

Méthodes de Partitionnement et d'apprentissage non supervisé

Classification Hiérarchique et Kmeans

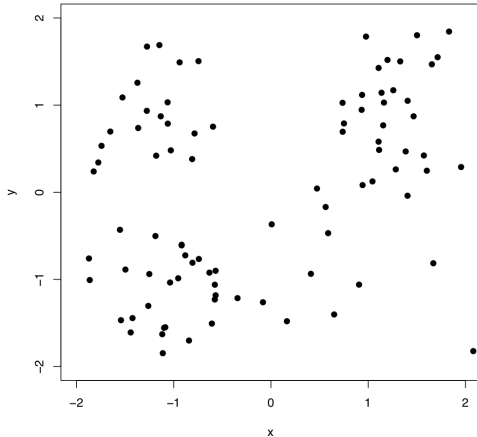
Anne Badel et Frédéric Guyon

2019-02-18

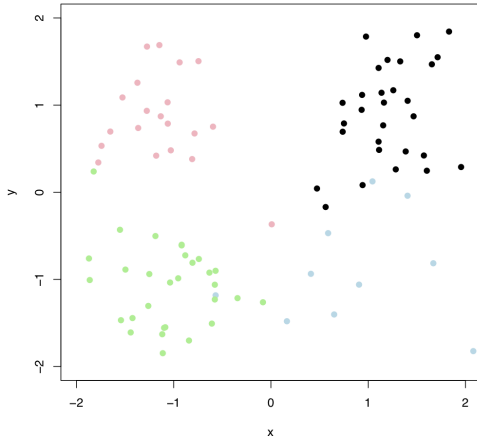
Partitionnement et apprentissage

- ▶ On a une **représentation** des données
 - ▶ sous forme de valeurs réelles=vecteur de
 - ▶ sous forme de catégories
- ▶ **Clustering**: on cherche a priori des groupes dans les données
- ▶ **Apprentissage**:
 - ▶ on connaît le partitionnement sur un jeu de données
 - ▶ on cherche le groupe (la classe) de nouvelles données

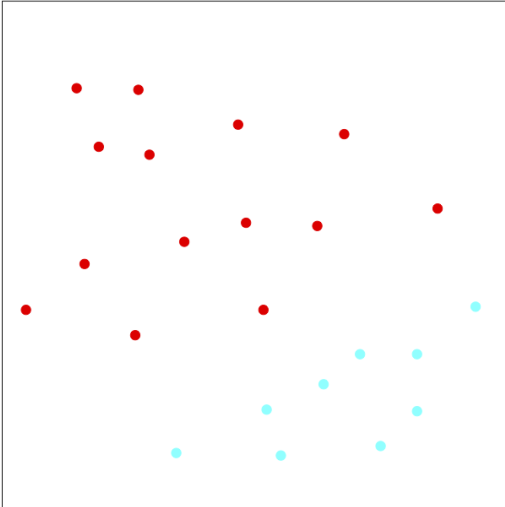
Partitionnement linéaire



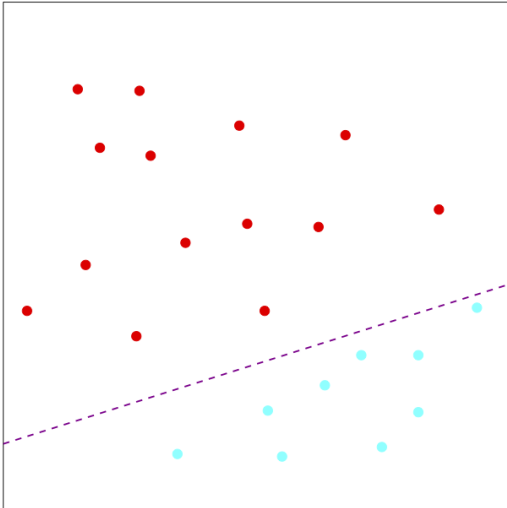
Partitionnement linéaire



Apprentissage



Apprentissage: Séparation linéaire



Méthodes

Trois grands principes de méthodes basées sur:

- ▶ La géométrie
- ▶ Les probabilités (statistique)
- ▶ Les graphes

En fait, trois façons de voir les mêmes algorithmes

Géométrie et distances

On considère les données comme des points de R^n :

- ▶ géométrie donnée par distances
- ▶ distances = dissimilarités imposées par le problème
- ▶ dissimilarités \longrightarrow permettent visualisation de l'ensemble des points
- ▶ Détermination visuelle des groupes

Géométrie et distances

Sur la base d'une distance (souvent euclidienne)

- ▶ Partitionnement:
 - ▶ Moyennes mobiles ou K-means : séparation optimale des groupes connaissant le nombre de groupe
 - ▶ Méthode agglomérative ou hierarchical clustering
- ▶ Classification:
 - ▶ attribution K plus proches voisins (K Nearest Neighbor)
 - ▶ séparation linéaire ou non linéaire

Distances

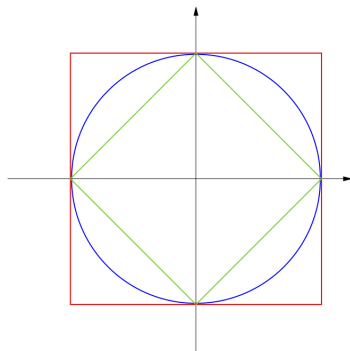
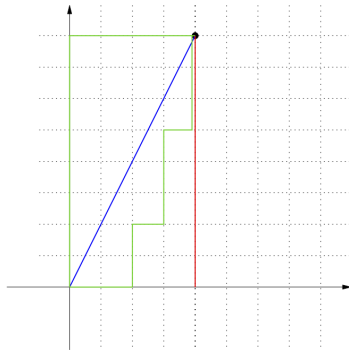
Définition d'une distance : fonction positive de deux variables

1. $d(x,y) \geq 0$
2. $d(x,y)=d(y,x)$
3. $d(x,y)=0 \iff x=y$
4. inégalité triangulaire: $d(x,z) \leq d(x,y)+d(y,z)$

Si 1,2,3 : dissimilarité

Distances utilisées dans R

- ▶ distance euclidienne ou distance l_2 : $d(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$
- ▶ distance de manhattan ou distance l_1 : $d(x, y) = \sum_i |x_i - y_i|$
- ▶ distance du maximum ou l-infini, l_∞ : $d(x, y) = \max_i |x_i - y_i|$



— distance euclidienne

— distance de manhattan

Distances utilisées dans R

- ▶ distance de Minkowski l_p :

$$d(x, y) = \sqrt[p]{\sum_i (|x_i - y_i|^p)}$$

- ▶ distance de Canberra (x et y valeurs positives):

$$d(x, y) = \sum_i \frac{x_i - y_i}{x_i + y_i}$$

- ▶ distance binaire ou distance de Jaccard ou Tanimoto:
proportion de propriétés communes

Autres distances non géométriques (pour information)

Utilisées en bio-informatique:

- ▶ Distance de Levenshtein: nombre de substitutions, insertions, deletions entre deux chaînes de caractères

$$d(\text{"BONJOUR"}, \text{"BONSOIR"}) = 2$$

- ▶ Distance d'alignements: distances de Levenshtein avec poids (par ex. matrices BLOSSUM)
- ▶ Distances d'arbre (Neighbor Joining)
- ▶ Distances ultra-métriques (phylogénie UPGMA)

Distances entre groupes

- ▶ Single linkage

$$D(C_1, C_2) = \min_{i \in C_1, j \in C_2} D(x_i, x_j)$$

- ▶ Complete linkage

$$D(C_1, C_2) = \max_{i \in C_1, j \in C_2} D(x_i, x_j)$$

- ▶ Group average

$$D(C_1, C_2) = \frac{1}{N_1 N_2} \sum_{i \in C_1, j \in C_2} D(x_i, x_j)$$

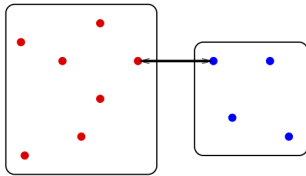
- ▶ Ward

$$d^2(C_i, C_j) = I_{intra}(C_i \cup C_j) - I_{intra}(C_i) - I_{intra}(C_j)$$

$$D(C_1, C_2) = \sqrt{\frac{N_1 N_2}{N_1 + N_2}} \|m_1 - m_2\|$$

Distances entre groupes

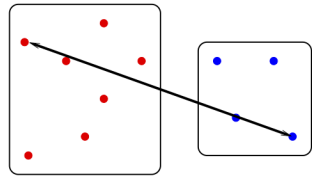
Single



Classe 1

Classe2

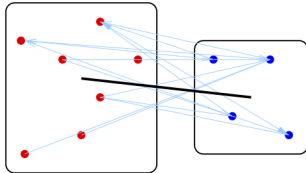
Complete



Classe 1

Classe2

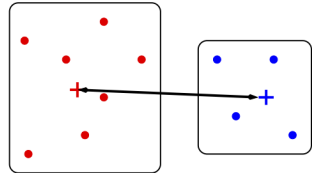
Average



Classe 1

Classe2

Ward



Classe 1

Classe2

Les données

Ces données sont un classique des méthodes d'apprentissage

Dans un premier temps, regardons les données

```
dim(mes.iris)
```

```
[1] 150   4
```

```
head(mes.iris)
```

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

```
str(mes.iris)
```

```
'data.frame':  150 obs. of  4 variables:
```

```
$ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9
```

```
$ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1
```

```
$ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1
```

```
$ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0
```

```
summary(mes.iris)
```

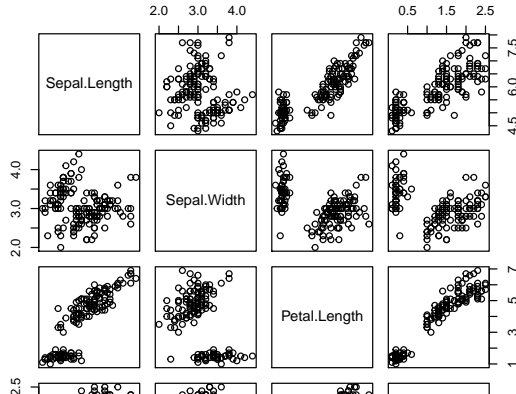
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Min. :4.300	Min. :2.000	Min. :1.000	Min. :0.100
1st Qu.:5.100	1st Qu.:2.800	1st Qu.:1.600	1st Qu.:0.100
Median :5.800	Median :3.000	Median :4.350	Median :1.300
Mean :5.843	Mean :3.057	Mean :3.758	Mean :1.203
3rd Qu.:6.400	3rd Qu.:3.300	3rd Qu.:5.100	3rd Qu.:1.800
Max. :7.900	Max. :4.400	Max. :6.900	Max. :2.500

Visualisation des données

On peut ensuite essayer de visualiser les données

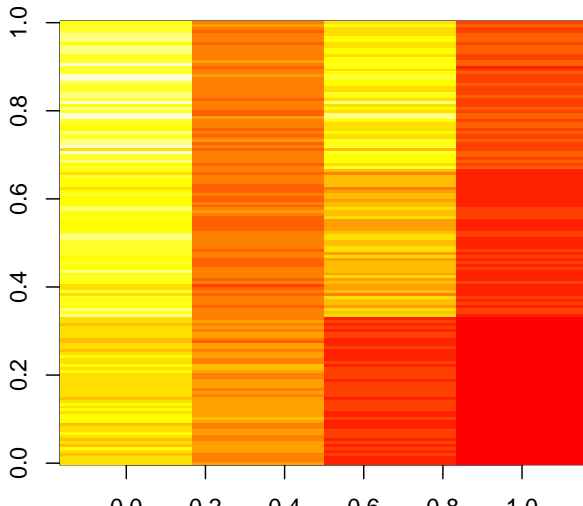
- ▶ par un plot

```
plot(mes.iris)
```



- ▶ par une image

```
image(t(as.matrix(mes.iris)))
```



Nettoyage des données (1)

Avant de commencer à travailler, il est nécessaire de commencer par vérifier que :

- ▶ il n'y a pas de données manquantes

```
sum(is.na(mes.iris))
```

```
[1] 0
```

Nettoyage des données (2)

- ▶ aucune variable n'est constante

```
iris.var <- apply(mes.iris, 2, var)
kable(iris.var, digits = 3)
```

	x
Sepal.Length	0.686
Sepal.Width	0.190
Petal.Length	3.116
Petal.Width	0.581

```
sum(apply(mes.iris, 2, var) == 0)
```

```
[1] 0
```

Normalisation

Afin de pouvoir considérer que toutes les variables sont à la même échelle, il est parfois nécessaire de normaliser les données.

- ▶ soit
 - ▶ en centrant (moyenne "0")

```
mes.iris.centre <- scale(mes.iris, center=TRUE, scale=FALSE)
```

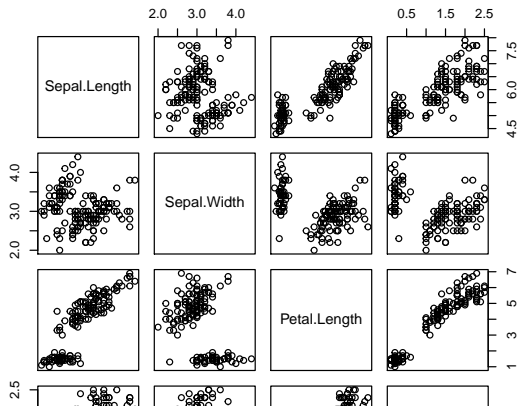
- ▶ soit
 - ▶ en centrant (moyenne "0")
 - ▶ et réduisant (variance "1")

```
mes.iris.scaled <- scale(mes.iris, center=TRUE, scale=TRUE)
```

On peut visuellement regarder l'effet de la normalisation :

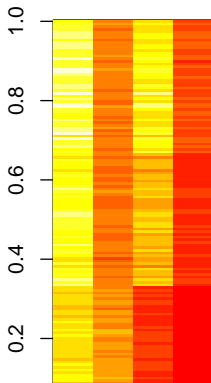
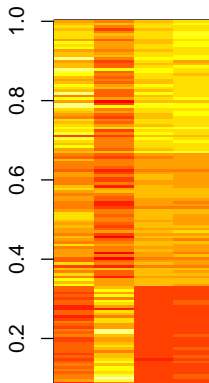
- par un plot des données

```
par(mfrow=c(1,2))  
plot(mes.iris)
```



- ▶ par une image

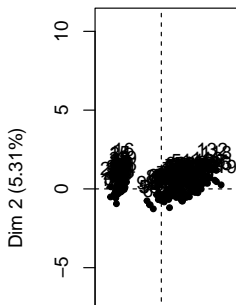
```
par(mfrow=c(1,2))  
image(t(as.matrix(mes.iris)), main="données brutes")  
image(t(as.matrix(mes.iris.scaled)), main="données normalisées")
```

données brutes**données normalisées**

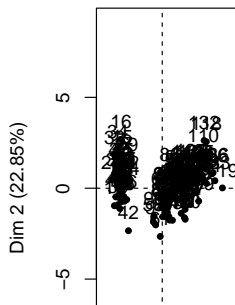
- ▶ par une projection sur une ACP

```
iris.acp <- FactoMineR::PCA(mes.iris, graph=F, scale.unit=F)
iris.scaled.acp <- FactoMineR::PCA(mes.iris.scaled, graph=F)
par(mfrow=c(1,2))
plot(iris.acp, title="données brutes", choix="ind")
plot(iris.scaled.acp, title="données normalisées", choix="ind")
```

données brutes



données normalisées



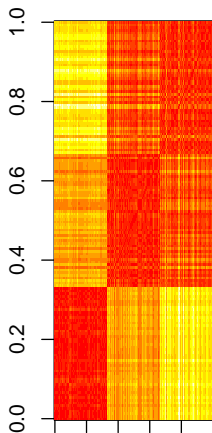
La matrice de distance

Nous utilisons la distance euclidienne

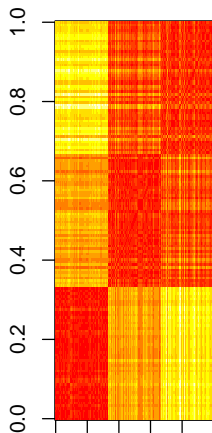
```
iris.euc <- dist(mes.iris)
iris.scale.euc <- dist(mes.iris.scaled)
```

```
par(mfrow=c(1,2))  
image(t(as.matrix(iris.euc)), main="données brutes")  
image(t(as.matrix(iris.scale.euc)), main="données normalisées")
```

données brutes



données normalisées



La classification hiérarchique

Principe

- ▶ classification hiérarchique : mettre en évidence des liens hiérarchiques entre les individus
- ▶ classification hiérarchique **ascendante** : partir des individus pour arriver à des classes / cluster
- ▶ classification hiérarchique **descendante** : partir d'un groupe qu'on subdivise en sous-groupes /cluster jusqu'à arriver à des individus.

Notion importante, cf distances

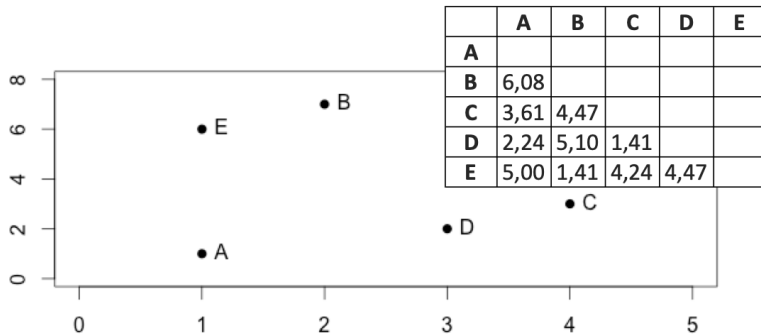
- ▶ ressemblance entre individus = distance
- ▶ ressemblance entre groupes d'individus = critère d'aggrégation
 - ▶ lien complet
 - ▶ lien moyen
 - ▶ critère de Ward

L'algorithme

étape 1 :

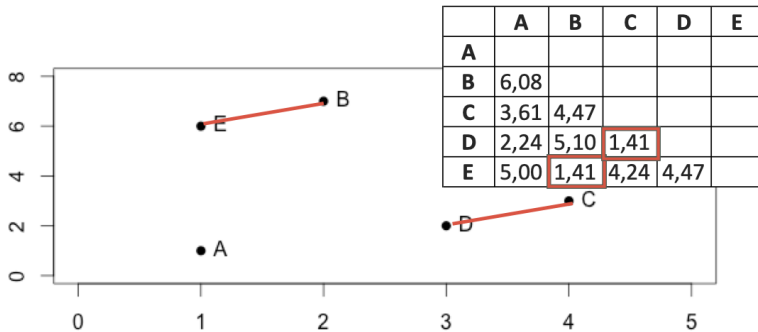
- ▶ départ : n individus = n clusters distincts
- ▶ calcul des distances entre tous les individus
 - ▶ choix de la métrique à utiliser en fonction du type de données
- ▶ regroupement des 2 individus les plus proches $\Rightarrow (n-1)$ clusters

au départ



initialisation : (A), (B), (C), (D), (E)

identification des individus les plus proches



première partition : (A), (BE), (CD)

construction du dendrogramme



étape j :

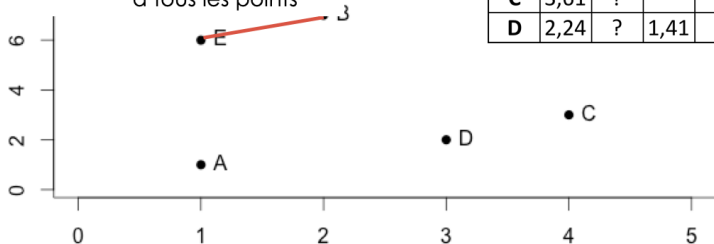
- ▶ calcul des dissemblances entre chaque groupe obtenu à l'étape (j-1)
- ▶ regroupement des deux groupes les plus proches \Rightarrow (n-j) clusters

calcul des nouveaux représentants BE et CD

par le lien moyen,

calcul de BE

calcul de la distance de BE
à tous les points



	A	BE	C	D
A				
BE	?			
C	3,61	?		
D	2,24	?	1,41	

calcul des distances de l'individu restant A aux points moyens

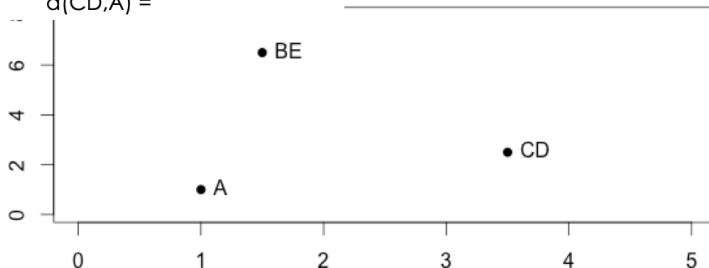
calcul des distances

$$d(BE, A) =$$

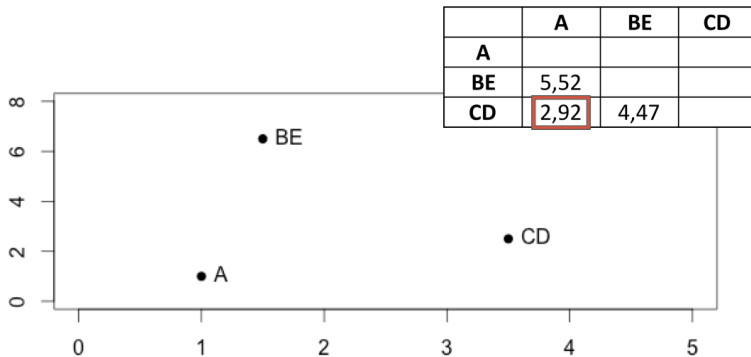
$$d(BE, CD) =$$

$$d(CD, A) =$$

$$d(BE, A) = \frac{d(B, A) + d(E, A)}{2} = \frac{6.08 + 5}{2} = 5.52$$



A est plus proche de ...



deuxième partition : (BE), (CDA)

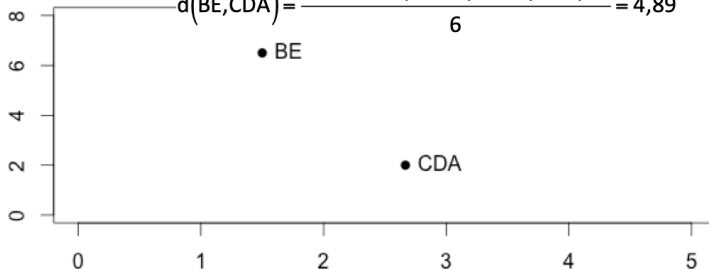
dendrogramme



pour finir

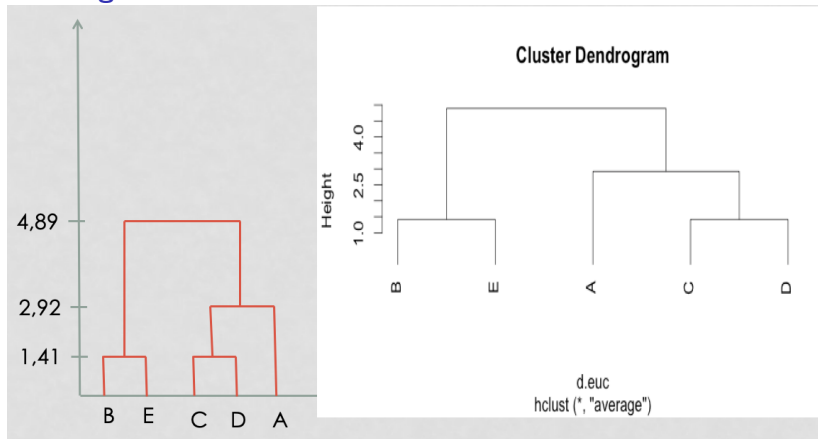
$$d(\text{BE}, \text{CDA}) = \frac{d(\text{B}, \text{C}) + d(\text{E}, \text{C}) + d(\text{B}, \text{D}) + d(\text{E}, \text{D}) + d(\text{B}, \text{A}) + d(\text{E}, \text{A})}{6}$$

$$d(\text{BE}, \text{CDA}) = \frac{6.08 + 5 + 4.47 + 4.24 + 5.1 + 4.47}{6} = 4.89$$



- ▶ à l'étape (n-1), tous les individus sont regroupés dans un même cluster

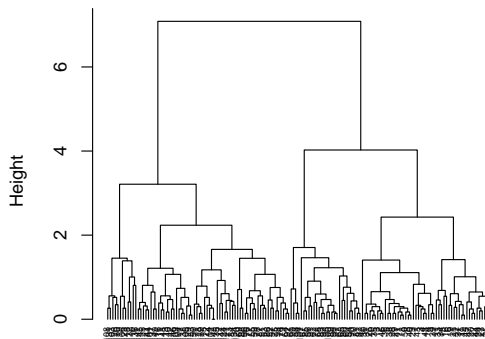
dendrogramme final



Je ne fais pas attention à ce que je fais

```
iris.hclust <- hclust(iris.euc)  
plot(iris.hclust, hang=-1, cex=0.5)
```

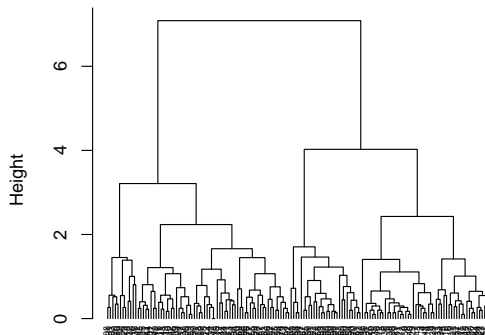
Cluster Dendrogram



Sur données normalisées

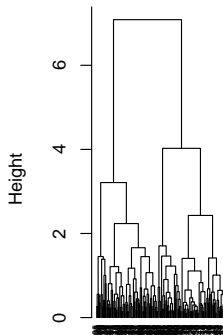
```
iris.scale.hclust <- hclust(iris.scale.euc)  
plot(iris.scale.hclust, hang=-1, cex=0.5)
```

Cluster Dendrogram



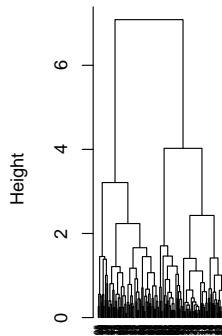
```
par(mfrow=c(1,2))  
plot(iris.hclust, hang=-1, cex=0.5)  
plot(iris.scale.hclust, hang=-1, cex=0.5)
```

Cluster Dendrogram



iris.euc

Cluster Dendrogram

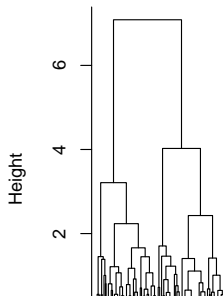


iris.scale.euc

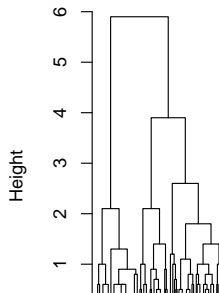
En utilisant une autre métrique

```
iris.scale.max <- dist(mes.iris.scaled, method="max")
iris.scale.hclust.max <- hclust(iris.scale.max)
par(mfrow=c(1,2))
plot(iris.scale.hclust, hang=-1, cex=0.5)
plot(iris.scale.hclust.max, hang=-1, cex=0.5)
```

Cluster Dendrogram



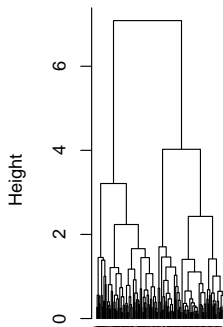
Cluster Dendrogram



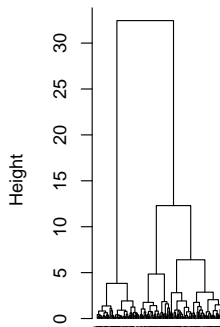
En utilisant un autre critère d'aggrégation

```
iris.scale.hclust.ward <- hclust(iris.scale.euc, method="ward")
par(mfrow=c(1,2))
plot(iris.scale.hclust, hang=-1, cex=0.5)
plot(iris.scale.hclust.ward, hang=-1, cex=0.5)
```

Cluster Dendrogram

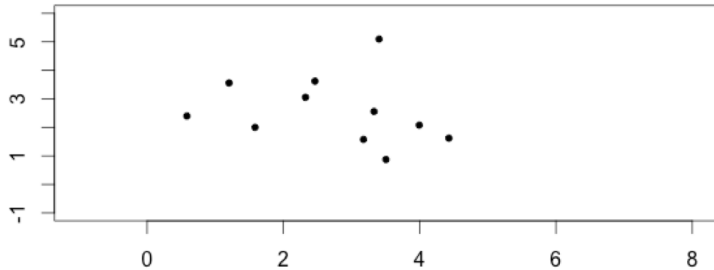


Cluster Dendrogram



Les k-means

Les individus dans le plan



L'algorithme

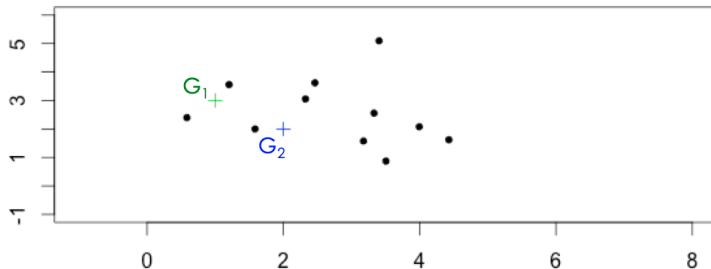
étape 1 :

- ▶ k centres provisoires tirés au hasard
- ▶ k clusters créés à partir des centres en regroupant les individus les plus proches de chaque centre
- ▶ obtention de la partition P_0

choix des centres provisoires

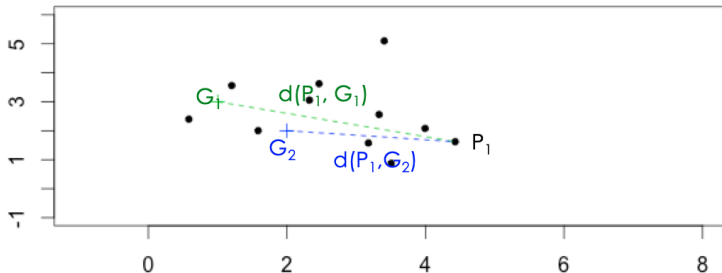
combien de cluster ?

les deux centres initiaux (G_1 et G_2) sont choisis au hasard



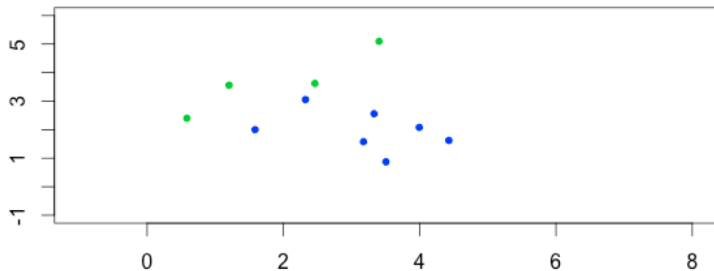
calcul des distances aux centres provisoires

- calcul des distances de chaque point aux centres G_1 et G_2 ,



le point P_1 est affecté au groupe 2

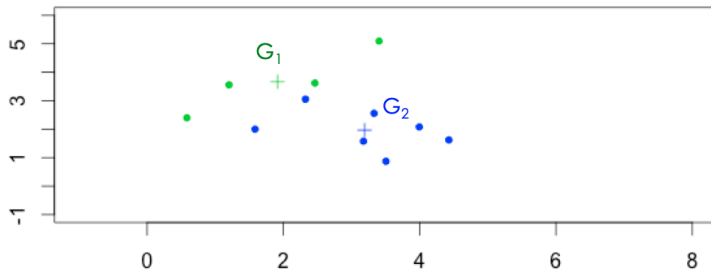
et affectation à un cluster



calcul des nouveaux centres de classes

étape j :

- ▶ construction des centres de gravité des k clusters construits à l'étape (j-1)
- ▶ k nouveaux clusters créés à partir des nouveaux centres suivant la même règle qu'à l'étape 0 obtention de la partition P_j

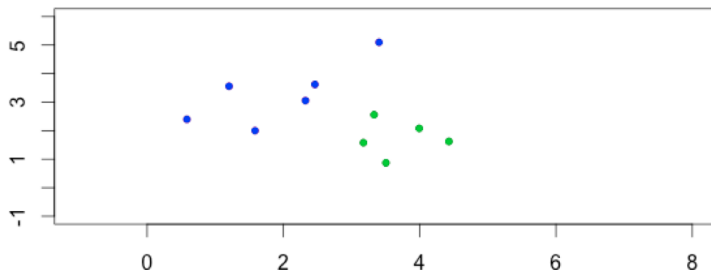


fin :

- ▶ l'algorithme converge vers une partition stable

arrêt :

- ▶ lorsque la partition reste la même, ou lorsque la variance intra-cluster ne décroît plus, ou lorsque le nombre maximal d'itérations est atteint.



Un premier k-means en 5 groupes

```
iris.scale.kmeans5 <- kmeans(mes.iris.scaled, center=5)
iris.scale.kmeans5
```

K-means clustering with 5 clusters of sizes 33, 27, 28, 17,

Cluster means:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
1	-1.0251515	0.17903030	-2.3246667	-0.9690303
2	1.1714815	0.03896296	2.1605185	0.9562222
3	-0.3111905	-0.42161905	0.2027143	0.0292381
4	-0.4727451	0.74266667	-2.2403529	-0.9228627
5	0.4211111	-0.17288889	1.1286667	0.4673333

Clustering vector:

```
[1] 1 1 1 1 1 4 1 1 1 1 4 1 1 1 4 4 4 1 4 4 4 4 1 1 1 1 1
[148] 5 5 5
```

```
iris.scale.kmeans5$cluster
```

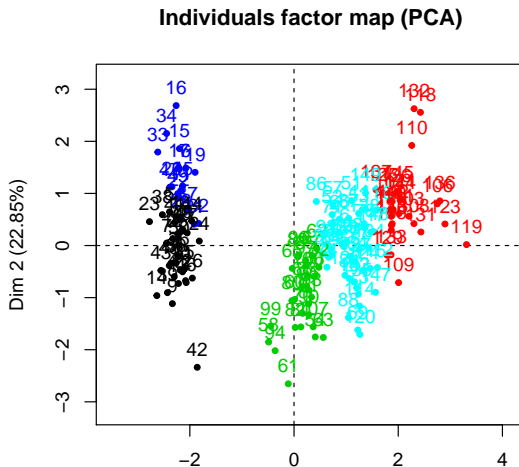
```
[1] 1 1 1 1 1 4 1 1 1 1 4 1 1 1 4 4 4 1 4 4 4 4 1 1 1 1 1  
[148] 5 5 5
```

```
table(iris.scale.kmeans5$cluster)
```

```
 1  2  3  4  5  
33 27 28 17 45
```

Visualisation des clusters

```
plot(iris.scaled.acp, col.ind=iris.scale.kmeans5$cluster, c
```



Combien de clusters ?

Quand une partition est-elle bonne ?

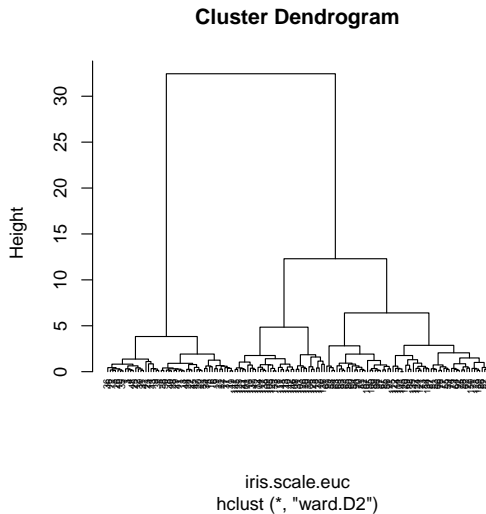
- ▶ si les individus d'un même cluster sont proches
 - ▶ homogénéité maximale à l'intérieur de chaque cluster
- ▶ si les individus de 2 clusters différents sont éloignés
 - ▶ hétérogénéité maximale entre chaque cluster

Classification hiérarchique

La coupure de l'arbre à un niveau donné construit une partition. la coupure doit se faire :

- ▶ après les agrégations correspondant à des valeurs peu élevées de l'indice
- ▶ avant les agrégations correspondant à des niveaux élevés de l'indice, qui dissocient les groupes bien distincts dans la population.

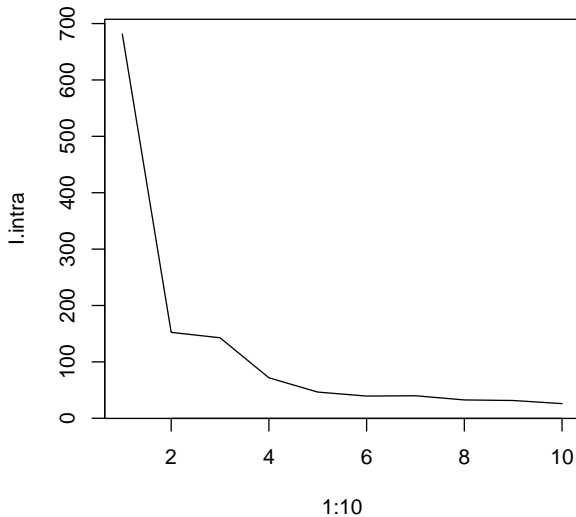
```
plot(iris.scale.hclust.ward, hang=-1, cex=0.5)
```



K-means

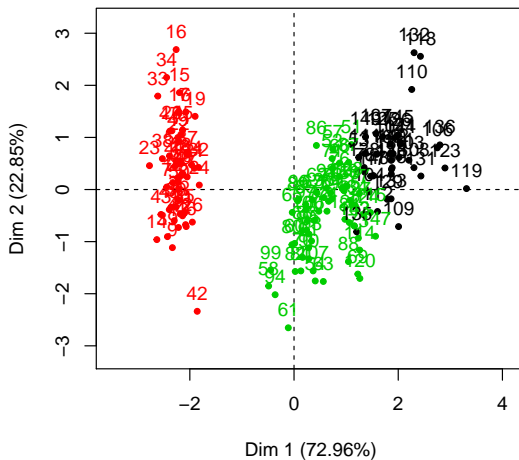
```
I.intra = numeric(length=10)
I.intra[1] = kmeans(mes.iris.scaled, centers=2)$totss
for (i in 2:10) {
  kmi <- kmeans(mes.iris.scaled, centers=i)
  I.intra[i] <- kmi$tot.withinss
}
```

```
plot(1:10, I.intra, type="l")
```



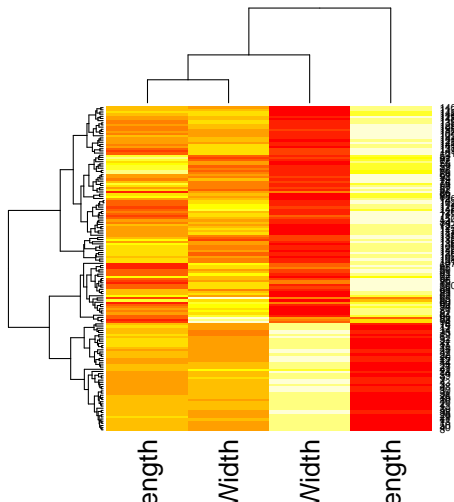
```
iris.scale.kmeans3 <- kmeans(mes.iris.scaled, center=3)
plot(iris.scaled.acp, col.ind=iris.scale.kmeans3$cluster, c
```

Individuals factor map (PCA)



Heatmap

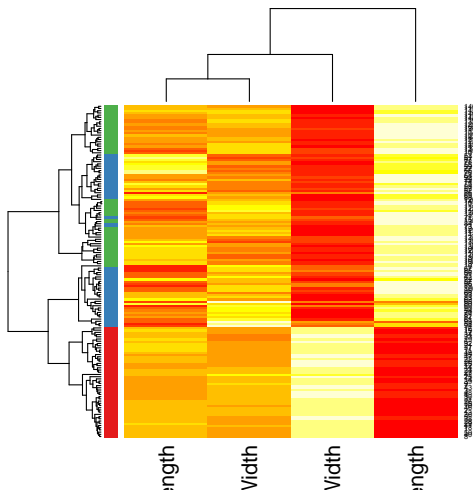
```
heatmap(mes.iris.scaled)
```



```

my_group=as.numeric(as.factor(substr(variete, 1 , 2)))
my_col=brewer.pal(3, "Set1")[my_group]
heatmap(mes.iris.scaled, RowSideColors=my_col)

```



Comparaison de clustering: Rand Index

Mesure de similarité entre deux clustering

à partir du nombre de fois que les classifications sont d'accord

$$R = \frac{m + s}{t}$$

- ▶ m =nombre de paires dans la même classe dans les deux classifications
- ▶ s =nombre de paires séparées dans les deux classifications
- ▶ t =nombre de paires totales

Comparaison de clustering: Adjusted Rand Index

- ▶ ARI = valeur attendue d'être d'accord
- ▶ Prend en compte la taille des classes
- ▶ $ARI = 1$ pour classification identique
- ▶ $ARI \simeq 0$ pour classification aléatoire (peut être < 0)
- ▶ Adapté pour nombre de classe différent entre les deux classifications et taille de classe différente

Comparaison des résultats des deux classifications

- ▶ par une table de confusion

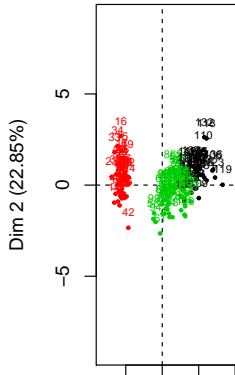
```
cluster.kmeans3 <- iris.scale.kmeans3$cluster  
cluster.hclust5 <- cutree(iris.scale.hclust.ward, k=5)  
table(cluster.hclust5, cluster.kmeans3)
```

	cluster.kmeans3			
cluster.hclust5	1	2	3	
1	0	50	0	
2	2	0	36	
3	0	0	26	
4	24	0	0	
5	12	0	0	

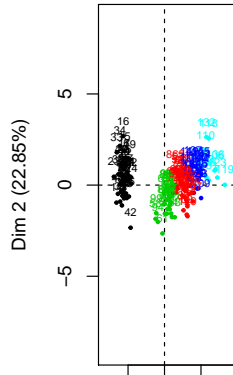
- ▶ par une visualisation

```
par(mfrow=c(1,2))
plot(iris.scaled.acp, col.ind=cluster.kmeans3, choix="ind")
plot(iris.scaled.acp, col.ind=cluster.hclust5, choix="ind")
```

kmeans en 3 groupes



hclust en 5 groupes



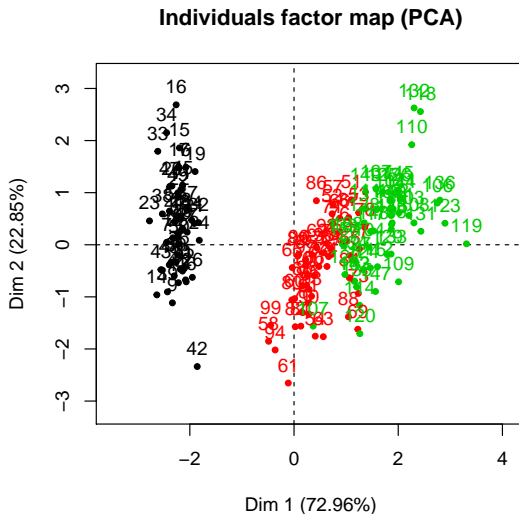
Comparaison avec la réalité

La réalité

```
variete <- iris[,5]  
table(variete)
```

```
variete  
      setosa versicolor  virginica  
        50         50         50
```

```
plot(iris.scaled.acp, col.ind=variete, choix="ind")
```



Comparer k-means avec la réalité

```
conf.kmeans <- table(variete, cluster.kmeans3)
kable(conf.kmeans)
```

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

Setosa vs !Setosa

Visualisation

```
variete2 <- rep("notSetosa", 150)
variete2[variete=="setosa"] <- "setosa"
variete2 = factor(variete2)
table(variete2)
```

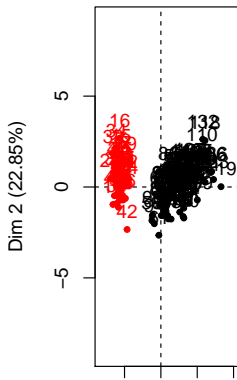
```
variete2
notSetosa    setosa
      100         50
```

```

par(mfrow=c(1,2))
plot(iris.scaled.acp, col.ind=variete2, title="variétés obs
cluster.kmeans2 <- kmeans(mes.iris.scaled, center=2)$cluste
plot(iris.scaled.acp, col.ind=cluster.kmeans2, title="kmean

```

variétés observés



kmeans en 2 groupes

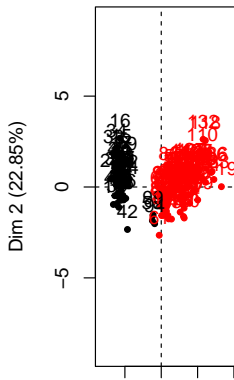


Table de confusion et calcul de performances

```
conf.kmeans <- table(variete2, cluster.kmeans2)
kable(conf.kmeans)
```

	1	2
notSetosa	3	97
setosa	50	0

- table de confusion, taux de bien prédits, spécificité, sensibilité, ...

```

TP <- conf.kmeans[1,1]
FP <- conf.kmeans[1,2]
FN <- conf.kmeans[2,1]
TN <- conf.kmeans[2,2]
P <- TP + FN           # nb positif dans la réalité
N <- TN + FP           # nb négatif dans la réalité
FPrate <- FP / N        # = false alarm rate
Spe <- TN / N           # = spécificité
Sens <- recall <- TPrate <- TP / P      # = hit rate ou re
PPV <- precision <- TP / (TP + FP)
accuracy <- (TP + TN) / (P + N)
F.measure <- 2 / (1/precision + 1/recall)
performance <- c(FPrate, TPrate, precision, recall, accuracy)
names(performance) <- c("FPrate", "TPrate", "precision", "recall", "accuracy")

```

```
kable(performance, digits=3)
```

	x
FPrate	1.000
TPrate	0.057
precision	0.030
recall	0.057
accuracy	0.020
F.measure	0.039
Spe	0.000
PPV	0.030

- ▶ rand index et adjusted rand index

```
clues::adjustedRand(as.numeric(variete2), cluster.kmeans2)
```

	Rand	HA	MA	FM	Jaccard
	0.9605369	0.9204051	0.9208432	0.9639434	0.9302767

Versicolor vs !Versicolor

Visualisation

```
variete2 <- rep("notVersicolor", 150)
variete2[variete=="versicolor"] <- "versicolor"
variete2 = factor(variete2)
table(variete2)
```

```
variete2
notVersicolor    versicolor
              100              50
```

```
par(mfrow=c(1,2))
plot(iris.scaled.acp, col.ind=variete2)
cluster.kmeans2 <- kmeans(mes.iris.scaled, center=2)$cluster
plot(iris.scaled.acp, col.ind=cluster.kmeans2)
```

Table de confusion et calcul de performances

```
conf.kmeans <- table(variete2, cluster.kmeans2)
kable(conf.kmeans)
```

	1	2
notVersicolor	50	50
versicolor	3	47

```
TP <- conf.kmeans[1,1]
FP <- conf.kmeans[1,2]
FN <- conf.kmeans[2,1]
TN <- conf.kmeans[2,2]
P <- TP + FN           # nb positif dans la réalité
N <- TN + FP           # nb négatif dans la réalité
FPrate <- FP / N        # = false alarm rate
Spe <- TN / N           # = spécificité
Sens <- recall <- TPrate <- TP / P      # = hit rate ou re
```

```
kable(performance, digits=3)
```

	x
FPrate	0.515
TPrate	0.943
precision	0.500
recall	0.943
accuracy	0.647
F.measure	0.654
Spe	0.485
PPV	0.500

```
clues::adjustedRand(as.numeric(variete2), cluster.kmeans2)
```

	Rand	HA	MA	FM	Jaccard
	0.53995526	0.07211421	0.07722223	0.57895580	0.40737752

Contact: anne.badel@univ-paris-diderot.fr