

# Statistique en grande dimension - partie non supervisée

---

Laurent Rouvière

30 novembre 2022

## Définition

Action de **répartir en classes**, en catégories, des choses, des objets, ayant des caractères communs afin notamment d'en faciliter l'étude.

## Nombreuses applications

- Astronomie : classification d'étoiles
- Médecine : diagnostic de maladies à partir d'observation cliniques
- Géographie : délimitation de zones homogènes
- Marketing : détermination de segments de marchés (groupes de consommateurs ayant les mêmes habitudes)
- Réseaux sociaux : extraction de communautés
- ...

## MOOC de François Husson

- disponible ici [https://husson.github.io/MOOC\\_AnaDo/classif.html](https://husson.github.io/MOOC_AnaDo/classif.html)
- vidéos - Quiz - Supports

- Beaucoup de groupes avec peu d'individus à l'intérieur (réduire  $n$ )

## Objectifs divers

- Beaucoup de groupes avec peu d'individus à l'intérieur (réduire  $n$ )
- Peu de groupes avec beaucoup d'individus à l'intérieur (extraire des profils que l'on interprète par la suite).

# Objectifs divers

- Beaucoup de groupes avec peu d'individus à l'intérieur (réduire  $n$ )
- Peu de groupes avec beaucoup d'individus à l'intérieur (extraire des profils que l'on interprète par la suite).

## Conséquence

Les algorithmes seront proches mais pas calibrés de la même façon.

- $n$  observations  $x_1, \dots, x_n$  à valeurs dans  $\mathbb{R}^d$ .

## Le problème

Trouver une **partition** de  $\{x_1, \dots, x_n\}$  : on cherche donc  $\mathcal{C}_1, \dots, \mathcal{C}_K$  tels que

$$\bigcup_{k=1}^K \mathcal{C}_k = \{x_1, \dots, x_n\} \quad \text{et} \quad \mathcal{C}_k \cap \mathcal{C}_{k'} = \emptyset \quad \text{si} \quad k \neq k'.$$

Chaque élément de la partition  $\mathcal{C}_k$  est appelé **cluster**.

- $n$  observations  $x_1, \dots, x_n$  à valeurs dans  $\mathbb{R}^d$ .

## Le problème

Trouver une **partition** de  $\{x_1, \dots, x_n\}$  : on cherche donc  $\mathcal{C}_1, \dots, \mathcal{C}_K$  tels que

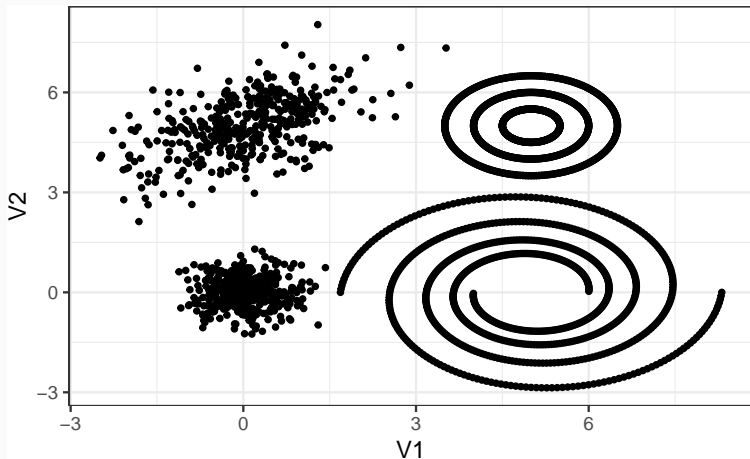
$$\bigcup_{k=1}^K \mathcal{C}_k = \{x_1, \dots, x_n\} \quad \text{et} \quad \mathcal{C}_k \cap \mathcal{C}_{k'} = \emptyset \quad \text{si} \quad k \neq k'.$$

Chaque élément de la partition  $\mathcal{C}_k$  est appelé **cluster**.

- Notions de ressemblance, **similarité**, **hiérarchie**.
- Choix du **nombre de classes**  $K$ .



## Un exemple jouet



## 4 types d'algorithmes

1. Méthodes de **partitionnement** ( $k$ -means)

## 4 types d'algorithmes

1. Méthodes de **partitionnement** ( $k$ -means)
2. Méthodes **hiérarchiques** (CAH)

## 4 types d'algorithmes

1. Méthodes de **partitionnement** ( $k$ -means)
2. Méthodes **hiérarchiques** (CAH)
3. Algorithmes basés sur les **densités** (DBSCAN)

## 4 types d'algorithmes

1. Méthodes de **partitionnement** ( $k$ -means)
2. Méthodes **hiérarchiques** (CAH)
3. Algorithmes basés sur les **densités** (DBSCAN)
4. Approches basées sur les **graphes** (clustering spectral)

## Partitionnement : les $k$ -means

---

## Le critère des $k$ -means

- **Idée** : Définir les **clusters**  $\mathcal{C}_k$  à partir de représentants  $c_k$ .
- Nombre de groupes  $K$  **fixé**.

# Le critère des $k$ -means

- **Idée** : Définir les **clusters**  $\mathcal{C}_k$  à partir de représentants  $c_k$ .
- Nombre de groupes  $K$  **fixé**.

## Le critère des $k$ -means

On cherche la partition  $\mathcal{C}$  et les représentants  $c$  qui **minimise le critère**

$$g(\mathcal{C}, c) = \sum_{k=1}^K \sum_{i \in \mathcal{C}_k} \|x_i - c_k\|^2.$$



# Le critère des $k$ -means

- **Idée** : Définir les **clusters**  $\mathcal{C}_k$  à partir de représentants  $c_k$ .
- Nombre de groupes  $K$  **fixé**.

## Le critère des $k$ -means

On cherche la partition  $\mathcal{C}$  et les représentants  $c$  qui **minimise le critère**

$$g(\mathcal{C}, c) = \sum_{k=1}^K \sum_{i \in \mathcal{C}_k} \|x_i - c_k\|^2.$$

- **Impossible** de trouver  $(\mathcal{C}^*, c^*)$  qui minimise  $g(\mathcal{C}, c)$ .

# Équivalences

1. Quand on fixe une partition  $\mathcal{C}^*$ , les meilleurs représentants [2] sont les moyennes  $\hat{c} = (\bar{x}_{c_1}, \dots, \bar{x}_{c_K})$

$$\forall c \quad g(\mathcal{C}^*, c) \geq g(\mathcal{C}^*, \hat{c})$$

# Équivalences

1. Quand on fixe une partition  $\mathcal{C}^*$ , les meilleurs représentants [2] sont les moyennes  $\hat{c} = (\bar{x}_{\mathcal{C}_1}, \dots, \bar{x}_{\mathcal{C}_K})$

$$\forall c \quad g(\mathcal{C}^*, c) \geq g(\mathcal{C}^*, \hat{c})$$

2. Quand on fixe des représentants  $c$ , la meilleure partition  $\hat{\mathcal{C}} = \{\hat{\mathcal{C}}_1, \dots, \hat{\mathcal{C}}_K\}$  est celle de la distance minimale (partition de Voronoi) définie par

$$\hat{\mathcal{C}}_k = \{i \in \{1, \dots, n\} \text{ tels que } \|x_i - c_k\|^2 = \min_j \|x_i - c_j\|^2\}$$

Elle réalise le minimum à représentants fixés:

$$\forall \mathcal{C} \quad g(\mathcal{C}, c) \geq g(\hat{\mathcal{C}}, c)$$

## Idée

Construire les centres et la partition de manière **réursive**.

## Idée

Construire les centres et la partition de manière **récursive**.

## Comment ?

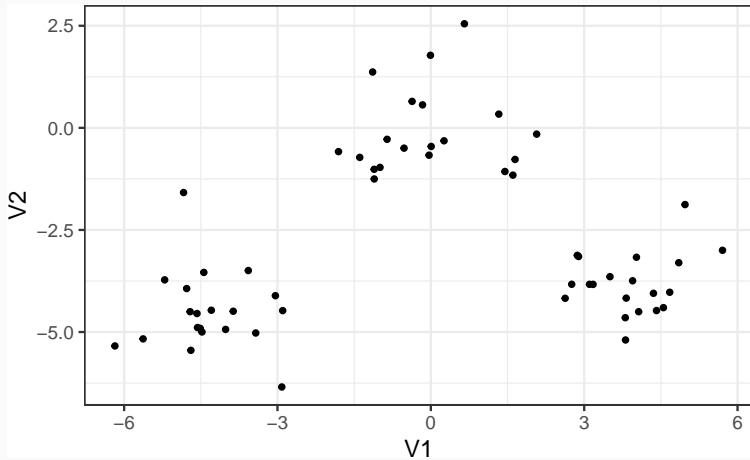
En utilisant un algorithme :

- LLOYD [7]
- FORGY [4]
- MACQUEEN [9]
- HARTIGAN et WONG [5]

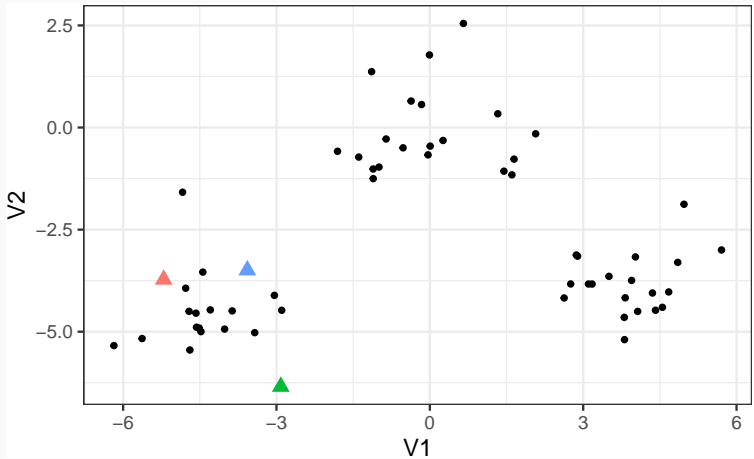
on pourra consulter <https://towardsdatascience.com/three-versions-of-k-means-cf939b65f4ea>.

1. **Initialisation** : choix de  $k$  points au hasard comme centroïde.
2. **Affectation** de chaque observation au centroïde le plus proche.
3. **Mise à jour** des centroïdes.

## Exemple

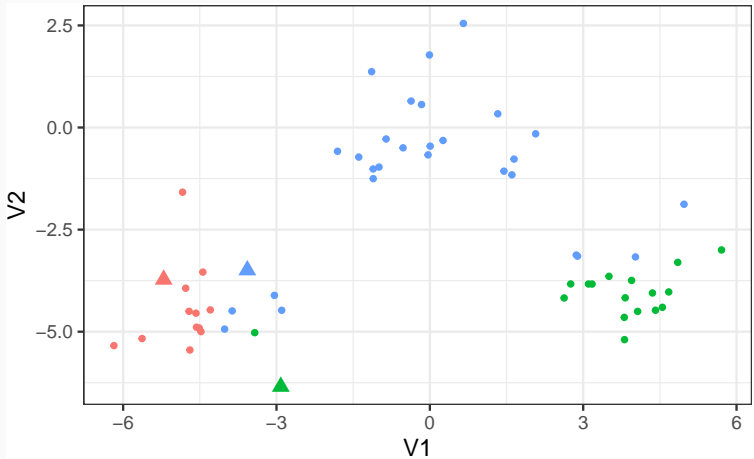


# Exemple

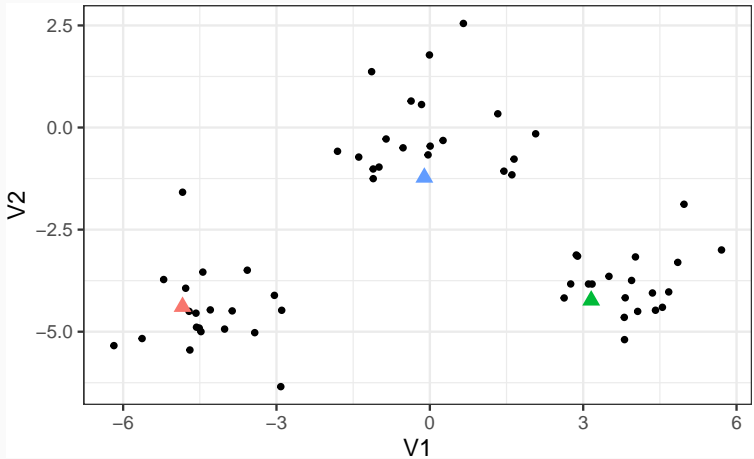




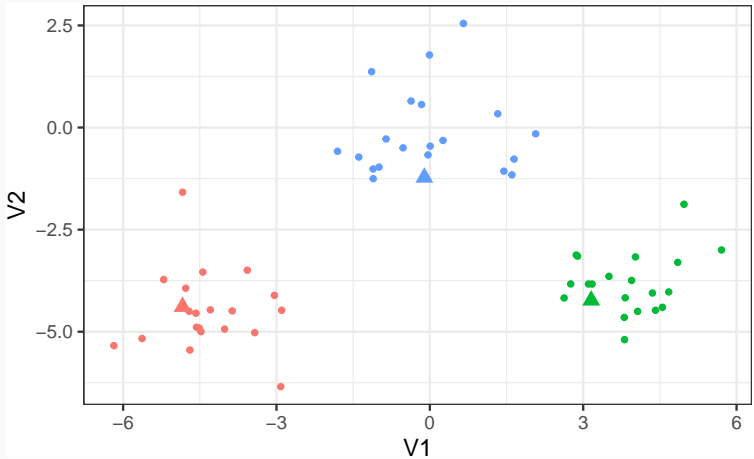
# Exemple



# Exemple



# Exemple



1. **Initialisation** : choix de  $K$  points au hasard comme centroïde
2. Pour  $i = 1, \dots, n$ 
  - a. Pour  $k = 1, \dots, K$ 
    - Affecter  $x_i$  à  $c_k$
    - calculer  $\alpha_k = \sum_{i=1}^n \|x_i - c_{k,i}\|^2$
  - b. Affecter  $x_i$  au centroïde qui minimise  $\alpha_k$
  - c. Mettre à jour les centroïdes
3. Répéter 2 jusqu'à convergence

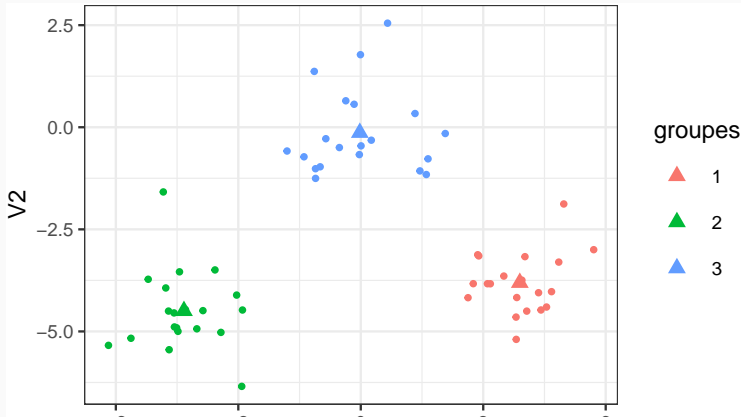
1. **Initialisation** : choix de  $K$  points au hasard comme centroïde
2. Pour  $i = 1, \dots, n$ 
  - a. Pour  $k = 1, \dots, K$ 
    - Affecter  $x_i$  à  $c_k$
    - calculer  $\alpha_k = \sum_{i=1}^n \|x_i - c_{k,i}\|^2$
  - b. Affecter  $x_i$  au centroïde qui minimise  $\alpha_k$
  - c. Mettre à jour les centroïdes
3. Répéter 2 jusqu'à convergence

### Remarque

Généralement recommandé avec **plusieurs initialisations**.

# Le coin R

```
> res <- kmeans(tbl[,2:3],centers = 3,  
+             algorithm = "Hartigan-Wong",nstart=100)  
> centres <- res$centers |> as_tibble() |> mutate(groupe=as.factor(1:3))  
> tbl |> mutate(groupe=as.factor(res$cluster)) |> ggplot()+  
+   aes(x=V1,y=V2,color=groupe)+geom_point() +  
+   geom_point(data=centres,shape=17,size=2)
```



## Avantages

- Facile à mettre en œuvre
- Faible complexité  $O(n)$

## Avantages

- Facile à mettre en œuvre
- Faible complexité  $O(n)$

## Inconvénients

- Clusters “sphériques”.
- Choix de  $k$ .
- Choix de la distance (grande dimension ?).



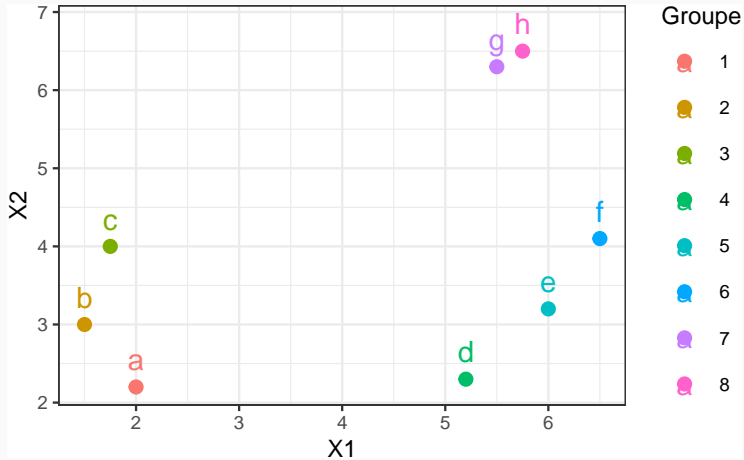
# Méthodes hiérarchiques

---

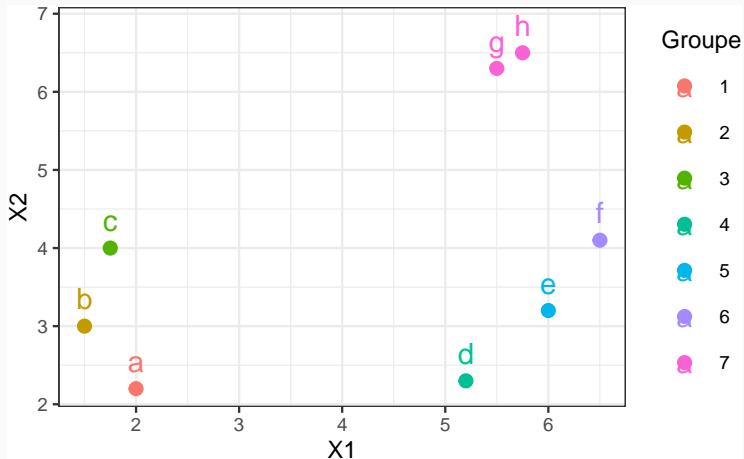
## Objectif

Créer une **suite de partitions emboîtées** en partant de la partition la plus **fine** ( $n$  classes) jusqu'à obtenir **une seule classe**.

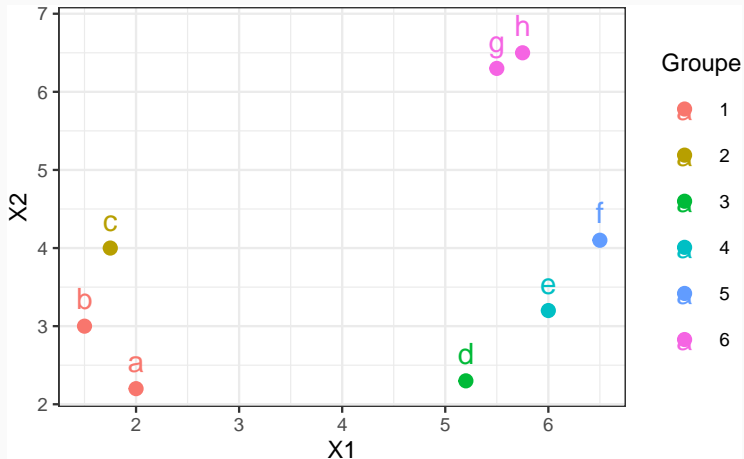
# Le processus hiérarchique



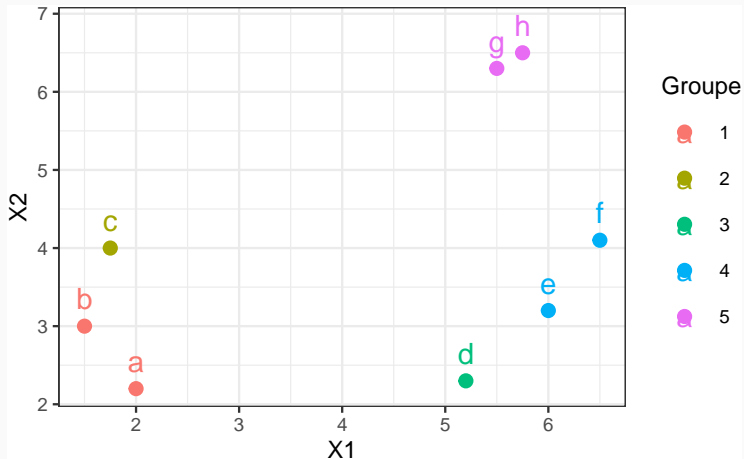
# Le processus hiérarchique



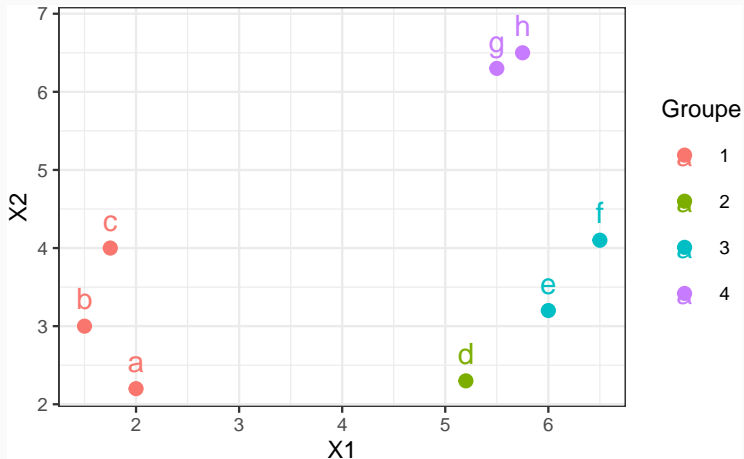
# Le processus hiérarchique



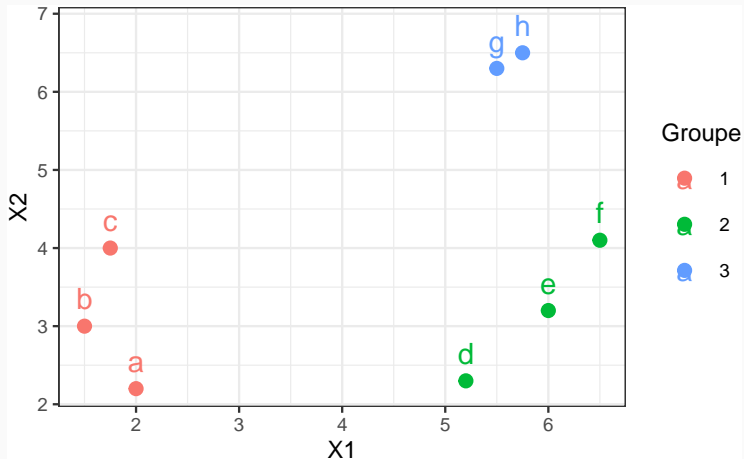
# Le processus hiérarchique



# Le processus hiérarchique

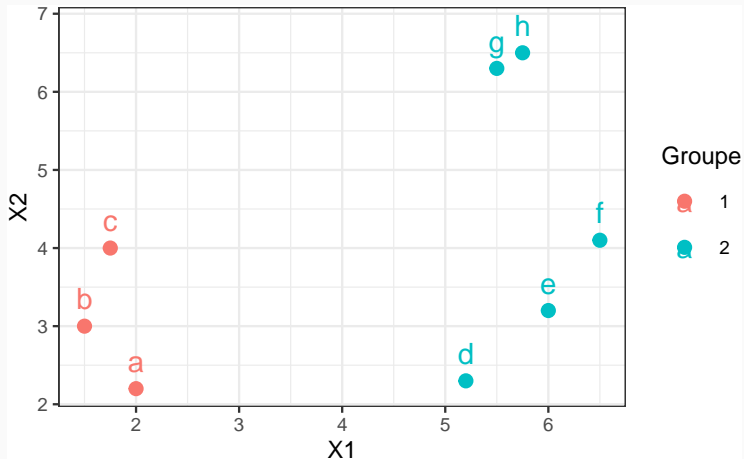


# Le processus hiérarchique

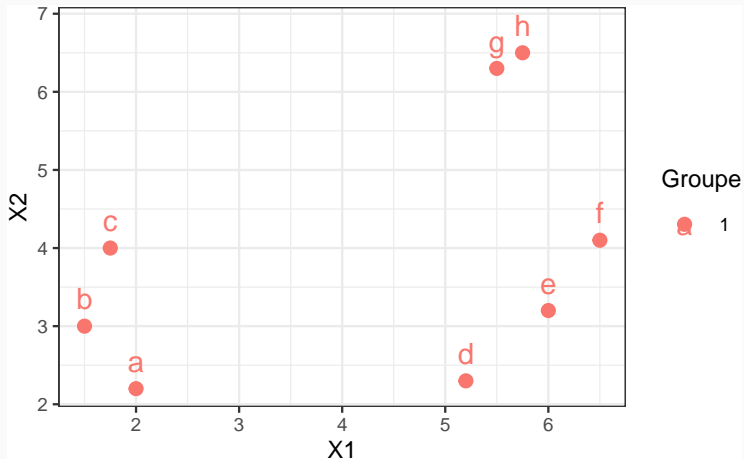




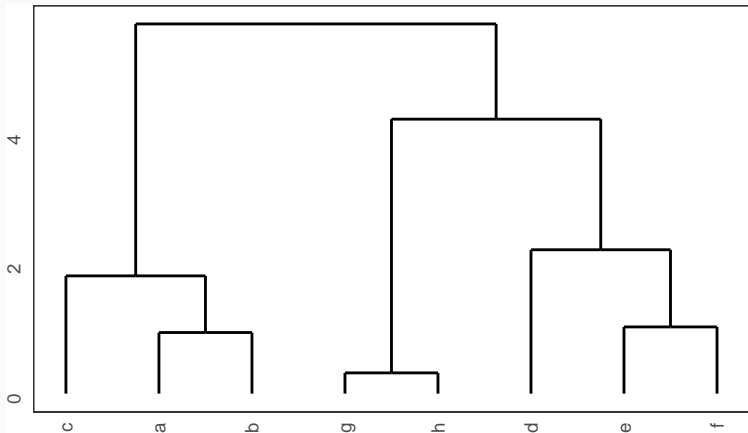
# Le processus hiérarchique



# Le processus hiérarchique



# Visualisation



# Méthodes hiérarchiques

---

## La Classification Ascendante Hiérarchique

## Algorithme

**Entrées :** données, distance (entre individus et clusters d'individus)

1. Calculer une matrice de **distances entre individus**.

## Algorithme

**Entrées :** données, distance (entre individus et clusters d'individus)

1. Calculer une matrice de **distances entre individus**.
2. Chaque observation forme 1 singleton.

## Algorithme

**Entrées :** données, distance (entre individus et clusters d'individus)

1. Calculer une matrice de **distances entre individus**.
2. Chaque observation forme 1 singleton.
3. **Agréger** les deux objets les plus proches.

## Algorithme

**Entrées :** données, distance (entre individus et clusters d'individus)

1. Calculer une matrice de **distances entre individus**.
2. Chaque observation forme 1 singleton.
3. **Agréger** les deux objets les plus proches.
4. Mettre à **jour la matrice de distances**.



## Algorithme

**Entrées :** données, distance (entre individus et clusters d'individus)

1. Calculer une matrice de **distances entre individus**.
2. Chaque observation forme 1 singleton.
3. **Agréger** les deux objets les plus proches.
4. Mettre à **jour la matrice de distances**.
5. Itérer jusqu'à obtenir **un seul groupe**.

## Algorithme

**Entrées :** données, distance (entre individus et clusters d'individus)

1. Calculer une matrice de **distances entre individus**.
2. Chaque observation forme 1 singleton.
3. **Agréger** les deux objets les plus proches.
4. Mettre à **jour la matrice de distances**.
5. Itérer jusqu'à obtenir **un seul groupe**.

**Sorties :** une suite de partitions emboîtées.

# De quoi a t-on besoin ?

Pas grand chose...

# De quoi a t-on besoin ?

Pas grand chose... il suffit de savoir calculer des distances et/ou indicateurs de similarités entre

- des observations. On notera  $d$  une telle distance
- des groupes d'observations, *i.e.* entre clusters. On notera  $\Delta$  une telle distance.

# Méthodes hiérarchiques

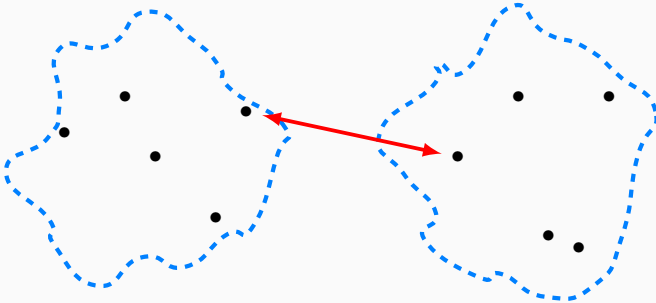
---

## Mesures de dissemblances

# Saut minimum

- Également appelé **minimu linkage** ou **single linkage**.

$$\Delta(\mathcal{C}_i, \mathcal{C}_j) = \min_{x_i \in \mathcal{C}_i, x_j \in \mathcal{C}_j} d(x_i, x_j)$$



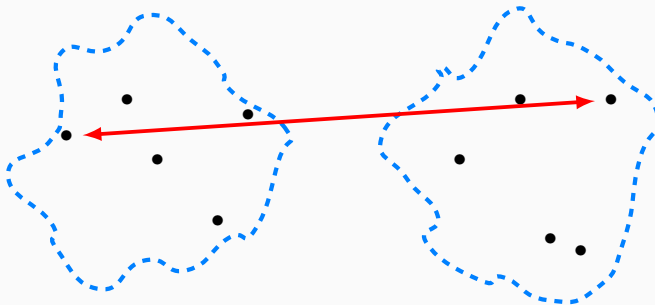
## Commentaires

- Groupes généralement "étirés".
- "Le voisin de mon voisin est mon voisin".

# Saut maximum

- Également appelé **complete linkage**.

$$\Delta(\mathcal{C}_i, \mathcal{C}_j) = \max_{x_i \in \mathcal{C}_i, x_j \in \mathcal{C}_j} d(x_i, x_j)$$



## Commentaires

Groupes généralement "compacts".

## Saut moyen (average linkage)

Moyenne de toutes les distances entre deux objets des deux groupes :

$$\Delta(\mathcal{C}_i, \mathcal{C}_j) = \frac{1}{|\mathcal{C}_i||\mathcal{C}_j|} \sum_{x_i \in \mathcal{C}_i, x_j \in \mathcal{C}_j} d(x_i, x_j)$$

### Commentaires

Intermédiaires entre le min et le max...



## Idée

Se baser sur l'inertie :

$$\begin{aligned}\mathcal{J}_{\text{tot}} &= \frac{1}{n} \sum_{i=1}^n d^2(x_i - \bar{x}) \\ &= \frac{1}{n} \sum_{k=1}^K \sum_{i \in \mathcal{C}_k} d^2(x_i, \bar{x}_{\mathcal{C}_k}) + \frac{1}{n} \sum_{k=1}^K n_k d^2(\bar{x}_{\mathcal{C}_k}, \bar{x}) \\ &= \mathcal{J}_{\text{intra}} + \mathcal{J}_{\text{inter}}\end{aligned}$$

en minimisant  $\mathcal{J}_{\text{intra}}$  et/ou maximisant  $\mathcal{J}_{\text{inter}}$ .

## Cas extrêmes

- $K = n \implies \mathcal{J}_{\text{intra}} = 0$  et  $\mathcal{J}_{\text{inter}} = \mathcal{J}_{\text{tot}}$ .
- $K = 1 \implies \mathcal{J}_{\text{intra}} = \mathcal{J}_{\text{tot}}$  et  $\mathcal{J}_{\text{inter}} = 0$ .

## Lien de Ward

Assembler les clusters de manière à **minimiser** la perte de  $\mathcal{J}_{\text{inter}}$

## Lien de Ward

Assembler les clusters de manière à **minimiser** la perte de  $\mathcal{J}_{\text{inter}}$   $\iff$   
minimiser le **lien de Ward** :

$$\Delta(\mathcal{C}_i, \mathcal{C}_j) = \frac{|\mathcal{C}_i| |\mathcal{C}_j|}{|\mathcal{C}_i| + |\mathcal{C}_j|} d^2(\bar{x}_{\mathcal{C}_i}, \bar{x}_{\mathcal{C}_j})$$

## Lien de Ward

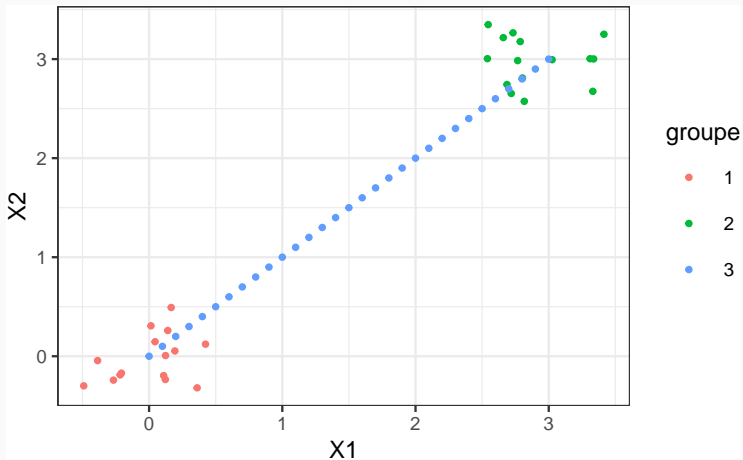
Assembler les clusters de manière à **minimiser** la perte de  $\mathcal{J}_{\text{inter}}$   $\iff$   
minimiser le **lien de Ward** :

$$\Delta(\mathcal{C}_i, \mathcal{C}_j) = \frac{|\mathcal{C}_i| |\mathcal{C}_j|}{|\mathcal{C}_i| + |\mathcal{C}_j|} d^2(\bar{x}_{\mathcal{C}_i}, \bar{x}_{\mathcal{C}_j})$$

## Commentaires

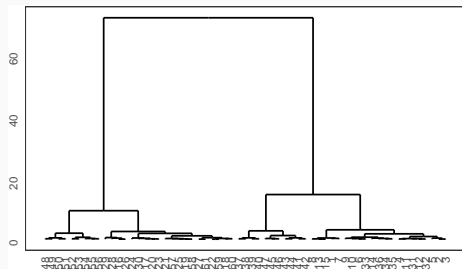
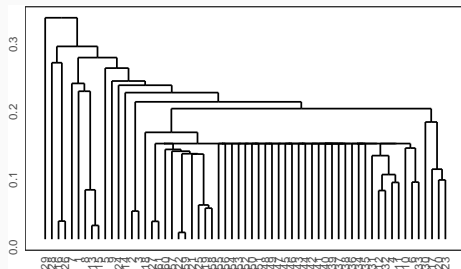
- Bien adapté à la **distance euclidienne**
- liens forts avec **l'ACP**.

# Exemple

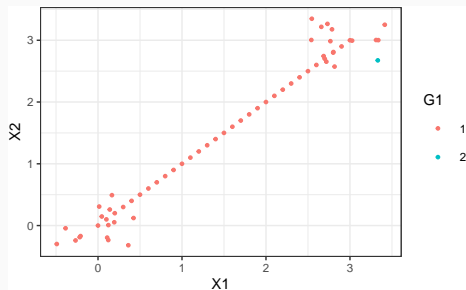


```
> class1 <- hclust(D,method = "single")
> ggdendrogram(class1)

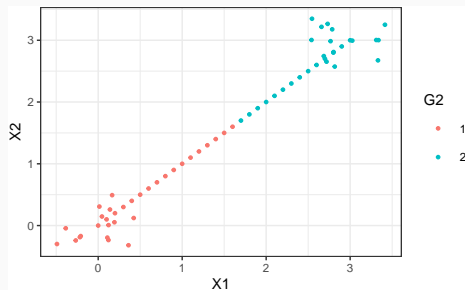
> class2 <- hclust(D,method = "ward")
> ggdendrogram(class2)
```



```
> G1 <- cutree(class1,k = 2)
```



```
> G2 <- cutree(class2,k =2)
```



## Choix du nombre de classes

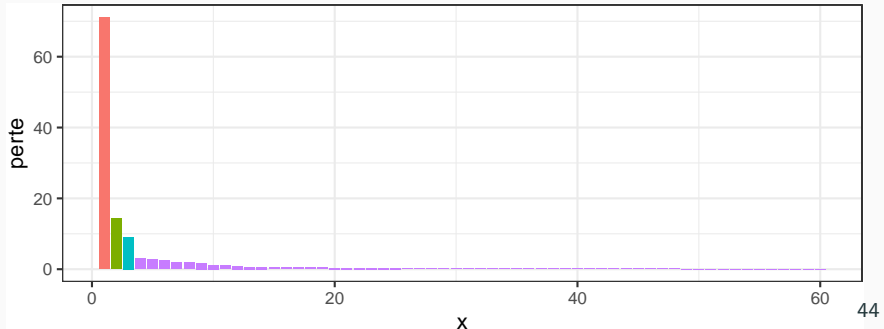
- Toujours difficile...



# Choix du nombre de classes

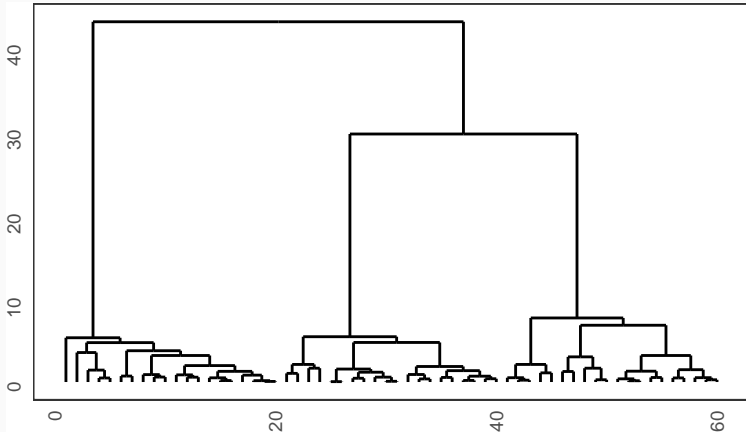
- Toujours **difficile**... On se base généralement sur la **perte d'inertie inter** obtenue en agrégeant les clusters :

```
> tibble(perte=rev(class2$height),x=1:60,  
+        col=as.factor(c(1:3,rep(4,57)))) |>  
+   ggplot()+aes(x=x,y=perte,fill=col)+  
+   geom_bar(stat = "identity",show.legend = FALSE)
```



# Le coin R : hclust

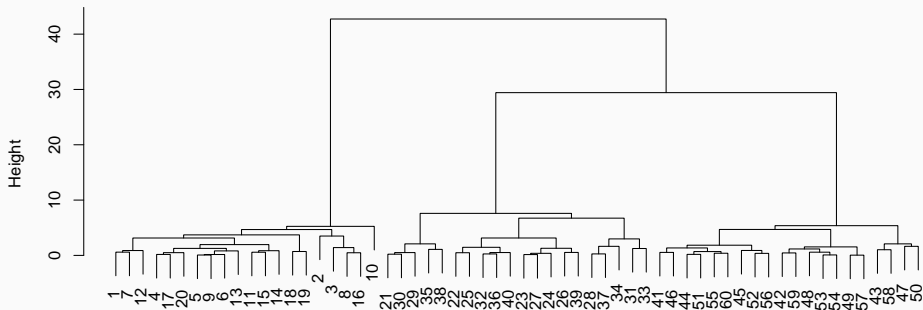
```
> DD <- dist(tbl)
> classif <- hclust(DD,method = "ward.D2")
> library(ggdendro)
> ggdendrogram(classif,labels = FALSE)
```



# Le coin R : agnes de cluster

```
> library(cluster)
> classif1 <- agnes(DD,method = "ward")
> plot(classif1,which.plots=2)
```

Dendrogram of `agnes(x = DD, method = "ward")`



DD

Agglomerative Coefficient = 0.98

## Avantages

- Pas besoin de connaître le nombre de classes a priori
- Visualisation dendrogramme

## Avantages

- Pas besoin de connaître le nombre de classes a priori
- Visualisation dendrogramme

## Inconvénients

- Coupure du dendrogramme pas toujours simple
- Complexité algorithmique élevée lorsque  $n$  est grand  $\implies O(n^3)$ .

# Méthodes hiérarchiques

---

## Compléments

- La CAH est souvent trop coûteuse en temps de calcul lorsque  $n$  est grand.

## $n$ grand $\implies$ Classification mixte

- La CAH est souvent trop coûteuse en temps de calcul lorsque  $n$  est grand.

### Classification Mixte

1. Faire un  $k$ -means sur les données avec  $k$  grand (par exemple  $k = 1000$ )
2. Lancer la CAH sur les centroïdes obtenus dans le  $k$ -means (en prenant en considération les effectifs des clusters)



## $n$ grand $\implies$ Classification mixte

- La CAH est souvent trop couteuse en temps de calcul lorsque  $n$  est grand.

### Classification Mixte

1. Faire un  $k$ -means sur les données avec  $k$  grand (par exemple  $k = 1000$ )
  2. Lancer la CAH sur les centroïdes obtenus dans le  $k$ -means (en prenant en considération les effectifs des clusters)
- Sur **R** on peut utiliser la fonction HCPC du package **FactoMineR**.

# Exemple

```
> dim(tbl)
[1] 70000      2
> aa <- dist(tbl)
Error: vecteurs de mémoire épuisés (limite atteinte ?)
```

```
> library(FactoMineR)
> classif <- HCPC(tbl,kk=100,nb.clust = 5,
+               description = FALSE,graph = FALSE)
> summary(classif$data.clust)
```

	V1	V2	clust
Min.	:-3.8960	Min. :-2.9089	1: 9968
1st Qu.:	0.7978	1st Qu.: 0.2049	2: 8851
Median :	4.4625	Median : 4.0371	3:11027
Mean :	3.5691	Mean : 2.8530	4: 9986
3rd Qu.:	5.4406	3rd Qu.: 5.1447	5:30168
Max.	: 8.3466	Max. : 8.6905	

- CAH et  $k$ -means reposent sur des distances entre individus.

- CAH et  $k$ -means reposent sur des distances entre individus.
- Les distances standards ne sont pas forcément pertinentes en grande dimension.

- CAH et  $k$ -means reposent sur des distances entre individus.
- Les distances standards ne sont pas forcément pertinentes en grande dimension.

### Réduction de dimension

- Souvent pertinent d'effectuer une analyse factorielle au préalable(ACP-ACM...) pour réduire la dimension.
- On fait ensuite le  $k$ -means et/ou la CAH sur les premiers axes de l'analyse factorielle.
- Sur R : fonction HCPC de **FactoMineR**.

# Fastcluster et flashClust

Packages qui proposent d'autres algorithmes pour le calcul de la CAH.

```
> tbl1 <- tbl |> slice(sample(70000,10000)); D <- dist(tbl1)
> system.time(aa <- stats::hclust(D,method = "ward.D2"))
  user  system elapsed
 2.555   0.163   2.732
> system.time(bb <- fastcluster::hclust(D,method="ward.D2"))
  user  system elapsed
 1.087   0.097   1.188
> system.time(cc <- flashClust::flashClust(D,method="ward"))
  user  system elapsed
 3.653   0.169   3.856
```

# Méthodes fondées sur la densité - DBSCAN

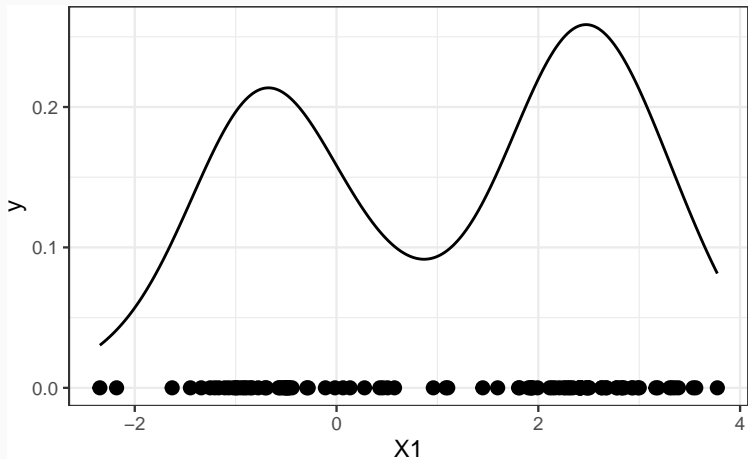
---

- Le principe est de déterminer les **classes** d'une partition à partir des **zones de forte densité**.
- Les zones de **faible densité** sont utilisées pour **délimiter les classes**.

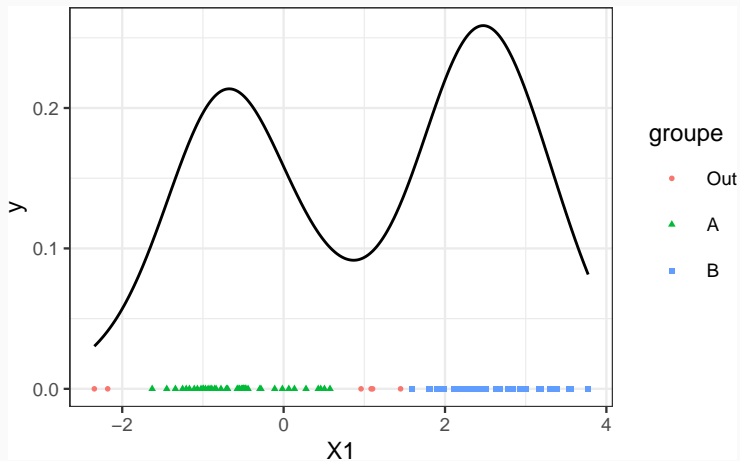


- Le principe est de déterminer les **classes** d'une partition à partir des **zones de forte densité**.
- Les zones de **faible densité** sont utilisées pour **délimiter les classes**.
- Les éléments sont **regroupés de proche en proche** et les éléments éloignés des zones de forte densité sont ignorés et considérés comme des **outliers**.
- ESTER et al. [3] : **DBSCAN** (Density-based spatial clustering of applications with noise)

# L'idée



# L'idée



- Soit  $\varepsilon > 0$  et  $\text{MinPts} \leq n$  fixés.
- On note  $B_\varepsilon(y)$  le voisinage centré sur  $y$  et de rayon  $\varepsilon$  et  $|B_\varepsilon(y)|$  le nombre de points dans  $B_\varepsilon(y)$ .

# Noyaux et points de bordure

- Soit  $\varepsilon > 0$  et  $\text{MinPts} \leq n$  fixés.
- On note  $B_\varepsilon(y)$  le voisinage centré sur  $y$  et de rayon  $\varepsilon$  et  $|B_\varepsilon(y)|$  le nombre de points dans  $B_\varepsilon(y)$ .

## Définition

- Si  $|B_\varepsilon(y)| \geq \text{MinPts}$  alors  $y$  est un **noyau** et est dans une zone de forte densité.
- Si  $|B_\varepsilon(y)| < \text{MinPts}$  alors  $y$  est un **point bordure** et n'est pas dans une zone de forte densité.

## Définition

- $x$  est **directement accessible** depuis  $y$  si  $x \in B_\varepsilon(y)$  et  $y$  est un noyau.

## Définition

- $x$  est **directement accessible** depuis  $y$  si  $x \in B_\varepsilon(y)$  et  $y$  est un noyau.
- $x$  est **accessible depuis**  $y$  si il existe une chaîne de points  $p_1 = y, p_2, \dots, p_k = x$  telle que  $\forall i, p_{i+1}$  est directement accessible depuis  $p_i$ .

## Définition

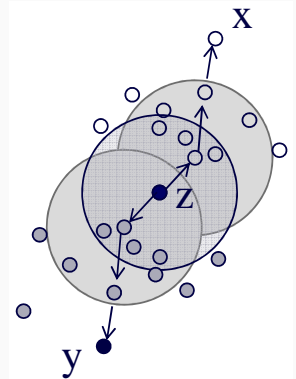
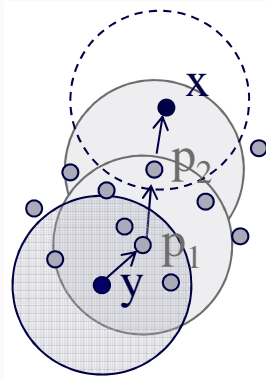
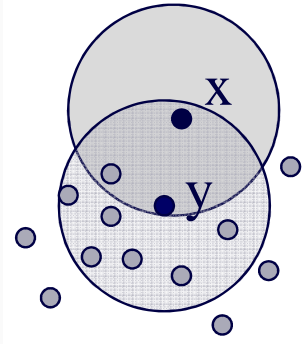
- $x$  est **directement accessible** depuis  $y$  si  $x \in B_\varepsilon(y)$  et  $y$  est un noyau.
- $x$  est **accessible depuis**  $y$  si il existe une chaîne de points  $p_1 = y, p_2, \dots, p_k = x$  telle que  $\forall i, p_{i+1}$  est directement accessible depuis  $p_i$ .

## Définition

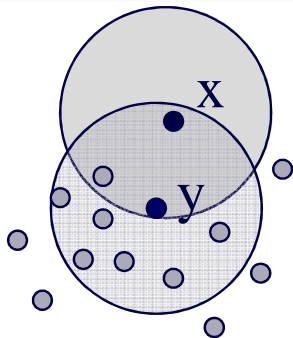
- Deux éléments  $x$  et  $y$  sont **connectés** s'ils sont tous les deux accessibles depuis un même élément  $z$  (l'éléments  $z$  peut éventuellement être  $x$  ou  $y$ ).
- Un cluster est **constitué** par un ensemble d'éléments connectés.



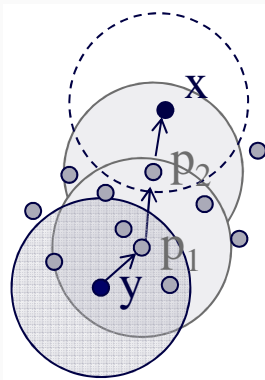
## Exemple : MinPts = 4



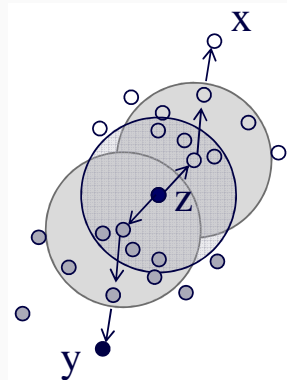
## Exemple : MinPts = 4



$x$  bordure,  $y$  noyau



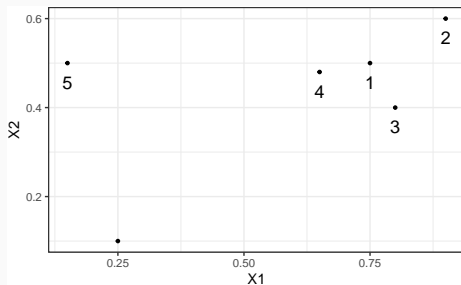
$x$  accessible depuis  $y$   
 $y$  non accessible depuis  
 $x$



$x$  et  $y$  connectés

# Un exemple

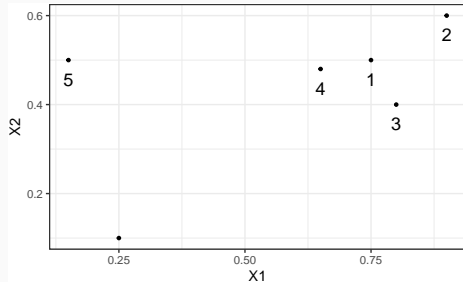
```
> round(dist(tbl[,1:2]),2)
      1      2      3      4      5
2 0.18
3 0.11 0.22
4 0.10 0.28 0.17
5 0.60 0.76 0.66 0.50
6 0.64 0.82 0.63 0.55 0.41
```



# Un exemple

```
> round(dist(tbl[,1:2]),2)
      1      2      3      4      5
2 0.18
3 0.11 0.22
4 0.10 0.28 0.17
5 0.60 0.76 0.66 0.50
6 0.64 0.82 0.63 0.55 0.41
```

```
> is.corepoint(tbl[,1:2],eps=0.25,minPts = 4)
[1] TRUE FALSE TRUE FALSE FALSE FALSE
```



# Résultats

```
> (db <- dbscan(tbl[,1:2],eps=0.25,minPts = 3))
DBSCAN clustering for 6 objects.
Parameters: eps = 0.25, minPts = 3
Using euclidean distances and borderpoints = TRUE
The clustering contains 1 cluster(s) and 2 noise points.

0 1
2 4

Available fields: cluster, eps, minPts, dist, borderPoints
> db$cluster
[1] 1 1 1 1 0 0
```

## Résultats

1 cluster de 4 points et 2 outliers.

- 2 paramètres sont à calibrer  $\epsilon$  et minPts ; Leur choix est crucial...

- 2 paramètres sont à calibrer  $\varepsilon$  et minPts ; Leur choix est crucial...
- minPts  $\nearrow \Rightarrow$  moins de noyaux  $\Rightarrow$  moins de clusters

- 2 paramètres sont à calibrer  $\varepsilon$  et minPts ; Leur choix est crucial...
- minPts  $\nearrow \Rightarrow$  moins de noyaux  $\Rightarrow$  moins de clusters
- $\varepsilon \searrow \Rightarrow$  moins de noyaux  $\Rightarrow$  plus d'outliers



# Choix des paramètres

- 2 paramètres sont à calibrer  $\varepsilon$  et minPts ; Leur choix est crucial...
- $\text{minPts} \nearrow \Rightarrow$  moins de noyaux  $\Rightarrow$  moins de clusters
- $\varepsilon \searrow \Rightarrow$  moins de noyaux  $\Rightarrow$  plus d'outliers

## Conséquence

Il faut calibrer ces paramètres

- On devra bien entendu faire plusieurs essais et analyser les résultats.

- On devra bien entendu faire **plusieurs** essais et **analyser les résultats**.

## Heuristique

- Choisir minPts de l'ordre de la **dimension des données** + 1

- On devra bien entendu faire **plusieurs** essais et **analyser les résultats**.

## Heuristique

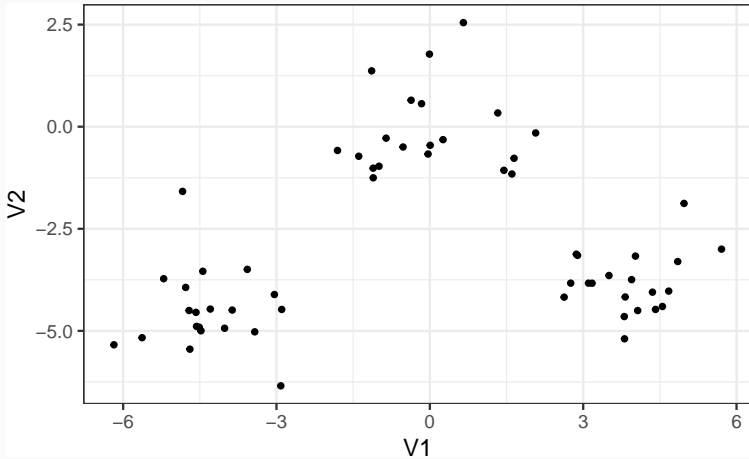
- Choisir minPts de l'ordre de la **dimension des données** + 1
- Tracer le graphe des `kNNdisplot` en utilisant  $k = \text{minPts} - 1$ .

- On devra bien entendu faire **plusieurs** essais et **analyser les résultats**.

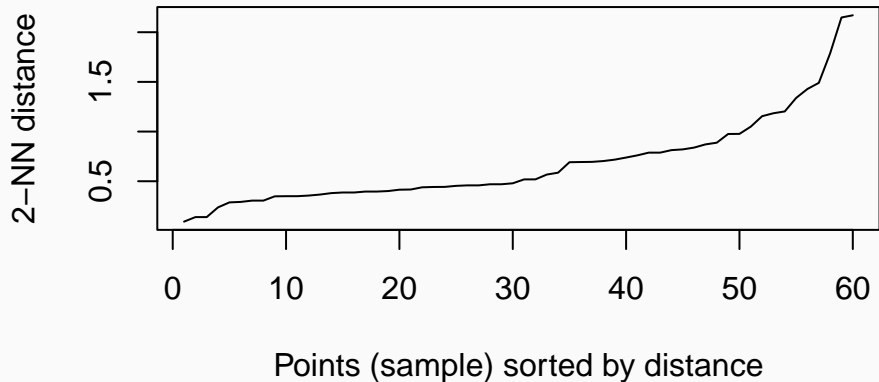
## Heuristique

- Choisir minPts de l'ordre de la **dimension des données** + 1
- Tracer le graphe des kNNdisplot en utilisant  $k = \text{minPts} - 1$ .
- Utiliser un **critère du coude** pour choisir  $\varepsilon$ .

## Example 1



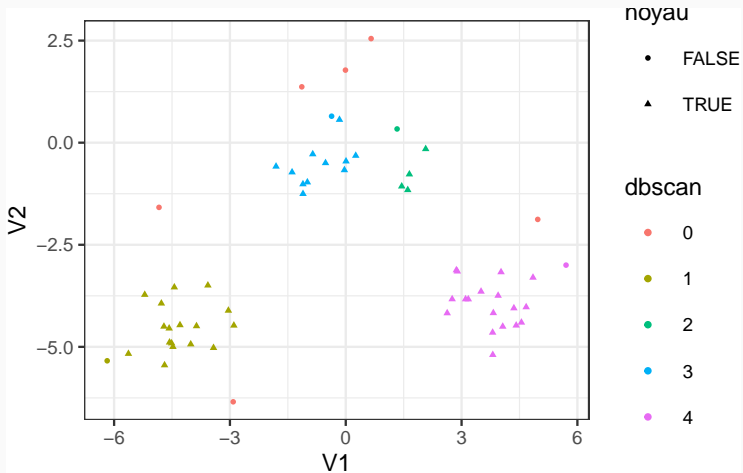
```
> kNNdistplot(tbl[,2:3],k=2)
```



⇒ on pourra prendre  $\varepsilon$  autour de 1

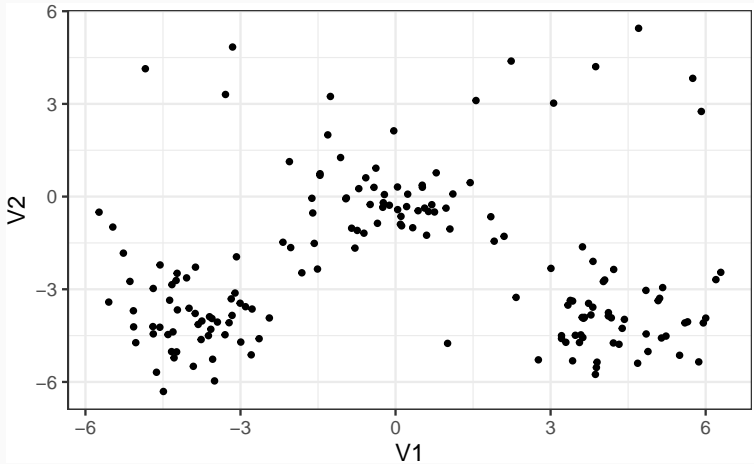
# Résultats

```
> db <- dbscan(tbl[,2:3],eps=1,minPts = 3)
> noyau <- is.corepoint(tbl[,2:3],eps=1,minPts = 3)
> tbl_db <- tbl |> mutate(dbscan=as.factor(db$cluster),noyau=noyau)
> ggplot(tbl_db)+aes(x=V1,y=V2,color=dbscan,shape=noyau)+geom_point()
```

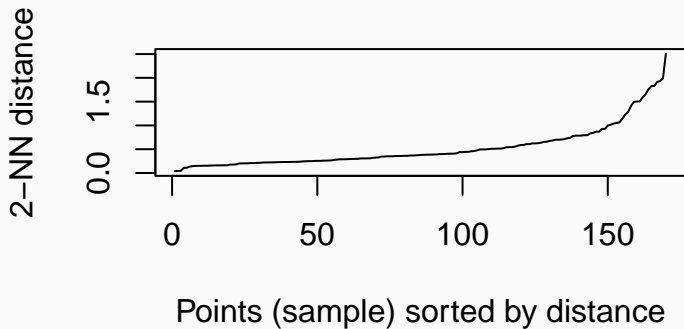




## Exemple 2



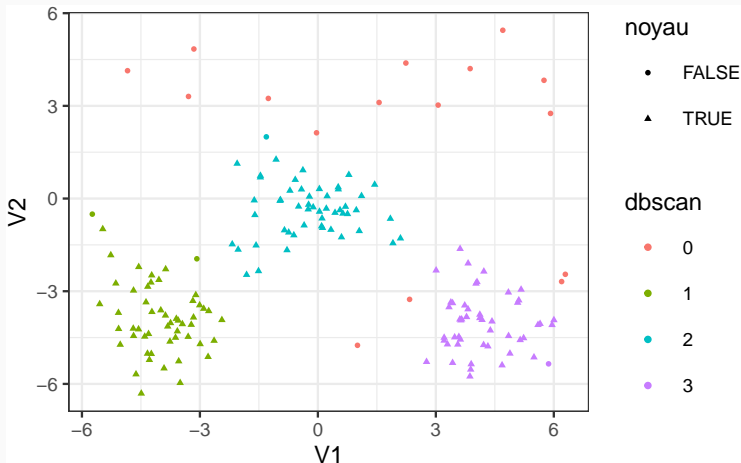
```
> kNNdistplot(tbl1[,2:3],k=2)
```



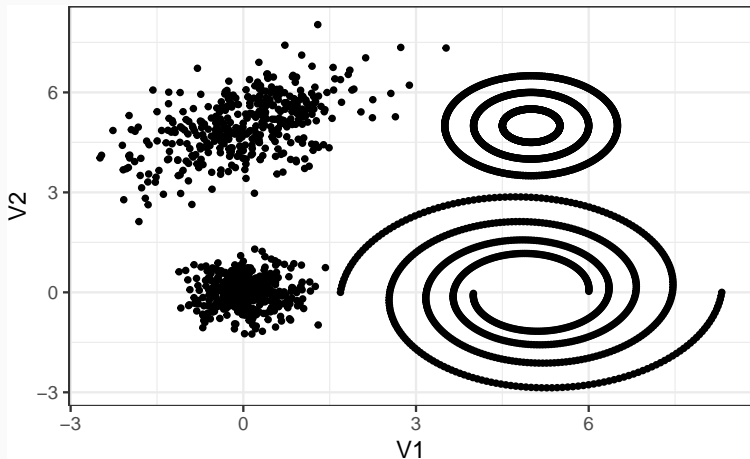
⇒ on pourra prendre  $\varepsilon$  autour de 1

# Résultats

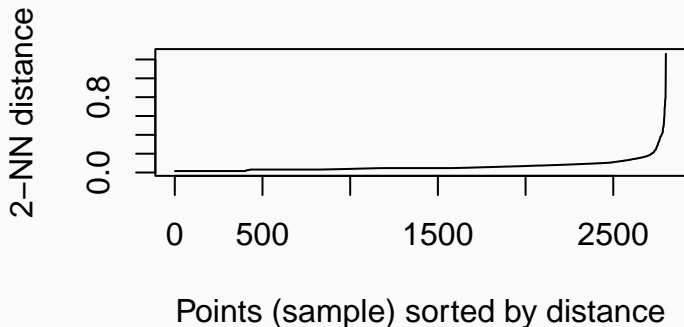
```
> db1 <- dbscan(tbl1[,2:3],eps=1,minPts = 3)
> noyau <- is.corepoint(tbl1[,2:3],eps=1,minPts = 3)
> tbl1_db <- tbl1 |> mutate(dbscan=as.factor(db1$cluster),noyau=noyau)
> ggplot(tbl1_db)+aes(x=V1,y=V2,color=dbscan,shape=noyau)+geom_point()
```



## Exemple 3



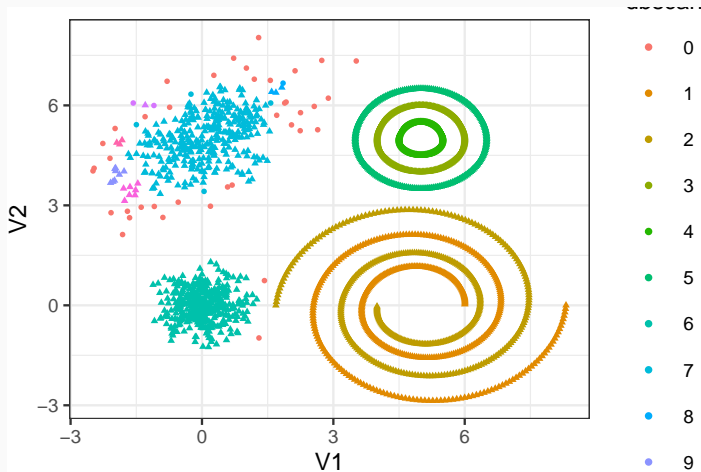
```
> kNNdistplot(tbl, k=2)
```



⇒ on pourra prendre  $\varepsilon$  autour de 0.3.

# Résultats

```
> db1 <- dbscan(tbl,eps=0.3,minPts = 3)
> noyau <- is.corepoint(tbl,eps=0.3,minPts = 3)
> tbl_db <- tbl |> mutate(dbscan=as.factor(db1$cluster),noyau=noyau)
> ggplot(tbl_db)+aes(x=V1,y=V2,color=dbscan,shape=noyau)+geom_point()
```



- Le nombre de groupes n'est pas un paramètre de l'algorithme.

- Le nombre de groupes n'est pas un paramètre de l'algorithme.

## Si trop de groupes ou d'outliers

- **assembler** les groupes à faibles effectifs à des groupes aux effectifs plus conséquents
- **affecter** les outliers aux clusters les plus proches.



- Le nombre de groupes n'est pas un paramètre de l'algorithme.

## Si trop de groupes ou d'outliers

- **assembler** les groupes à faibles effectifs à des groupes aux effectifs plus conséquents
- **affecter** les outliers aux clusters les plus proches.

⇒ par exemple avec un algorithme du **1 plus proche voisin**.

## Avantages

- Permet d'identifier différentes structures géométriques.

## Avantages

- Permet d'identifier différentes structures géométriques.
- Inutile de spécifier le nombre de clusters.

## Avantages

- Permet d'identifier différentes structures géométriques.
- Inutile de spécifier le nombre de clusters.
- Identifie les potentiels outliers.

## Avantages

- Permet d'identifier différentes structures géométriques.
- Inutile de spécifier le nombre de clusters.
- Identifie les potentiels outliers.

## Inconvénients

- 2 paramètres à choisir (comme toujours...).

## Avantages

- Permet d'identifier différentes structures géométriques.
- Inutile de spécifier le nombre de clusters.
- Identifie les potentiels outliers.

## Inconvénients

- 2 paramètres à choisir (comme toujours...).
- Trouver la "bonne" distance, en particulier en grande dimension.

# Clustering spectral

---

- **Cadre identique** :  $G = (V, E)$  un graphe et on veut trouver une partition de  $V$  en **clusters** ou **communautés**.



- **Cadre identique** :  $G = (V, E)$  un graphe et on veut trouver une partition de  $V$  en **clusters** ou **communautés**.
- Approche basée sur la **décomposition spectrale du Laplacien du graphe**.

- **Cadre identique** :  $G = (V, E)$  un graphe et on veut trouver une partition de  $V$  en **clusters** ou **communautés**.
- Approche basée sur la **décomposition spectrale du Laplacien du graphe**.
- Approche utilisée dans un **cadre plus large** :
  - **Problème** : clustering sur un jeu de données standards  $n \times p$  ;
  - L'approche peut être appliquée à une **matrice de similarité**.

- **Cadre identique** :  $G = (V, E)$  un graphe et on veut trouver une partition de  $V$  en **clusters** ou **communautés**.
- Approche basée sur la **décomposition spectrale du Laplacien du graphe**.
- Approche utilisée dans un **cadre plus large** :
  - **Problème** : clustering sur un jeu de données standards  $n \times p$  ;
  - L'approche peut être appliquée à une **matrice de similarité**.
- On pourra consulter [8] dont cette partie est fortement inspirée.

# Notations

- $G = (V, E)$  un graphe non dirigé valué avec  $n = |V|$ .
- $w_{ij} \geq 0$  poids de l'arête entre  $i$  et  $j$  et  $W = (w_{ij})_{1 \leq i, j \leq n}$  la matrice d'adjacence.
- $d_i = \sum_{j \neq i} w_{ij}$  degré du nœud  $i$  et  $D = \text{diag}(d_i)_{1 \leq i \leq n}$  la matrice des degrés.

# Notations

- $G = (V, E)$  un graphe **non dirigé valué** avec  $n = |V|$ .
- $w_{ij} \geq 0$  **poids de l'arête** entre  $i$  et  $j$  et  $W = (w_{ij})_{1 \leq i, j \leq n}$  la **matrice d'adjacence**.
- $d_i = \sum_{j \neq i} w_{ij}$  **degré** du nœud  $i$  et  $D = \text{diag}(d_i)_{1 \leq i \leq n}$  la matrice des degrés.

## Laplacien non normalisé

Le **Laplacien non normalisé** de  $G$  est la matrice  $n \times n$  définie par :

$$L = D - W.$$

## Quelques propriétés

Les deux propositions suivantes sont **fondamentales** pour l'algorithme de **clustering spectral**.

### Proposition 1

1. Pour tout vecteur  $f \in \mathbb{R}^n$  on a

$$f' L f = \frac{1}{2} \sum_{1 \leq i, j \leq n} w_{ij} (f_i - f_j)^2.$$

2.  $L$  est symétrique et semi définie positive.

## Quelques propriétés

Les deux propositions suivantes sont **fondamentales** pour l'algorithme de **clustering spectral**.

### Proposition 1

1. Pour tout vecteur  $f \in \mathbb{R}^n$  on a

$$f' L f = \frac{1}{2} \sum_{1 \leq i, j \leq n} w_{ij} (f_i - f_j)^2.$$

2.  $L$  est symétrique et semi définie positive.
3. La **plus petite valeur propre** de  $L$  est 0, le vecteur propre correspondant est  $\mathbf{1}_n$ .
4.  $L$  a  $n$  valeurs propres non nulles  $0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ .

### Proposition 2

Soit  $G$  un graphe non dirigé. Alors

1. le degrés de multiplicité  $k$  de la valeur propre 0 de  $L$  est égal au nombre de composantes connexes  $A_1, \dots, A_k$  dans  $G$ .



## Proposition 2

Soit  $G$  un graphe non dirigé. Alors

1. le degrés de multiplicité  $k$  de la valeur propre 0 de  $L$  est égal au nombre de composantes connexes  $A_1, \dots, A_k$  dans  $G$ .
2. l'espace propre associé à la valeur propre 0 est engendré par les vecteurs d'indicatrices  $\mathbf{1}_{A_1}, \dots, \mathbf{1}_{A_k}$ .

## Proposition 2

Soit  $G$  un graphe non dirigé. Alors

1. le degrés de multiplicité  $k$  de la valeur propre 0 de  $L$  est égal au nombre de composantes connexes  $A_1, \dots, A_k$  dans  $G$ .
2. l'espace propre associé à la valeur propre 0 est engendré par les vecteurs d'indicatrices  $\mathbf{1}_{A_1}, \dots, \mathbf{1}_{A_k}$ .

## Conséquence importante

Le spectre de  $L$  permet d'identifier les composantes connexes de  $G$ .

- En pratique : 1 communauté n'est pas forcément égale à une composante connexe.
- On peut par exemple vouloir extraire des communautés dans un graphe à une composante connexe.

- En pratique : 1 communauté n'est pas forcément égale à une composante connexe.
- On peut par exemple vouloir extraire des communautés dans un graphe à une composante connexe.

### Idée

Considérer les  $k$  plus petites valeurs propres du Laplacien.

# Spectral clustering non normalisé

## Algorithme

**Entrées** : un graphe non dirigé  $G$ ,  $k$  le nombre de clusters.

1. Calculer le Laplacien non normalisé  $L$  de  $G$ .

# Spectral clustering non normalisé

## Algorithme

**Entrées** : un graphe non dirigé  $G$ ,  $k$  le nombre de clusters.

1. Calculer le Laplacien non normalisé  $L$  de  $G$ .
2. Calculer les  $k$  premiers vecteurs propres  $u_1, \dots, u_k$  de  $G$ .
3. On note  $U$  la matrice  $n \times k$  qui contient les  $u_k$  et  $y_i$  la  $i$ ème ligne de  $U$ .

# Spectral clustering non normalisé

## Algorithme

**Entrées** : un graphe non dirigé  $G$ ,  $k$  le nombre de clusters.

1. Calculer le Laplacien non normalisé  $L$  de  $G$ .
2. Calculer les  $k$  premiers vecteurs propres  $u_1, \dots, u_k$  de  $G$ .
3. On note  $U$  la matrice  $n \times k$  qui contient les  $u_k$  et  $y_i$  la  $i$ ème ligne de  $U$ .
4. Faire un  $k$ -means avec les points  $y_i, i = 1, \dots, n \implies A_1, \dots, A_k$ .

**Sortie** : clusters  $C_1, \dots, C_k$  avec

$$C_j = \{i | y_i \in A_j\}.$$

- Si  $G$  ne possède pas  $k$  composantes connexes alors  $U$  n'est pas composé que de 1 et de 0.
- On ne peut donc pas extraire directement les composantes à cette étape.



- Si  $G$  ne possède pas  $k$  composantes connexes alors  $U$  n'est pas composé que de 1 et de 0.
- On ne peut donc pas extraire directement les composantes à cette étape.
- Mais

## Remarque

- Si  $G$  ne possède pas  $k$  composantes connexes alors  $U$  n'est pas composé que de 1 et de 0.
- On ne peut donc pas extraire directement les composantes à cette étape.
- Mais si il existe (presque)  $k$  composantes, alors les  $y_i \in \mathbb{R}^k$  risquent de se rapprocher de cette configuration 0-1.

## Remarque

- Si  $G$  ne possède pas  $k$  composantes connexes alors  $U$  n'est pas composé que de 1 et de 0.
- On ne peut donc pas extraire directement les composantes à cette étape.
- Mais si il existe (presque)  $k$  composantes, alors les  $y_i \in \mathbb{R}^k$  risquent de se rapprocher de cette configuration 0-1.
- C'est pourquoi on fait un  $k$ -means en 4.

- Il existe **plusieurs versions** d'algorithmes de **clustering spectral**.
- Les plus utilisées s'appliquent à une **version normalisée du Laplacien**, par exemple :

$$L_{\text{norm}} = I - D^{-1/2} W D^{-1/2}.$$

- Les propriétés de  $L_{\text{norm}}$  sont **proches** de celles de  $L$ . On a par exemple la propriété suivante.

- Il existe **plusieurs versions** d'algorithme de **clustering spectral**.
- Les plus utilisées s'appliquent à une **version normalisée du Laplacien**, par exemple :

$$L_{\text{norm}} = I - D^{-1/2} W D^{-1/2}.$$

- Les propriétés de  $L_{\text{norm}}$  sont **proches** de celles de  $L$ . On a par exemple la propriété suivante.

### Proposition 3

Soit  $G$  un graphe **non dirigé**. Alors

1. le **degrés de multiplicité  $k$**  de la valeur propre 0 de  $L_{\text{norm}}$  est égal au **nombre de composantes connexes  $A_1, \dots, A_k$**  dans  $G$ .

- Il existe **plusieurs versions** d'algorithme de **clustering spectral**.
- Les plus utilisées s'appliquent à une **version normalisée du Laplacien**, par exemple :

$$L_{\text{norm}} = I - D^{-1/2} W D^{-1/2}.$$

- Les propriétés de  $L_{\text{norm}}$  sont **proches** de celles de  $L$ . On a par exemple la propriété suivante.

### Proposition 3

Soit  $G$  un graphe **non dirigé**. Alors

1. le **degrés de multiplicité**  $k$  de la valeur propre 0 de  $L_{\text{norm}}$  est égal au **nombre de composantes connexes**  $A_1, \dots, A_k$  dans  $G$ .
2. l'**espace propre** associé à la valeur propre 0 est engendré par les vecteurs d'indicatrices  $D^{1/2} \mathbf{1}_{A_1}, \dots, D^{1/2} \mathbf{1}_{A_k}$ .

# Clustering spectral normalisé

- On déduit de cette propriété la version la plus courante de clustering spectral du à [10].

## Algorithme

Entrées : un graphe non dirigé  $G$ ,  $k$  le nombre de clusters.

1. Calculer le Laplacien normalisé  $L_{\text{norm}}$  de  $G$ .

# Clustering spectral normalisé

- On déduit de cette propriété la version la plus courante de clustering spectral du à [10].

## Algorithme

Entrées : un graphe non dirigé  $G$ ,  $k$  le nombre de clusters.

1. Calculer le Laplacien normalisé  $L_{\text{norm}}$  de  $G$ .
2. Calculer les  $k$  premiers vecteurs propres  $u_1, \dots, u_k$  de  $G$ . On note  $U$  la matrice  $n \times k$  qui les contient.
3. Calculer  $T$  en normalisant les lignes de  $U$  :  $t_{ij} = u_{ij} / (\sum_{\ell} u_{i\ell}^2)^{1/2}$ .



# Clustering spectral normalisé

- On déduit de cette propriété la version la plus courante de clustering spectral du à [10].

## Algorithme

Entrées : un graphe non dirigé  $G$ ,  $k$  le nombre de clusters.

1. Calculer le Laplacien normalisé  $L_{\text{norm}}$  de  $G$ .
2. Calculer les  $k$  premiers vecteurs propres  $u_1, \dots, u_k$  de  $G$ . On note  $U$  la matrice  $n \times k$  qui les contient.
3. Calculer  $T$  en normalisant les lignes de  $U$  :  $t_{ij} = u_{ij} / (\sum_{\ell} u_{i\ell}^2)^{1/2}$ .
4. Faire un  $k$ -means avec les points  $y_i, i = 1, \dots, n$  ( $i$ ème ligne de  $T$ )  
 $\Rightarrow A_1, \dots, A_k$ .

Sortie : clusters  $C_1, \dots, C_k$  avec

$$C_j = \{i | y_i \in A_j\}.$$

- Algorithme **quasi similaire** au clustering spectral **non normalisé**.
- Une étape de **normalisation** en plus.
- Cette étape se justifie par la **théorie de la perturbation** du spectre d'une matrice.
- On pourra consulter [8] pour des justifications.

- Comme souvent en **clustering**, cet algorithme nécessite de connaître le **nombre de groupes**.

- Comme souvent en **clustering**, cet algorithme nécessite de connaître le **nombre de groupes**.
- Utilisation de **connaissances métier** pour ce choix
- ou

- Comme souvent en **clustering**, cet algorithme nécessite de connaître le **nombre de groupes**.
- Utilisation de **connaissances métier** pour ce choix
- ou étude des **valeurs propres** du Laplacien.

## Remarque importante

- L'algorithme n'utilise pas nécessairement la structure du graphe.

## Remarque importante

- L'algorithme n'utilise pas nécessairement la structure du graphe.
- Il est entièrement basé sur la matrice (d'adjacence)  $W$  des poids qui contient des arêtes.

## Remarque importante

- L'algorithme n'utilise pas nécessairement la structure du graphe.
- Il est entièrement basé sur la matrice (d'adjacence)  $W$  des poids qui contient des arêtes.
- Cette matrice peut également être vue comme une matrice de similarité.



## Remarque importante

- L'algorithme n'utilise pas nécessairement la structure du graphe.
- Il est entièrement basé sur la matrice (d'adjacence)  $W$  des poids qui contient des arêtes.
- Cette matrice peut également être vue comme une matrice de similarité.

## Conséquence

- On peut donc généraliser cet algorithme à n'importe quel problème où on possède une matrice de similarité.
- Exemple : problème de clustering standard sur des données  $n \times p$  (il "suffit" de construire une matrice de similarité).

# Clustering spectral sur un tableau de données

- **Données** : tableau  $n \times p$   $n$  individus,  $p$  variables.
- **Problème** : classification non supervisée des  $n$  individus.

# Clustering spectral sur un tableau de données

- **Données** : tableau  $n \times p$   $n$  individus,  $p$  variables.
- **Problème** : classification non supervisée des  $n$  individus.
- **Méthodes classiques** :  $k$ -means, CAH...

# Clustering spectral sur un tableau de données

- **Données** : tableau  $n \times p$   $n$  individus,  $p$  variables.
- **Problème** : classification non supervisée des  $n$  individus.
- **Méthodes classiques** :  $k$ -means, CAH...

## Alternative : clustering spectral

1. construire un **graphe de similarité** ;
2. lancer l'algorithme de **clustering spectral** sur **ce graphe** (ou plutôt sur sa **matrice de similarité**).

## Construction du graphe de similarités

- On peut utiliser les techniques vues dans la section ?? :  
 $\epsilon$ -neighborhood graph ou plus proches voisins (mutuels ou non).

# Construction du graphe de similarités

- On peut utiliser les techniques vues dans la **section ??** :  **$\varepsilon$ -neighborhood graph** ou **plus proches voisins (mutuels ou non)**.
- De façon plus générale, la matrice de similarités s'obtient souvent à partir d'un **noyau  $K$**  :

$$K : \mathbb{R}^p \times \mathbb{R}^p \rightarrow \mathbb{R}$$
$$(x, y) \mapsto \langle \Phi(x), \Phi(y) \rangle_{\mathcal{H}}$$

où  $\Phi : \mathbb{R}^p \rightarrow \mathcal{H}$  est une fonction qui **plonge** les observations dans un espace de Hilbert  $\mathcal{H}$  appelé **feature space**.

## Exemples de noyau

- Linéaire (vanilladot) :  $K(x, y) = \langle x, y \rangle$ .
- Gaussien (rfbdot) :  $K(x, y) = \exp(-\sigma \|x - y\|^2)$ .
- Polynomial (polydot) :  $K(x, y) = (\text{scale} \langle x, y \rangle + \text{offset})^{\text{degree}}$ .
- ...

# Exemples de noyau

- Linéaire (vanilladot) :  $K(x, y) = \langle x, y \rangle$ .
- Gaussien (rfbdot) :  $K(x, y) = \exp(-\sigma \|x - y\|^2)$ .
- Polynomial (polydot) :  $K(x, y) = (\text{scale} \langle x, y \rangle + \text{offset})^{\text{degree}}$ .
- ...

## Références

On pourra trouver dans exemples de noyau dans [6].



## Matrice de similarités avec un noyau

- Etant données  $n$  observations  $x_i \in \mathbb{R}^p$  et un noyau  $K$
- on peut construire une matrice de similarité, par exemple pour un noyau Gaussien :

$$w_{ij} = \begin{cases} \exp(-\sigma \|x_i - x_j\|^2) & \text{si } i \neq j \\ 0 & \text{sinon.} \end{cases}$$

# Matrice de similarités avec un noyau

- Etant données  $n$  observations  $x_i \in \mathbb{R}^p$  et un noyau  $K$
- on peut construire une matrice de similarité, par exemple pour un noyau Gaussien :

$$w_{ij} = \begin{cases} \exp(-\sigma \|x_i - x_j\|^2) & \text{si } i \neq j \\ 0 & \text{sinon.} \end{cases}$$

## Clustering spectral

Le clustering spectral consiste à appliquer l'algorithme vu précédemment en calculant le Laplacien normalisé à partir de cette matrice de similarités (voir [10, 1]).

# Clustering spectral sur des données $n \times p$

## Algorithme

**Entrées** : tableau de données  $n \times p$ ,  $K$  un noyau,  $k$  le nombre de clusters.

1. Calculer la matrice de similarités  $W$  sur les données avec le noyau  $K$ .
2. Calculer le Laplacien normalisé  $L_{\text{norm}}$  à partir de  $W$ .

# Clustering spectral sur des données $n \times p$

## Algorithme

**Entrées** : tableau de données  $n \times p$ ,  $K$  un noyau,  $k$  le nombre de clusters.

1. Calculer la matrice de **similarités**  $W$  sur les données avec le **noyau**  $K$ .
2. Calculer le **Laplacien normalisé**  $L_{\text{norm}}$  à partir de  $W$ .
3. Calculer les  **$k$  premiers vecteurs propres**  $u_1, \dots, u_k$  de  $G$ . On note  $U$  la matrice  $n \times k$  qui les contient.
4. Calculer  $T$  en **normalisant les lignes** de  $U$  :  $t_{ij} = u_{ij} / (\sum_{\ell} u_{i\ell}^2)^{1/2}$ .
5. Faire un  **$k$ -means** avec les points  $y_i, i = 1, \dots, n$  ( $i$ ème ligne de  $T$ )  
 $\implies A_1, \dots, A_k$ .

**Sortie** : clusters  $C_1, \dots, C_k$  avec

$$C_j = \{i | y_i \in A_j\}.$$

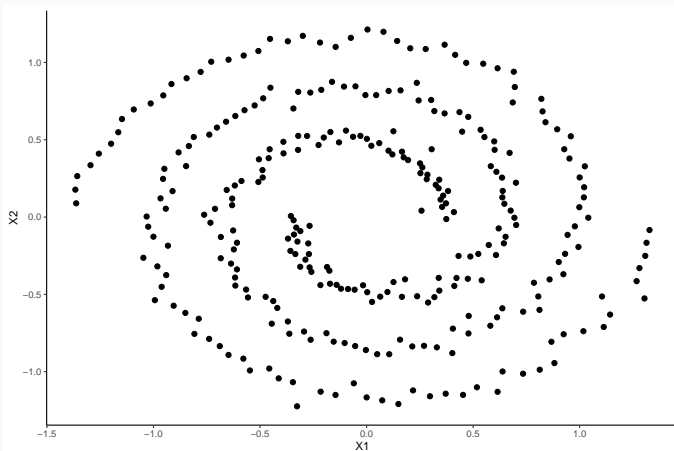
- La fonction `specc` du package `kernlab` permet de faire le clustering spectral.

- La fonction `specc` du package `kernlab` permet de faire le clustering spectral.
- **Exemple** : données `spirals`

```
> library(kernlab)
> data(spirals)
> spirals1 <- data.frame(spirals)
> head(spirals1)
##           X1           X2
## 1  0.8123568 -0.98712687
## 2 -0.2675890 -0.32552004
## 3  0.3739746 -0.01293652
## 4  0.2576481  0.04130805
## 5 -0.8472613  0.32939461
## 6  0.4097649  0.03205686
```

# Visualisation du nuage de points

```
> ggplot(spirals1)+aes(x=X1,y=X2)+geom_point()+theme_classic()
```



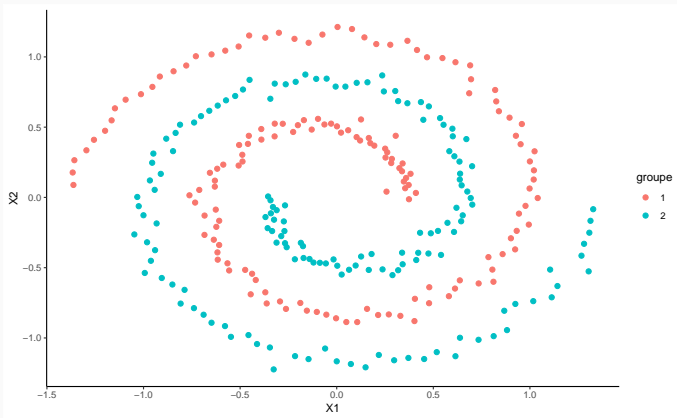
# Le clustering spectral

```
> groupe <- specc(spirals,centers=2,kernel="rbfdot")
> head(groupe)
## [1] 2 2 1 1 2 1
> spirals1 <- spirals1 %>% mutate(groupe=as.factor(groupe))
> ggplot(spirals1)+aes(x=X1,y=X2,color=groupe)+geom_point(size=2)+theme_classic()
```



# Le clustering spectral

```
> groupe <- specc(spirals,centers=2,kernel="rbfdot")  
> head(groupe)  
## [1] 2 2 1 1 2 1  
> spirals1 <- spirals1 %>% mutate(groupe=as.factor(groupe))  
> ggplot(spirals1)+aes(x=X1,y=X2,color=groupe)+geom_point(size=2)+theme_classic()
```



- [1] E. ARIAS-CASTRO. "Clustering based on pairwise distances when the data is of mixed dimensions". In : *IEEE Transaction on Information Theory* 57.3 (2011), p. 1692-1706.
- [2] Hans-Hermann BOCK. "Origins and extensions of the  $k$ -means algorithm in cluster analysis". In : *Electronic journal for history of probability and statistics* 4 (2008), p. 1-18.
- [3] M. ESTER et al. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In : Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96). 1996.

- [4] E. W. FORGY. "Cluster analysis of multivariate data: efficiency vs interpretability of classifications". In : *Biometrics* 21 (1965), p. 768-769.
- [5] J. A. HARTIGAN et M. A. WONG. "Algorithm AS 136: A K-means clustering algorithm". In : *Applied Statistics* 28 (1979), p. 100-108.
- [6] A KARATZOGLOU et al. "kernlab – An S4 Package for Kernel Methods in R". In : *Journal of Statistical Software* 11.9 (2004).
- [7] S. P. LLOYD. "Least squares quantization in PCM". In : *IEEE Transactions on Information Theory* 28 (1982), p. 128-137.
- [8] U. von LUXBURG. "A tutorial on spectral clustering". In : *Statistics and computing* 17 (2017), p. 395-416.

- [9] J. MACQUEEN. "Some methods for classification and analysis of multivariate observations". In : *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*. Sous la dir. de L. M. Le Cam & J. NEYMAN. T. 28. Berkeley, 1967, p. 281-297.
- [10] A. NG, M. JORDAN et Y. WEISS. "On spectral clustering analysis". In : *Advances in Neural Information Processing Systems (NIPS)*, t. 14. 2002, p. 849-856.