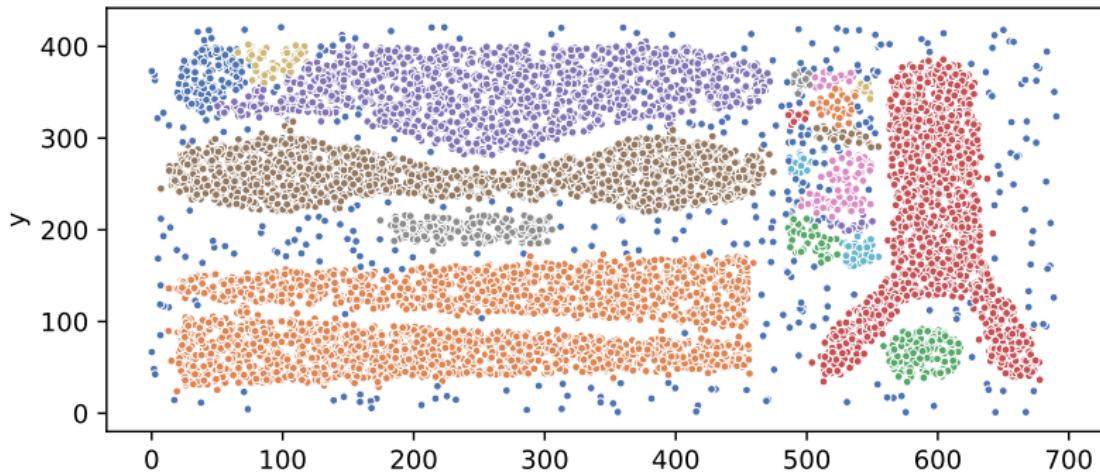
```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.neighbors import NearestNeighbors
pyDS8=r.DS8
neighbors = NearestNeighbors(n_neighbors=9)
neighbors_fit = neighbors.fit(pyDS8)
distances, indices = neighbors_fit.kneighbors(pyDS8)
distancesmean = np.sort(distances.sum(axis=1)/8, axis=0)
```



# Exemple avec



```
from sklearn.cluster import DBSCAN
import seaborn as sns
import matplotlib.pyplot as plt
pyDS8=r.DS8
clusters = DBSCAN(eps=10, min_samples=9).fit(pyDS8)
fig = sns.scatterplot(data=pyDS8, x="x", y="y", hue=clusters.labels_, legend="brief", palette="deep", s=10)
plt.legend([],[], frameon=False)
plt.show()
```



## References |

- [1] Edwin Diday. "Une nouvelle méthode en classification automatique et reconnaissance des formes la méthode des nuées dynamiques". In: *Revue de statistique appliquée* 19.2 (1971), pp. 19–33.
- [2] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise.". In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.
- [3] Edward W Forgy. "Cluster analysis of multivariate data: efficiency versus interpretability of classifications". In: *biometrics* 21 (1965), pp. 768–769.
- [4] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, 1990.
- [5] James McQueen. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1967*. 1967, pp. 281–297.

## References II

- [6] Robert Tibshirani, Guenther Walther, and Trevor Hastie. "Estimating the number of clusters in a data set via the gap statistic". In: *Journal of the Royal Statistical Society. Series B. Statistical Methodology* 63.2 (2001), pp. 411–423.