

# Introduction à l'apprentissage automatique - Machine Learning

Emmanuel Grolleau

Observatoire de Paris – LESIA – Service d'Informatique Scientifique

08/2017

# Sommaire

- **Introduction**
- Apprentissage supervisé
- Apprentissage non supervisé
- Séries temporelles
- Conclusion

# Définition

- L'**apprentissage automatique** (en anglais *Machine Learning*) est un type d'intelligence artificielle qui confère aux ordinateurs la capacité d'apprendre sans être explicitement programmés.
- Il consiste en *la* mise en place **d'algorithmes** ayant pour objectif d'obtenir une **analyse prédictive** à partir de données, dans un but précis.

# Exemples

- La **voiture autonome** de Google
- Classification des **emails** dans gmail
- Moteur de recherche de Google
- La **traduction en temps réel** de Skipe / La **reconnaissance vocale** Siri d'Apple
- Détection de **fraude** dans le monde de la banque
- Reconnaissance **faciale**

- Les algorithmes de Machine Learning utilisent donc nécessairement une phase dite d'apprentissage.
- Les programmes d'apprentissage automatique détectent des schémas dans les données et ajustent leur fonctionnement en conséquence.

# Apprentissage supervisé / non supervisé

- Les algorithmes d'apprentissage peuvent se catégoriser selon le mode d'apprentissage qu'ils emploient :
  - **Apprentissage supervisé**
    - les *classes* sont prédéterminées et les *exemples* connus, le système apprend à classer selon un *modèle* de classement.
    - Le processus se passe en deux phases. Lors de la première phase (dite d'*apprentissage*), il s'agit de déterminer un modèle des données étiquetées.
    - La seconde phase (dite de *test*) consiste à prédire l'étiquette d'une nouvelle donnée, connaissant le modèle préalablement appris.

# Apprentissage supervisé / non supervisé

- **Apprentissage non supervisé**
  - Quand le système ou l'opérateur ne disposent que d'exemples, mais non d'étiquettes, et que le nombre de classes et leur nature n'ont pas été prédéterminés.
  - L'algorithme doit découvrir par lui-même la structure plus ou moins *cachée* des données. Le partitionnement de données, *data clustering* en anglais, est un algorithme d'apprentissage non supervisé.

# Différence Machine Learning / Data Mining

- Data Mining : retraiter les données déjà connues pour en sortir des propriétés et des précisions encore inconnues.
- Machine Learning : apprendre aux systèmes à prédire ce que pourrait être le résultat sorti de données encore inconnues à partir de données connues.

# Catégorisation par type de variable en sortie

- **Régression**: la variable de sortie est continue.
- **Classification**: la variable de sortie prend ses valeurs dans des classes.

# Apprentissage supervisé

- **Règle de Bayes**
- Classification naïve bayésienne
- Régression multivariée
- Régression régularisée
- Protocole d'apprentissage
- Les k plus proches voisins
- Dilemme biais/variance
- Arbre de décision
- Bagging
- Forêt aléatoire
- Perceptron
- Perceptron multicouche
- Les réseaux de neurones
- Deep learning

# Classification : Règle de décision bayésienne

- On souhaite trouver une fonction de classification qui associe une classe  $C(x)$  à une valeur de  $x$ .
  - Ex : en fonction de la variable aléatoire « taille de l'individu » on souhaite savoir s'il s'agit d'un homme ( $C_H$ ) ou d'une femme ( $C_F$ ).
- On note  $P(C_k)$  les probabilités des classes  $C$ . On parle de probabilité a priori.

$$\sum_{k=1}^n P(C_k) = 1$$

- Ex : Dans la population française :

$$P(C_F) = 0.51; P(C_H) = 0.49$$

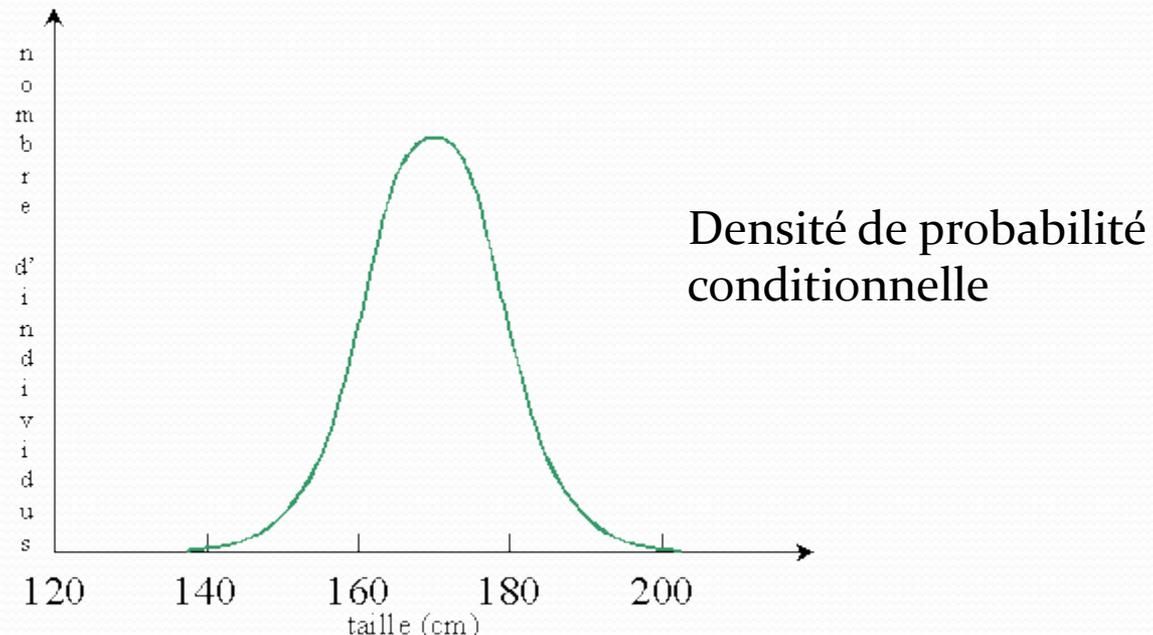
# Classification : Règle de décision bayésienne

- On note  $P(x | C_k)$  la densité de probabilité conditionnelle, c.-à-d. la probabilité d'obtenir  $x$  sachant que l'individu appartient à la classe  $C_k$
- Dans le cas où  $X$  est une variable aléatoire continue on a :

$$\forall k, \int_{x \in X} p(x | C_k) dx = 1$$

# Classification : Règle de décision bayésienne

- Exemple de densité de probabilité : la taille des hommes français.
- $Classe = C_H; X = \text{taille} \Rightarrow P(\text{taille} | C_H)$

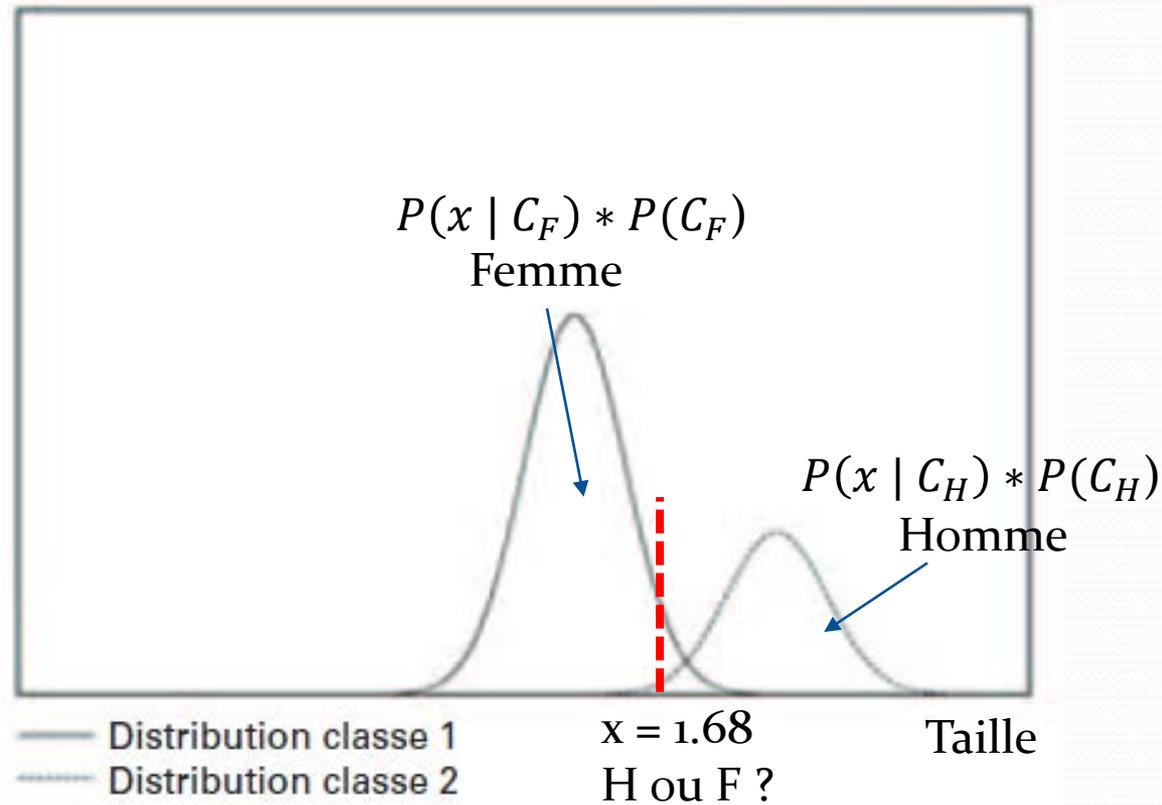


# Classification : Règle de décision bayésienne

- Nous cherchons à classifier, c.-à-d. à trouver la classe la plus probable d'un individu en fonction de sa taille.
- Autrement dit, nous cherchons  $P(C_k | x)$ . On parle de probabilité a posteriori.

$$\forall x \in X, \sum_{k=1}^n P(C_k | x) = 1$$

# Classification : Règle de décision bayésienne



*N.B : La différence entre  $P(C_F)$  et  $P(C_H)$  a été volontairement exagérée*

Distribution de taille des classes  $C_H$  (Homme) et  $C_F$  (Femme).

# Classification : Règle de décision bayésienne

- Règle de Bayes :

$$P(C_k | x) = \frac{P(x | C_k) * P(C_k)}{P(x)} = \frac{P(x | C_k) * P(C_k)}{\sum_{j=1}^N P(x | C_j) * P(C_j)}$$

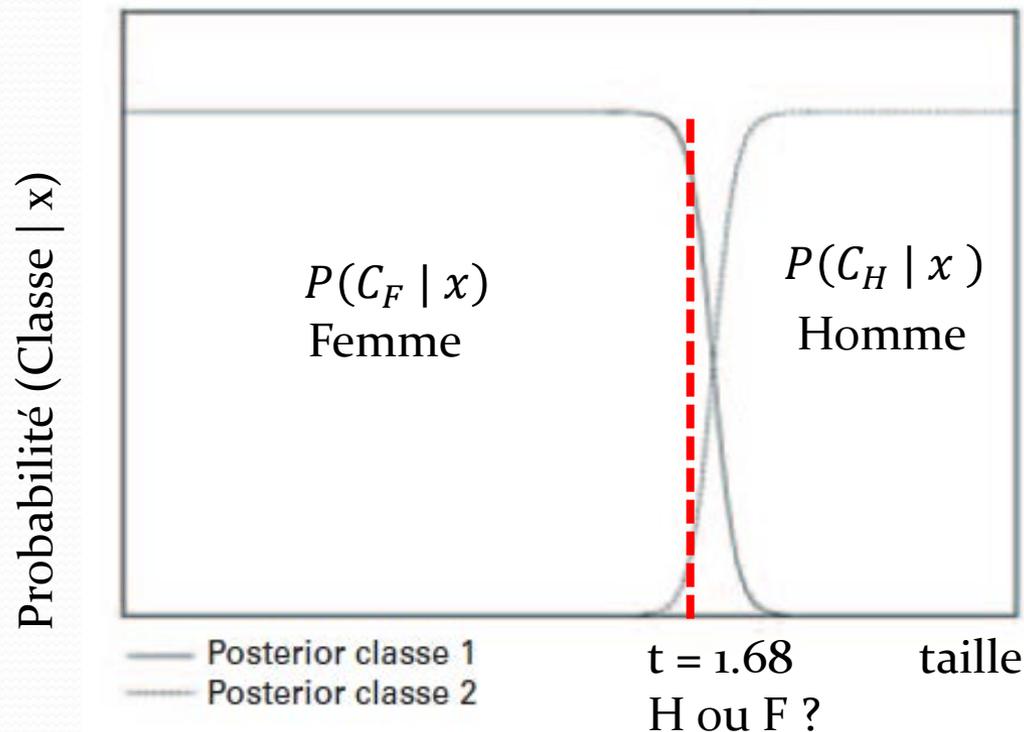
- Exemple

$$P(C_H | 1.68) = \frac{P(1.68 | C_H) * P(C_H)}{P(1.68)}$$

# Classification : Règle de décision bayésienne

## bayésienne

La règle de décision bayésienne consiste à classer un exemple dans la classe maximisant la probabilité a posteriori.



# Classification : Règle de décision bayésienne

- On peut estimer la qualité d'une règle de classification par la probabilité qu'elle produise le bon résultat.

$$P(\text{erreur} | x) = \begin{cases} P(C_F | x) & \text{si règle a classifié en } C_H \\ P(C_H | x) & \text{si règle a classifié en } C_F \end{cases}$$

$$P(\text{erreur}) = \int_{x \in X} P(\text{erreur} | x) p(x) dx$$

Pour la règle de Bayes, on a (**Bayes error rate**) :

$$P(\text{erreur}) = 1 - \max_k P(C_k | x) = \min_k [1 - P(C_k | x)]$$

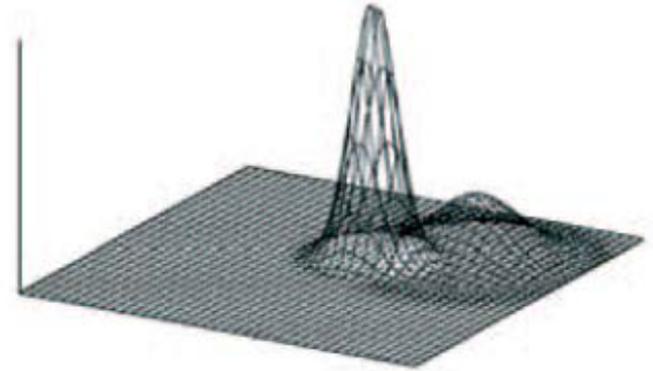
=> **On ne peut pas faire mieux !**

# Classification : Règle de décision bayésienne

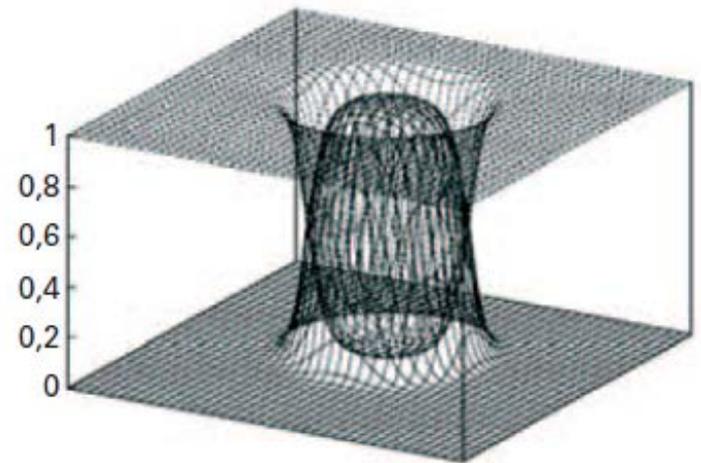
- On ne peut faire mieux que la règle de décision bayésienne mais le problème est que généralement nous n'avons pas accès aux probabilités a priori et aux probabilités conditionnelles.
- Il faut faire autrement.
- On travaille généralement dans un espace à  $n$  dimensions et  $X$  est un vecteur de variables aléatoires.

# Classification : Règle de décision bayésienne

En 2 dimensions :  
Taille et poids des  
individus.



— Distribution classe 1  
- - - Distribution classe 2



— Posterior classe 1  
- - - Posterior classe 2

# Apprentissage supervisé

- Règle de Bayes
- **Classification naïve bayésienne**
- Régression multivariée
- Régression régularisée
- Protocole d'apprentissage
- Les k plus proches voisins
- Dilemme biais/variance
- Arbre de décision
- Bagging
- Forêt aléatoire
- Perceptron
- Perceptron multicouche
- Les réseaux de neurones
- Deep learning

# Classification naïve bayésienne

- La classification dépend généralement de  $n$  variables (taille, poids, ...).  $X$  est donc un vecteur de V.A.

$$P(C_k | X_1, \dots, X_n) = \frac{P(X_1, \dots, X_n | C_k) * P(C_k)}{P(X_1, \dots, X_n)}$$

- L'algorithme de classification naïve bayésienne suppose que toutes les V.A. sont indépendantes et que donc :

$$P(X_1, \dots, X_n | C_k) = \prod_{i=1}^n P(X_i | C_k)$$

# Classification naïve bayésienne

- Par exemple, pour classifier un email en SPAM, on doit théoriquement résoudre :

$$P(SPAM | \text{texte}) = \frac{P(\text{texte} | SPAM) * P(SPAM)}{P(\text{texte})}$$

- $P(SPAM)$  est évaluable,
- mais comment évaluer  $P(\text{texte} | SPAM)$  et  $P(\text{texte})$  ?

# Classification naïve bayésienne

- On s'affranchit de l'obligation d'estimer  $P(\text{texte})$  via la formule :

$$\frac{P(SPAM | \text{texte})}{P(GOOD | \text{texte})} = \frac{P(\text{texte} | SPAM) * P(SPAM)}{P(\text{texte} | GOOD) * P(GOOD)} > \text{threshold}$$

- Si nous fixons par exemple  $\text{threshold} = 2$  cela signifie que si la probabilité que nous ayons affaire à un SPAM avec ce texte est 2 fois supérieure à celle d'avoir un email normal, nous classifions l'email en SPAM.

# Classification naïve bayésienne

- Il ne nous reste plus qu'à estimer  $P(\text{texte} | SPAM)$  et  $P(\text{texte} | GOOD)$  .
- Pour ce faire nous utilisons l'algorithme de classification naïve bayésienne en supposant que tous les mots dans le texte sont indépendants (ce qui est bien sûr faux, d'où l'adjectif naïf).

$$P(\text{texte} | SPAM) = \prod_{i=1}^n P(\text{Mot}_i | SPAM)$$

Idem pour  $P(\text{texte} | GOOD)$ . On peut améliorer un peu la détection en tenant également en compte la position du mot dans l'email.

# Apprentissage supervisé

- Règle de Bayes
- Classification naïve bayésienne
- **Régression multivariée**
- Régression régularisée
- Protocole d'apprentissage
- Les k plus proches voisins
- Dilemme biais/variance
- Arbre de décision
- Bagging
- Forêt aléatoire
- Perceptron
- Perceptron multicouche
- Les réseaux de neurones
- Deep learning

# Régression linéaire multivariée

- Soit  $p$  V.A.  $X_1$  à  $X_p$ ,
- Nos données d'apprentissage sont composées de  $n$  échantillons avec une valeur (continue ou catégorisée) pour chaque V.A. et d'une valeur cible.
- On cherche à établir un modèle reliant les entrées aux sorties.



# Régression linéaire multivariée

$X_1$  à  $X_p$

Mass	Age	log(g)	ZAMS	$Y$
1.10±0.02	7.07±0.46	4.295	5677	$T_{\text{eff}}$
1.06±0.02	6.82±0.28	4.360	5629	
1.13±0.04	2.23±0.17	4.388	5985	
1.17±0.03	8.13±0.59	4.246	5642	
1.27±0.04	4.17±0.40	4.270	5905	
1.01±0.03	6.23±0.37	4.270	5838	
1.23±0.04	5.51±0.71	4.053	6064	
1.04±0.02	6.24±0.37	4.118	5933	
1.13±0.03	7.19±0.70	4.184	5801	
0.84±0.02	9.15±0.47	4.479	5253	
1.04±0.02	5.04±0.17	4.502	5165	
1.19±0.04	7.93±0.94	4.178	5695	
1.00±0.03	7.28±0.51	4.314	5718	
1.14±0.03	1.71±0.19	4.376	6092	
0.96±0.01	6.43±0.47	4.506	5307	
1.19±0.04	2.03±0.29	4.315	6138	
1.16±0.05	6.35±1.37	4.072	5943	
1.13±0.05	1.07±0.25	4.402	6001	
1.07±0.02	4.36±0.46	4.294	6063	
1.10±0.05	6.43±0.58	4.077	6023	
1.03±0.05	5.85±0.93	4.043	6094	

# Régression linéaire multivariée

- Supposons que notre modèle est de type linéaire (par rapport aux V.A.), il s'écrit :

$$\hat{Y} = \theta_0 * 1 + \theta_1 X_1 + \dots + \theta_p X_p$$

- On peut représenter nos données d'apprentissage par des vecteurs  $\theta$  et  $Y$  et matrice  $X$ :

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, \begin{bmatrix} 1 & a_{1,1} & \dots & a_{1,p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & a_{n,1} & \dots & a_{n,p} \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_p \end{bmatrix}$$

- L'objectif est donc de trouver  $\theta_i, \forall i = 0 \dots p$

# Régression linéaire multivariée

- Pour trouver les  $\theta_i$ , on va minimiser l'erreur globale du modèle.
- Pour représenter l'erreur entre une valeur obtenue par le modèle et la vraie valeur on utilise généralement la distance euclidienne :

$$\varepsilon_i = (\hat{y}_i - y_i)^2$$

- L'erreur globale du modèle est donc :

$$\varepsilon = \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

# Régression linéaire multivariée

- Pour pouvoir comparer des modèles avec un nombre d'échantillons différent, on moyenne :

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

- $J(\theta)$  est appelée :
  - **Fonction de coût du modèle**
  - C'est cette fonction que l'on va minimiser pour trouver les  $\theta_i$

# Régression linéaire multivariée

- L'algorithme permettant de minimiser cette fonction est appelé « **descente de gradient** » :

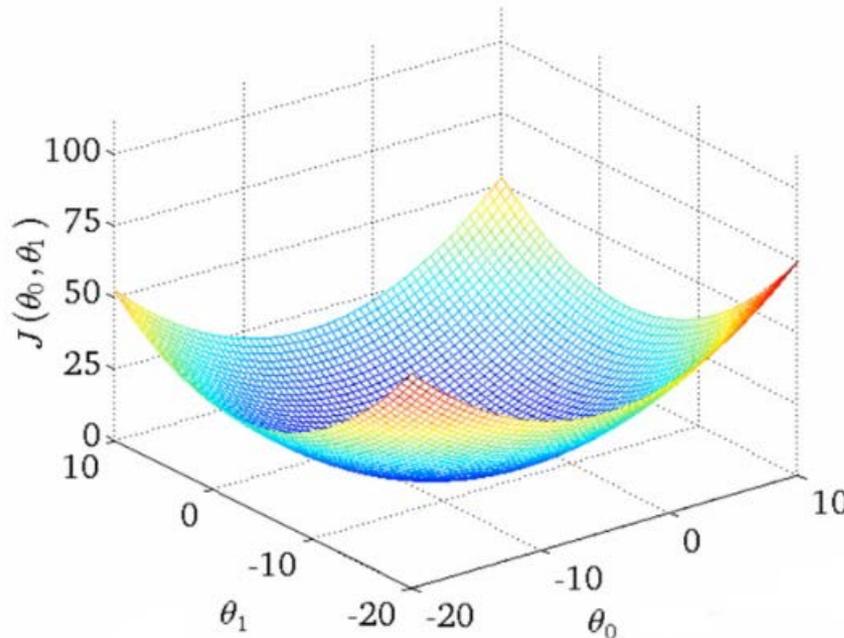
- On initialise tous les  $\theta_i$
- On itère jusqu'à convergence de tous les  $\theta_i$  :

$$\theta_i = \theta_i - \alpha * \frac{\delta}{\delta \theta_i} J(\theta), \text{ pour } j = 0 \dots p$$

- $\alpha$  est appelé le « **learning rate** »
  - Plus  $\alpha$  est grand plus le pas entre 2 itérations est grand et plus on va descendre rapidement la pente au risque de passer au dessus du minimum, voire de diverger.

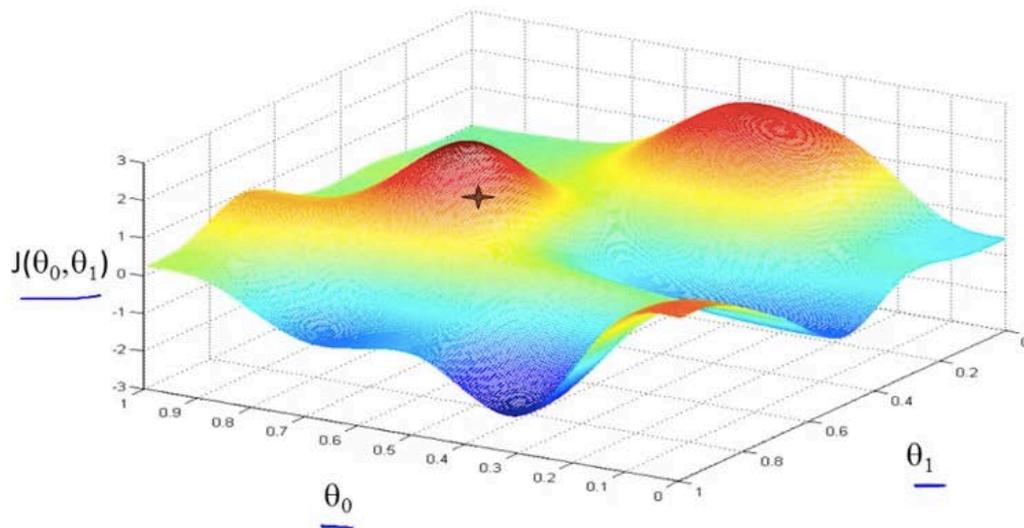
# Régression linéaire multivariée

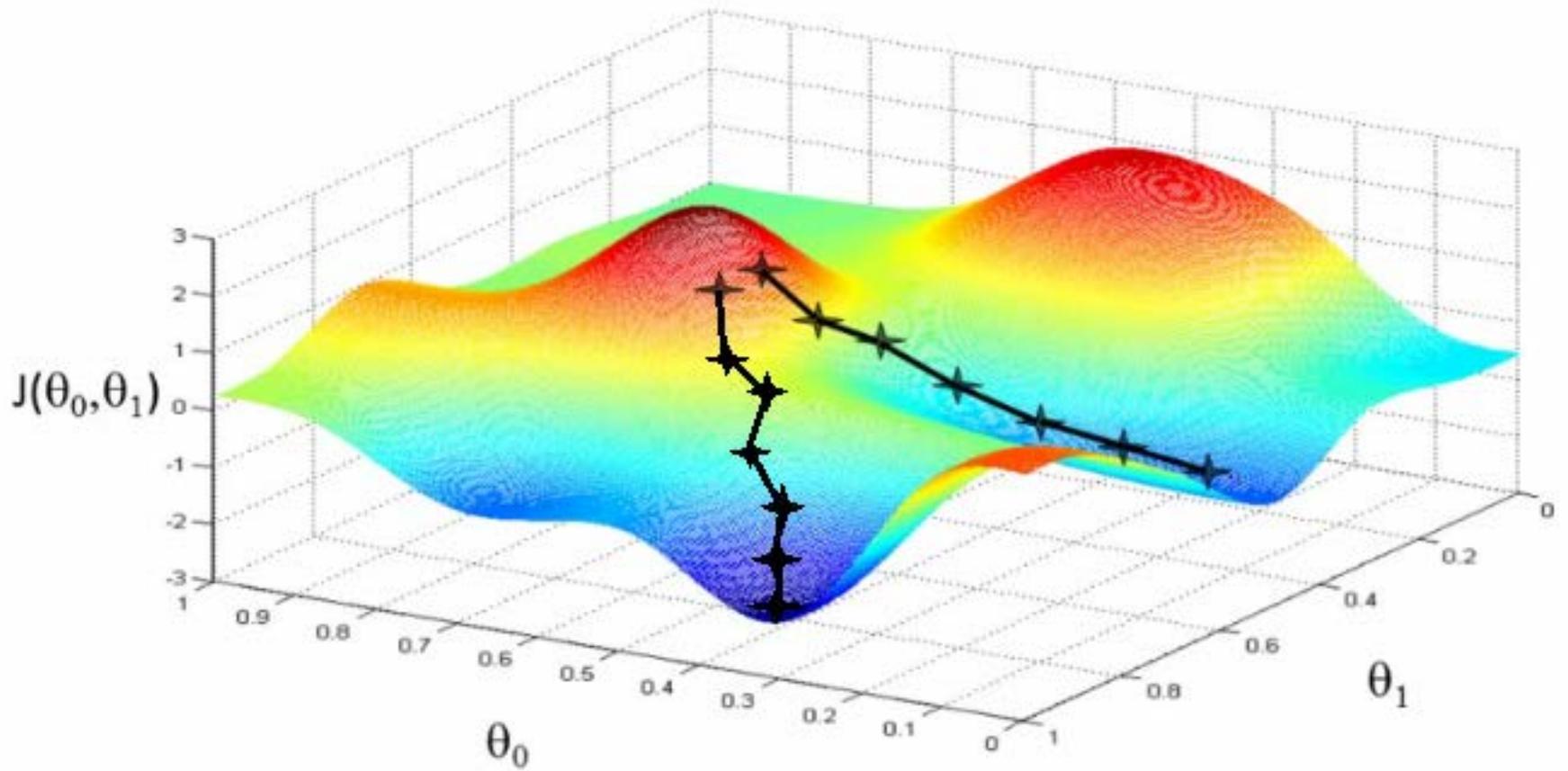
- Pour une régression univariée, on peut représenter cette descente par la recherche du minimum dans la fonction de cout  $J(\theta_0, \theta_1)$



# Régression linéaire multivariée

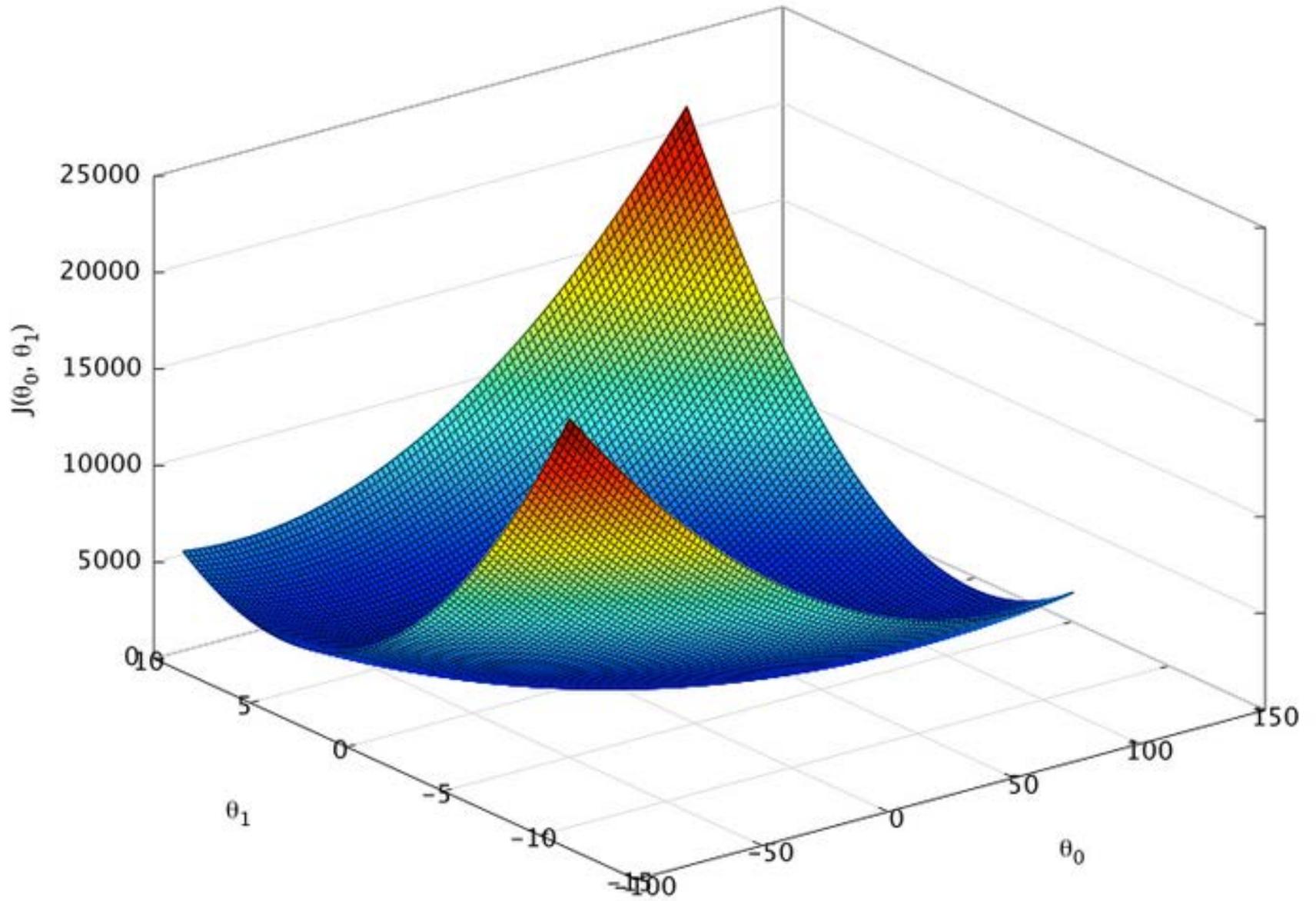
- **Problème 1** : Cette recherche suppose que l'espace des solutions est une fonction de coût **convexe**, sinon nous risquons de tomber dans un minimum local suite à une initialisation hasardeuse.





# Régression linéaire multivariée

- **Problème 2** : la fonction de coût peut présenter des arêtes ou des vallées, c'est-à-dire un grand nombre de solutions de minimisation possible.
- Ce phénomène fait qu'un petit changement dans les données d'entrées produit un modèle totalement différent.
- Notre modèle est **instable**.



# Apprentissage supervisé

- Règle de Bayes
- Classification naïve bayésienne
- Régression multivariée
- **Régression régularisée**
- Protocole d'apprentissage
- Les k plus proches voisins
- Dilemme biais/variance
- Arbre de décision
- Bagging
- Forêt aléatoire
- Perceptron
- Perceptron multicouche
- Les réseaux de neurones
- Deep learning

# Régression **régularisée**

- Ce problème est lié au fait que notre minimisation n'applique aucune contrainte sur les paramètres du modèle,  $\theta_i$
- Pour palier ce problème nous allons contraindre les  $\theta_i$  par une **régularisation**, c'est-à-dire l'ajout d'un terme dans notre fonction de coût.

# Régression **régularisée**

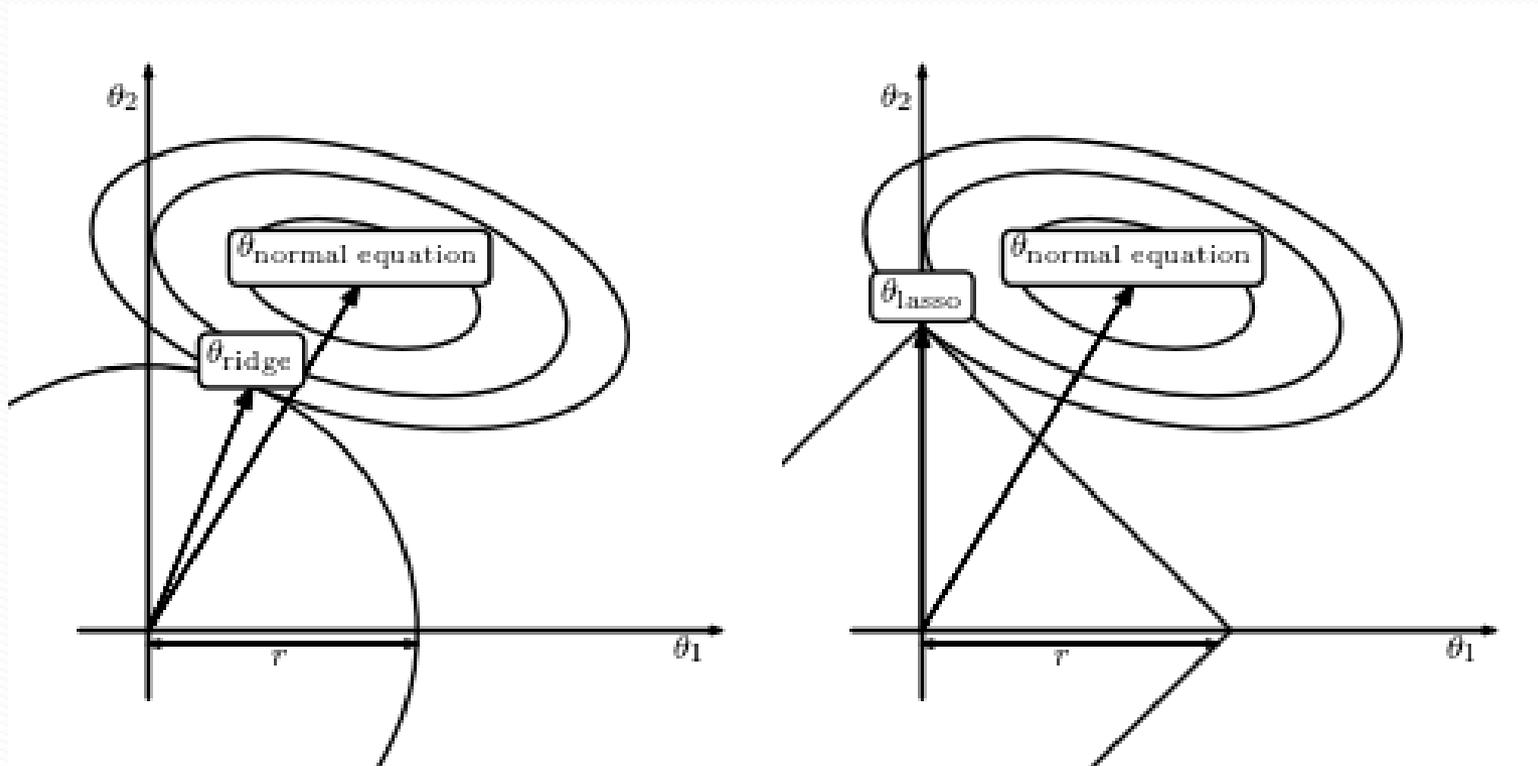
- Régression **Ridge** (performant si V.A. corrélée)

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 + \lambda \sum_{j=1}^p \theta_j^2$$

- Régression **LASSO** (peut annuler certains  $\theta_j$ )

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 + \lambda \sum_{j=1}^p |\theta_j|$$

# Régression **régularisée**



# Régression **régularisée**

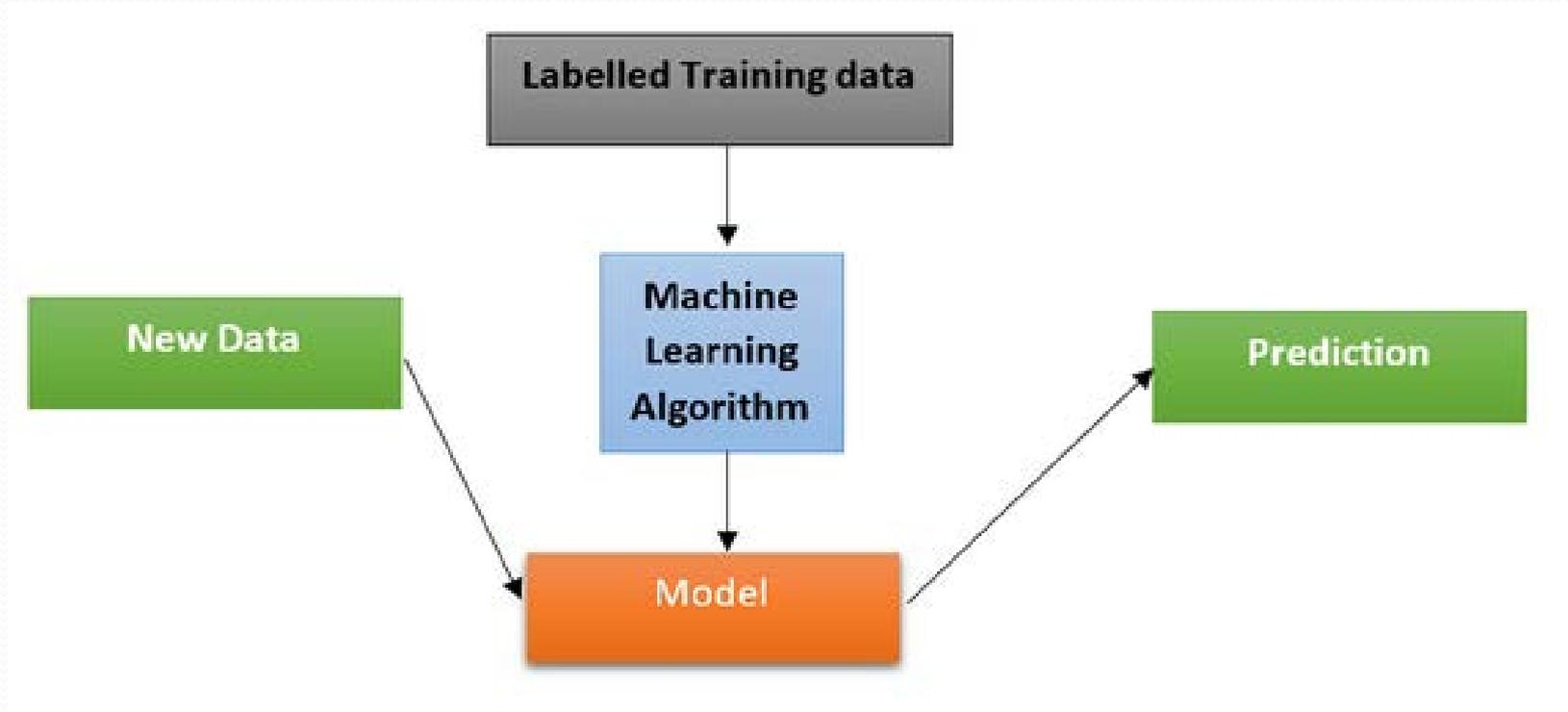
- Régression **ElasticNet** (Ridge + LESSO)
- $\alpha$  permet de moduler entre Ridge et LASSO, plus  $\alpha$  est grand plus on annulera de paramètre  $\theta$

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 + \lambda \sum_{j=1}^p \left[ \frac{1}{2} (1 - \alpha) \theta_j^2 + \alpha |\theta_j| \right]$$

# Apprentissage supervisé

- Règle de Bayes
- Classification naïve bayésienne
- Régression multivariée
- Régression régularisée
- **Protocole d'apprentissage**
- Les k plus proches voisins
- Dilemme biais/variance
- Arbre de décision
- Bagging
- Forêt aléatoire
- Perceptron
- Perceptron multicouche
- Les réseaux de neurones
- Deep learning

# Protocole d'apprentissage

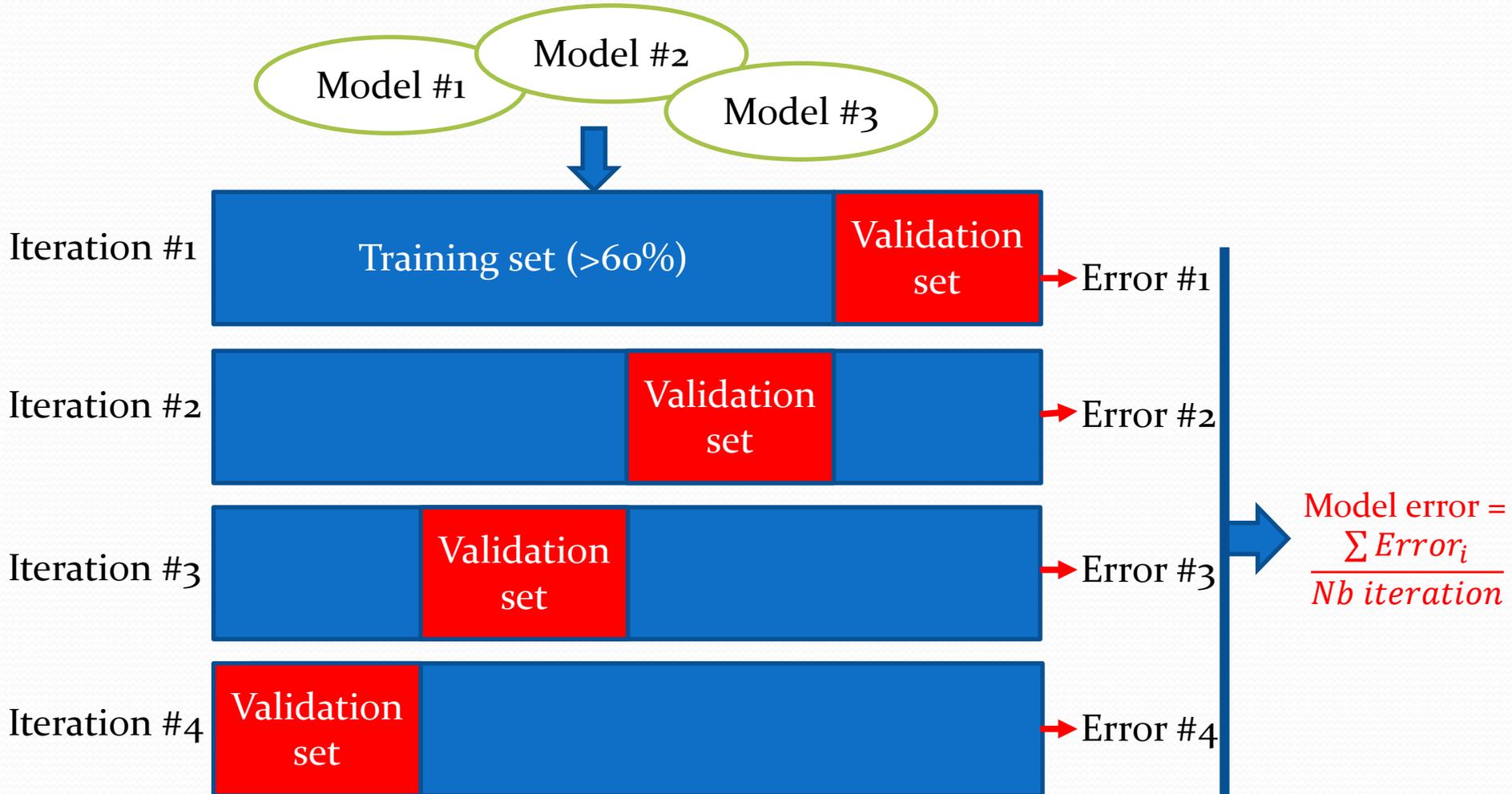


# Protocole d'apprentissage

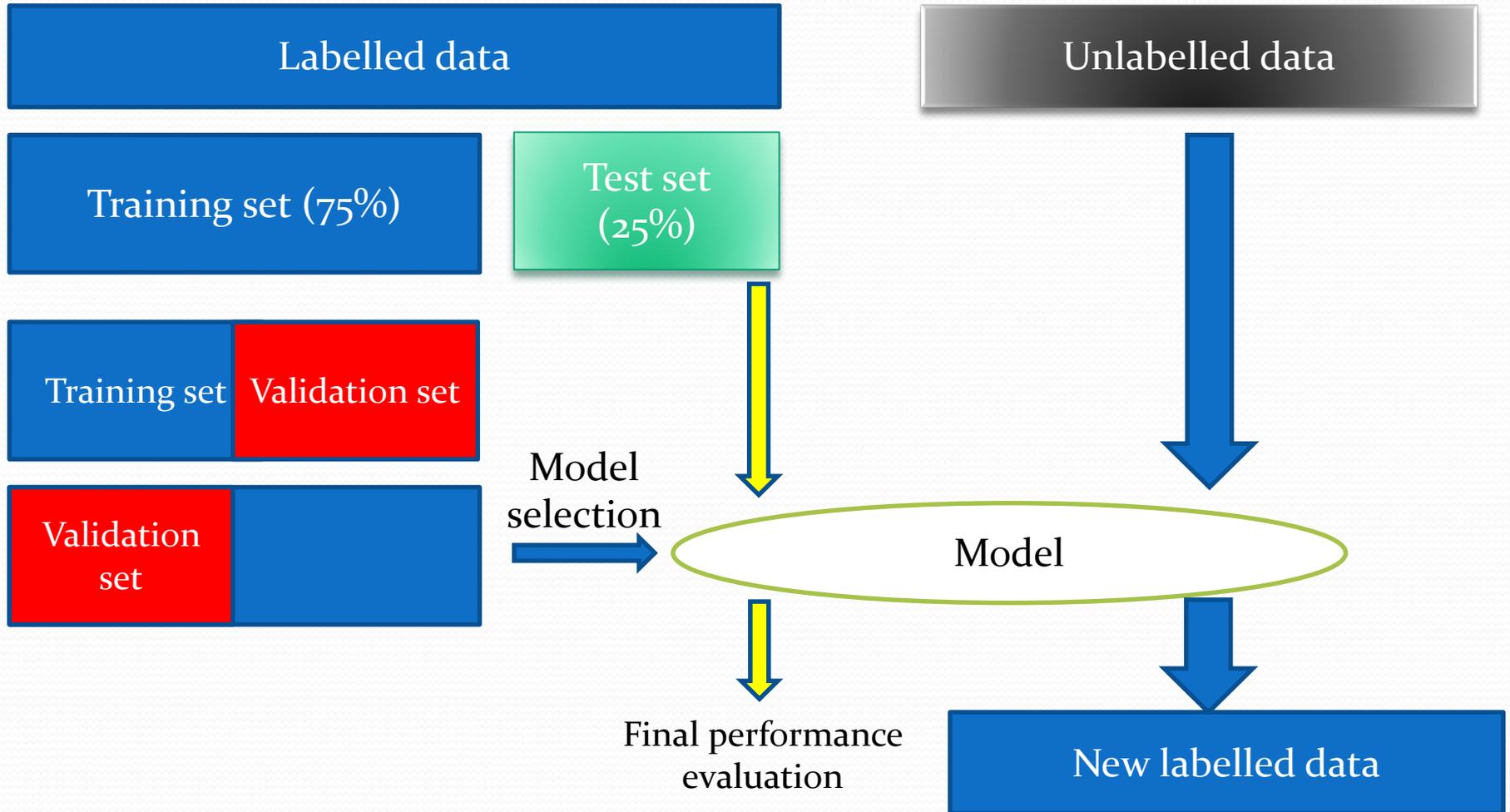


# Protocole d'apprentissage

## Validation croisée (k-fold cross validation)



# Protocole d'apprentissage



# Prétraitement des données

- Pour chaque paramètre , on normalise :

$$X_{norm} = \frac{X - \mu}{\sigma}$$

- Ou on standardise la valeur entre 0 et 1 :

$$X_{std} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

- Codification

#	Color
0	Red
1	Green
2	Blue
3	Red
4	Blue



#	Red	Green	Blue
0	1	0	0
1	0	1	0
2	0	0	1
3	1	0	0
4	0	0	1

# Apprentissage supervisé

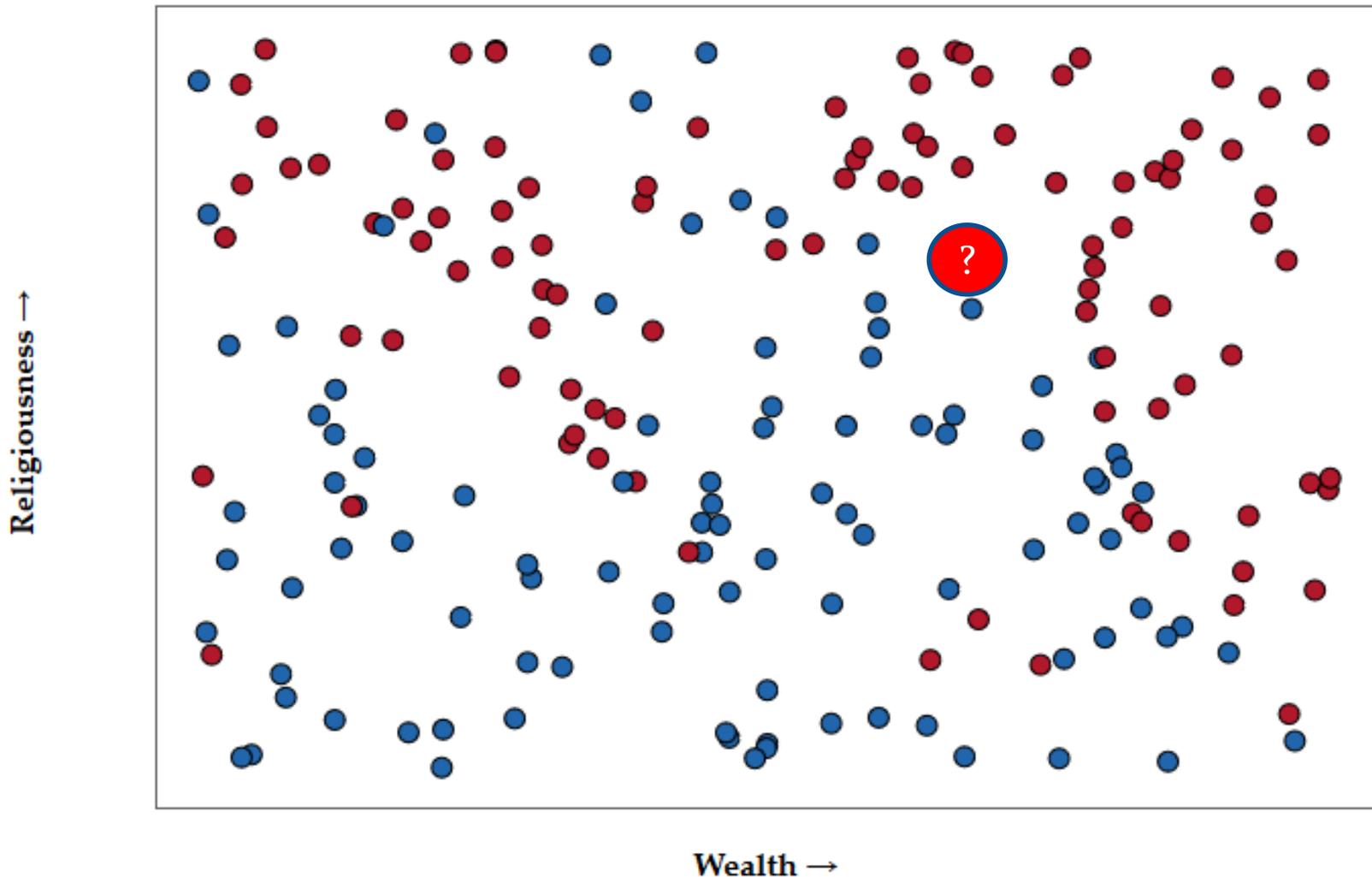
- Règle de Bayes
- Classification naïve bayésienne
- Régression multivariée
- Régression régularisée
- Protocole d'apprentissage
- **Les k plus proches voisins**
- Dilemme biais/variance
- Arbre de décision
- Bagging
- Forêt aléatoire
- Perceptron
- Perceptron multicouche
- Les réseaux de neurones
- Deep learning

# K plus proche voisin - KNN

- *Algorithme d'apprentissage supervisé pour la classification*
- KNN, de l'anglais *k-nearest neighbor*
- *Le modèle ne possède qu'un seul paramètre : le nombre  $k$  de voisins à prendre en compte.*

# K plus proche voisin – KNN

## Problématique



# K plus proche voisin – KNN

## Algorithme

- Soit  $m$  classes  $C_j$ ,  $j$  variant de 1 à  $m$
- Soit  $n$  points d'apprentissage nommés  $P_i$  appartenant chacun une classe  $C_j$ ,  $i$  variant de 1 à  $n$
- Soit un point cible nommé  $T$  de classe inconnue
- Pour chaque  $P_i$  :
  - Calculer la distance entre  $T$  et  $P_i$
- Pour les  $k$  points  $P_i$  les plus proches de  $T$ :
  - Compter le nombre d'occurrence de chaque classe
- Attribuer à  $T$  la classe la plus fréquente

# K plus proche voisin – KNN

## Distances

### Mesures de distance utilisées :

Soit 2 points X et Y de coordonnées respectives  $(x_1, \dots, x_n)$ ,  $(y_1, \dots, y_n)$  dans un espace de paramètres  $\mathbb{R}^n$

- **Distance euclidienne** (Pythagore)

$$d_{X,Y} = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Exemple : Si 2 paramètres en entrée  $x(x_1, x_2)$ ,  $y(y_1, y_2)$ , dans  $\mathbb{R}^2$

$$d_{X,Y} = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

# K plus proche voisin – KNN

## Distances

- **Distance de Manhattan :**

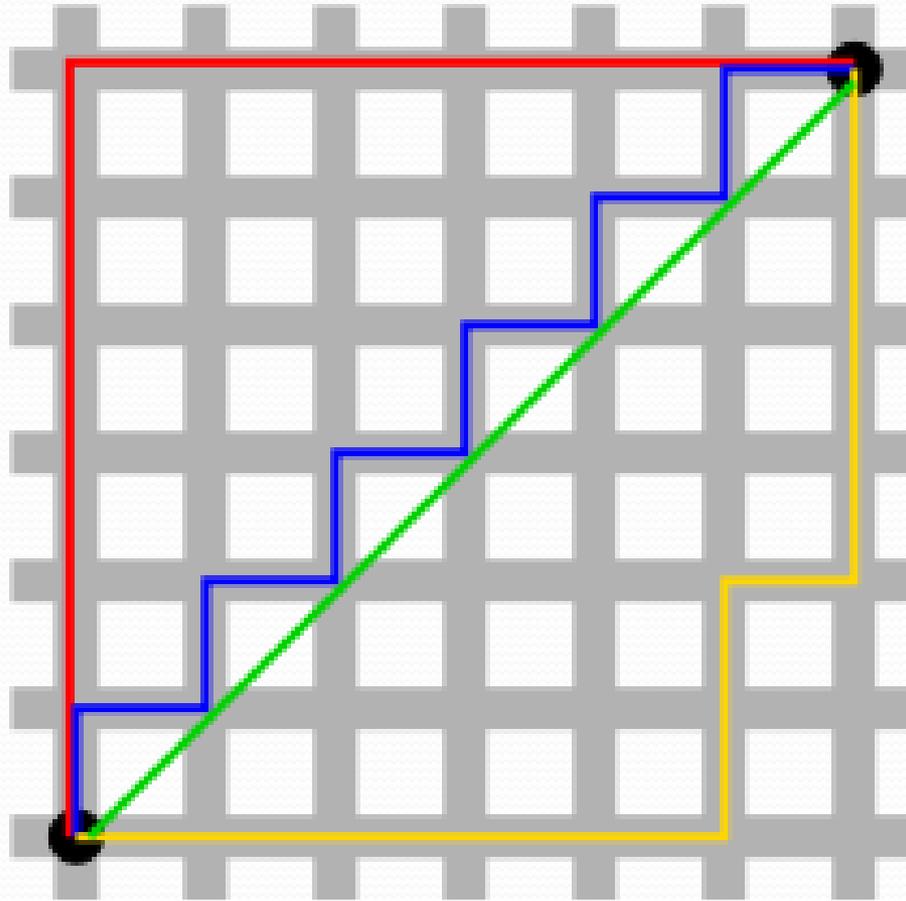
$$d_{X,Y} = \sum_{i=1}^n |x_i - y_i|$$

Exemple : Si 2 paramètres en entrée  $x(x_1, x_2)$ ,  $y(y_1, y_2)$ , dans  $\mathbb{R}^2$

$$d_{X,Y} = |x_1 - y_1| + |x_2 - y_2|$$

# K plus proche voisin – KNN

## Distances

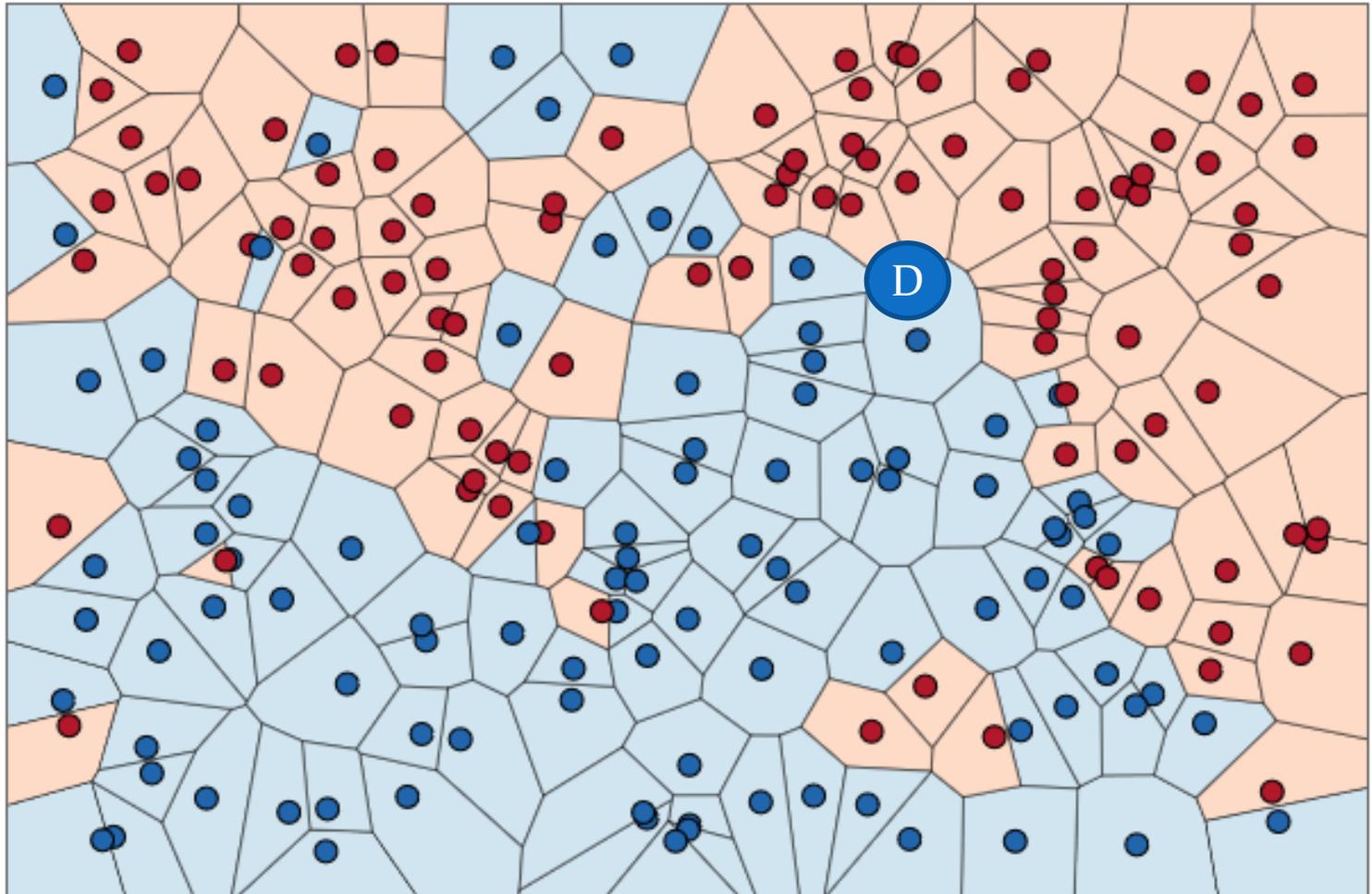


# K plus proche voisin – KNN

## Frontière de décision

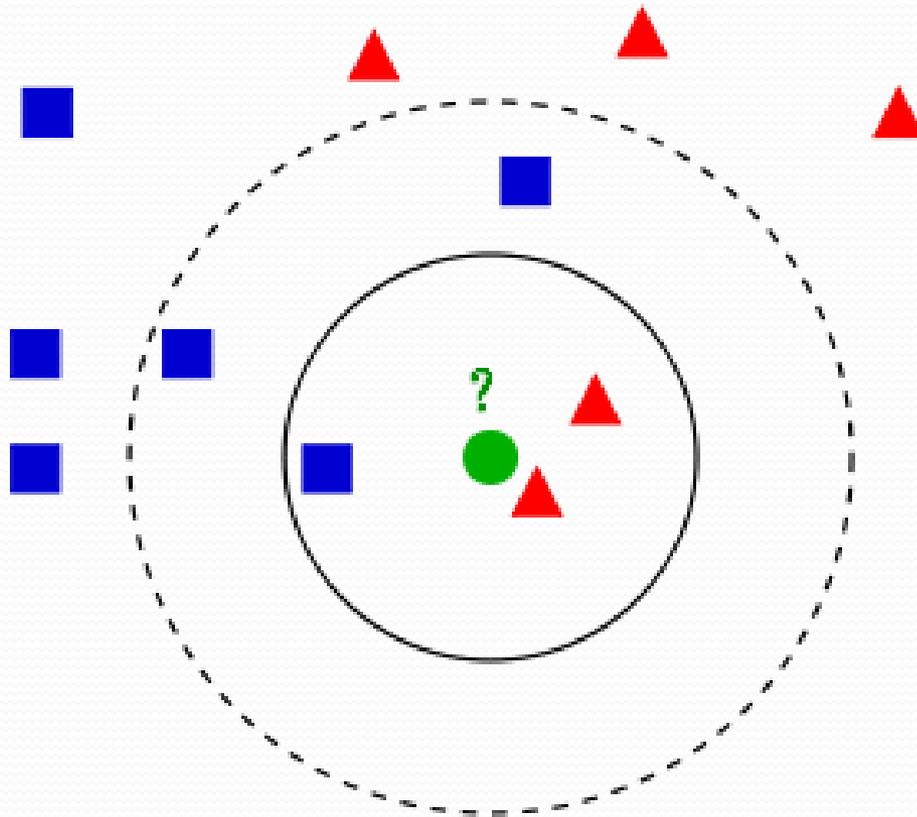
Decision boundaries

$K=1$



# K plus proche voisin – KNN

## Choix de k



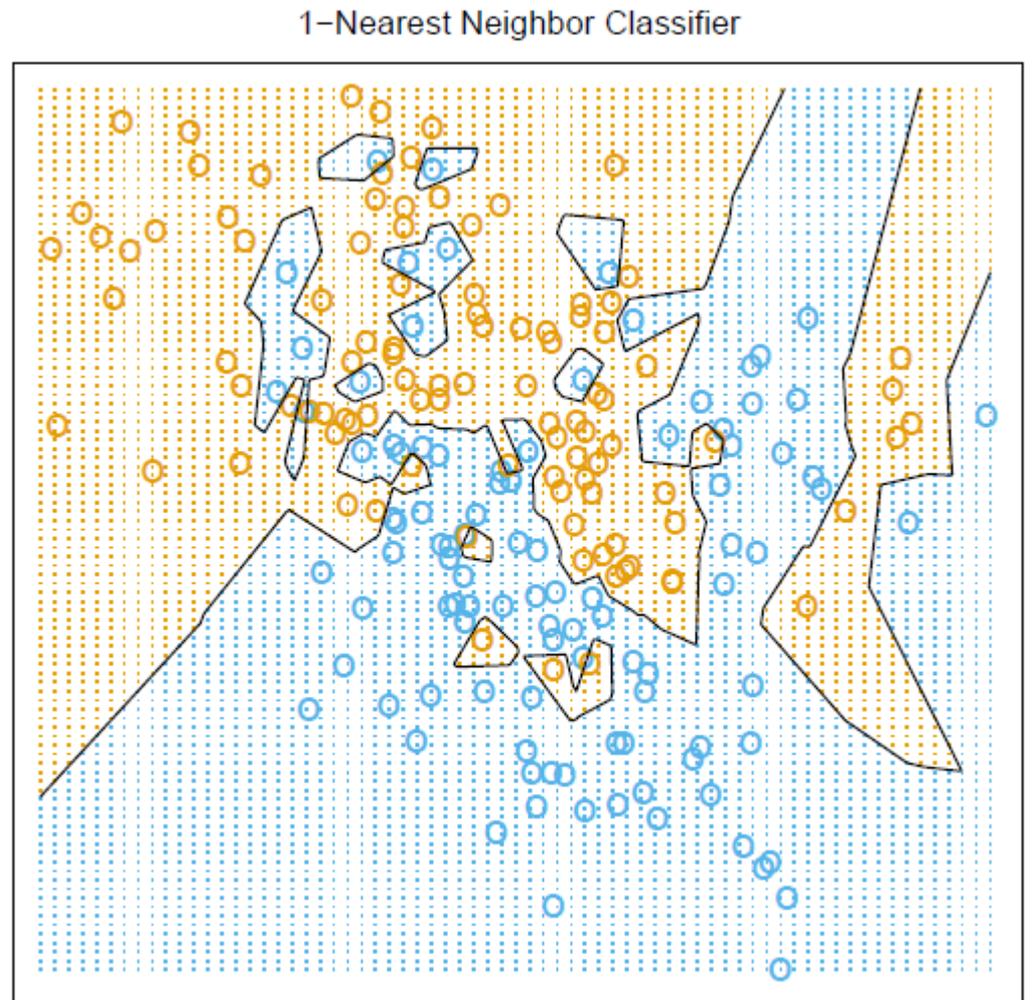
A quelle classe appartient le rond vert :

- si  $K = 3$  ?
- Si  $k = 5$  ?

# K plus proche voisin – KNN

## Choix de k

$K=1$

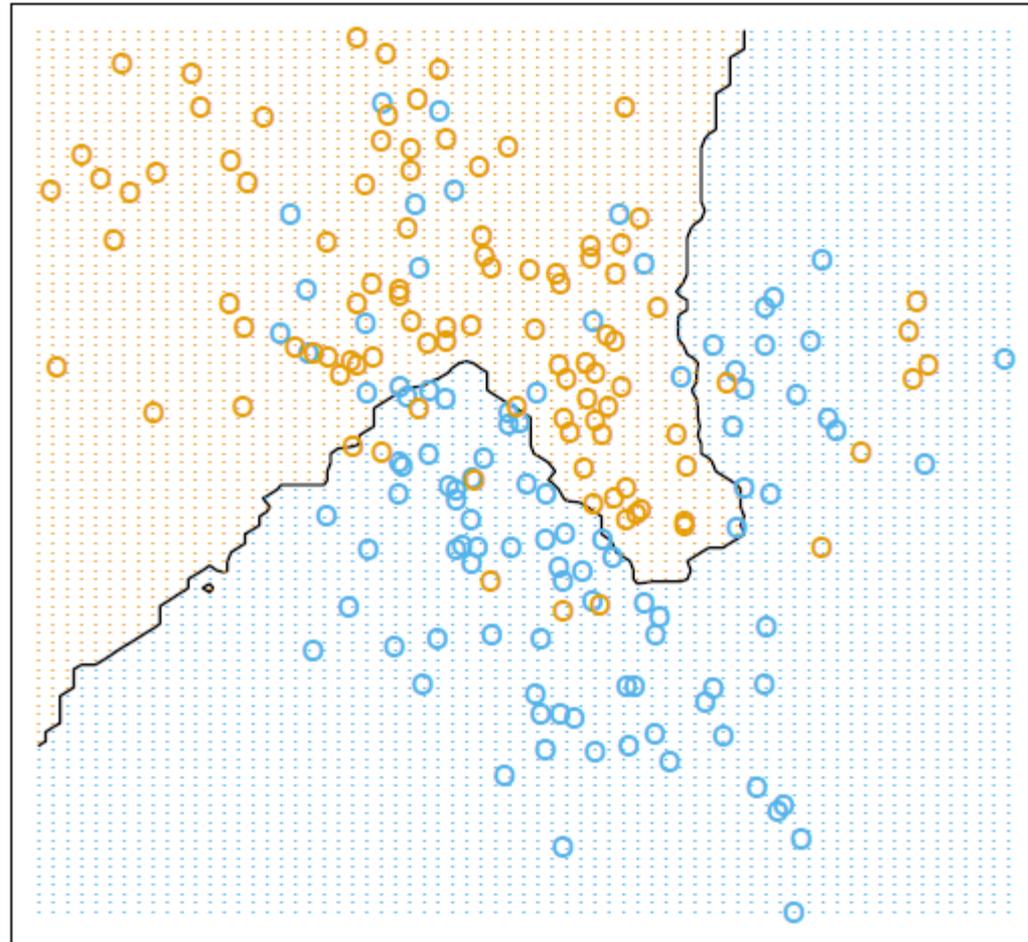


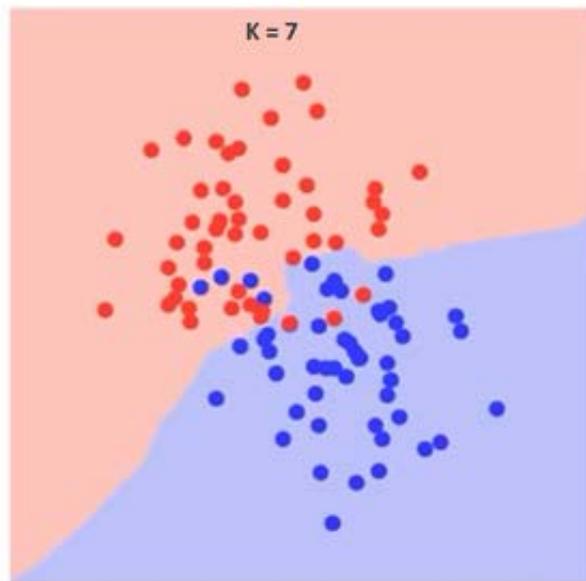
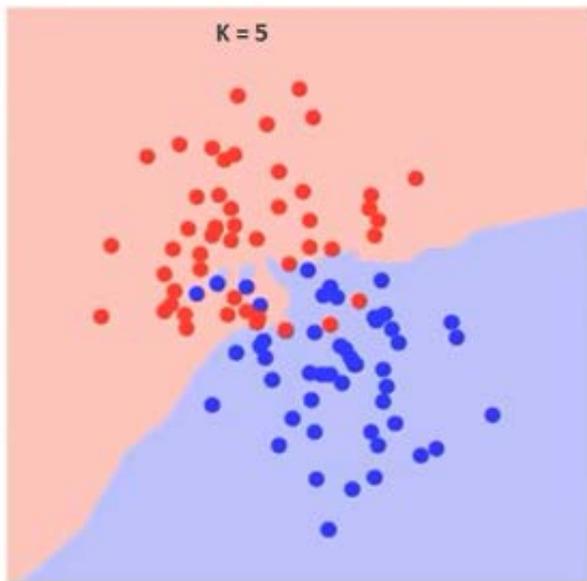
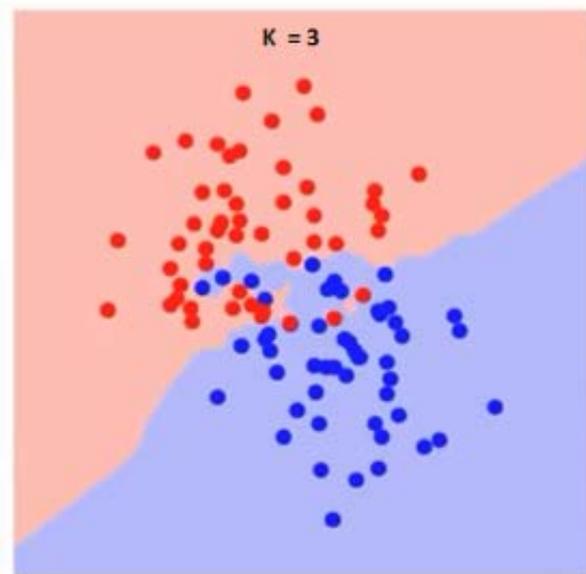
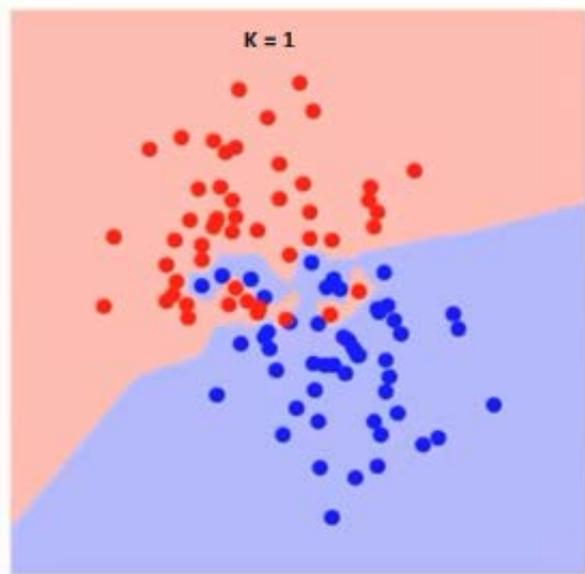
# K plus proche voisin – KNN

## Choix de k

K=15

15-Nearest Neighbor Classifier

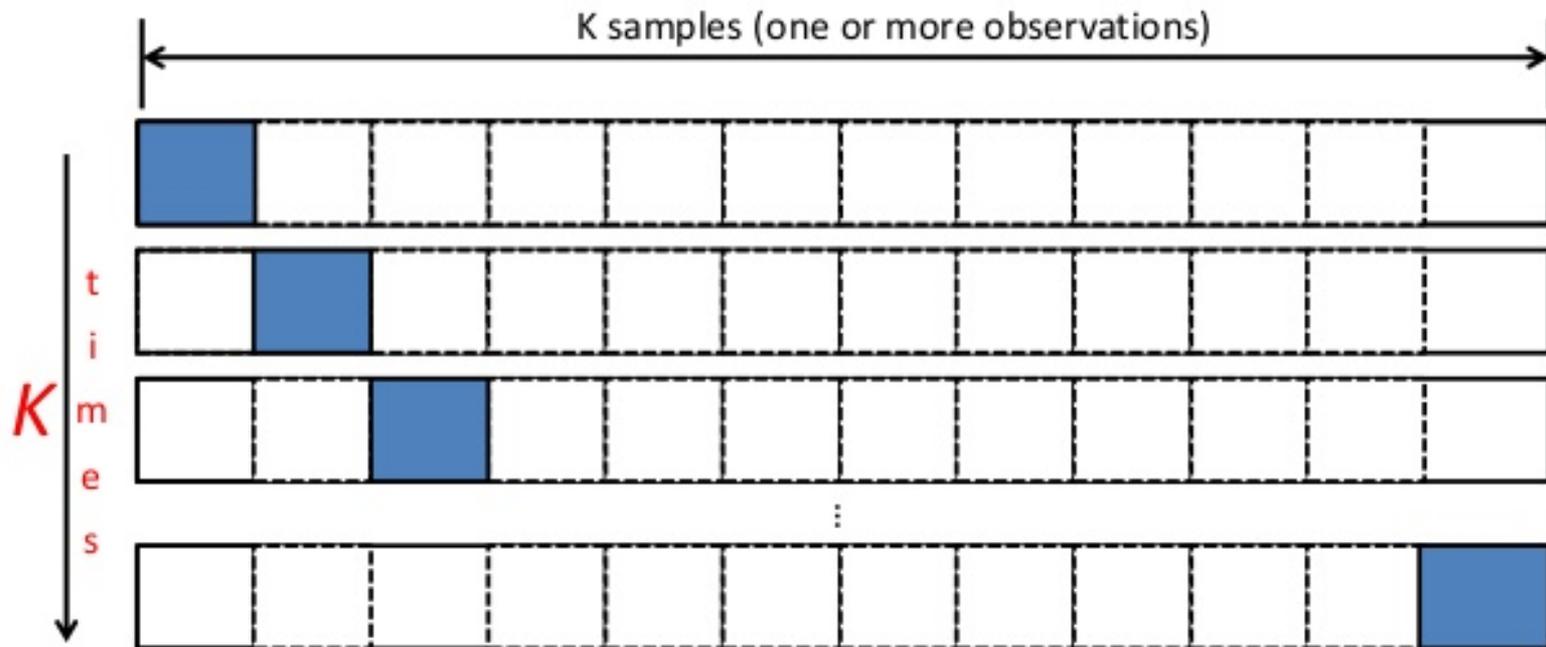




# K plus proche voisin – KNN

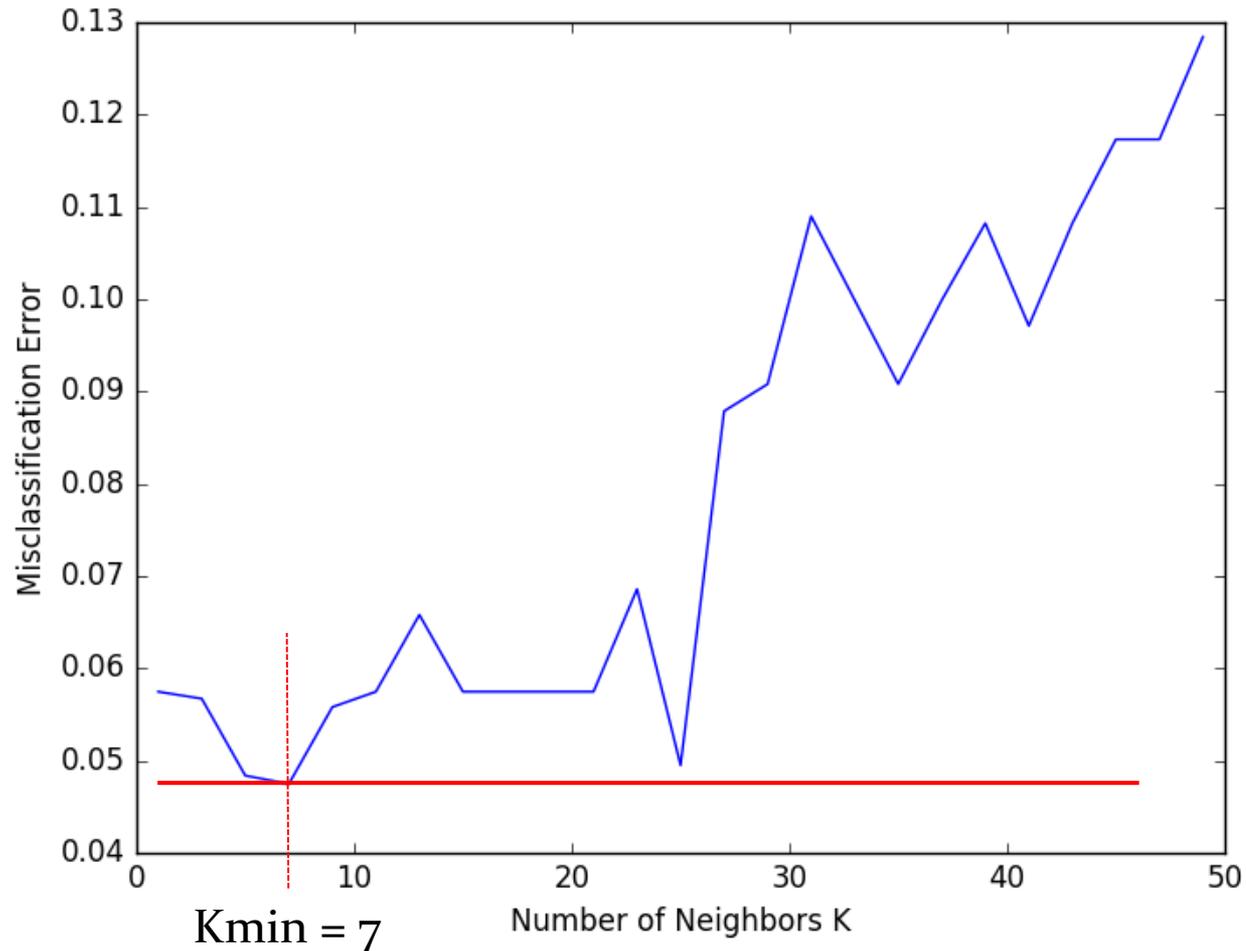
## Choix de k

- K-fold cross-validation:



# K plus proche voisin – KNN

## Choix de k



# K plus proche voisin – KNN

## Utilisation pour la régression

- Soit  $n$  points d'apprentissage nommés  $P_i$  possédant une valeur  $V$ ,  $i$  variant de 1 à  $n$
- Soit un point cible nommé  $T$  de valeur inconnue
- Pour chaque  $P_i$  :
  - Calculer la distance entre  $T$  et  $P_i$
- Pour les  $k$  points  $P_i$  les plus proches de  $T$ :
  - Faire une moyenne **pondérée par l'inverse de la distance des valeurs** des points  $P_i$
- Attribuer à  $T$  la classe la plus fréquente

# Apprentissage supervisé

- Règle de Bayes
- Classification naïve bayésienne
- Régression multivariée
- Régression régularisée
- Protocole d'apprentissage
- Les k plus proches voisins
- **Dilemme biais/variance**
- Arbre de décision
- Bagging
- Forêt aléatoire
- Perceptron
- Perceptron multicouche
- Les réseaux de neurones
- Deep learning

# Problématique

- Soit un ensemble d'apprentissage constitué d'un ensemble de points  $x_1, \dots, x_n$  et de valeurs réelles  $y_i$  associée à chaque point  $x_i$
- On suppose que la variable aléatoire  $Y$  est liée à la variable aléatoire  $X$  par la relation :

$$Y = f(X) + \varepsilon$$

En effet, la variable aléatoire  $Y$  possède sa propre variance autour de la fonction  $f(X)$ . On suppose que  $\varepsilon \sim N(0, \sigma)$ .

# Problématique

- $\hat{f}(x)$  est une fonction issue d'un algorithme de ML qui modélise le problème (c.à.d. qui modélise  $f(x)$ ) et qui généralise à des points extérieurs à l'ensemble d'apprentissage.
- $\hat{f}(x)$  est un prédicteur qui approxime la vraie fonction  $f(x)$

# Estimation de l'erreur du modèle

- La fonction d'estimation de l'erreur du modèle la plus couramment utilisée est l'erreur de prédiction quadratique moyenne (Mean Squared Prediction Error - MSPE).

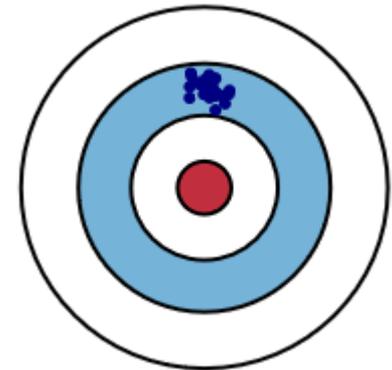
$$MSPE(x) = E[(Y - \hat{f}(x))^2]$$

- Objectif : minimiser une fonction de coût représentant l'erreur du modèle.
- Nous utilisons pour cela uniquement des données d'apprentissage. Cela ne couvre pas l'ensemble des possibilités.

# Dilemme/compromis biais-variance

- Lorsque l'on minimise la fonction de coût du modèle, on minimise en fait simultanément deux sources d'erreur :
- **Le Biais** : c'est l'erreur provenant d'hypothèses erronées dans l'algorithme d'apprentissage.

$$\text{Bias}[\hat{f}(x)] = E[\hat{f}(x)] - f(x)$$

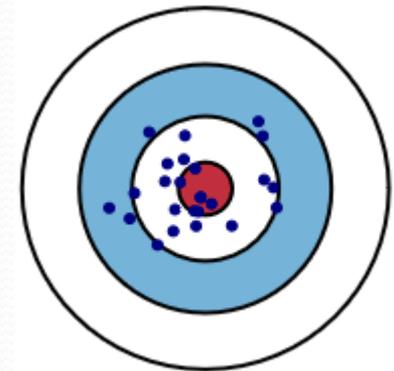


- Biais élevé => sous-apprentissage ou underfitting

# Dilemme/compromis biais-variance

- La **variance** : c'est l'erreur dû à la sensibilité de notre modèle aux petites fluctuations de l'échantillon d'apprentissage.

$$\text{Var}[\hat{f}(x)] = E \left[ (\hat{f}(x) - E[\hat{f}(x)])^2 \right]$$



- Variance élevée => sur-apprentissage ou overfitting

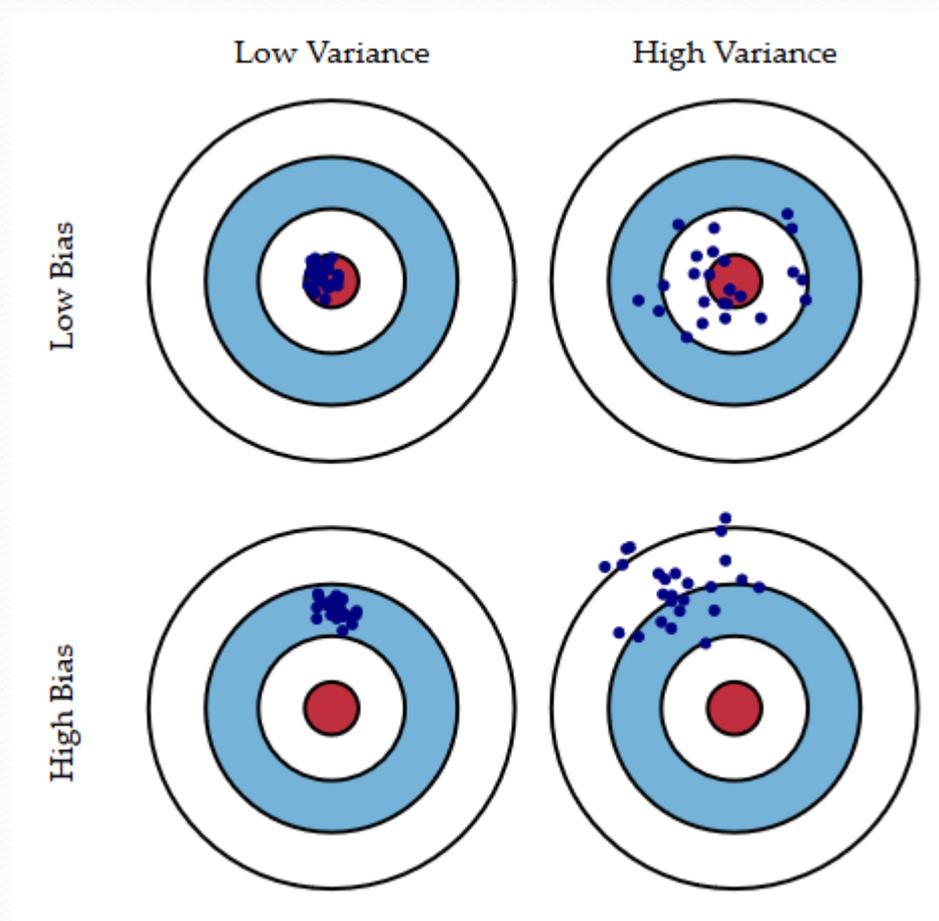
# Dilemme/compromis biais-variance

- On montre que :

$$MSPE(x) = (Bias[\hat{f}(x)])^2 + Var[\hat{f}(x)] + \sigma^2$$

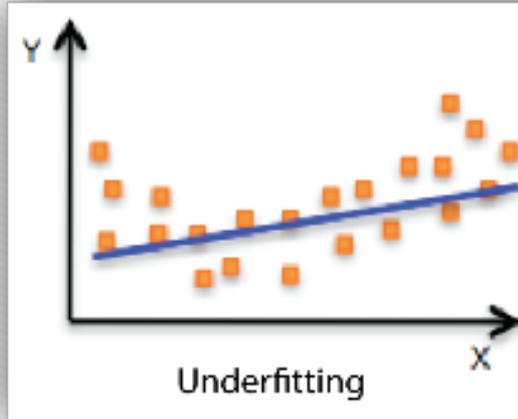
$\sigma^2$  est l'erreur irréductible provenant de la variabilité interne des données.

# Dilemme/compromis biais-variance



# Dilemme/compromis biais-variance

Régression linéaire



Biais élevé  
Variance faible

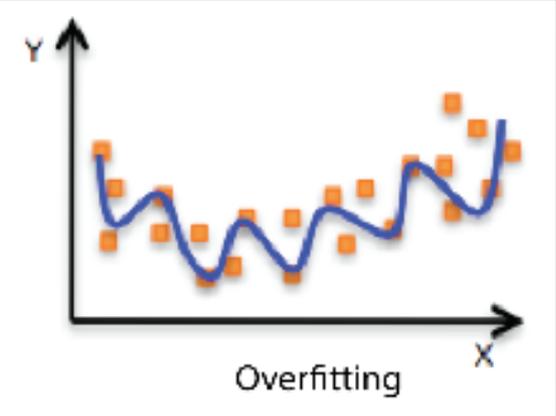


⇒ Faible capacité  
de représentation



⇒ Bonne capacité  
de généralisation

Régression  
polynomiale à  
nombreux degrés



Biais faible,  
Variance élevée

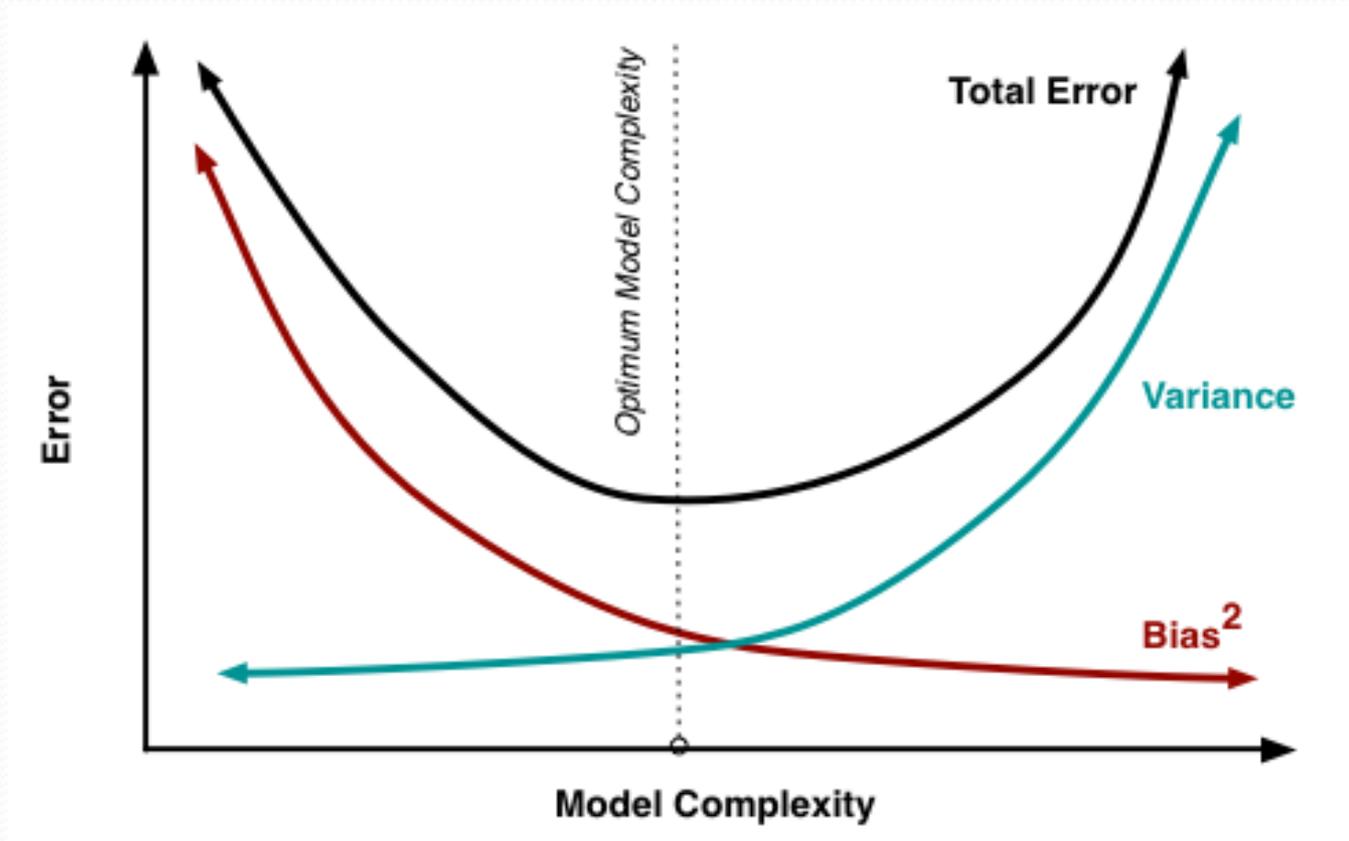


⇒ Bonne capacité  
de représentation



⇒ Mauvaise capacité  
de généralisation

# Dilemme/compromis biais-variance



# Dilemme/compromis biais-variance

- Dans le cas de l'algorithme knn, on montre que :

$$Err(x) = \left( f(x) - \frac{1}{k} \sum_{i=1}^k f(x_i) \right)^2 + \frac{\sigma^2}{k} + \sigma$$

$$(\mathbf{Bias}[\hat{f}(x)])^2 + \mathbf{Var}[\hat{f}(x)] + \sigma^2$$

Si k augmente => biais élevé, variance faible

# Dilemme/compromis biais-variance

- Objectif : choisir un modèle qui reflète avec précision les régularités dans les données d'apprentissage (biais faible), mais qui se généralise aussi aux futures données à prédire (variance faible).

# Dilemme/compromis biais-variance

- Intuitivement on aura souvent tendance à vouloir privilégier la réduction du biais, en complexifiant notre algorithme d'apprentissage.

**=> ERREUR car perte de la capacité de généralisation du modèle.**

# Apprentissage supervisé

- Règle de Bayes
- Classification naïve bayésienne
- Régression multivariée
- Régression régularisée
- Protocole d'apprentissage
- Les k plus proches voisins
- Dilemme biais/variance
- **Arbre de décision**
- Bagging
- Forêt aléatoire
- Perceptron
- Perceptron multicouche
- Les réseaux de neurones
- Deep learning

# Arbre de décision

- **L'apprentissage par arbre de décision** (Decision tree learning) est basé sur l'utilisation d'un arbre comme modèle prédictif.
- L'arbre de décision est construit grâce aux données d'apprentissage.

# Arbre de décision

- Il existe deux principaux types d'arbre de décision :
  - Les **arbres de classification** (*Classification Tree*) permettent de prédire à quelle classe la variable-cible appartient, dans ce cas la prédiction est une **étiquette de classe**.
  - Les **arbres de régression** (*Regression Tree*) permettent de prédire une quantité réelle (par exemple, le prix d'une maison ou la durée de séjour d'un patient dans un hôpital), dans ce cas la prédiction est **une valeur numérique**.

# Arbre de décision

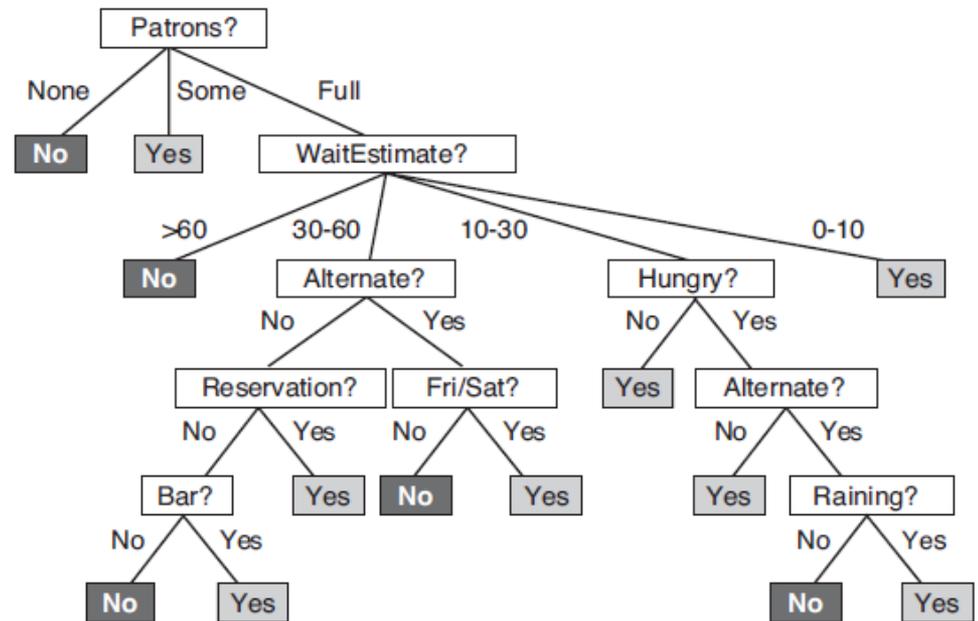
Exemple : from *Artificial Intelligence, a Modern Approach*, S. Russell.

Training set : 12 clients

Example	<b>X</b> Input Attributes										<b>Y</b> Goal
	<i>Alt</i>	<i>Bar</i>	<i>Fri</i>	<i>Hun</i>	<i>Pat</i>	<i>Price</i>	<i>Rain</i>	<i>Res</i>	<i>Type</i>	<i>Est</i>	<i>WillWait</i>
$x_1$	<i>Yes</i>	<i>No</i>	<i>No</i>	<i>Yes</i>	<i>Some</i>	<i>\$\$\$</i>	<i>No</i>	<i>Yes</i>	<i>French</i>	<i>0–10</i>	$y_1 = \text{Yes}$
$x_2$	<i>Yes</i>	<i>No</i>	<i>No</i>	<i>Yes</i>	<i>Full</i>	<i>\$</i>	<i>No</i>	<i>No</i>	<i>Thai</i>	<i>30–60</i>	$y_2 = \text{No}$
$x_3$	<i>No</i>	<i>Yes</i>	<i>No</i>	<i>No</i>	<i>Some</i>	<i>\$</i>	<i>No</i>	<i>No</i>	<i>Burger</i>	<i>0–10</i>	$y_3 = \text{Yes}$
$x_4$	<i>Yes</i>	<i>No</i>	<i>Yes</i>	<i>Yes</i>	<i>Full</i>	<i>\$</i>	<i>Yes</i>	<i>No</i>	<i>Thai</i>	<i>10–30</i>	$y_4 = \text{Yes}$
$x_5$	<i>Yes</i>	<i>No</i>	<i>Yes</i>	<i>No</i>	<i>Full</i>	<i>\$\$\$</i>	<i>No</i>	<i>Yes</i>	<i>French</i>	<i>&gt;60</i>	$y_5 = \text{No}$
$x_6$	<i>No</i>	<i>Yes</i>	<i>No</i>	<i>Yes</i>	<i>Some</i>	<i>\$\$</i>	<i>Yes</i>	<i>Yes</i>	<i>Italian</i>	<i>0–10</i>	$y_6 = \text{Yes}$
$x_7$	<i>No</i>	<i>Yes</i>	<i>No</i>	<i>No</i>	<i>None</i>	<i>\$</i>	<i>Yes</i>	<i>No</i>	<i>Burger</i>	<i>0–10</i>	$y_7 = \text{No}$
$x_8$	<i>No</i>	<i>No</i>	<i>No</i>	<i>Yes</i>	<i>Some</i>	<i>\$\$</i>	<i>Yes</i>	<i>Yes</i>	<i>Thai</i>	<i>0–10</i>	$y_8 = \text{Yes}$
$x_9$	<i>No</i>	<i>Yes</i>	<i>Yes</i>	<i>No</i>	<i>Full</i>	<i>\$</i>	<i>Yes</i>	<i>No</i>	<i>Burger</i>	<i>&gt;60</i>	$y_9 = \text{No}$
$x_{10}$	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>	<i>Full</i>	<i>\$\$\$</i>	<i>No</i>	<i>Yes</i>	<i>Italian</i>	<i>10–30</i>	$y_{10} = \text{No}$
$x_{11}$	<i>No</i>	<i>No</i>	<i>No</i>	<i>No</i>	<i>None</i>	<i>\$</i>	<i>No</i>	<i>No</i>	<i>Thai</i>	<i>0–10</i>	$y_{11} = \text{No}$
$x_{12}$	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>	<i>Full</i>	<i>\$</i>	<i>No</i>	<i>No</i>	<i>Burger</i>	<i>30–60</i>	$y_{12} = \text{Yes}$

# Arbre de décision

- Les nœuds correspondent à un test sur un attribut unique.
- Les branches correspondent à une modalité de l'attribut.
- Les feuilles représentent les valeurs de la variable-cible.
- Le parcours d'un chemin qui va de la racine à une feuille correspond à une combinaison de modalité d'attribut menant à une valeur cible.

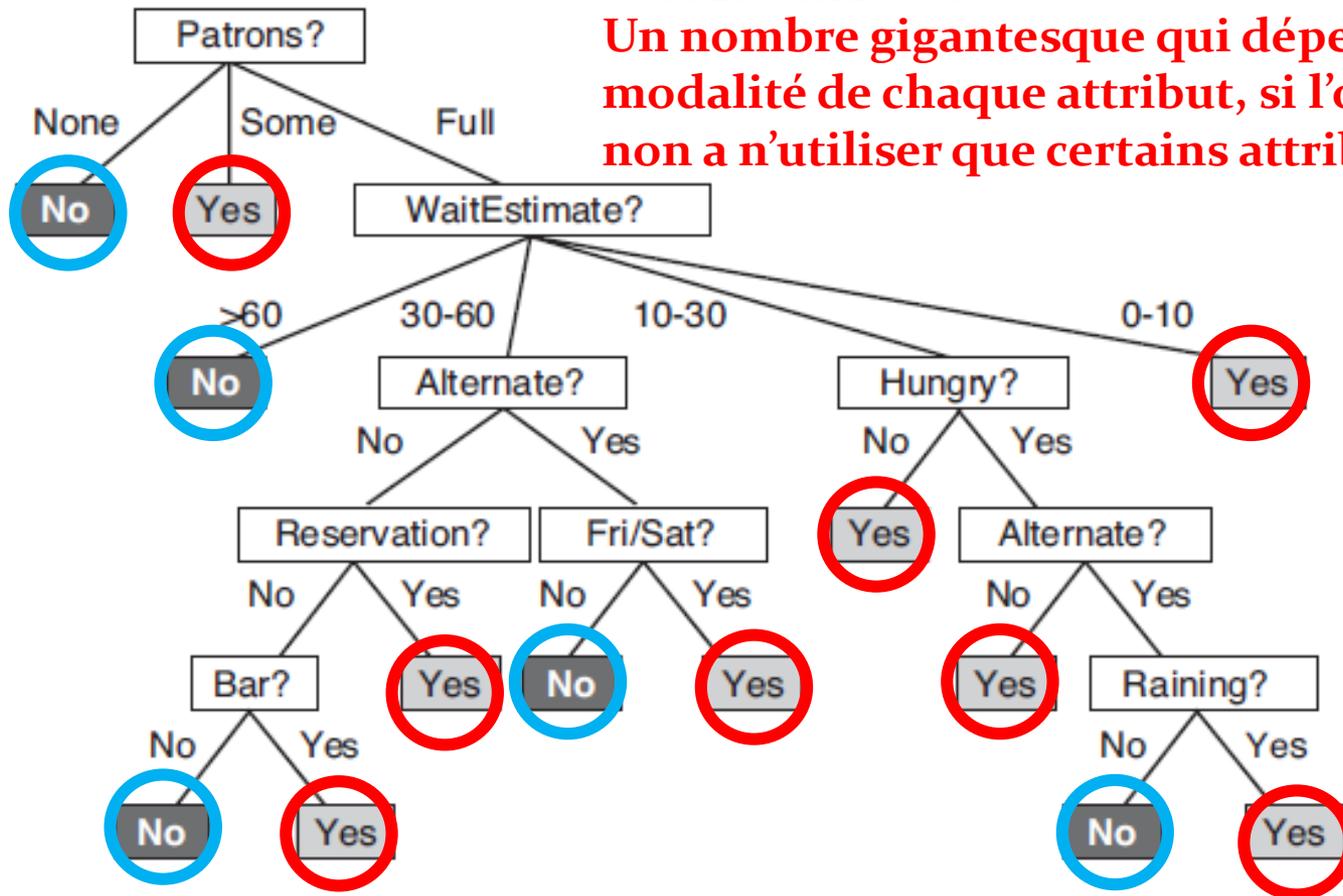


Arbre de décision : **Les clients vont-ils attendre pour une table ?**

# Arbre de décision

Combien d'arbres différents peut-on construire avec 10 attributs ?

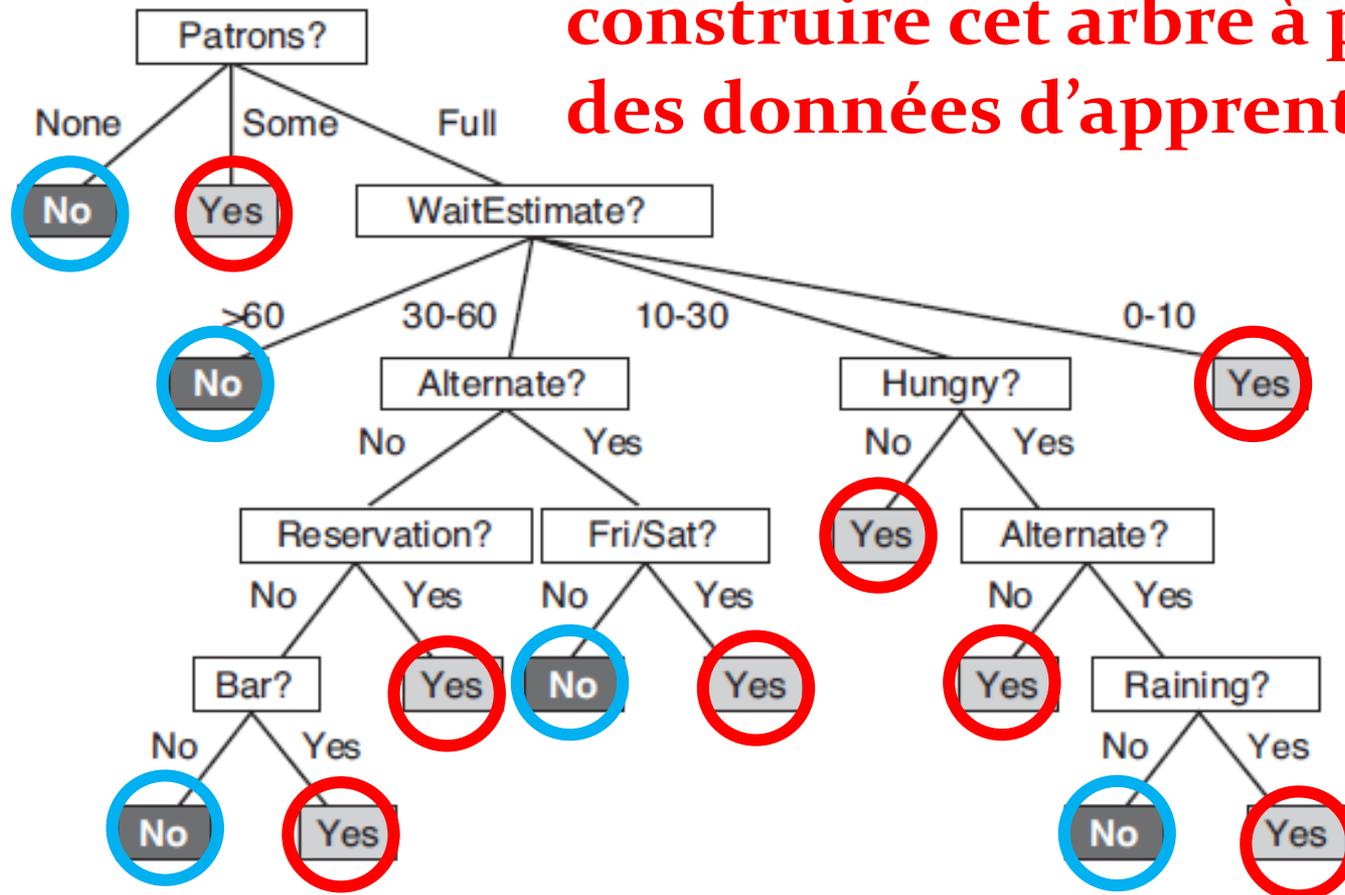
Un nombre gigantesque qui dépend du nombre de modalité de chaque attribut, si l'on s'autorise ou non à n'utiliser que certains attributs.



Arbre de décision : Les clients vont-ils attendre pour une table ?

# Arbre de décision

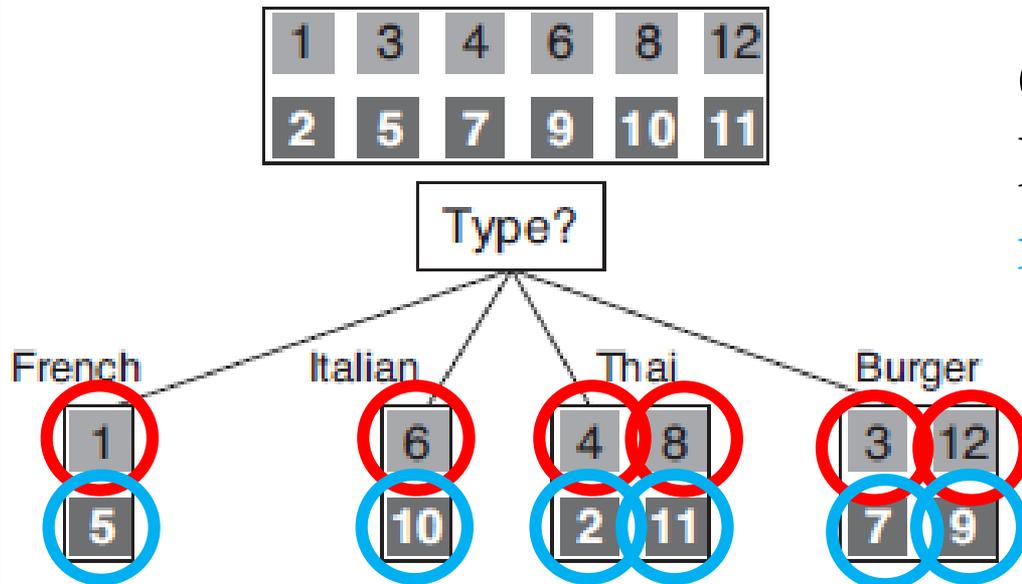
**Il nous faut une stratégie pour construire cet arbre à partir des données d'apprentissage.**



Arbre de décision : **Les clients vont-ils attendre pour une table ?**

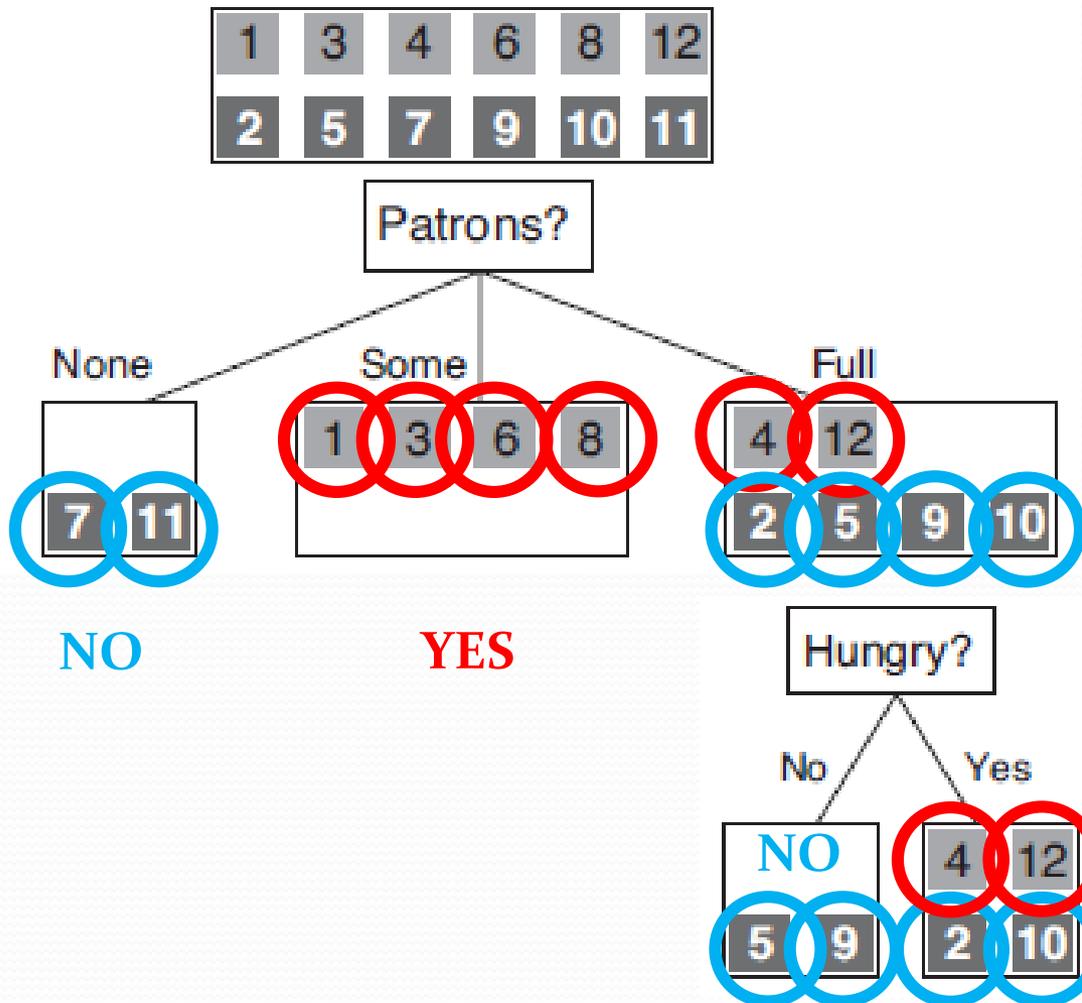
# Arbre de décision

- Objectif : Utiliser en premier les attributs les plus importants c'est-à-dire ceux qui découpent de manière la plus nette l'échantillon d'apprentissage au regard de la valeur cible.



Que pensez-vous de l'attribut **Type de restaurant** ?

# Arbre de décision



Que pensez-vous de l'attribut **Patrons** (Nombre de clients dans le restaurant)?

Que pensez-vous de l'attribut **Patrons** suivi de l'attribut **Hungry**?

# Arbre de décision

- Algorithme **DECISION-TREE-LEARNING**
  1. Sélectionner l'attribut le plus important
  2. **Pour chaque modalité** de l'Attribut sélectionné
    1. Appeler récursivement **DECISION-TREE-LEARNING**

Algorithme glouton (Greedy Algorithm). Ce n'est pas forcément le meilleur arbre car le problème de construction de l'arbre est NP complet.

# Arbre de décision

**DECISION-TREE-LEARNING**(exemples, attributs, exemples\_parents)

Si les exemples ont tous la même classification retourner **classification**

AttributSelectionné := argmax(**IMPORTANT**(attributs, exemples))

Arbre := nouvelle arbre de racine « AttributSelectionné »

**Pour chaque modalité** de AttributSelectionné faire:

ExemplesValeur := {exemples.AttributSelectionné = modalité}

SousArbre := **DECISION-TREE-LEARNING**(ExemplesValeur,  
Attributs – AttributSelectionné, exemples)

Ajouter une branche à l'arbre avec label (Attribut = modalité)

Ajouter un sous arbre à la branche := SousArbre

**Retourner** Arbre

# Arbre de décision

- Choix de l'attribut le plus important :
  - Choisir l'attribut qui réduit le plus l'entropie (donc l'incertitude) de la décision. Entropie au sens de Shannon.
- L'entropie d'une variable aléatoire pouvant prendre  $k$  distinctes modalités  $v_k$  de probabilité  $P(v_k)$  vaut :

$$\text{Entropie} = H = - \sum_k P(v_k) * \log_2 P(v_k)$$

- L'entropie d'une variable aléatoire booléenne dont la probabilité d'être vraie vaut  $q$  :

$$B(q) = -[ q * \log_2(q) + (1 - q) * \log_2(1 - q) ]$$

# Arbre de décision

- Dans notre exemple, la variable cible est une variable booléenne. Notre jeu d'apprentissage comporte 6 valeurs YES et 6 valeurs NO,  $q=0.5$ , l'entropie de notre variable cible est donc de **1 bit** :

```
In [4]: - (.5*log2 (.5) + (1-.5)*log2 (1-.5) )  
Out [4]: 1.0
```

- L'objectif est donc de choisir l'attribut qui réduit le plus possible cette entropie.

# Arbre de décision

- On montre que la réduction d'entropie, ou **gain d'information par l'attribut A**, par un test sur l'attribut A prenant d modalités distinctes vaut :

$$\text{Gain}(A) = B\left(\frac{p}{n+p}\right) - \sum_{k=1}^d \frac{p_k + n_k}{p+n} B\left(\frac{p_k}{p_k + n_k}\right)$$

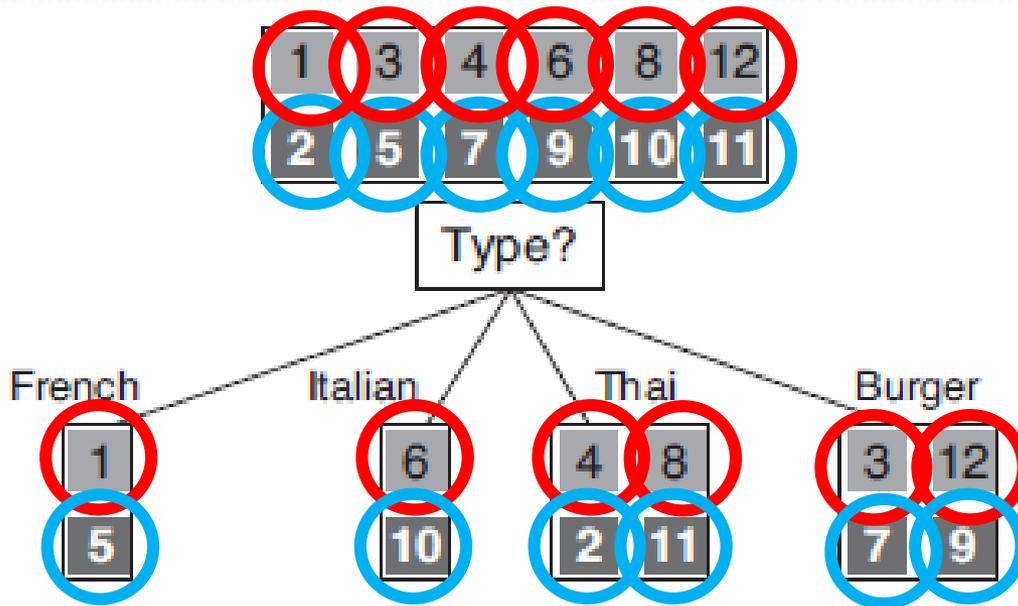
$p$  := nombre de valeurs YES dans le parent

$n$  := nombre de valeurs NO dans le parent

$p_k$  := nombre de valeurs YES dans le fils k

$n_k$  := nombre de valeurs NO dans le fils k

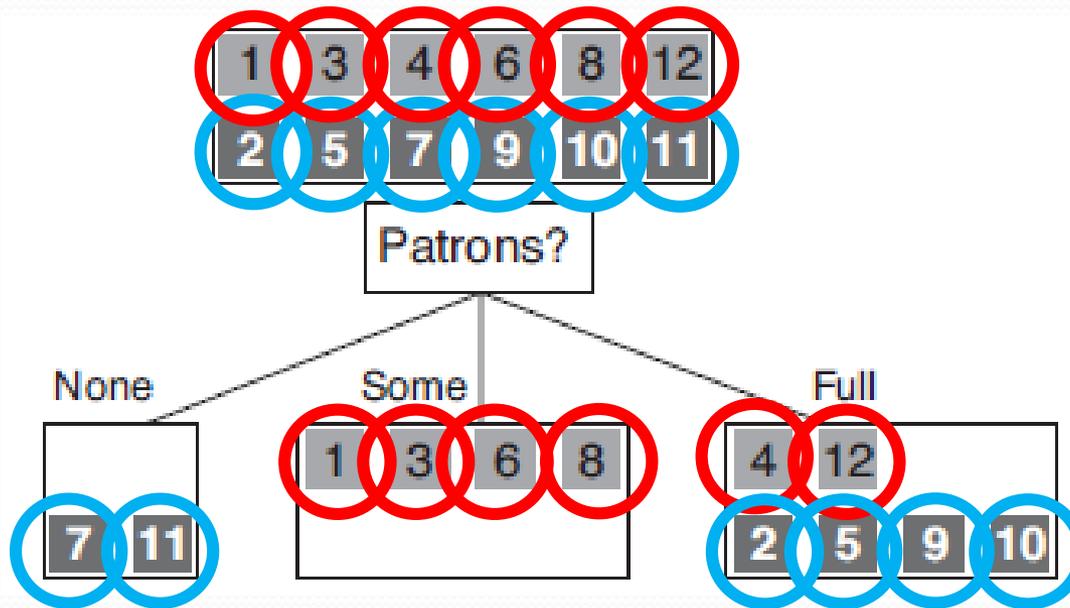
# Arbre de décision



$$Gain(\mathbf{Type}) = 1 - \left[ \frac{2}{12} B\left(\frac{1}{2}\right) + \frac{2}{12} B\left(\frac{1}{2}\right) + \frac{4}{12} B\left(\frac{2}{4}\right) + \frac{4}{12} B\left(\frac{2}{4}\right) \right] = \mathbf{0bits}$$

```
In [51]: B(6/12.) - (2/12.*B(1/2.)+2/12.*B(1/2.)+4/12.*B(2/4.)+4/12.*B(2/4.))
Out[51]: 0.0
```

# Arbre de décision



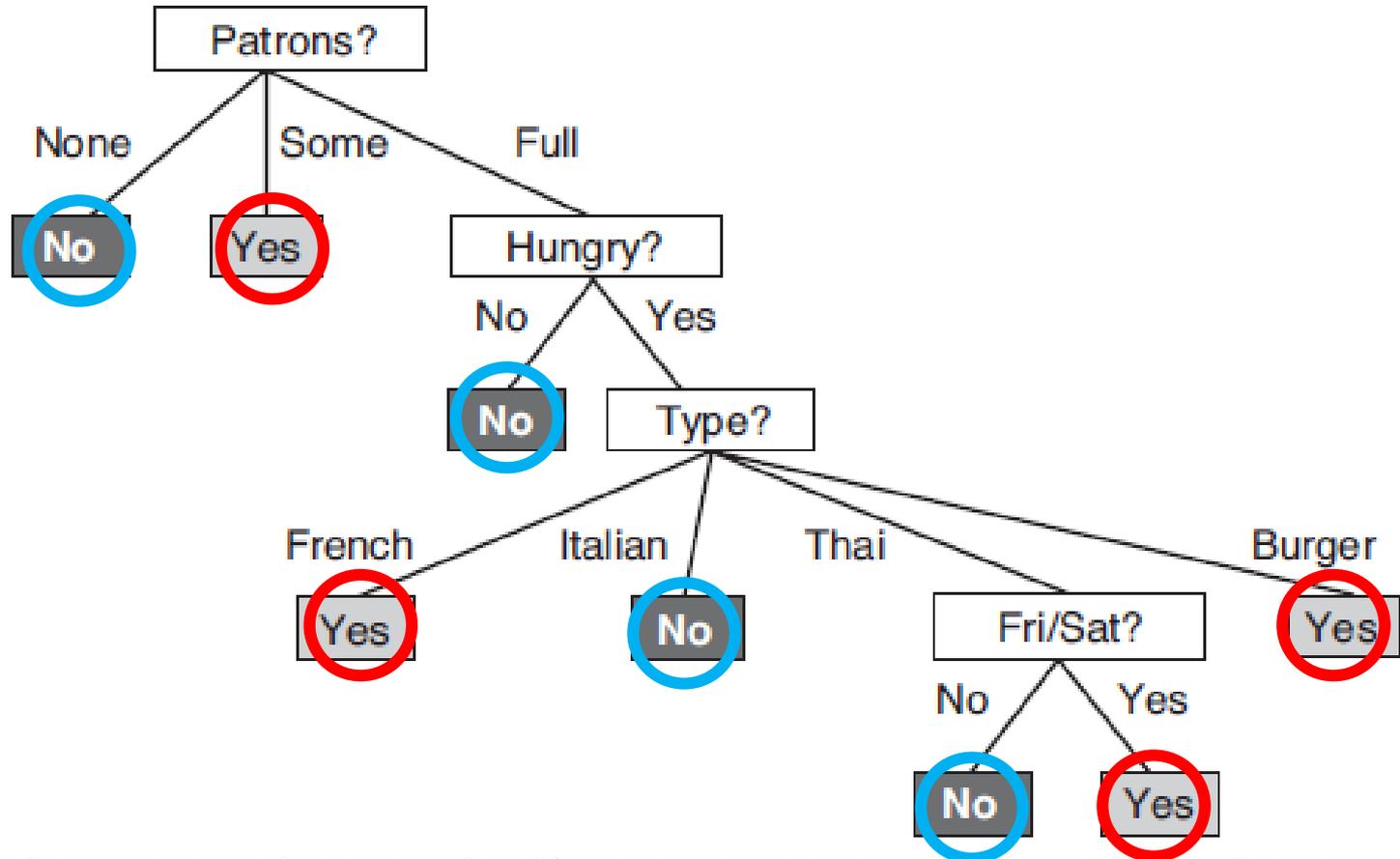
On choisit donc « Patrons »  
comme premier attribut de tri.

$$\text{Gain}(\textit{Patrons}) = 1 - \left[ \frac{2}{12} B\left(\frac{0}{2}\right) + \frac{4}{12} B\left(\frac{4}{4}\right) + \frac{6}{12} B\left(\frac{2}{6}\right) \right] = 0,541 \text{ bits}$$

```
In [49]: def B(q):
.....:     if(q == 0 or q == 1): return 0
.....:     else: return -(q*log2(q) + (1-q)*log2(1-q))
.....:
```

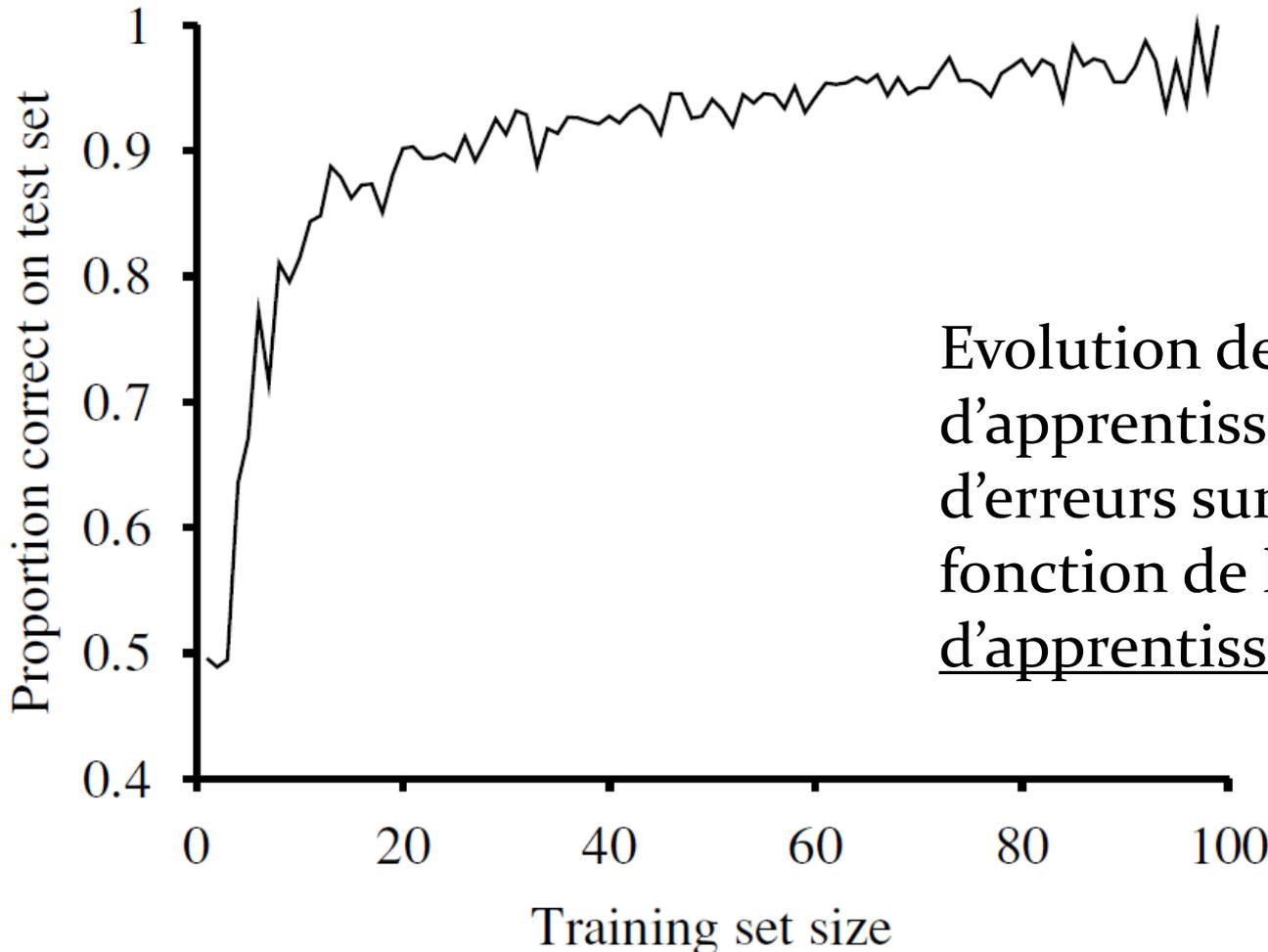
```
In [50]: B(6/12.) - (2/12.*B(0/2.)+4/12.*B(4/4.)+6/12.*B(2/6.))
Out [50]: 0.54085208297275522
```

# Arbre de décision



Arbre résultant de l'apprentissage sur 12 clients.

# Arbre de décision



Evolution de la courbe  
d'apprentissage : proportion  
d'erreurs sur le jeu de test en  
fonction de la taille du jeu  
d'apprentissage

# Arbre de décision

- Différentes méthodes de sélection de l'attribut :
  - **Critère d'entropie de Shannon** : méthodes **C4.5**, **ID3**
  - **Indice d'impureté de Gini** : Classification and Regression Tree (**CART**)

# Arbre de décision

- **L'indice d'impureté de Gini** (Gini Index) mesure l'inégalité de répartition de valeurs distinctes (classe de la valeur cible) au sein d'une distribution (modalités de notre attribut).
- **Impureté** : caractérisation du degré de mélange du nœud. Un nœud est dit pur si le segment qui lui est associé ne contient que des descriptions d'éléments d'une même classe (c'est ce que l'on veut !).
- On souhaite donc minimiser l'impureté des nœuds et donc sélectionner les nœuds dont le découpage apporte en moyenne les nœuds le plus purs possible.

# Arbre de décision

- On montre que **la réduction d'impureté** engendrée par l'attribut A prenant d modalités distinctes vaut :

$$\text{réduction d'impureté}(A) = GINI(\text{Système}) - GINI(A)$$

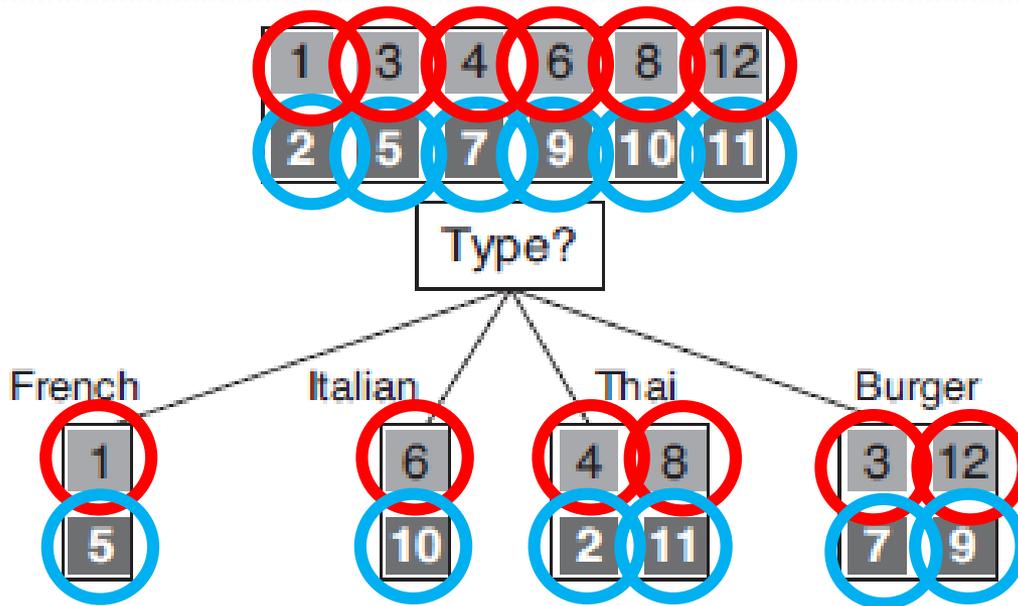
- Pour une variable cible prenant 2 valeurs distinctes (n et p) :

$$GINI(\text{Système}) = 1 - \left[ \left( \frac{p}{n+p} \right)^2 + \left( \frac{n}{n+p} \right)^2 \right] = 1 - \left[ \left( \frac{6}{12} \right)^2 + \left( \frac{6}{12} \right)^2 \right] = 0.5$$

$$GINI(A) = \sum_{k=1}^d \frac{p_k + n_k}{p + n} * \left[ 1 - \left( \left( \frac{p_k}{p_k + n_k} \right)^2 + \left( \frac{n_k}{p_k + n_k} \right)^2 \right) \right]$$

$p$  := nombre de valeurs YES dans le parent,  $n$  := nombre de valeurs NO dans le parent  
 $p_k$  := nombre de valeurs YES dans le fils k,  $n_k$  := nombre de valeurs NO dans le fils k

# Arbre de décision



On retrouve comme pour l'entropie, un gain de 0 pour l'attribut « Type »

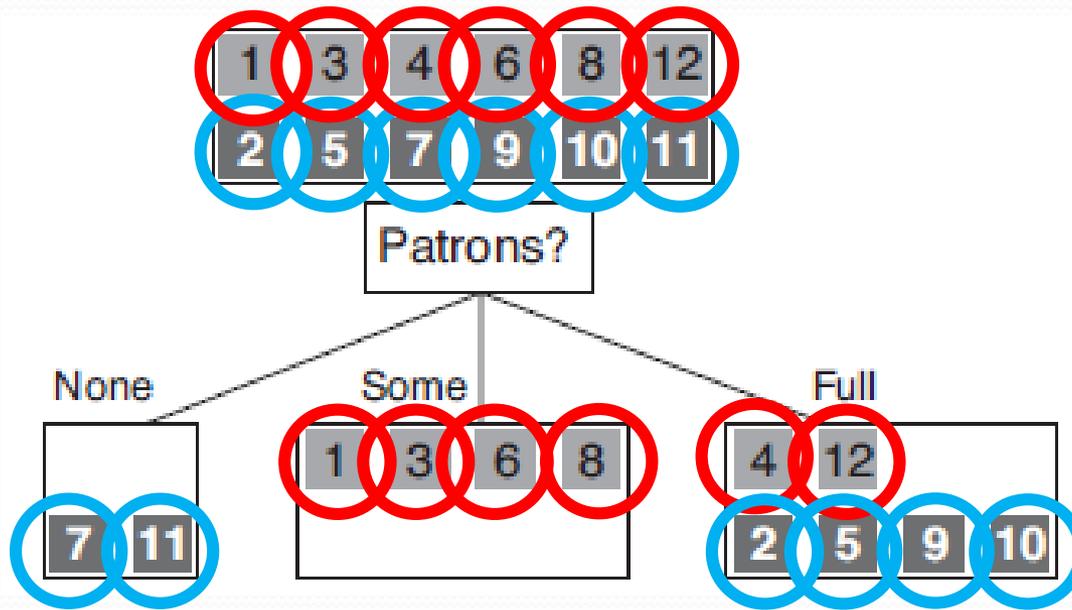
*Réduction d'impureté*(Type)

$$= 0,5 - \left[ 2 * \frac{2}{12} * \left( 1 - \left[ \left( \frac{1}{2} \right)^2 + \left( \frac{1}{2} \right)^2 \right] \right) + 2 * \frac{4}{12} * \left( 1 - \left[ \left( \frac{2}{4} \right)^2 + \left( \frac{2}{4} \right)^2 \right] \right) \right]$$

In [67]: 0.5 - (2 \* 2 / 12 \* (1 - (0.5 \*\* 2 + 0.5 \*\* 2)) + 2 \* 4 / 12 \* (1 - (0.5 \*\* 2 + 0.5 \*\* 2)))

Out [67]: 0.0

# Arbre de décision



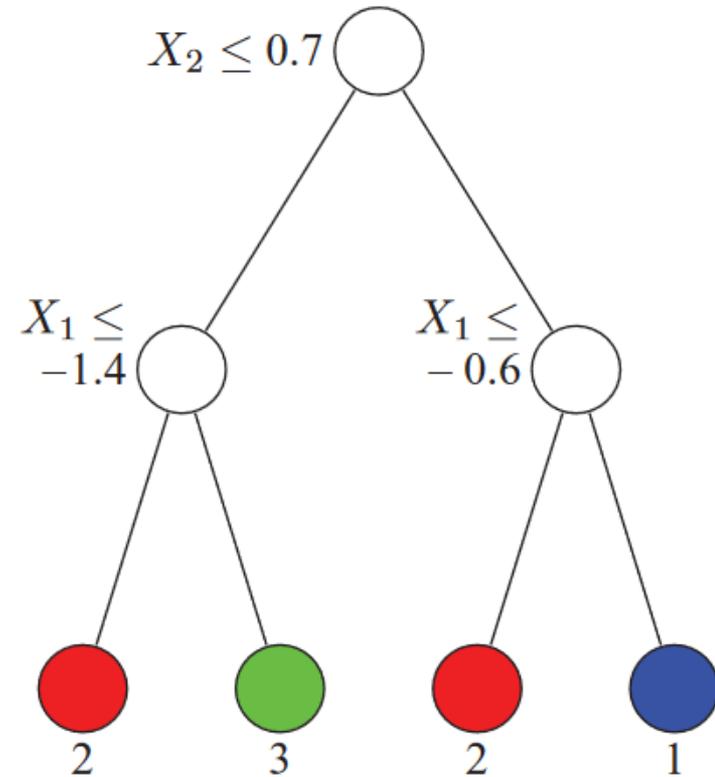
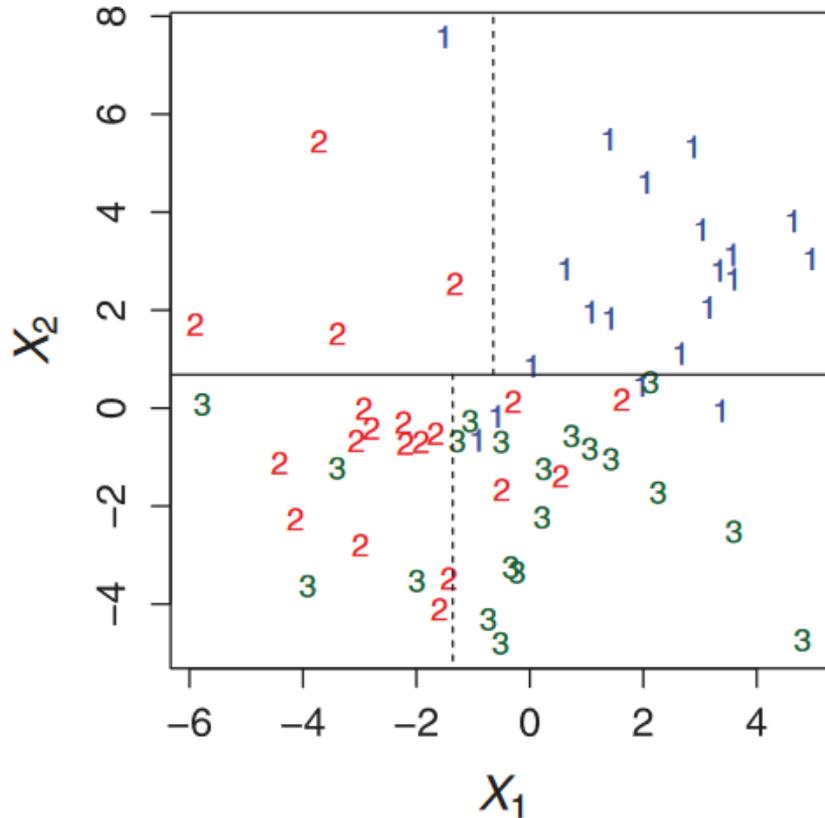
*Réduction d'impureté*(Patrons)

$$= 0.5 - \left[ \frac{2}{12} * \left( 1 - \left[ \left( \frac{0}{2} \right)^2 + \left( \frac{2}{2} \right)^2 \right] \right) + \frac{4}{12} * \left( 1 - \left[ \left( \frac{4}{4} \right)^2 + \left( \frac{0}{4} \right)^2 \right] \right) + \frac{6}{12} * \left( 1 - \left[ \left( \frac{2}{6} \right)^2 + \left( \frac{4}{6} \right)^2 \right] \right) \right]$$

In [69]: 0.5-6/12.\*(1-((2/6.)\*\*2+(4/6.)\*\*2))

Out [69]: 0.27777777777777778

# Arbre de décision



Représentation graphique possible avec uniquement 2 attributs

# Apprentissage supervisé

- Règle de Bayes
- Classification naïve bayésienne
- Régression multivariée
- Régression régularisée
- Protocole d'apprentissage
- Les k plus proches voisins
- Dilemme biais/variance
- Arbre de décision
- **Bagging**
- Forêt aléatoire
- Perceptron
- Perceptron multicouche
- Les réseaux de neurones
- Deep learning

# Bootstrap aggregating (Bagging)

- Algorithme utilisable en combinaison avec d'autres méthodes de Machine Learning.
- Objectif : réduire la variance et le surajustement.
- Méthode :
  - Soit un training set  $D$  de taille  $n$
  - Tirer aléatoirement et avec remise (dans  $D$ )  $m$  échantillons  $D_i$  de taille  $n'$ . Un échantillon est appelé échantillon **bootstrap**.
  - Entraîner l'algorithme de ML sur chacun des échantillons créant ainsi  $m$  modèles.
  - Exécuter les  $m$  modèles sur les nouvelles données.
  - Agréger le résultat,
    - Par vote pour la classification (classe majoritaire)
    - Par moyenne pour la régression

# Bootstrap aggregating (Bagging)

Planet	Mass ( $M_{Jup}$ )	Radius ( $R_{Jup}$ )	Period (day)	$a$ (AU)	$e$	$i$ (deg)
WASP-103 b	1.49	1.528	0.925542	0.01985	—	86.3
OGLE-2016-BLG-1112L b	31.7	—	—	9.63	—	—
OGLE-2016-BLG-1469L b	13.6	—	—	0.33	—	—
OGLE-2016-BLG-0693L b	49	—	—	5	—	—
SAND 364 b	1.54	—	121.71	—	0.35	—
HD 41248 c	0.0223	—	13.365	0.107	0.0117	—
EPIC 210894022 b	0.027	0.17	5.35117	0.0621	—	86.2
K2-110 b	0.0525	0.2312	13.86375	0.1021	0.079	89.35
Kepler 1651 A b	—	0.164	9.87863917	0.0619	0.13	85.94
Kepler-1319 A b	—	0.126	2.88676239	—	0.33	—
WASP-167 b	8	1.51	2.0219591	0.0365	—	79.9
Kepler-1650 b	—	0.067	1.538181528	0.0163	—	89.38
YZ Cet d	0.00359	—	4.65627	0.02764	0.129	—
YZ Cet c	0.0031	—	3.06008	0.0209	0.04	—
YZ Cet b	0.0024	—	1.96876	0.01557	0	—
tau Cet g	0.00551	—	20	0.133	0.06	—

Bootstrap #1

Bootstrap #2

Bootstrap #3

MLAlgo

MLAlgo

MLAlgo

Vote ou  
Moyenne

# Bootstrap aggregating (Bagging)

- Remarque : Le tirage est réalisé avec remise, on peut donc avoir  $n$  fois le même exemple (ligne) dans l'échantillon  $D_i$ .
- En poussant cette logique, on peut réaliser des échantillons de même taille que le training set d'origine, c.-à-d.  $n=n'$ , dans ce cas, on montre que la fraction des exemples uniques provenant de  $D$  est en moyenne :

$$1 - \frac{1}{e} \approx 63,2\%$$

Le reste des exemples étant des duplications.

# Bootstrap aggregating (Bagging)

- On peut montrer que :
  - soit  $m$  le nombre de bootstrap,
  - en supposant que les  $m$  modèles sont non corrélés,
  - nous réduirions la variance de notre modèle (ou estimateur) sans affecter le biais par :

$$VarBagging = Var(Bagging(Modèle)) = \frac{Var(Modèle)}{m}$$

En réalité, puisque les modèles sont corrélés suivant un coefficient de corrélation  $\rho$ , on aura :

$$VarBagging = \rho * Var(Modèle) + \frac{1 - \rho}{m} * Var(Modèle)$$

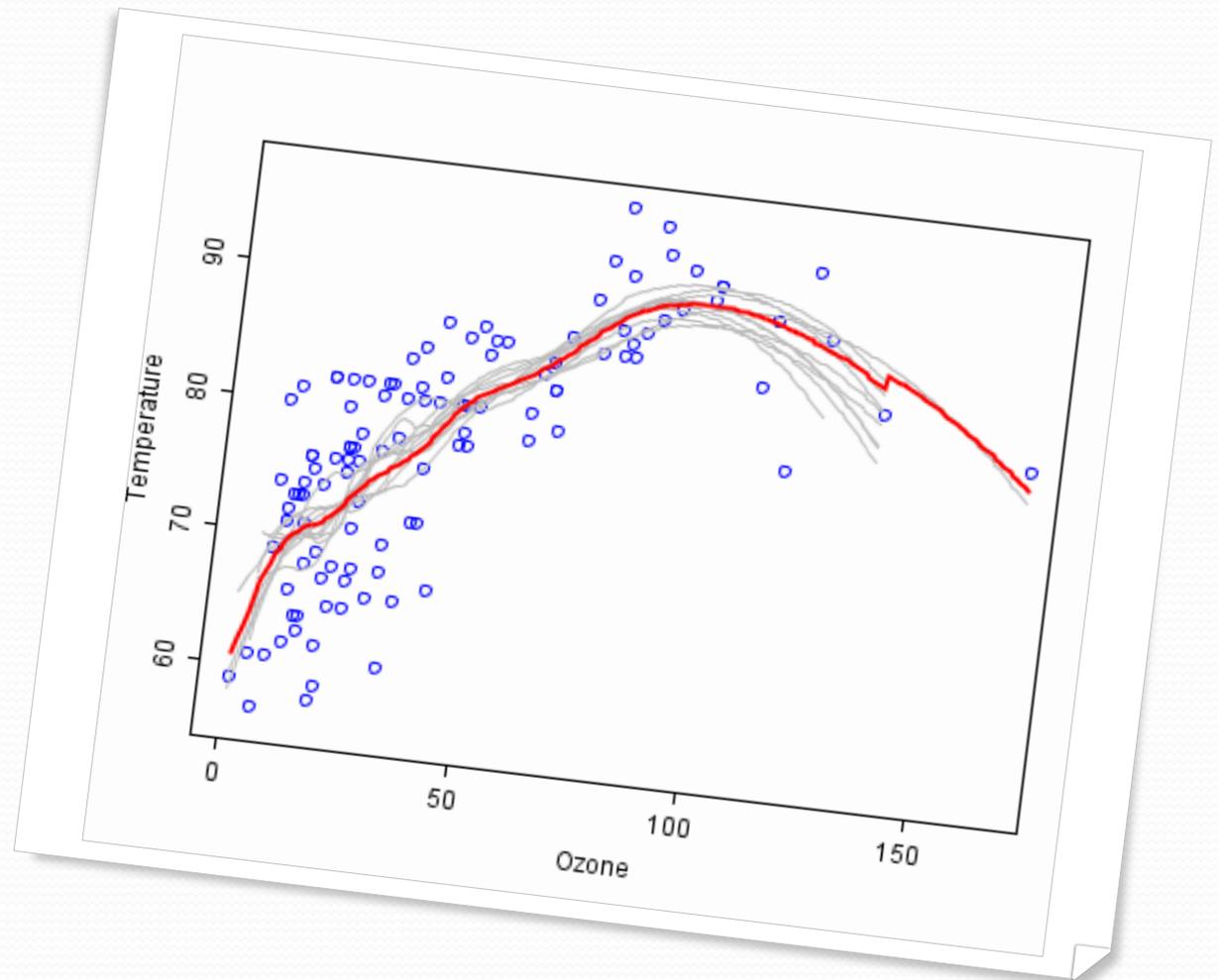
Réduction de la variance par bagging

## Bootstrap aggregating (Bagging)

Relation entre l'Ozone  
et la Température.

100 bootstrap ont été réalisés (10 affichés en gris).

Les lignes grises issues de chaque bootstrap surajustent, alors que leur moyenne est plus stable.



# Apprentissage supervisé

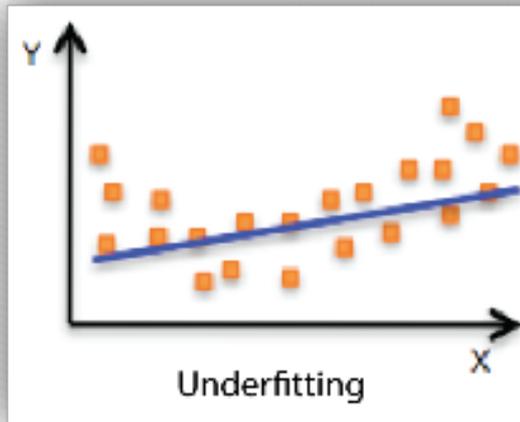
- Règle de Bayes
- Classification naïve bayésienne
- Régression multivariée
- Régression régularisée
- Protocole d'apprentissage
- Les k plus proches voisins
- Dilemme biais/variance
- Arbre de décision
- Bagging
- **Forêt aléatoire**
- Perceptron
- Perceptron multicouche
- Les réseaux de neurones
- Deep learning

# Forêt aléatoire (Random forest)

- Les arbres de décision ont tendance à avoir une forte variance, c'est à dire qu'ils sont très sensibles aux valeurs particulières de leur « training set » et ont donc une faible capacité de généralisation.

# Dilemme/compromis biais-variance

Régression linéaire



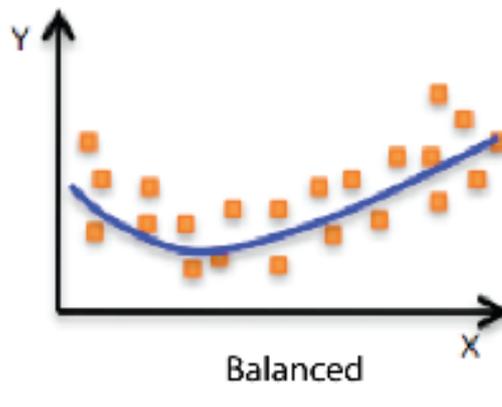
Biais élevé  
Variance faible



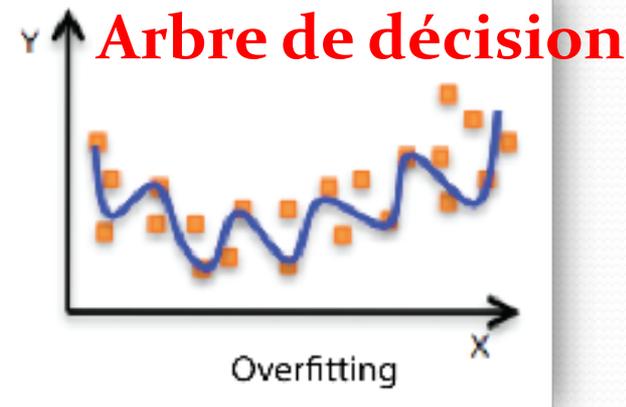
⇒ Faible capacité  
de représentation



⇒ Bonne capacité  
de généralisation



Régression  
polynomiale à  
nombreux degrés



Biais faible,  
Variance élevée



⇒ Bonne capacité  
de représentation



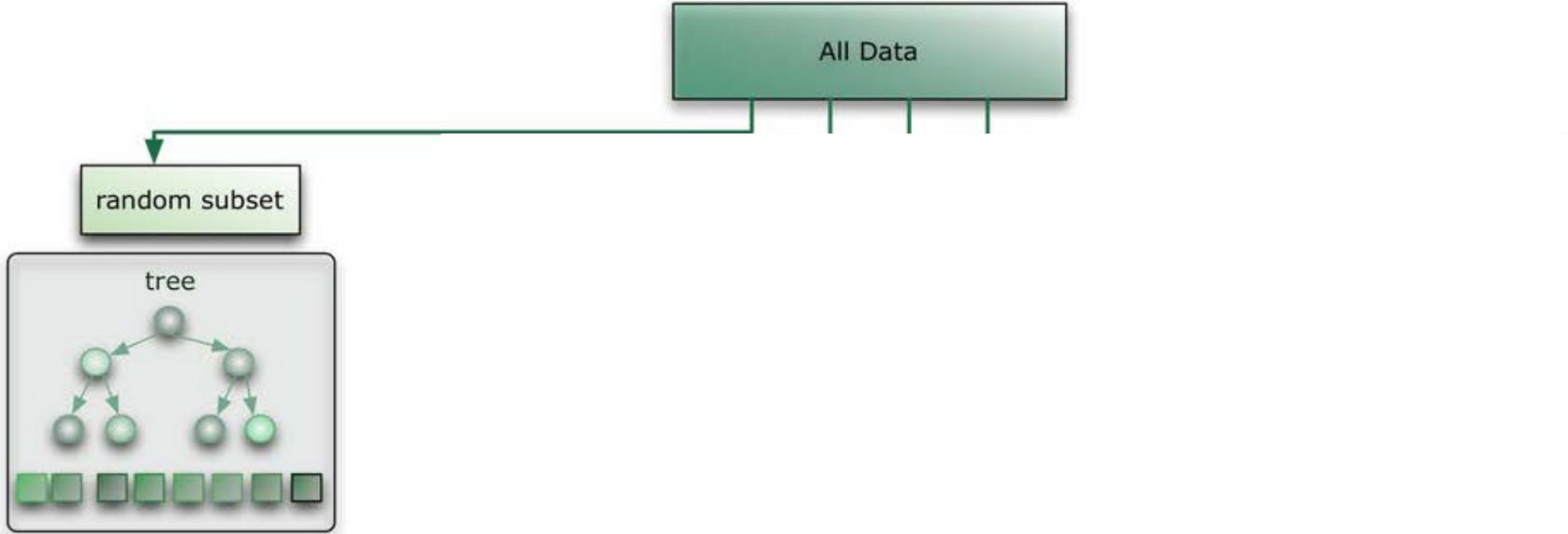
⇒ Mauvaise capacité  
de généralisation

# Forêt aléatoire (Random forest)

- **L'algorithme de forêt aléatoire (Random forest)** effectue un apprentissage sur de **multiples arbres de décision** entraînés sur des sous-ensembles de données légèrement différents.
- L'algorithme de forêt aléatoire utilise 2 techniques :
  - Bagging
  - Sélection d'attributs (Feature sampling)

# Forêt aléatoire (Random forest)

- Soit un « training set »  $D$  de  $n$  exemples et de  $p$  attributs.
- Tirer aléatoirement et avec remise (dans  $D$ )  $m$  échantillons (bootstrap)  $D_i$  **de taille  $n$** .
- Créer un arbre de décision pour chaque bootstrap respectant la règle suivante :
  - A chaque sélection d'un attribut de split pour un nœud, **sélectionner l'attribut dans un sous-ensemble des  $p$  attributs tiré aléatoirement**.
  - On utilise généralement  $p' = \sqrt{d}$  (pour la classification),  $\frac{p}{3}$  pour la régression.
- Exécuter les  $m$  modèles sur les nouvelles données.
- Agréger le résultat,
  - Par vote pour la classification (classe majoritaire)
  - Par moyenne pour la régression



# Forêt aléatoire (Random forest)

- Bagging

$$Var = \rho * Var(\text{Modèle}) + \frac{1 - \rho}{m} * Var(\text{Modèle})$$

Réduction de la variance par bagging

Réduction de la variance par feature sampling

- Sélection d'attributs (Feature sampling)
  - Réduction de la variance du modèle lui-même.

# Apprentissage supervisé

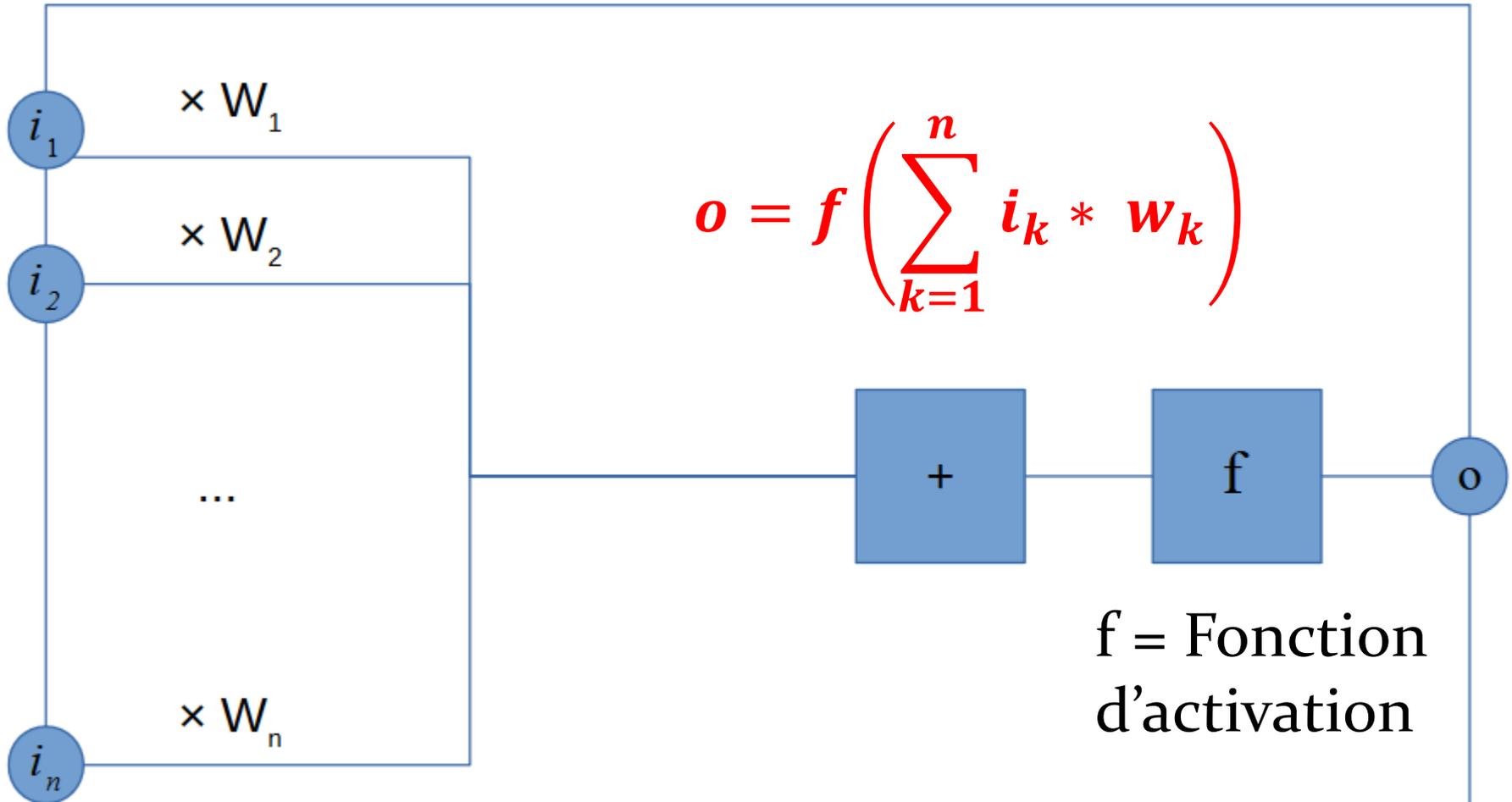
- Règle de Bayes
- Classification naïve bayésienne
- Régression multivariée
- Régression régularisée
- Protocole d'apprentissage
- Les k plus proches voisins
- Dilemme biais/variance
- Arbre de décision
- Bagging
- Forêt aléatoire
- **Perceptron**
- Perceptron multicouche
- Les réseaux de neurones
- Deep learning

# Le perceptron



- Le **perceptron** a été inventé en 1957 par **Frank Rosenblatt**
- Le cerveau humain contient jusqu'à cent milliards de neurones. Chaque neurone est connecté en moyenne à 10.000 autres par le biais de synapses.
- Les stimulations synaptiques perçues par un neurone peuvent activer celui-ci qui transmet alors un signal électrique aux neurones suivants.
- Le perceptron imite la stimulation d'un neurone par des neurones voisins.

# Le perceptron



# Le perceptron

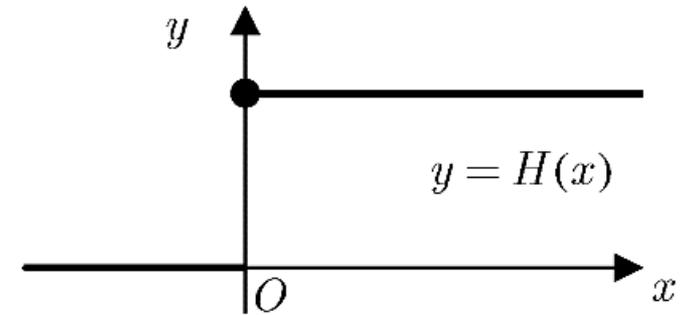
- Fonctions d'activation :

- fonction de **Heaviside (seuil)**

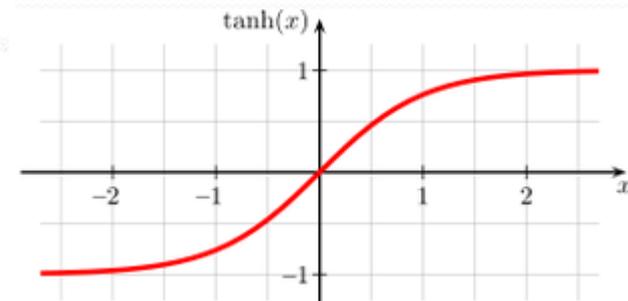
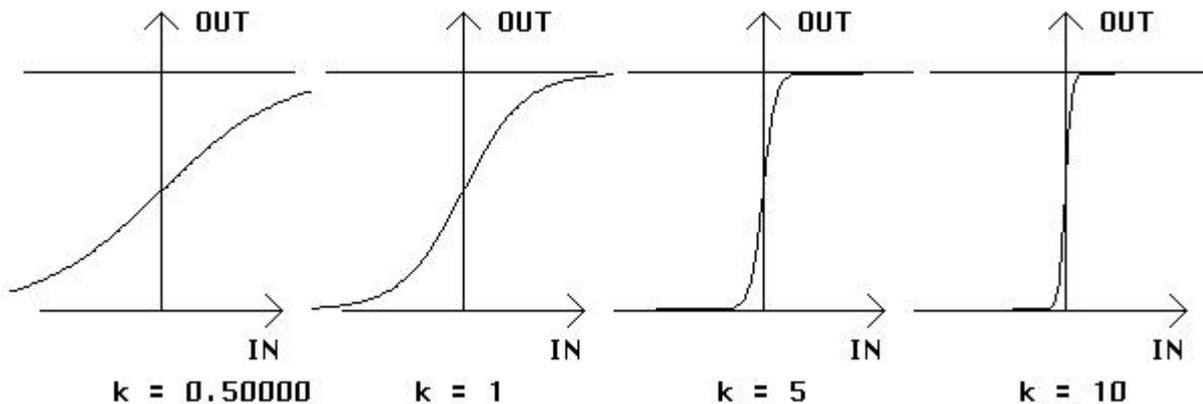
- $y = Heaviside(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$

- fonction **sigmoïde**

- $y(x) = \frac{1}{1 + e^{-k * x}}$ ;  $k > 0$  ou



$$y(x) = \tanh(x)$$



# Le perceptron

## Seuil d'activation

- On ne souhaite pas forcément le déclenchement dès que  $\sum_{k=1}^n i_k * w_k > 0$ . Ce que l'on souhaite plutôt c'est le déclenchement à partir d'un seuil  $\theta$  :

$$y = \left\{ \begin{array}{l} 1 \text{ si } \sum_{k=1}^n i_k * w_k > \theta \\ 0 \text{ sinon} \end{array} \right\}$$

ou

$$y = Heaviside \left( \sum_{k=1}^n i_k * w_k - \theta \right)$$

# Le perceptron

- Par simplicité on pose :

$$w_0 = \theta$$

$$y = \text{Heaviside} \left( \sum_{k=0}^n i_k * w_k \right), i_0 = -1$$

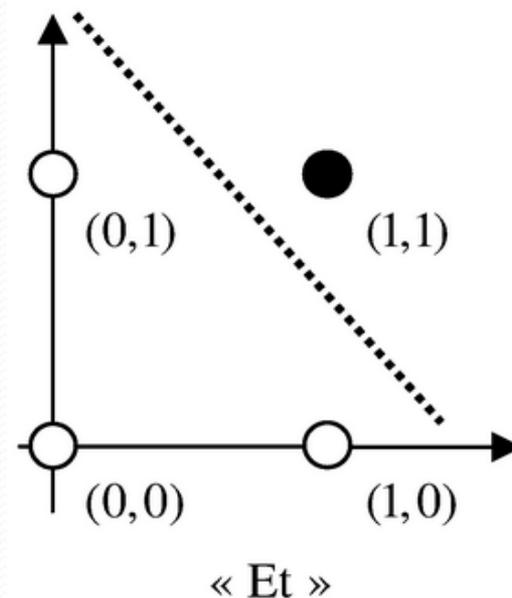
- $w_0$  est une entrée correspondant au seuil (parfois appelé biais).  **$w_0$  ne dépend pas des entrées.**

# Le perceptron

## Exemple

- Soit une base d'apprentissage correspondant à l'opérateur logique ET. Nous disposons de 2 entrées et d'une sortie binaire.
- $(0,0) \mapsto 0$ ,  $(1,0) \mapsto 0$ ,  $(0,1) \mapsto 0$  et  $(1,1) \mapsto 1$

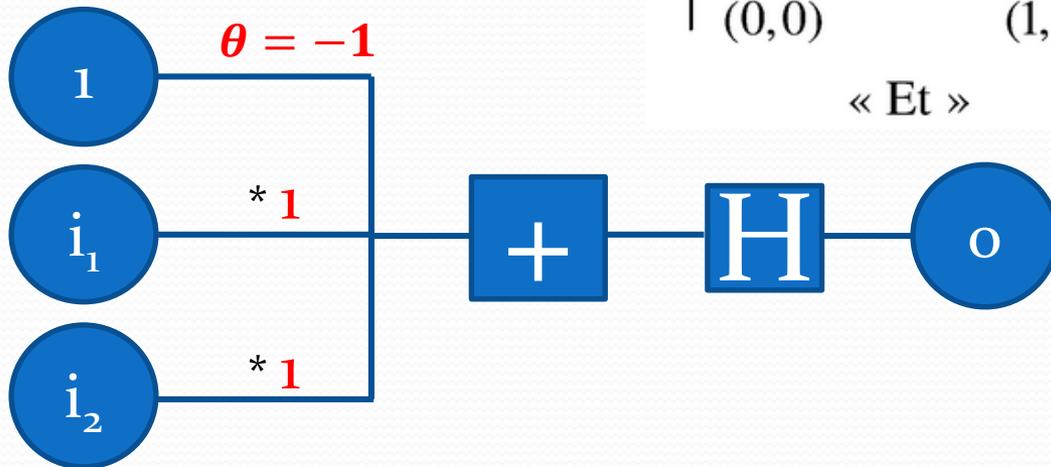
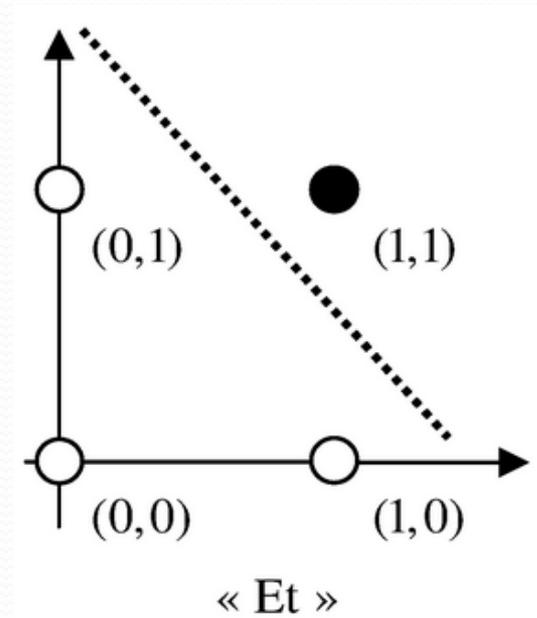
	$E_1 = 0$	$E_1 = 1$
$E_2 = 0$	0	0
$E_2 = 1$	0	1



# Le perceptron

## Exemple

« Et » logique

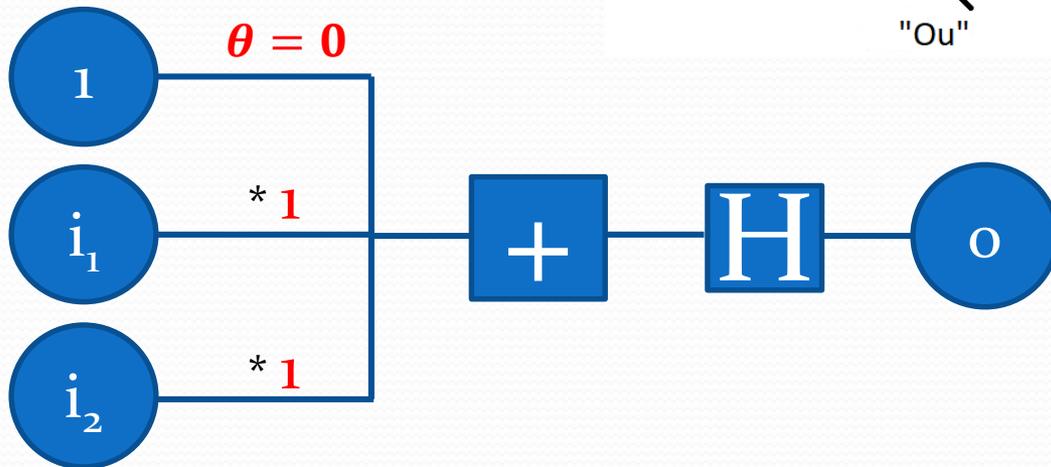
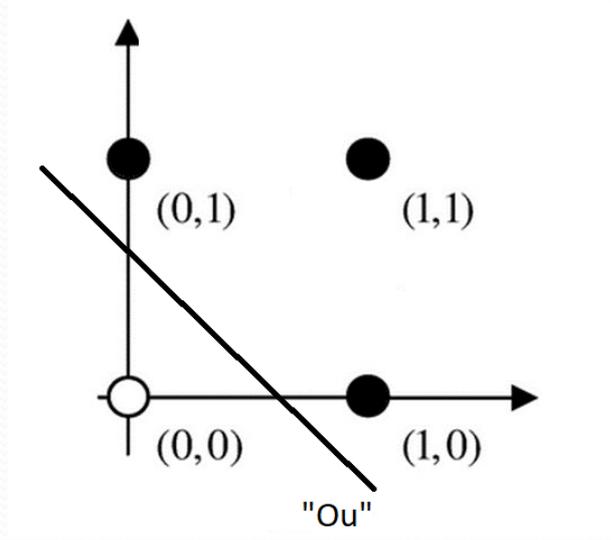


<b>i<sub>1</sub></b>	<b>i<sub>2</sub></b>	<b>o</b>
0	0	0
0	1	0
1	0	0
1	1	1

# Le perceptron

## Exemple

« Ou » logique



$i_1$	$i_2$	$o$
0	0	0
0	1	1
1	0	1
1	1	1

# Le perceptron

## Exemple

- Nous pouvons tout de suite entrevoir une limite dans la modélisation du perceptron. En effet, comment pourrions nous simuler un opérateur XOR ?
- XOR :  $(0,0) \mapsto 0$ ,  $(1,0) \mapsto 1$ ,  $(0,1) \mapsto 1$  et  $(1,1) \mapsto 0$

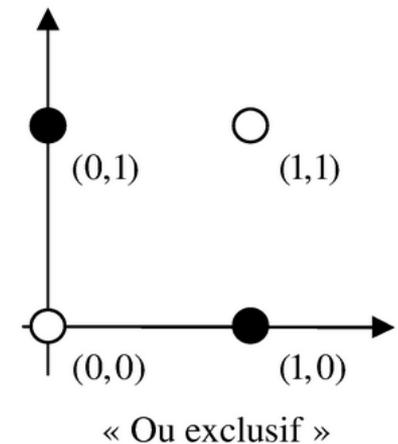
	$E_1 = 0$	$E_1 = 1$
$E_2 = 0$	0	1
$E_2 = 1$	1	0

$w_1 * E_1 > \theta$

$w_2 * E_2 > \theta$

$w_1 * E_1 + w_2 * E_2 > \theta$

(Note: A red circle with a minus sign is placed between the two intermediate equations, indicating that the simple sum of individual weights does not work for XOR.)



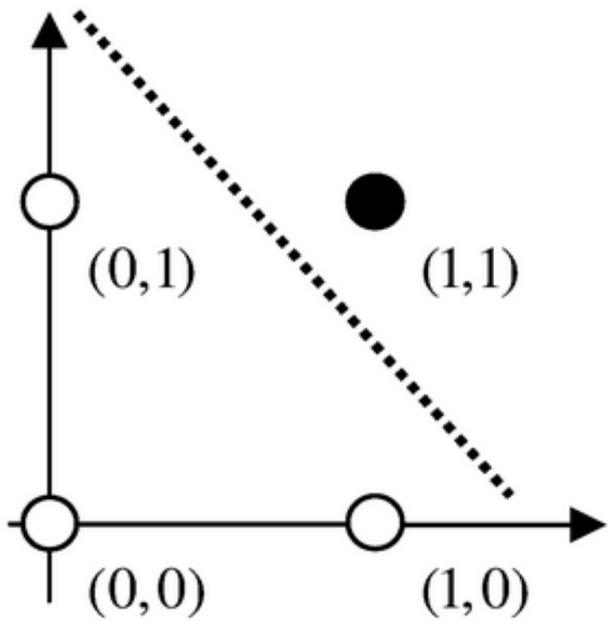
# Le perceptron

$\sum_{k=1}^n i_k * w_k > \theta$  est l'équation d'un hyperplan dans un espace  $\mathbb{R}^n$  (c.-à-d. une droite dans  $\mathbb{R}^2$ ). Cette droite sépare l'espace en deux sous espace.

- On montre que :

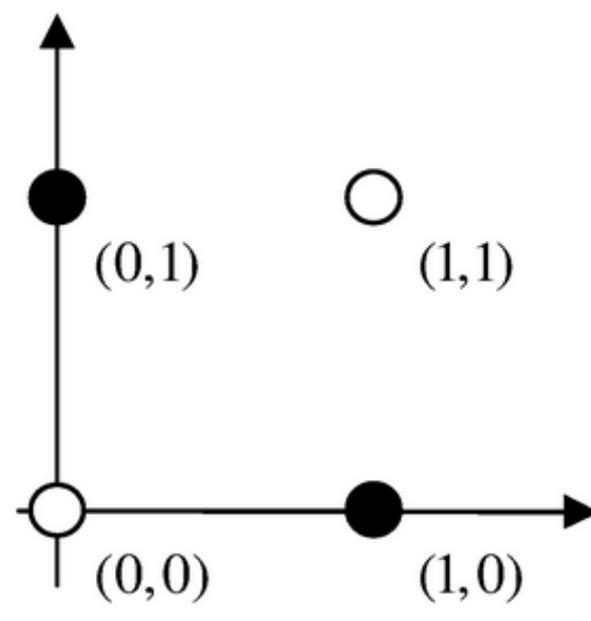
*Un échantillon d'exemples pourra être appris par un perceptron si, et seulement si, il est linéairement séparable, c'est-à-dire que les éléments envoyés sur 0 peuvent être séparés de ceux envoyés sur 1 par un hyperplan.*

# Le perceptron



« Et »

Modélisable par un perceptron



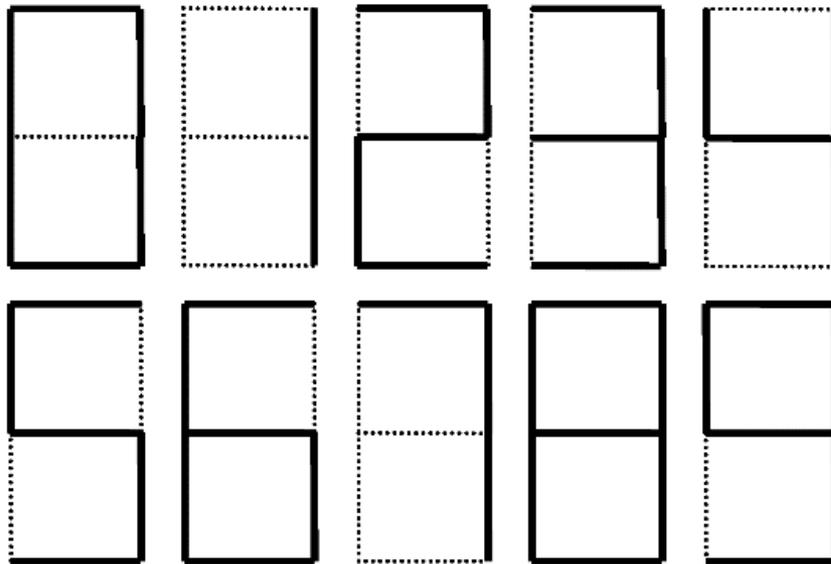
« Ou exclusif »

Non modélisable par un perceptron

# Le perceptron

## Protocole d'apprentissage

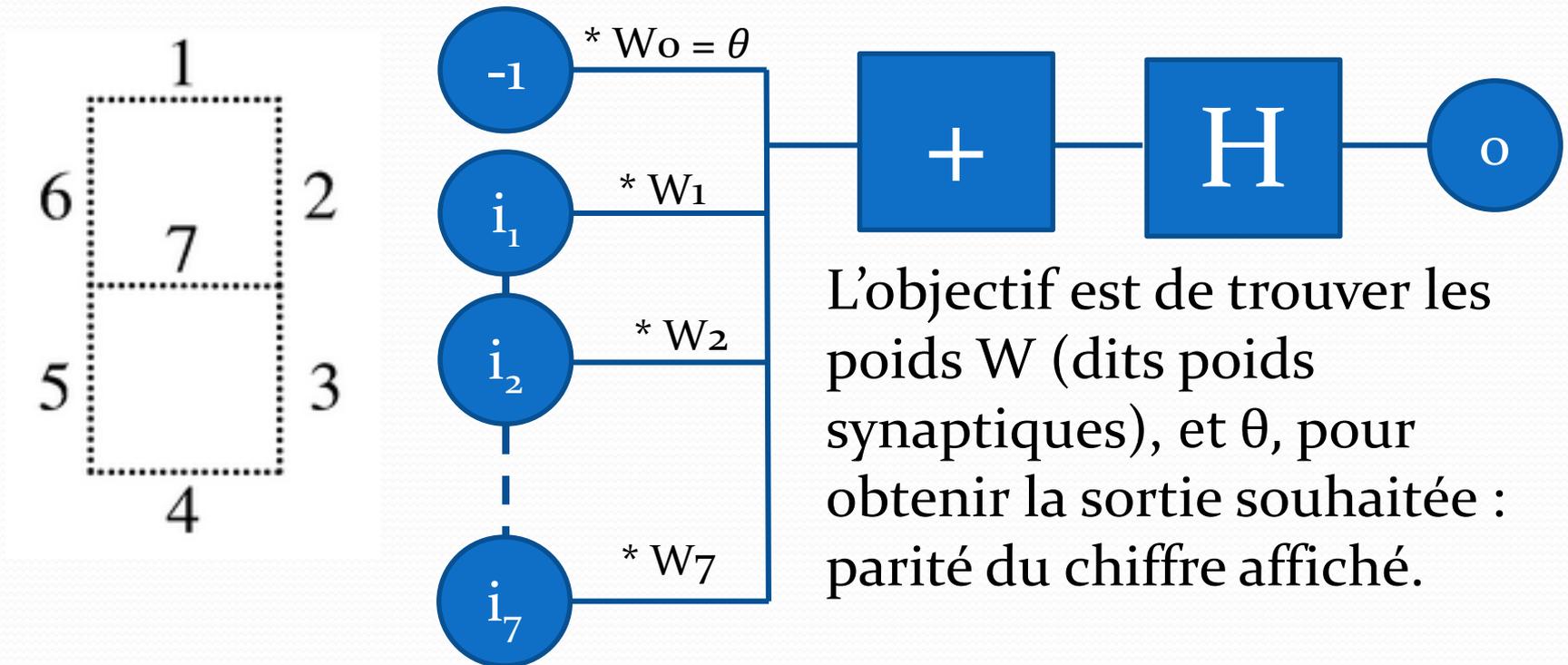
- Soit un afficheur numérique à sept segments.
- Nous souhaitons concevoir un perceptron donnant la parité du chiffre affiché, à savoir 0 s'il est pair et 1 sinon.



# Le perceptron

## Protocole d'apprentissage

- Notre modèle : chaque segment correspond à une entrée de notre perceptron, numérotée de 1 à 7.



# Le perceptron

## Protocole d'apprentissage

- Initialiser tous les  $W$  (par exemple à la valeur 0)
- Pour chaque exemple d'une base d'apprentissage :
  - Comparer la réponse fournie par le perceptron avec la réponse attendue (vraie parité).

Parité attendue	Sortie	Action	
0	0	Aucune	<b>On parle de</b>
1	1	Aucune	<b>rétropropagation de l'erreur</b>
0	1	Retrancher 1 à tous les poids synaptiques des entrées actives	
1	0	Ajouter 1 à tous les poids synaptiques des entrées actives	

# Le perceptron

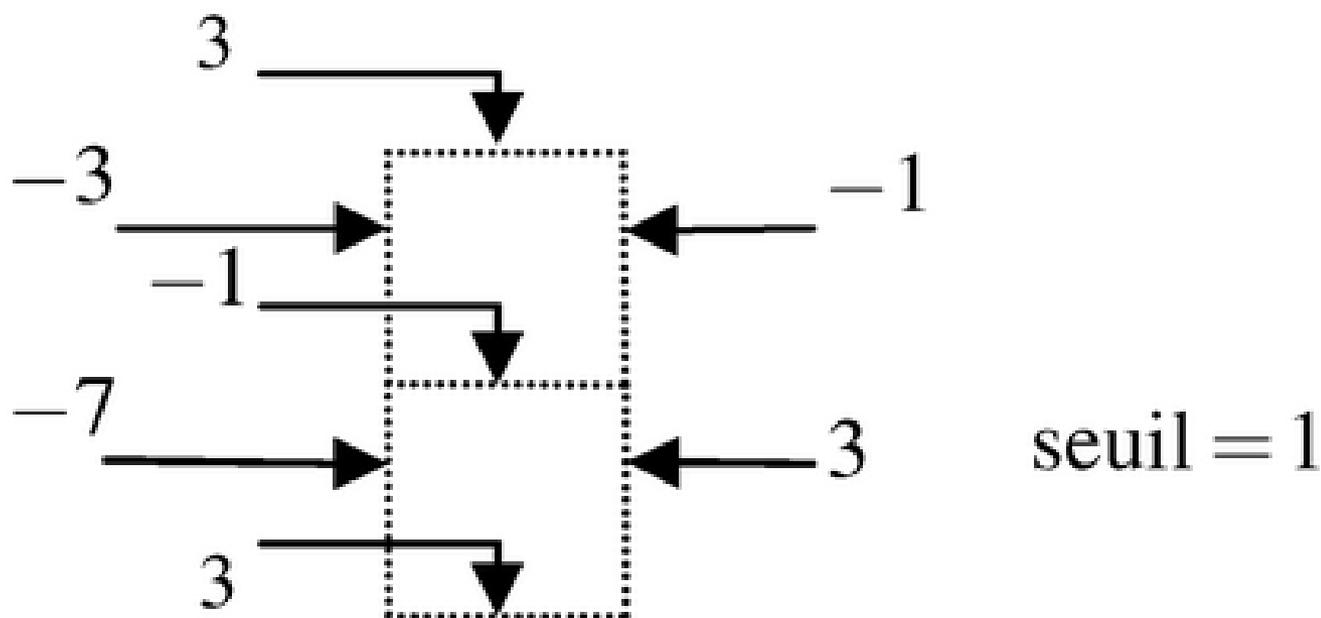
## Protocole d'apprentissage

- L'algorithme d'apprentissage se poursuit jusqu'à ce que tous les éléments de la base d'apprentissage soient étudiés sans qu'aucun poids synaptiques ne soient modifiés.
  - Stabilisation du perceptron, le modèle a convergé.
- L'algorithme s'arrête si et seulement si la base d'apprentissage est linéairement séparable.

# Le perceptron

## Protocole d'apprentissage

Résultat obtenu après convergence pour les poids synaptiques



# Apprentissage supervisé

- Règle de Bayes
- Classification naïve bayésienne
- Régression multivariée
- Régression régularisée
- Protocole d'apprentissage
- Les k plus proches voisins
- Dilemme biais/variance
- Arbre de décision
- Bagging
- Forêt aléatoire
- Perceptron
- **Perceptron multicouche**
- Les réseaux de neurones
- Deep learning

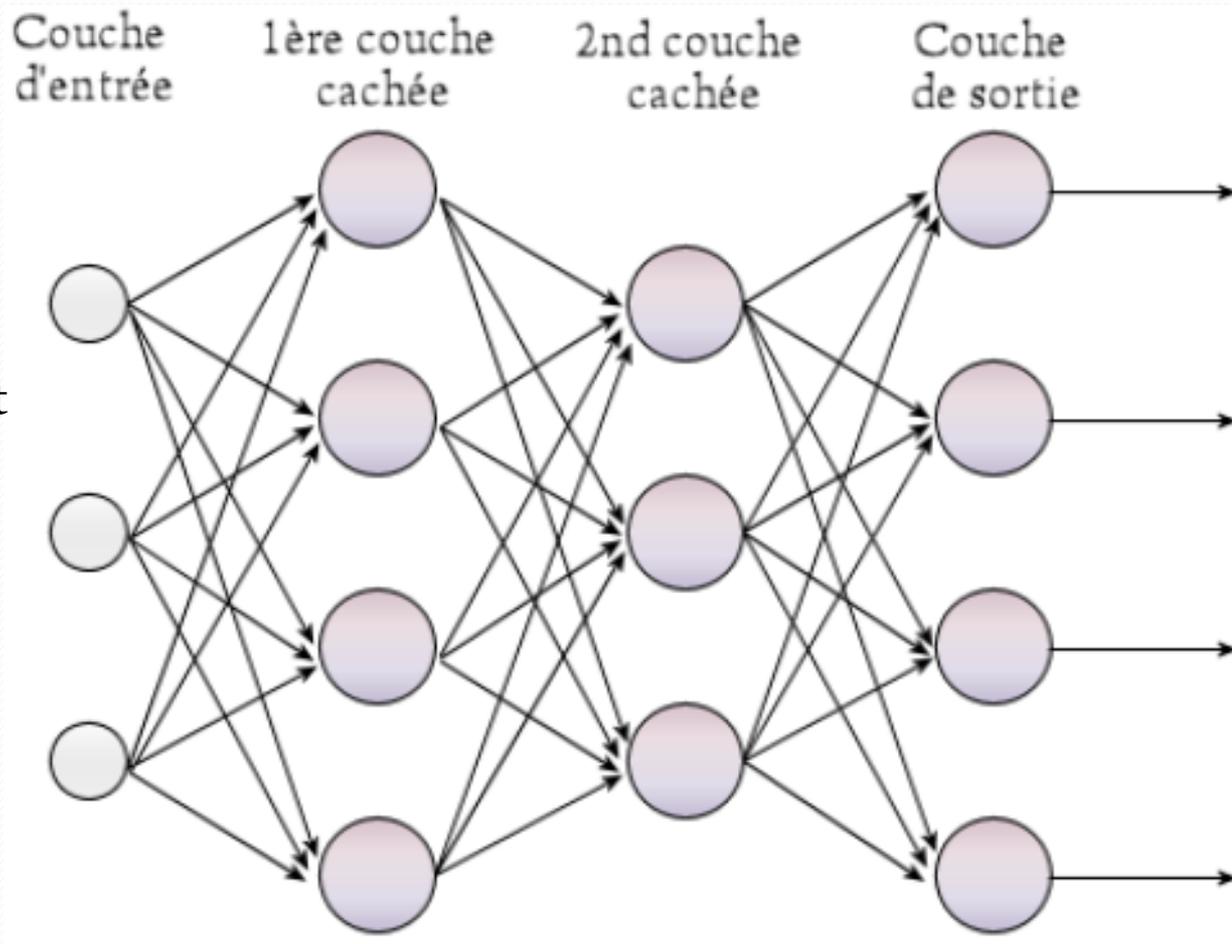
# Le perceptron multicouche (MLP)

- Le perceptron multicouche apparait en 1986 introduit par **David Rumelhart**, et, simultanément, chez **Yann Le Cun** (Collège de France).
- Principe : On utilise plusieurs perceptrons en couche successives, les couches centrales étant dites cachées.
- Chaque nœud de la couche  $i$  est connecté à tous les nœuds de la couche  $i+1$  et l'information circule toujours de la couche  $i$  à la couche  $i+1$  (**feedforward connection**).

# Le perceptron multicouche

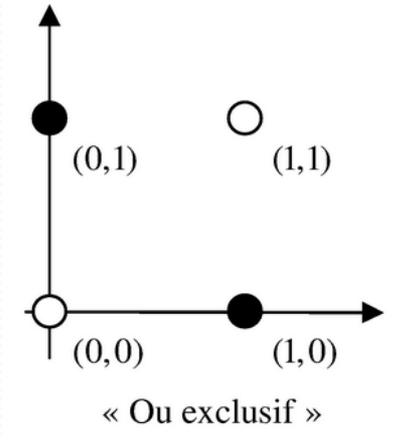
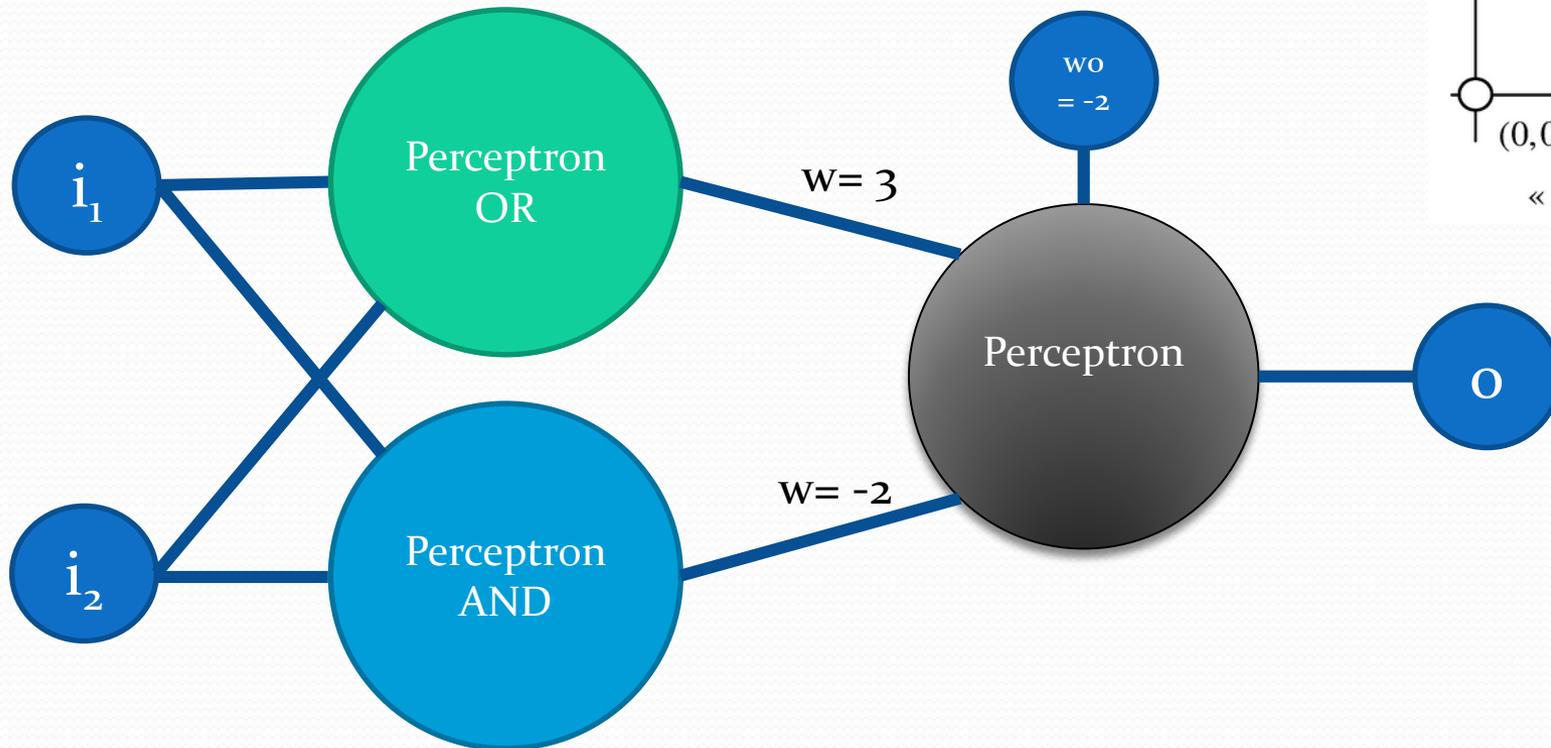


David Rumelhart



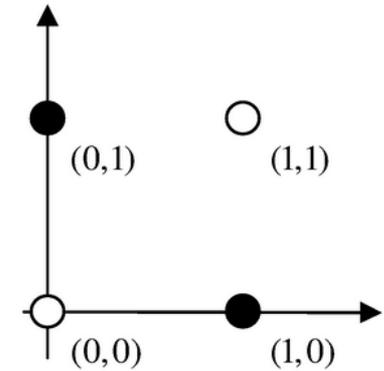
# Le perceptron multicouche

Pour résoudre notre problème de linéarité :

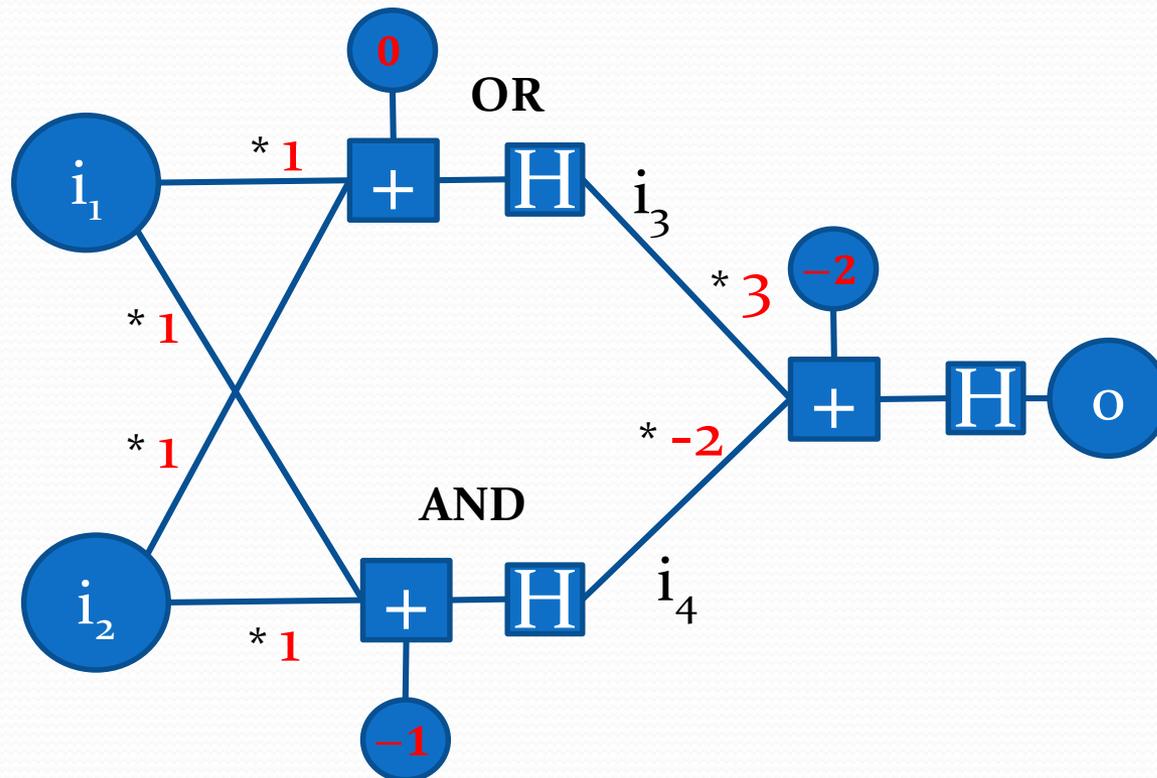


# Le perceptron multicouche

Pour résoudre notre problème de linéarité :



« Ou exclusif »



$i_1$	$i_2$	$i_3$	$i_4$	$o$
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

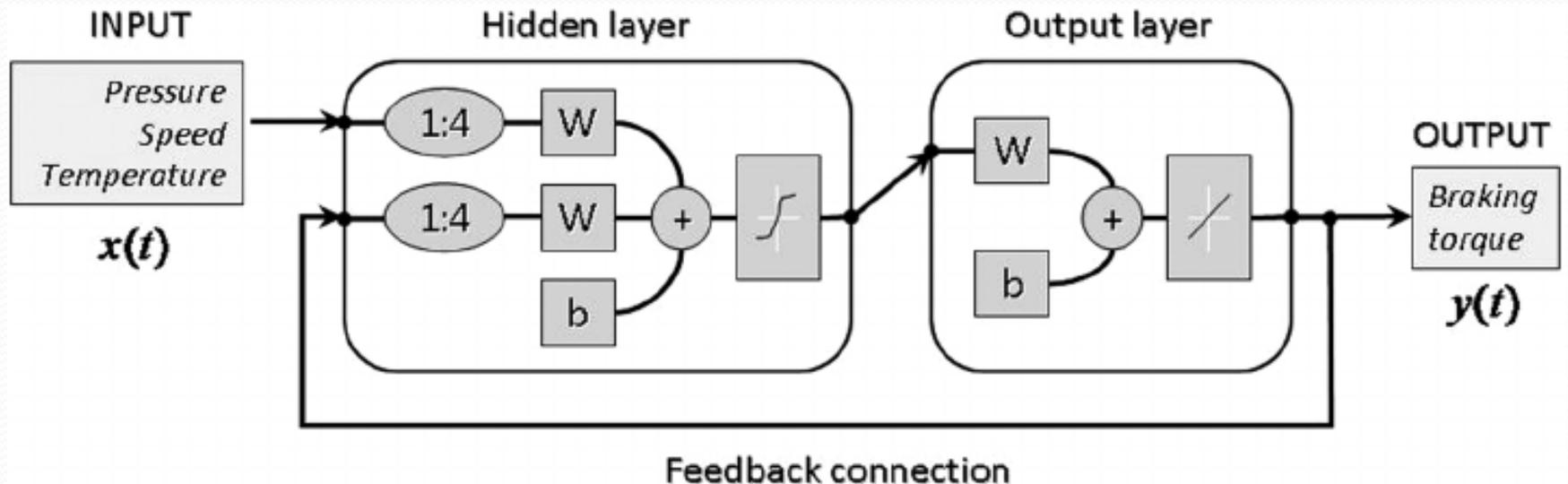
**XOR**

# Apprentissage supervisé

- Règle de Bayes
- Classification naïve bayésienne
- Régression multivariée
- Régression régularisée
- Protocole d'apprentissage
- Les k plus proches voisins
- Dilemme biais/variance
- Arbre de décision
- Bagging
- Forêt aléatoire
- Perceptron
- Perceptron multicouche
- **Les réseaux de neurones**
- Deep learning

# Les réseaux de neurones (NN)

- Les réseaux de neurones
  - Les MLP sont un type de réseau de neurones.
  - Possibilité de faire des boucles (**feedback connection**) ou réseaux récurrents.

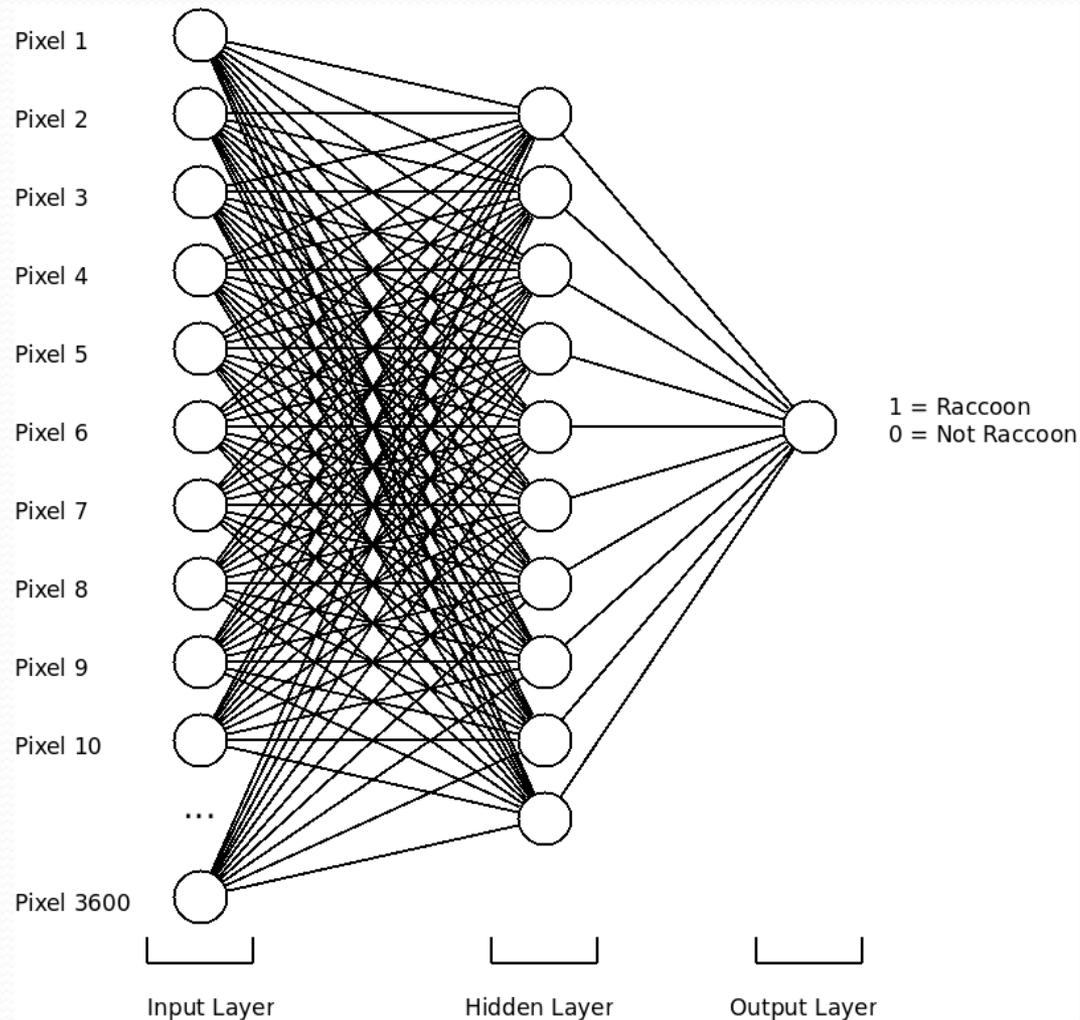


# Les réseaux de neurones (NN)

- Application : Reconnaissance de forme dans une image.
- **Image d'un chat => labélisée "Chat"**
- Première approche : chaque pixel est connecté à une entrée du réseau de neurone.

# Les réseaux de neurones (NN)

## Pattern recognition



# Les réseaux de neurones (NN)

- Solution trop lourde en charge CPU.
- On extrait d'abord des caractéristiques de l'image, ce sont ces caractéristiques (features) que l'on injecte dans le NN. **Feature Engineering.**

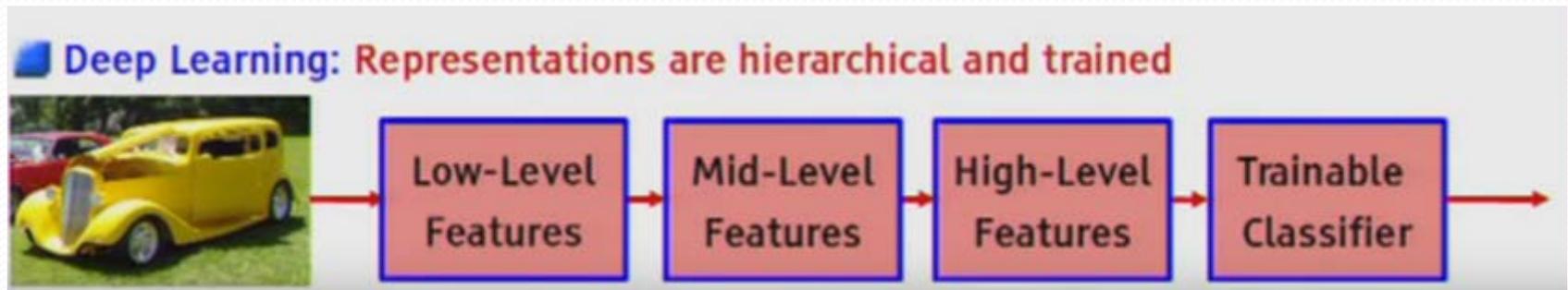


# Apprentissage supervisé

- Règle de Bayes
- Classification naïve bayésienne
- Régression multivariée
- Régression régularisée
- Protocole d'apprentissage
- Les k plus proches voisins
- Dilemme biais/variance
- Arbre de décision
- Bagging
- Forêt aléatoire
- Perceptron
- Perceptron multicouche
- Les réseaux de neurones
- **Deep learning**

# Deep learning

- Modèle plus gros (deep learning) rendu possible par :
  - Accroissement de la performance de calcul des machines (GPU)
  - Base de données d'apprentissage de plus en plus gigantesque (<http://www.image-net.org/>)



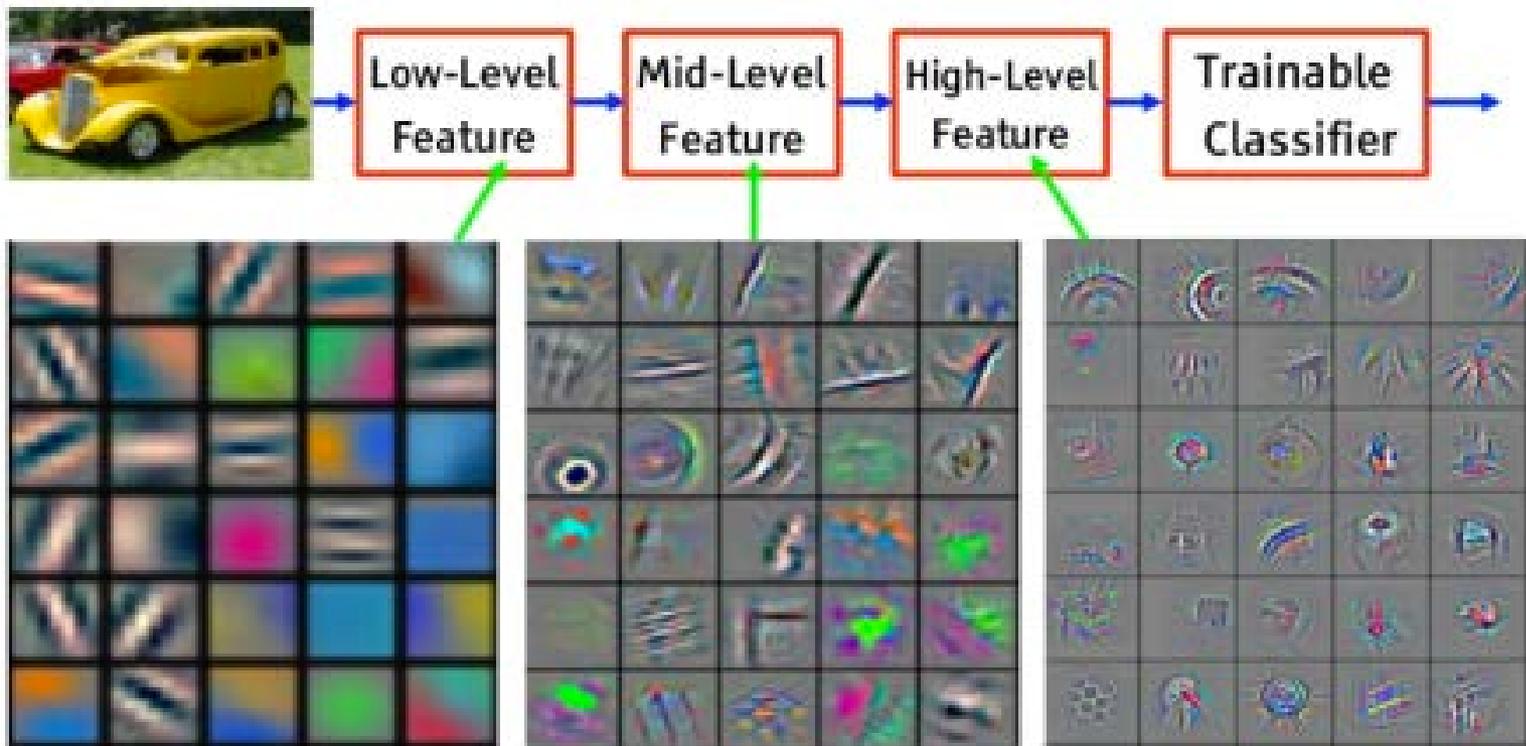
# Deep learning



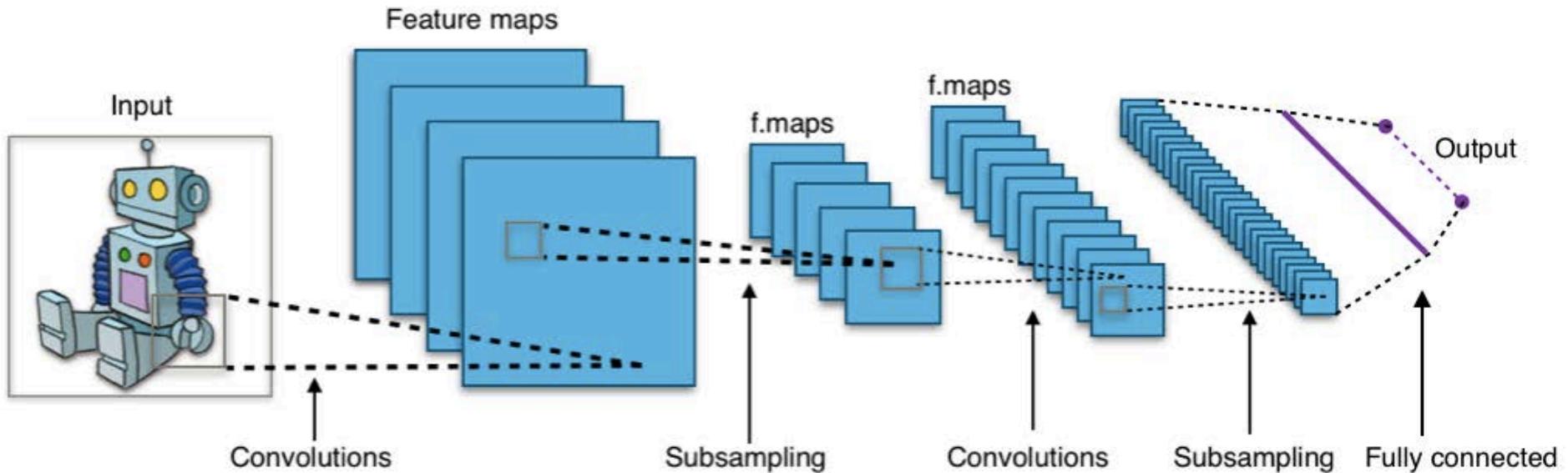
Yan Le Cun

- Les caractéristiques de l'image sont extraites automatiquement par le réseau de neurones. On parle de **Hierarchical Feature Learning**.
- **Pour ce faire on utilise généralement des Réseaux de Neurones Convolutif (CNN):**
  - La couche de **convolution** (CONV) qui traite les données d'un champ récepteur. => Permet d'extraire des caractéristiques déjà observées
  - La couche de pooling (POOL), qui permet de compresser l'information en réduisant la taille de l'image intermédiaire (souvent par **sous-échantillonnage**). => réduit la sensibilité aux bruits.
  - La couche de correction (ReLU), en référence à la fonction d'activation (Unité de rectification linéaire).=> comment le signal circule d'un neurone à l'autre.
  - La couche "entièrement connectée" (FC), qui est une couche de type **multi layer perceptron**.=> toutes les connexions entre neurones sont étudiées.
  - La couche de perte (LOSS). Elle spécifie comment l'entraînement du réseau pénalise l'écart entre le signal prévu et réel.

# Deep learning



# Deep learning



# Deep learning anecdote

- En 2012, les chercheurs du Google X Lab ont exposé un réseau de neurones gigantesque (**Google Brain**, 16000 machines, 1 milliard de connexions) à 10 millions de vidéos choisies aléatoirement sur Internet et non étiquetées (apprentissage non supervisé).
- Le réseau de neurones s'est donc progressivement modifié pour extraire automatiquement les caractéristiques des images/vidéos reçues.
- Résultat : l'un des concepts que le réseau de neurones avait trouvé et qu'il recherchait systématiquement dans les nouvelles images présentées était...

# Deep learning anecdote



Image d'un chat générée par Google Brain par apprentissage non supervisée.

# Dilemme/compromis biais-variance

- Les modèles linéaires généralisés peuvent être régularisés afin d'en diminuer la variance mais au prix de l'augmentation du biais.
- Les réseaux de neurones : la variance augmente et le biais diminue avec le nombre de couches cachées
  - Comme dans le modèle linéaire généralisé, une régularisation est généralement appliquée.
- Avec la méthode des  $k$  plus proches voisins, une valeur élevée de  $k$  conduit à un biais élevé et une variance faible.
- Avec les arbres de décision, la profondeur de l'arbre détermine la variance. Les arbres de décision sont généralement élagués pour contrôler la variance.

# Apprentissage non supervisé

- **K-moyennes**
- Cartes auto-organisatrices

# Clustering

- On distingue :
  - Les méthodes de clustering hiérarchique : utilisées lorsque l'on ne connaît pas le nombre de classes finales.
  - Les méthodes de clustering non hiérarchique : utilisées lorsque l'on connaît le nombre  $k$  de classes finales.

# K-moyennes (k-means)

- Clustering non hiérarchique (on connaît  $k$ )
- Étant donné des points et un entier  $k$ , le problème est de diviser les points en  $k$  groupes, souvent appelés partitions (*clusters*), de façon à minimiser une fonction de coût.
- L'objectif est de minimiser la distance entre les points à l'intérieur de chaque partition.

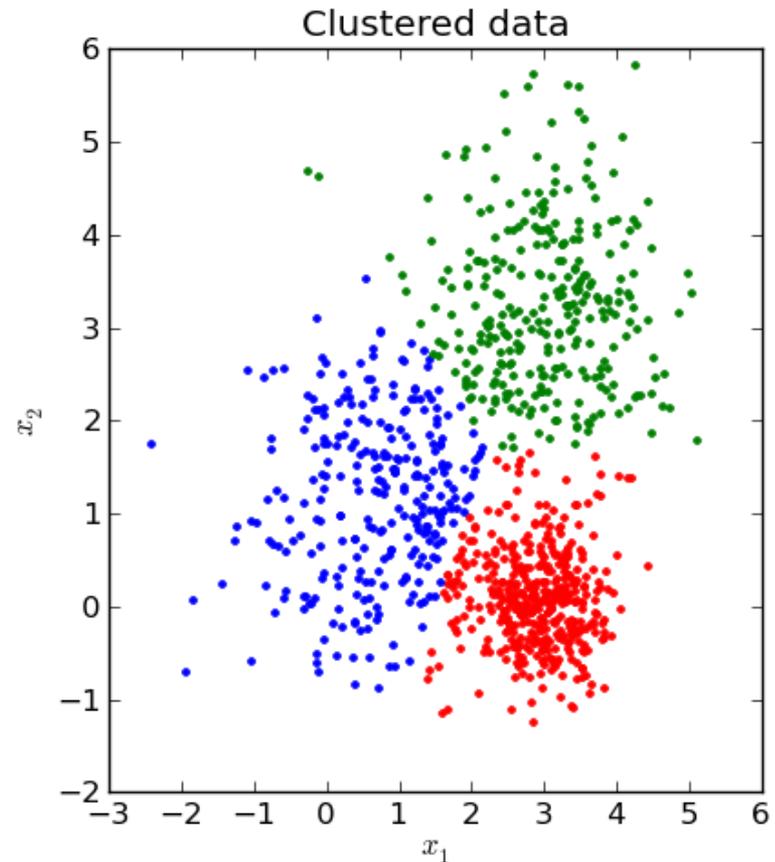
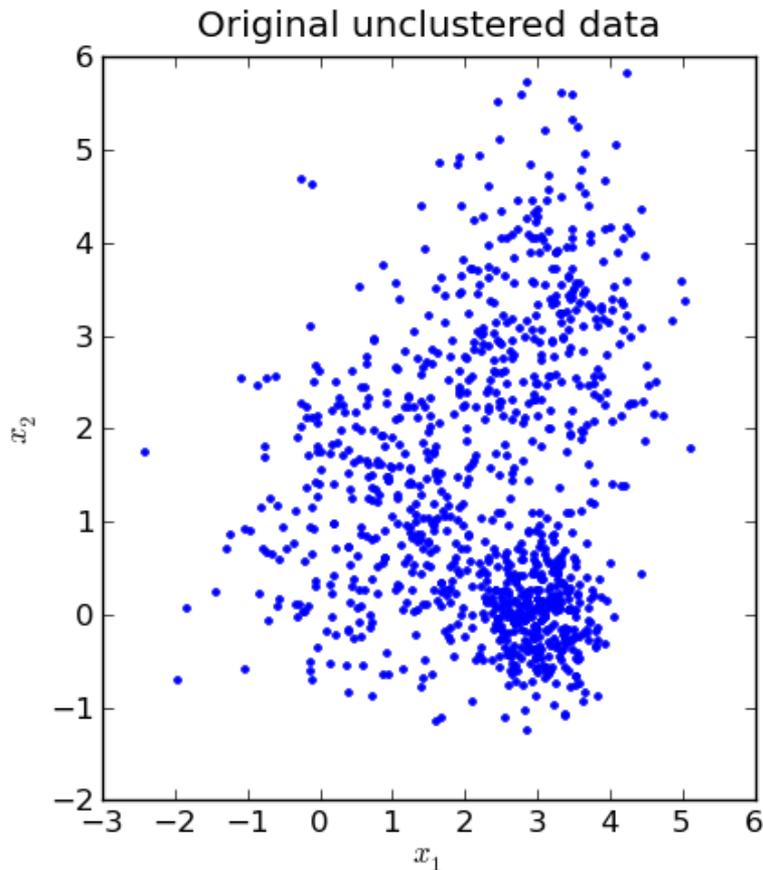
$$\operatorname{argmin} \sum_{i=1}^k \sum_{x_j \in S_i} \|x_j - \mu_i\|^2$$

Où  $\mu_i$  est le barycentre des points dans  $S_i$

# K-moyennes (k-means)

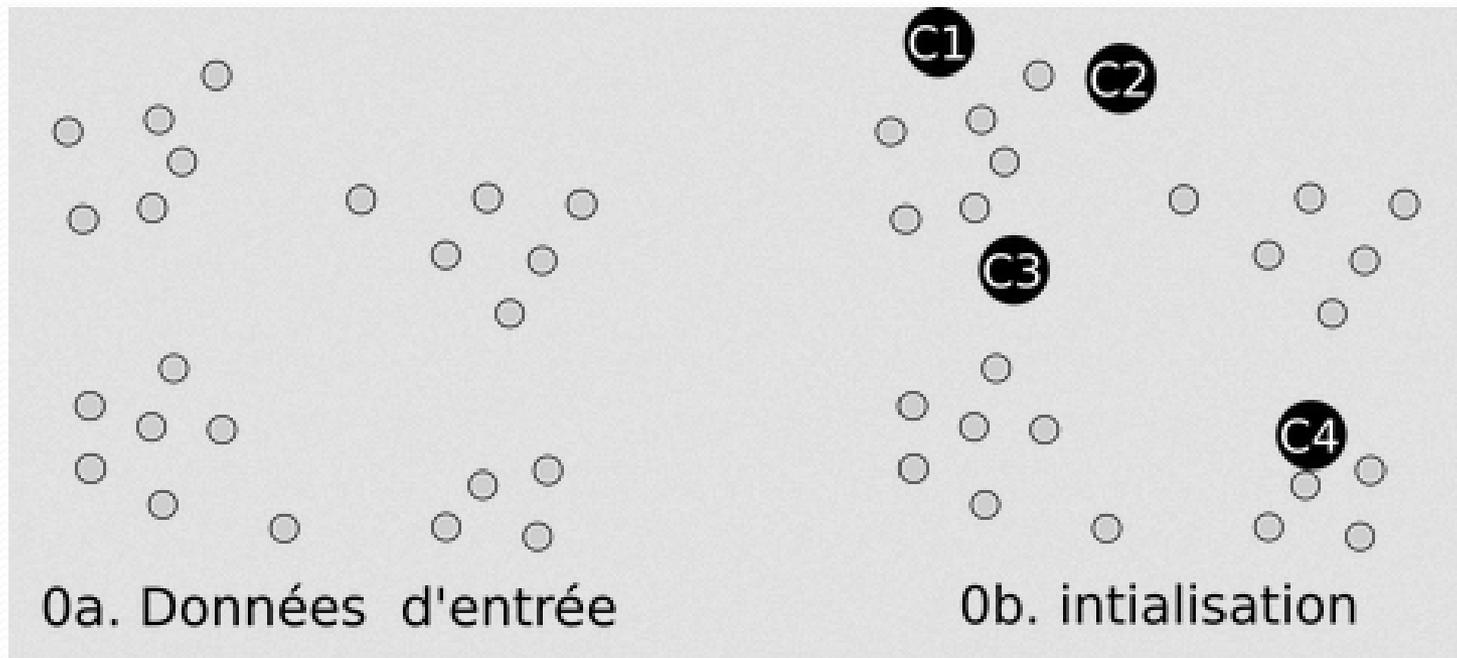
Input :  $k$ ,  $n$  observations dans un espace  $\mathbb{R}^m$ .

Ci-dessous : espace à 2 dimensions et  $k=3$



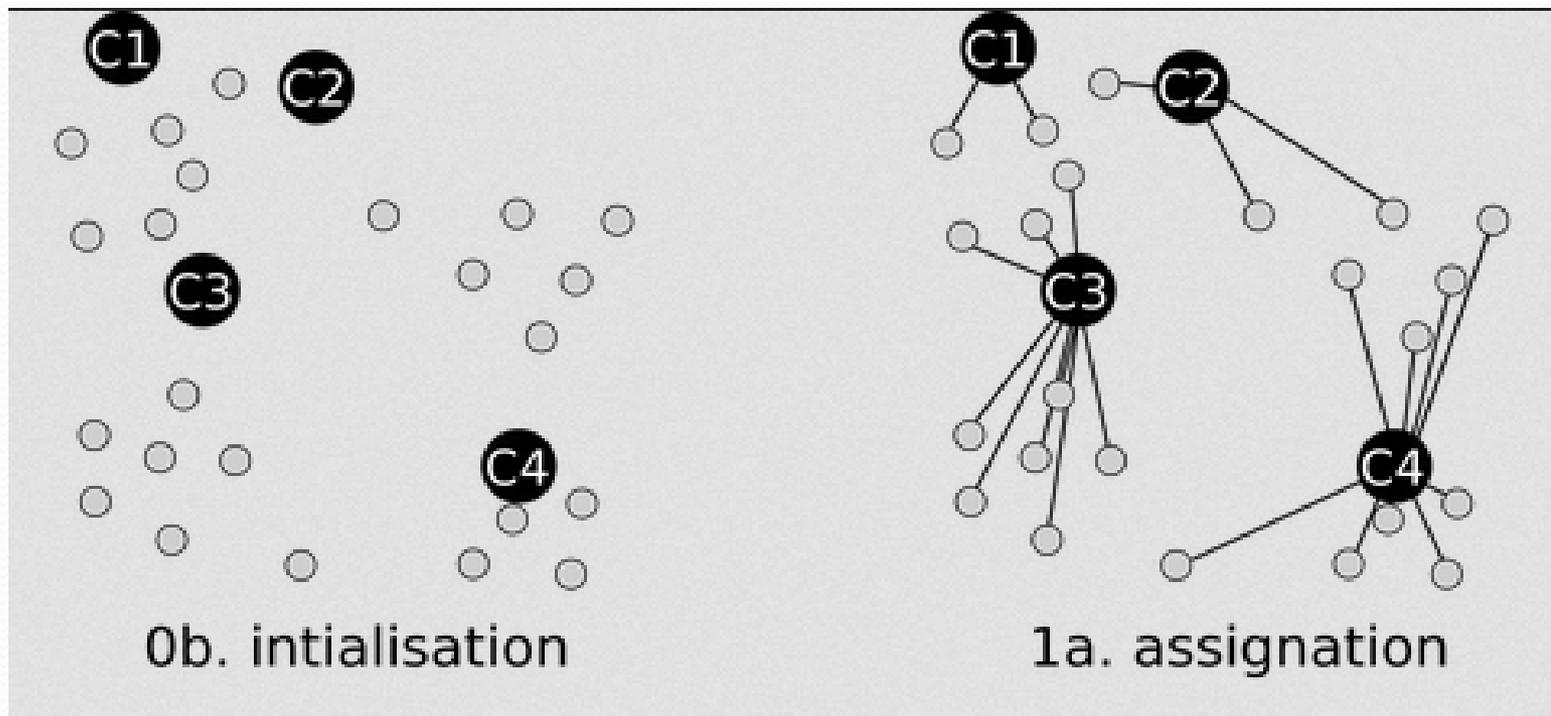
# K-moyennes (k-means)

- Algorithme
- Choisir  $k$  points qui représentent la position moyenne des partitions  $\mu_1, \mu_2, \dots, \mu_k$



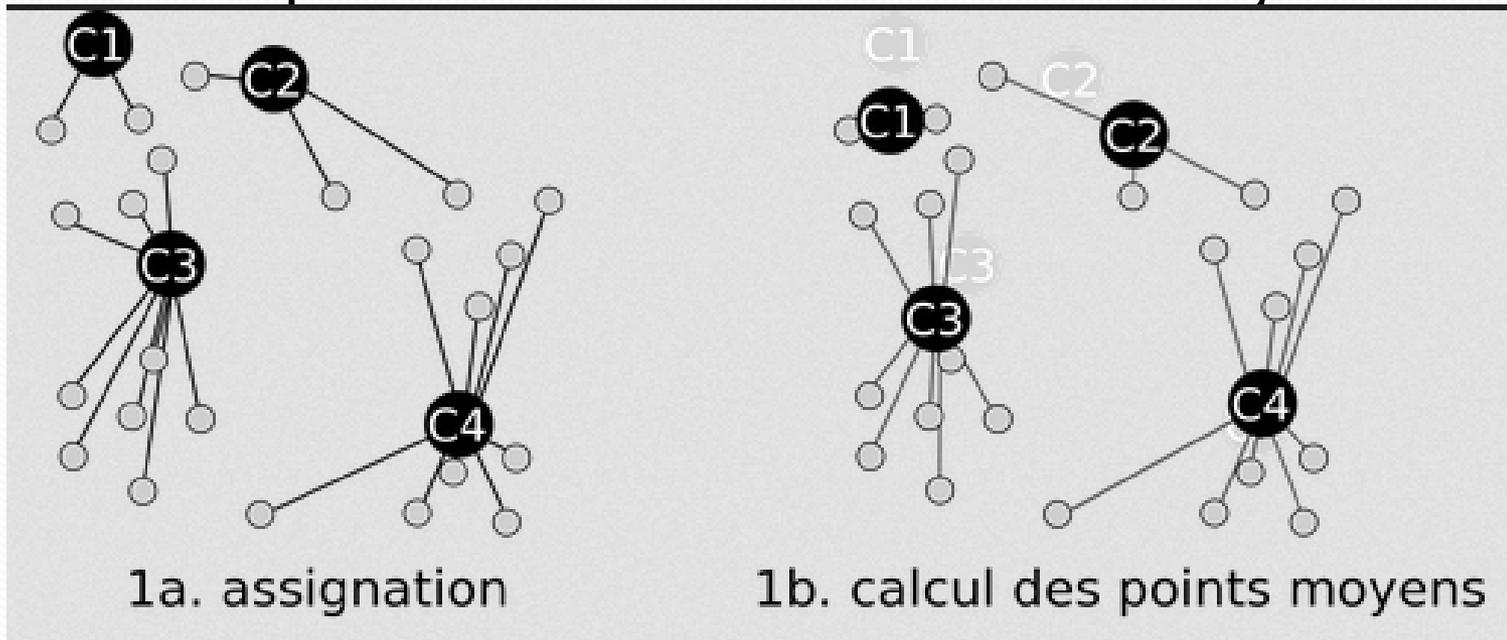
# K-moyennes (k-means)

- Répéter jusqu'à ce qu'il y ait convergence :
  - Assigner à chaque observation la partition la plus proche



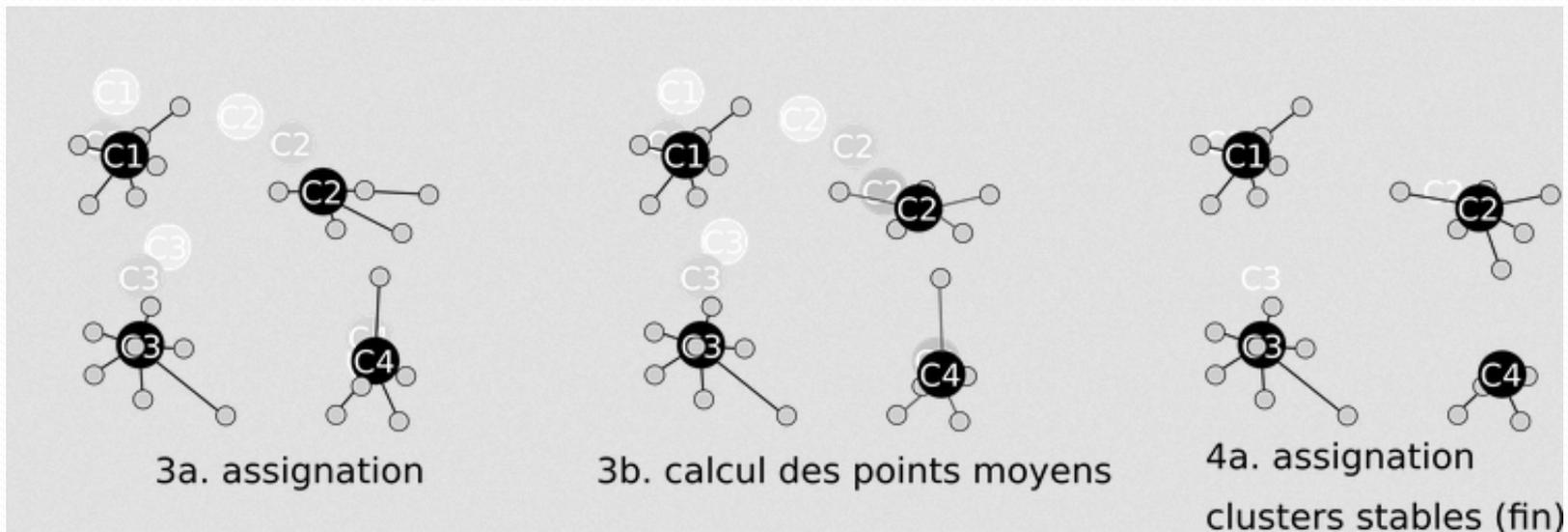
# K-moyennes (k-means)

- Répéter jusqu'à ce qu'il y ait convergence :
  - Assigner à chaque observation la partition la plus proche
  - Pour chaque cluster : calculer la nouvelle moyenne



# K-moyennes (k-means)

- Répéter jusqu'à ce qu'il y ait convergence :
  - Assigner à chaque observation la partition la plus proche
  - Pour chaque cluster : calculer la nouvelle moyenne
  - S'arrêter lorsque plus aucun assignement ne change



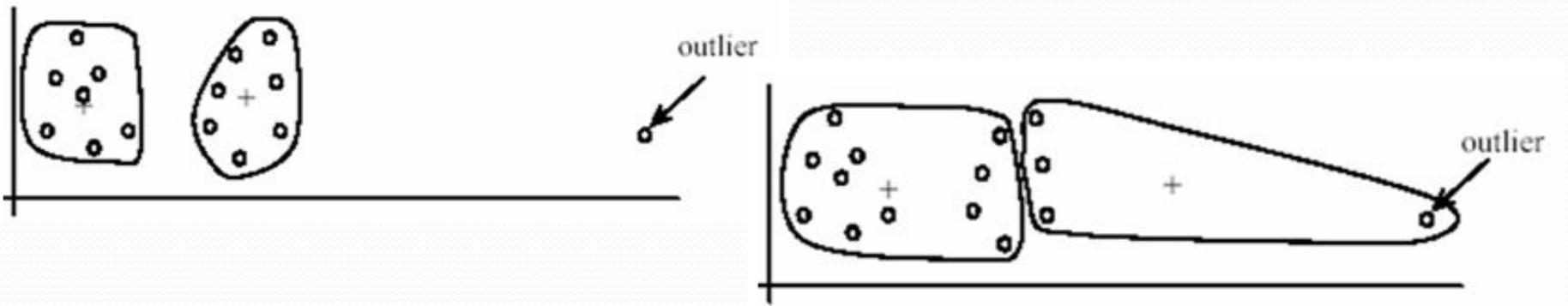
# K-moyennes (k-means)

- L'initialisation est un facteur déterminant dans la qualité des résultats (risque de minimum local).
  - Différentes méthodes d'initialisation
- Puisque  $k$  est fini, on démontre aisément que chaque itération ne fait que diminuer la fonction de coût qui est strictement positive.
  - L'algorithme converge toujours mais parfois vers un minimum local.
- Complexité polynomiale
- Obligation d'avoir une même variance sur chaque axe.

# K-moyennes (k-means)

- CONS :

- k peut-il être fixé a priori ?
  - Puisqu'on ne connaît pas les données, on ne connaît pas le nombre de clusters !
- Très sensible aux valeurs absurdes (outliers)



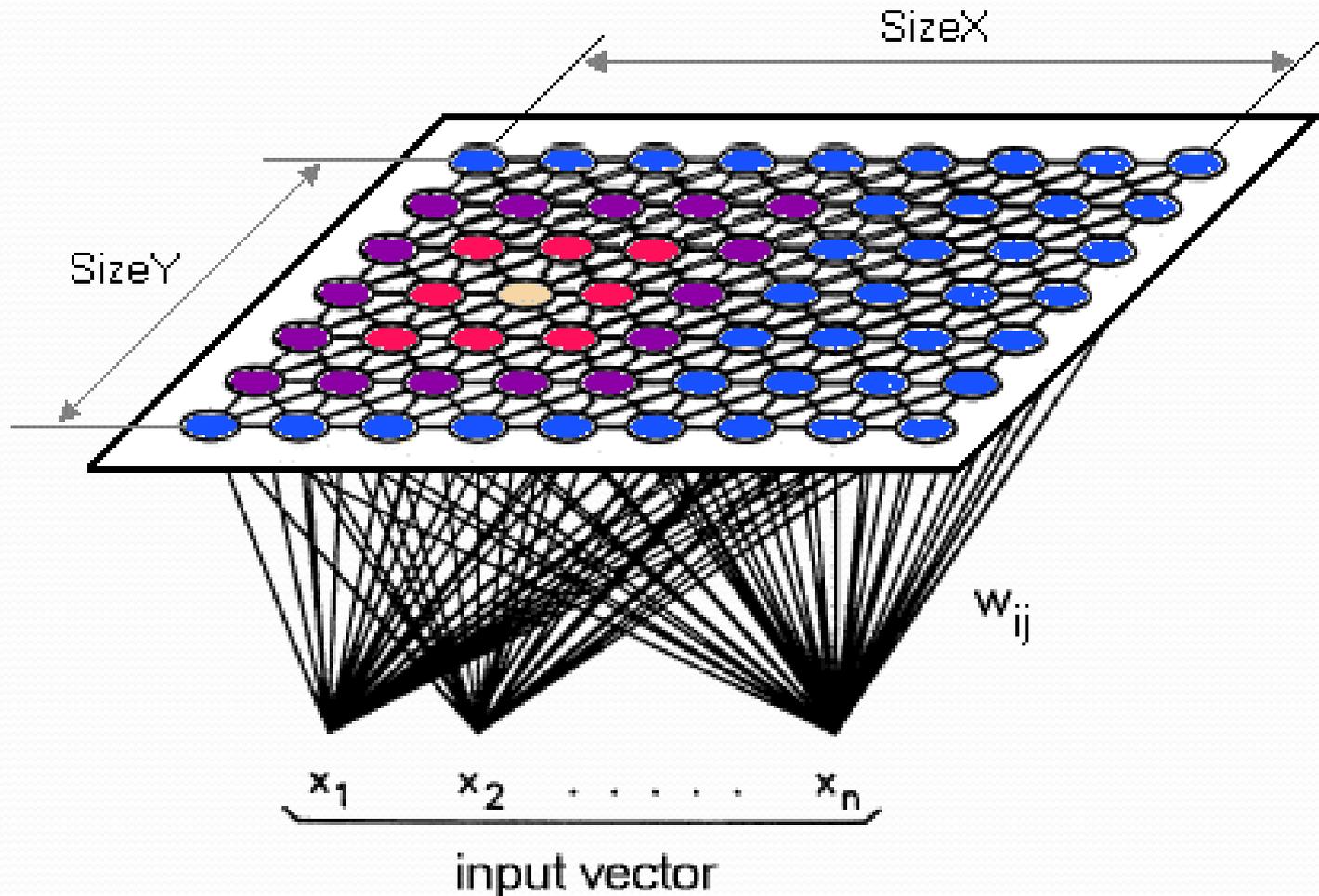
# Apprentissage non supervisé

- K-moyennes
- **Cartes auto-organisatrices**

# Cartes auto-organisatrices de Kohonen (SOM)

- Réseau de neurones fondé sur des méthodes d'apprentissage **non-supervisées**.
- Chaque neurone est connecté aux entrées et les neurones ne sont pas connectés entre eux.
- Les neurones sont placés sur une grille carré.
- Chaque neurone possède un poids par variable d'entrée.

# Cartes auto-organisatrices de Kohonen (SOM)



# Cartes auto-organisatrices de Kohonen (SOM)

- **Objectif** : trouver le neurone qui s'ajuste le mieux à chaque point observé.
  - Les points similaires vont donc naturellement se raccorder au même neurone.
- **Algorithme**
- On initialise chaque poids de chaque neurone par une variable aléatoire entre 0 et 1.
- Tant que l'on n'a pas atteint le nombre limite d'itération.
  - $k := \text{indice de l'itération}$ 

Soit la  $k$ ème observation avec ses  $n$  paramètres (chaque paramètre correspondant à une V.A.). L'observation est tirée aléatoirement dans le dataset.

$$X(k) = [x_1, \dots, x_n]$$
  - On trouve le neurone dont les poids sont les plus proches de l'observation. Ce neurone est appelé Best Machine Unit (**BMU**).
  - La distance est calculée par la distance euclidienne classique :

$$d_j = \sqrt{\sum_{i=1}^n (x_i - w_{ij})^2}, j \text{ index of unit, } i \text{ index of feature}$$

# Cartes auto-organisatrices de Kohonen (SOM)

- Comme pour un réseau de neurones classique, on met à jour les poids du neurone BMU et de chaque neurone. Plus un nœud est proche de la BMU, plus la mise à jour de ses poids sera importante.

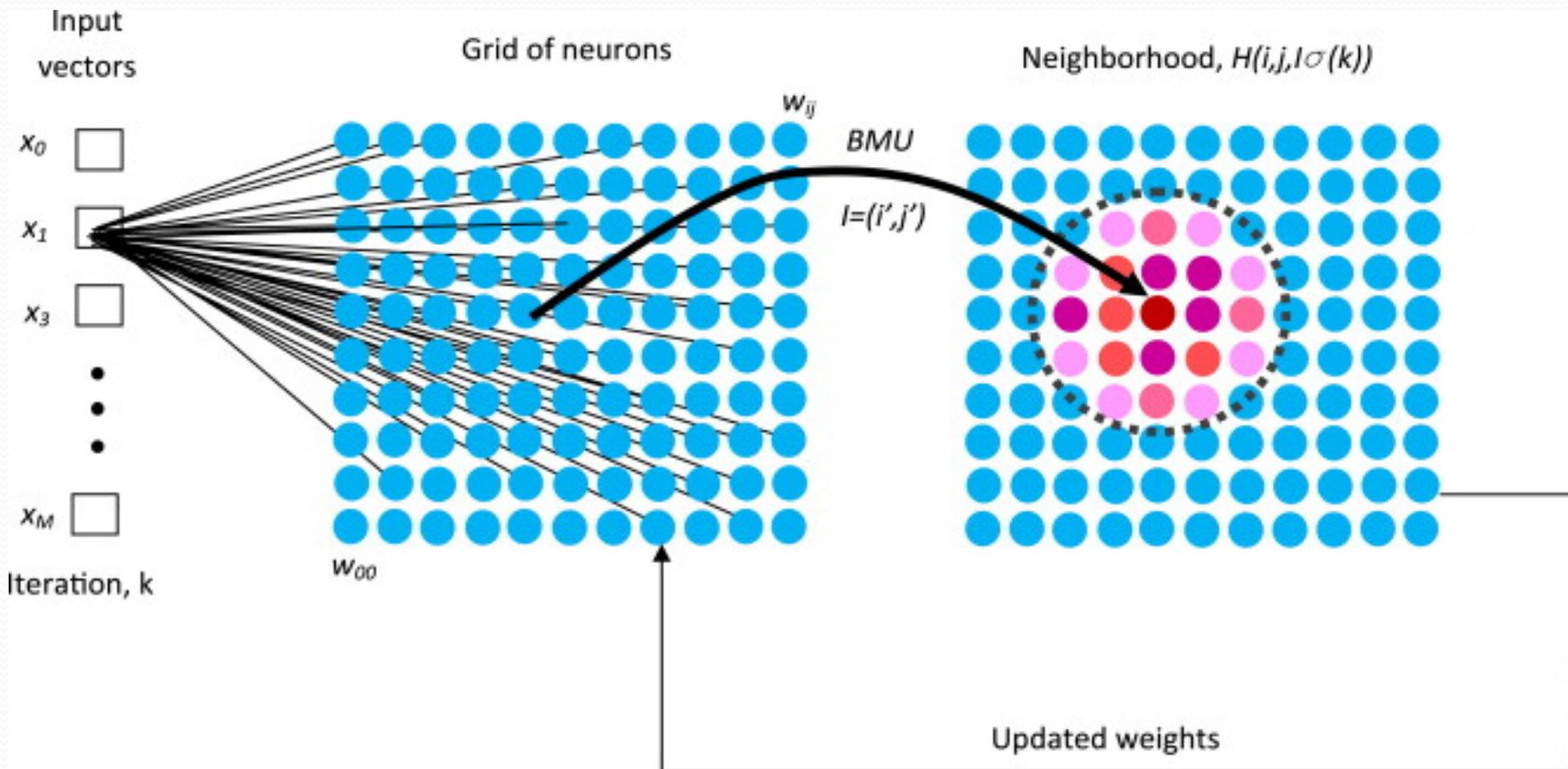
$$W_{ij}(k + 1) = W_{ij}(k) + \eta(k) * H_j(k) * (x_i(k) - W_{ij}(k))$$

- $\eta(k)$  => coefficient d'apprentissage (variant à chaque itération)
- $H_j(k)$  => fonction de voisinage, typiquement variant comme une gaussienne.

$$H_j(k) = \text{fonction de voisinage} = e^{-\frac{\|r_j - r_{BMU}\|^2}{\sigma(k)^2}}$$

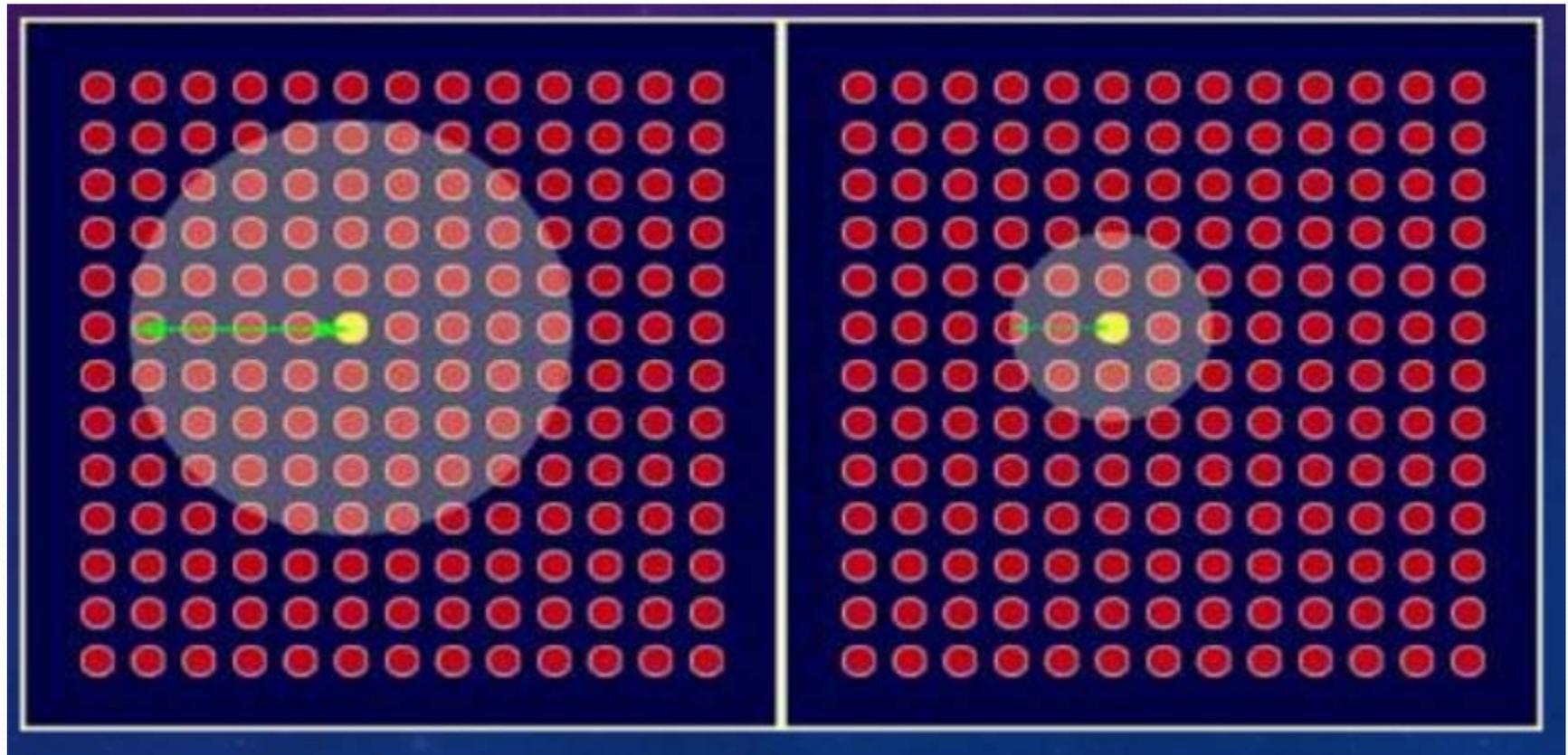
$\sigma(k)$  = coefficient de voisinage, fonction de l'itération

# Cartes auto-organisatrices de Kohonen (SOM)



# Cartes auto-organisatrices de Kohonen (SOM)

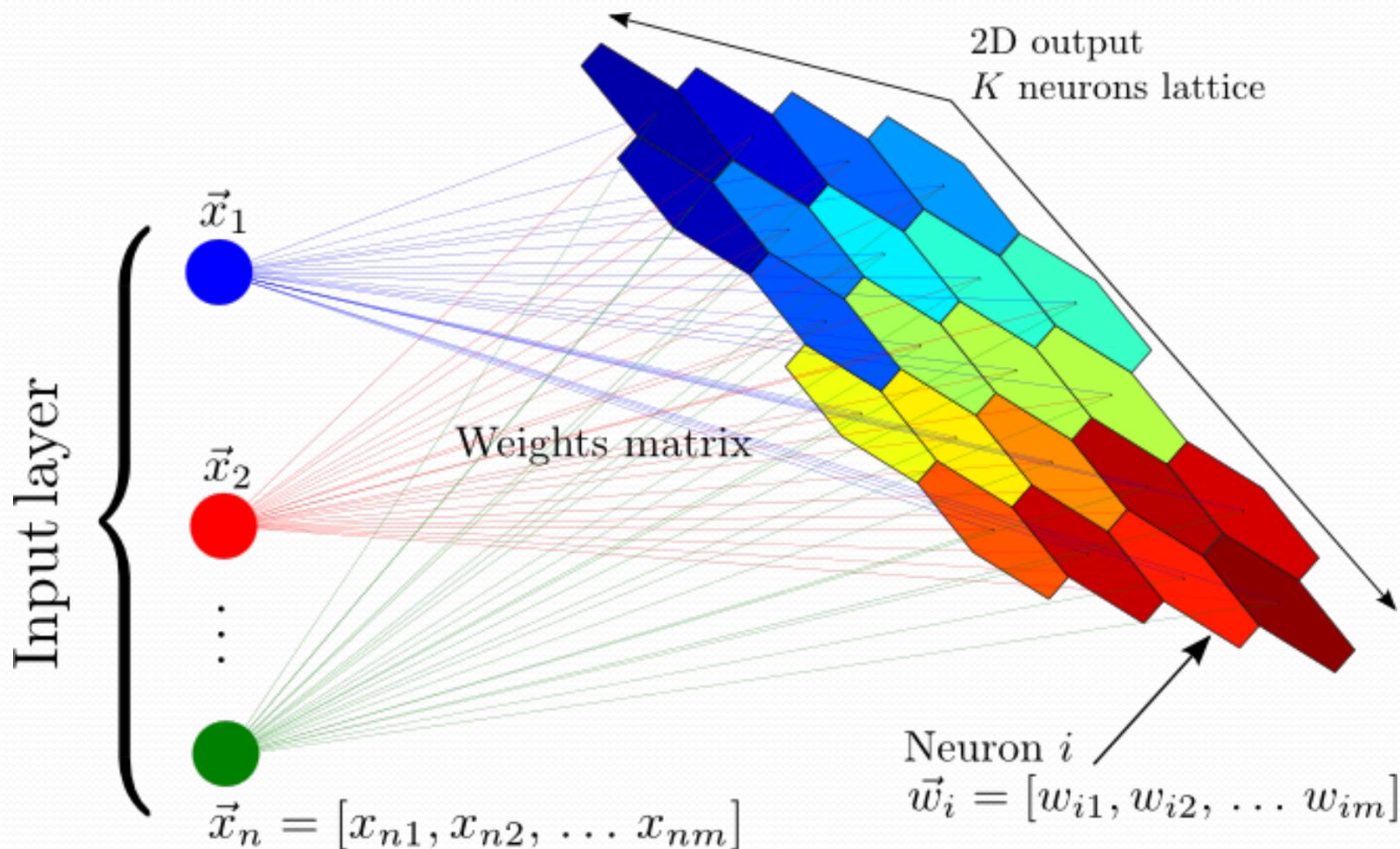
L'aire d'influence du voisinage décroît en fonction de l'itération



$k=1$

$k=100$

# Cartes auto-organisatrices de Kohonen (SOM)



# Cartes auto-organisatrices de Kohonen (SOM)

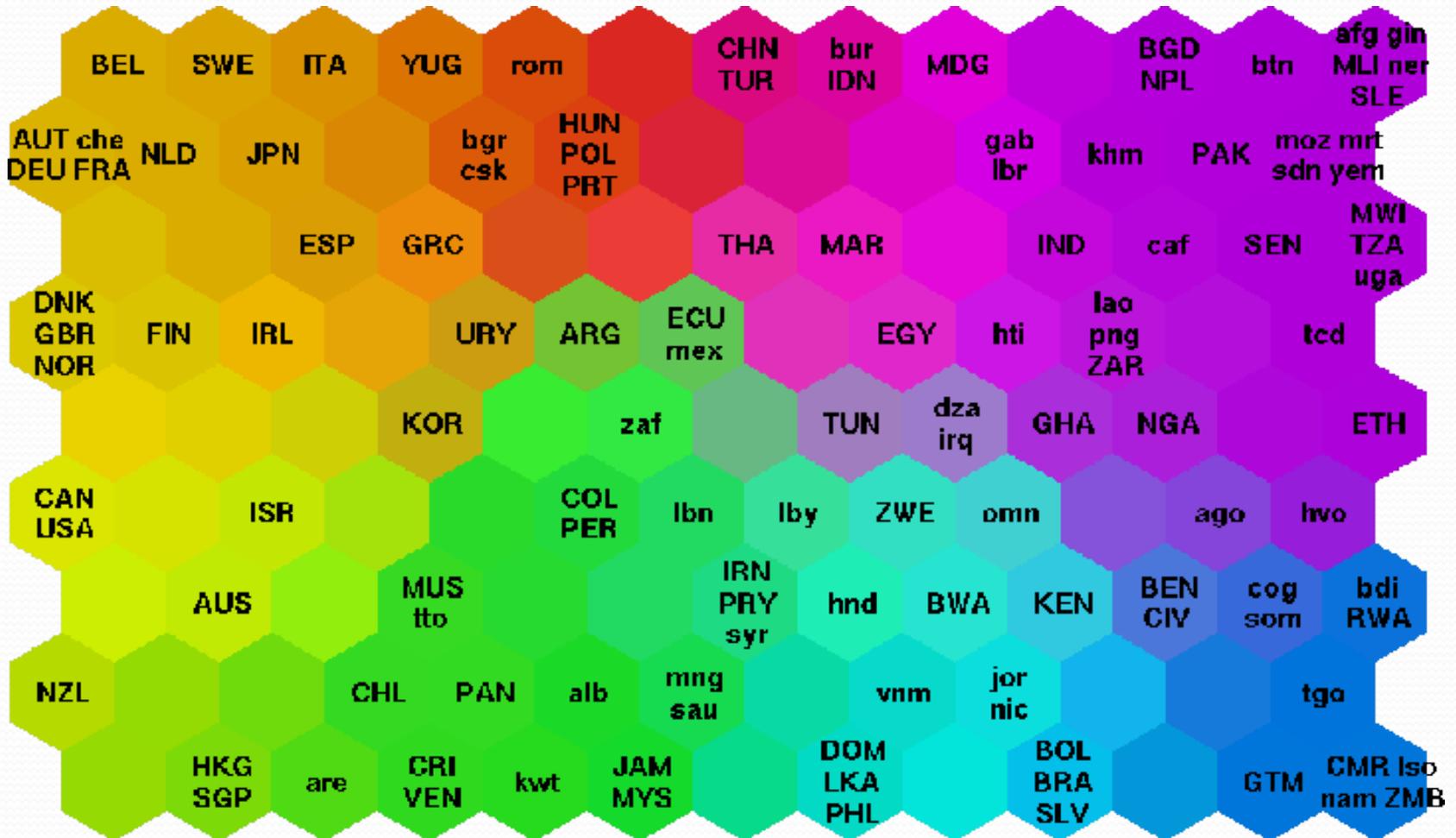
- Au final, les observations ayant des entrées similaires seront toutes rattachées à des neurones voisins.
- Les cartes auto-organisatrices pourront à la fois servir :
  - En partitionnement (clustering)
  - En classification (si on injecte une nouvelle donnée).

# Cartes auto-organisatrices de Kohonen (SOM)

- Exemple :
  - Utilisation d'un jeu de données des statistiques de la banque mondiale (1992).
  - 39 paramètres utilisés décrivant :
    - des facteurs de qualité de vie, état de santé, nutrition, système éducatif, etc.
    - jamais la notion de richesse ou de pauvreté n'apparaît explicitement dans un paramètre.

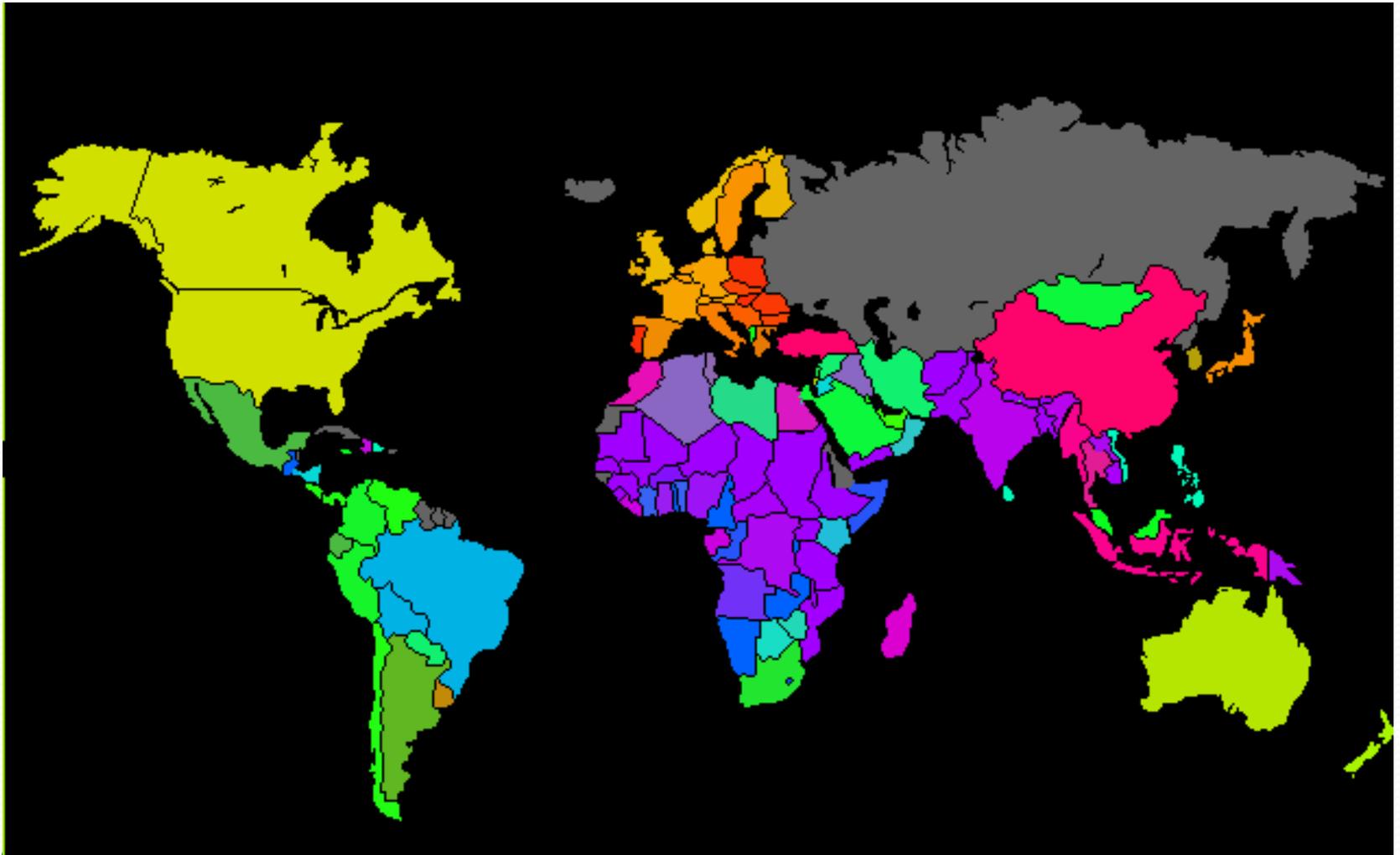
# Cartes auto-organisatrices de Kohonen (SOM)

Résultats

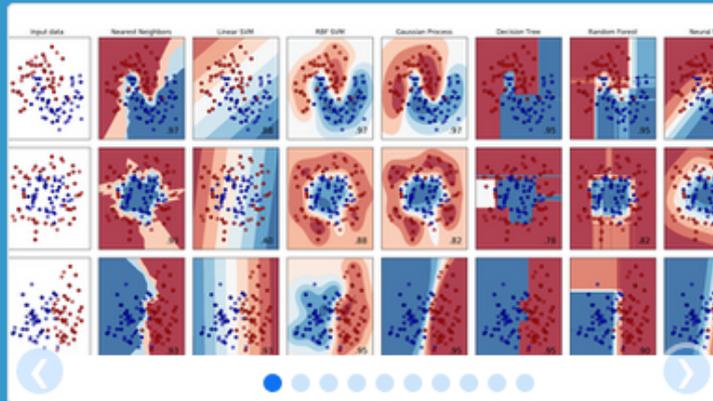


# Cartes auto-organisatrices de Kohonen (SOM)

L'ordinateur a retrouvé le concept de répartition géographique de la pauvreté/richeesse !







# scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

## Classification

Identifying to which category an object belongs to.

**Applications:** Spam detection, Image recognition.

**Algorithms:** SVM, nearest neighbors, random forest, ... — Examples

## Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.

**Algorithms:** SVR, ridge regression, Lasso, ... — Examples

## Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes

**Algorithms:** k-Means, spectral clustering, mean-shift, ... — Examples

## Dimensionality reduction

Reducing the number of random variables to consider.

**Applications:** Visualization, Increased efficiency

**Algorithms:** PCA, feature selection, non-negative matrix factorization. — Examples

## Model selection

Comparing, validating and choosing parameters and models.

**Goal:** Improved accuracy via parameter tuning

**Modules:** grid search, cross validation, metrics. — Examples

## Preprocessing

Feature extraction and normalization.

**Application:** Transforming input data such as text for use with machine learning algorithms.

**Modules:** preprocessing, feature extraction. — Examples

# Séries temporelles

- **Modèle ARMA**
- Séries temporelles et Machine Learning

# Séries temporelles

- **Séries temporelles**

- Analyse dans le domaine fréquentiel
  - Transformée de Fourier, Ondelettes
- Analyse dans le domaine temporel
  - Modèle d'autocorrélation

- **Modèle ARMA** (Analyse dans le domaine temporel )

- Objectif : Modéliser la série temporelle pour prédire les valeurs suivantes.
- La série temporelle est considérée comme un processus stochastique, c.-à-d. où chaque observation est considérée comme une variable aléatoire.
- Principe : La valeur cible évolue en fonction du temps avec généralement une forte corrélation entre une valeur et les valeurs immédiatement précédentes.

# Modèle ARMA

- **Bruit blanc faible :**

$\varepsilon_t \sim \text{WhiteNoise}(0, \sigma^2)$  signifie :

$$E[\varepsilon_t] = 0$$

$$E[\varepsilon_t^2] = \sigma_\varepsilon^2 \neq 0 \text{ et constante}$$

$$\text{Cov}(\varepsilon_t, \varepsilon_s) = 0 \text{ si } t \neq s$$

Les variables aléatoires  $\varepsilon_t$  sont donc de moyenne nulle, de variance constante et non corrélées dans le temps.

- **Bruit blanc fort :**

- On parle de **bruit blanc fort** s'il s'agit d'un bruit blanc faible et que les V.A. sont de plus indépendantes et identiquement distribuées (i.i.d).

# Modèle ARMA

- **Modèle AutoRegressive (AR)**

- AR(p) => Une valeur à un instant t dépend linéairement des valeurs précédentes :

$$Y_t = \beta_0 + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \dots + \beta_p Y_{t-p} + \varepsilon_t$$
$$\varepsilon_t \sim \text{WhiteNoise}(0, \sigma^2)$$

- **Modèle Moving Average (MA)**

- MA(q) : On utilise uniquement les termes d'erreurs linéairement pour prédire la valeur de Y.

$$Y_t = \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \dots + \theta_p \varepsilon_{t-p} + \varepsilon_t$$

# Modèle ARMA

- ARMA(p,q)
- Une valeur à un instant t dépend des p valeurs précédentes et des q bruits blancs précédents :

$$Y_t = \beta_0 + \beta_1 Y_{t-1} + \dots + \beta_p Y_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q}$$

# Modèle ARMA

- Les paramètres d'un processus ARMA à estimer sont :
  - les coefficients  $\beta$ ;
  - les coefficients  $\theta$ ;
  - la variance  $\sigma^2$  du bruit blanc  $\varepsilon$ .

Et avant tout, l'ordre  $(p, q)$  du processus ARMA

# Modèle ARMA

## Estimation de $\mathbf{q}$

- **Fonction d'autocorrélation (ACF)**

$$\rho(k) = \text{Corr}(Y_t, Y_{t+k}) = \frac{\text{Covariance}(Y_t, Y_{t+k})}{\text{Var}(Y_t)}$$

On estime la covariance sur la série par :

$$\widehat{\text{Covariance}}(k) = \frac{\sum_{t=1}^{n-k} (Y_t - \bar{Y}_T)(Y_{t+k} - \bar{Y}_T)}{n - k}$$

$n$  = nombre d'observations

# Modèle ARMA

## Estimation de $q$

- Estimateur de la fonction d'autocorrélation (ACF)

$$ACF(k) = \hat{\rho}(k) = \frac{\widehat{Covariance}(k)}{\widehat{Var}(Y_t)}$$

- *MA d'ordre  $q$*

on montre que  $\rho(k) \begin{cases} \neq 0 & \text{Si } k \leq q \\ = 0 & \text{Si } k > q \end{cases}$

Ainsi on peut estimer  $q$  en traçant l'ACF en fonction de  $k$ .

# Modèle ARMA

## Estimation de $q$

### Autocorrélogramme

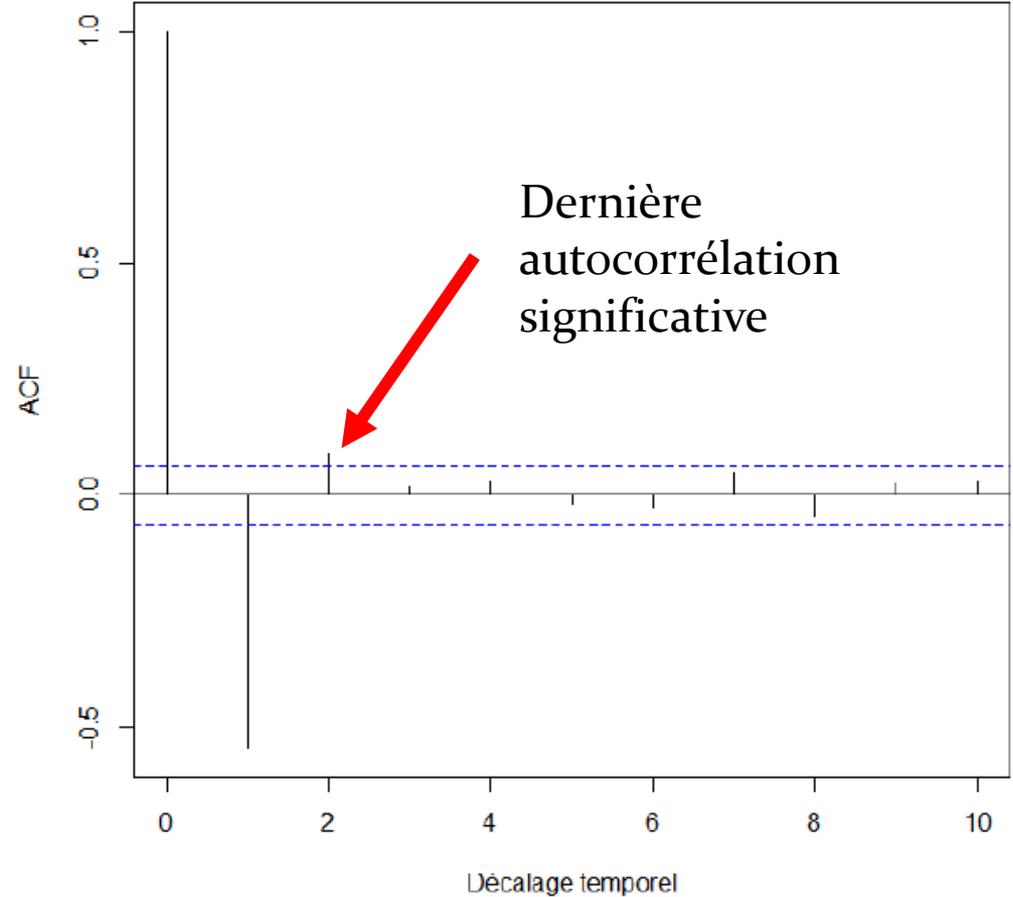
Représentation de l'ACF en fonction de  $k$ .

Les lignes pointillées indiquent la limite d'autocorrélation significativement  $> 0$ .

On voit ici qu'après  $k=2$ , la corrélation n'est plus significative.

**Nous avons affaire à un processus MA(2)**

**$q=2$**



# Modèle ARMA

## Estimation de $p$

- Pour un processus AR( $p$ ) , on ne peut rien dire de sa fonction ACF si ce n'est que :

$\rho(k) \neq 0$  pour tout  $k$ , inintéressant

- **Fonction d'autocorrélation partielle (PACF)**

- PACF( $k$ ) est la corrélation entre :

- Le résidu de la régression de la série  $Y_{t+k}$  par les séries  $Y_{t+1}, \dots, Y_{t+k-1}$ , donc  $U$  dans :

$$Y_{t+k} = \alpha_1 Y_{t+1} + \alpha_2 Y_{t+2} + \dots + \alpha_{t+k-1} Y_{t+k-1} + U$$

- Le résidu de la régression de la série  $Y_t$  par les séries  $Y_{t+1}, \dots, Y_{t+k-1}$ , donc  $V$  dans :

$$Y_t = \beta_1 Y_{t+1} + \beta_2 Y_{t+2} + \dots + \beta_{t+k-1} Y_{t+k-1} + V$$

- **$PACF(k) = Corr(U, V)$**

# Modèle ARMA

## Estimation de **p**

- *AR d'ordre p*

on montre que  $PACF(k) \begin{cases} \neq 0 & \text{Si } k \leq p \\ = 0 & \text{Si } k > p \end{cases}$

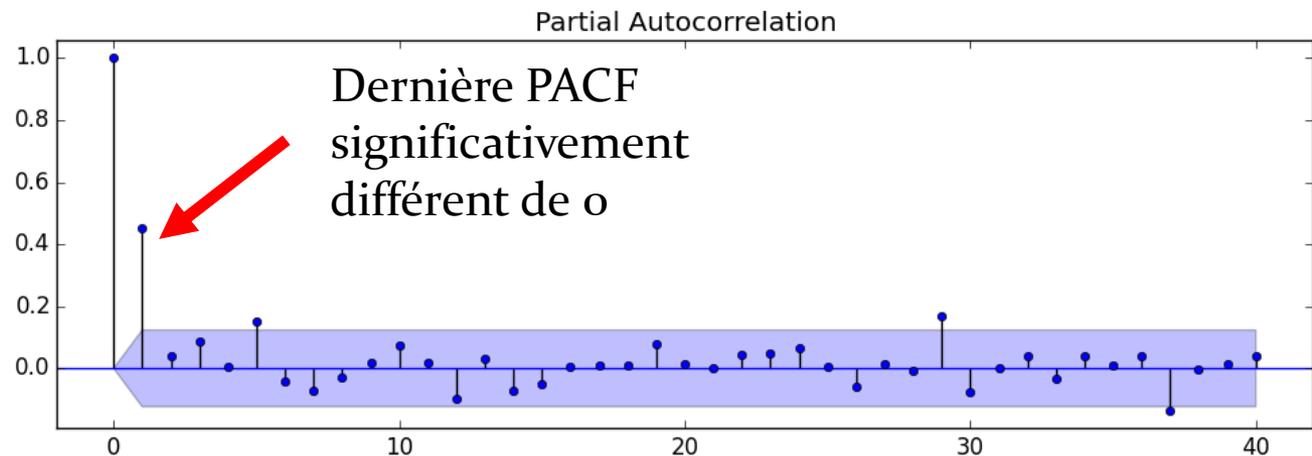
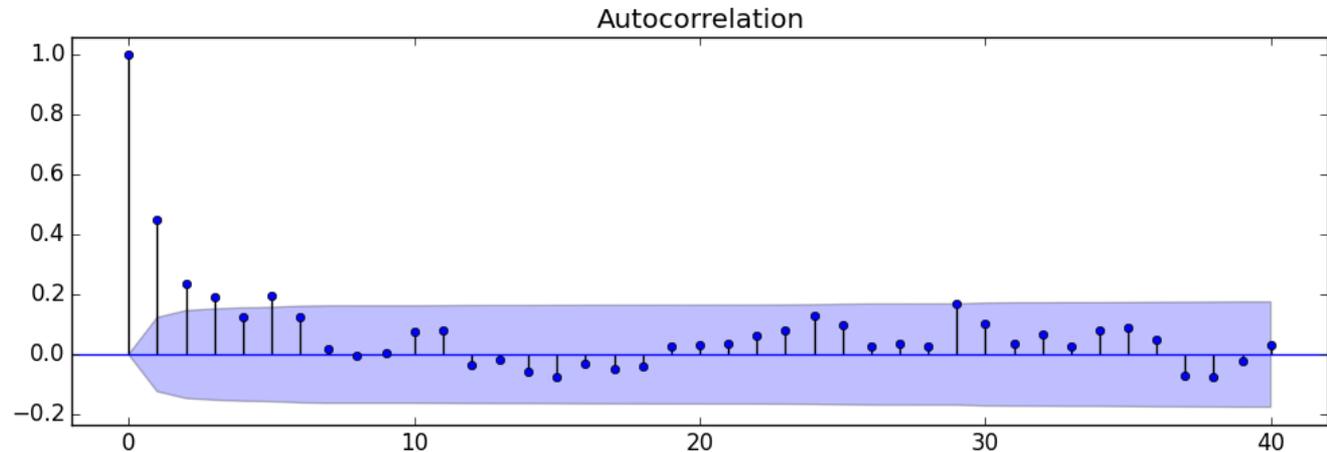
Ainsi on peut estimer p en traçant la PACF en fonction de k

- En résumé:

	AR(p)	MA(q)
ACF		=0 pour tout k>q
PACF	=0 pour tout k>p	

# Modèle ARMA

## Estimation de $p$



# Modèle ARMA

- Dans notre exemple  $p=2$  et  $q=2$ , ce qui donne le modèle suivant :

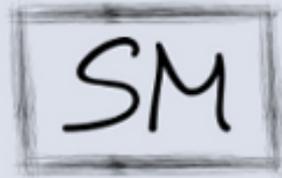
$$Y_t = \beta_0 + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2}$$

- Détermination des coefficients  $\beta$  du processus AR( $p$ )
  - Utilisation des équations de Yule-Walker
  - Soit  $\gamma_k = \text{Covariance}(Y_t, Y_{t+k})$

$$\begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_p \end{bmatrix} = \begin{bmatrix} \gamma_0 & \gamma_1 & \cdots & \gamma_{p-1} \\ \gamma_1 & \gamma_0 & \cdots & \gamma_{p-2} \\ \vdots & \vdots & \ddots & \gamma_1 \\ \gamma_{p-1} & \gamma_{p-2} & \gamma_1 & \gamma_0 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}$$

# Modèle ARMA

- Modélisation uniquement valable pour les séries stationnaires:
  - l'espérance est constante au cours du temps, il n'y a donc pas de tendance.
$$E[Y_i] = \mu, \forall t = 1 \dots t$$
  - la variance est constante au cours du temps et non infinie.
$$Var[Y_i] = \sigma^2 \neq \infty, \forall t = 1 \dots t$$
  - la position dans le temps ne joue pas de rôle sur l'autocorrélation entre 2 points qui ne dépend que de l'ampleur du décalage k
$$Cov[Y_i, Y_{i-k}] = \rho_k, \forall t = 1 \dots t, \forall k = 1 \dots t$$
- Séries non stationnaires => Modèle **ARIMA**



StatsModels  
Statistics in Python

Install | Support | Bugs | Develop | Examples | FAQ |

## ARMA Process

The following are tools to work with the theoretical properties of an ARMA process for given lag-polynomials.

<code>arma_process.ArmaProcess(ar, ma[, nobs])</code>	Represent an ARMA process for given lag-polynomials
<code>arma_process.ar2arma(ar_des, p, q[, n, ...])</code>	find arma approximation to ar process
<code>arma_process.arma2ar(ar, ma[, nobs])</code>	get the AR representation of an ARMA process
<code>arma_process.arma2ma(ar, ma[, nobs])</code>	get the impulse response function (MA representation) for ARMA process
<code>arma_process.arma_acf(ar, ma[, nobs])</code>	theoretical autocorrelation function of an ARMA process
<code>arma_process.arma_acovf(ar, ma[, nobs])</code>	theoretical autocovariance function of ARMA process
<code>arma_process.arma_generate_sample(ar, ma, ...)</code>	Generate a random sample of an ARMA process
<code>arma_process.arma_impulse_response(ar, ma)</code>	get the impulse response function (MA representation) for ARMA process
<code>arma_process.arma_pacf(ar, ma[, nobs])</code>	partial autocorrelation function of an ARMA process
<code>arma_process.arma_periodogram(ar, ma[, ...])</code>	periodogram for ARMA process given by lag-polynomials ar and ma
<code>arma_process.deconvolve(num, den[, n])</code>	Deconvolves divisor out of signal, division of polynomials for n terms
<code>arma_process.index2lpol(coeffs, index)</code>	expand coefficients to lag poly
<code>arma_process.lpol2index(ar)</code>	remove zeros from lagpolynomial, squeezed representation with index
<code>arma_process.lpol_fiar(d[, n])</code>	AR representation of fractional integration
<code>arma_process.lpol_fima(d[, n])</code>	MA representation of fractional integration
<code>arma_process.lpol_sdiff(s)</code>	return coefficients for seasonal difference (1-L <sup>s</sup> )
<code>sandbox.tsa.fftarma.ArmaFft(ar, ma, n)</code>	fft tools for arma processes

# Séries temporelles

- Modèle ARMA
- **Séries temporelles et Machine Learning**

# Machine Learning et séries temporelles

- Ajout d'une ou plusieurs colonnes décrivant le temps dans le training set.
- On pourrait également imaginer faire une Transformée de Fourier de la série et ajouter des colonnes correspondant à des poids évoluant en fonction des fréquences les plus significatives.
- Puis exécution classique de l'algorithme de ML pour prévision.

# Conclusion

# En astrophysique

- Supervisé : Reconnaissance de forme sur image (sursaut, ...)
- Non supervisé : partitionnement de plusieurs types d'objets dans un ensemble

# Bibliographie

- The Elements of Statistical Learning, Data Mining, Inference, and Prediction – (Hastie, Tibshirani, Friedman)
- Apprentissage statistique, Gérard Dreyfus et al., Eyrolles, 2008
- Understanding the Bias-Variance Tradeoff (Scott Fortmann-Roe)
- Le perceptron : Prépas Dupuy de Lôme
- Indice de GINI :  
[https://www.researchgate.net/post/How\\_to\\_compute\\_impurity\\_using\\_Gini\\_Index](https://www.researchgate.net/post/How_to_compute_impurity_using_Gini_Index)

# Annexes

# Arbre de décision

- Algorithme d'approximation rapide de l'indice de GINI (donnant la moitié de l'indice ?)
- On montre que le **réduction d'impureté** de l'attribut A prenant d modalités distinctes vaut :

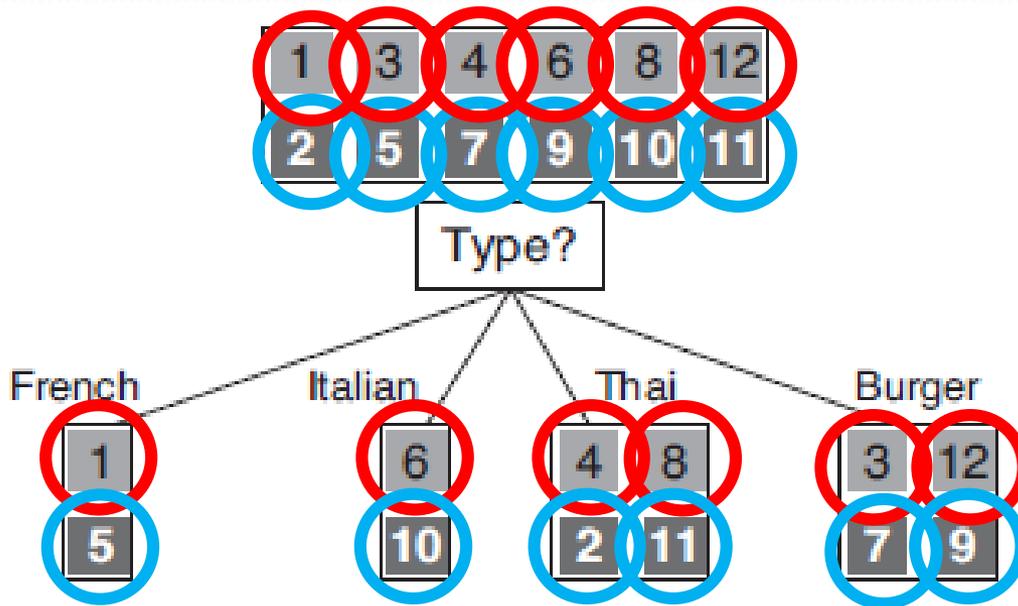
$$\text{réduction d'impureté}(A) = GINI(\text{Système}) - GINI(A)$$

- Pour une variable cible prenant 2 valeurs distinctes n et p la réduction d'impureté de GINI est approximable par :

$$GINI(\text{Système}) = P(p) * P(n) = \frac{p}{p+n} * \frac{n}{p+n} = \frac{p * n}{(p+n)^2}$$

$$\text{réduction}(A) = \frac{p * n}{(p+n)^2} - \sum_{k=1}^d \frac{p_k + n_k}{p+n} * \left( \frac{p_k}{p_k + n_k} * \frac{n_k}{p_k + p_k} \right)$$

# Arbre de décision



On retrouve comme pour l'entropie, un gain d'information de 0 pour l'attribut « Type »

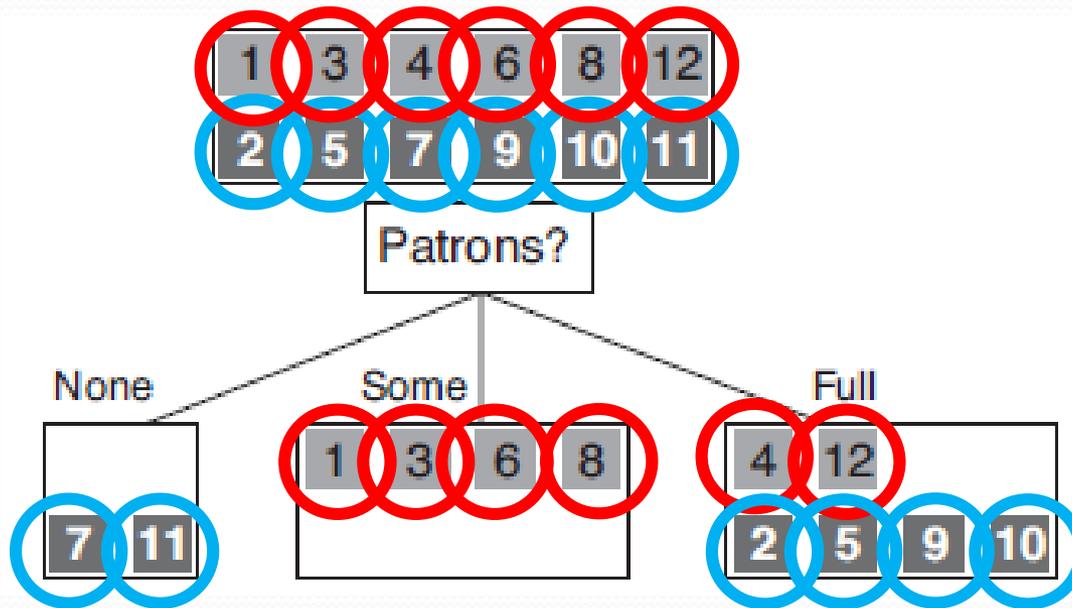
Réduction d'impureté (Type)

$$= \frac{36}{12^2} - \left[ \frac{2}{12} * \left( \frac{1}{2} * \frac{1}{2} \right) + \frac{2}{12} * \left( \frac{1}{2} * \frac{1}{2} \right) + \frac{4}{12} * \left( \frac{2}{4} * \frac{2}{4} \right) + \frac{4}{12} * \left( \frac{2}{4} * \frac{2}{4} \right) \right]$$

In [19]: 36./12\*\*2 - (2\*2/12.\*1/2.\*1/2.+2\*4/12.\*2/4.\*2/4.)

Out [19]: 0.0

# Arbre de décision



*Réduction d'impureté (Patrons)*

$$= \frac{36}{12^2} - \left[ \frac{2}{12} * \left( \frac{0}{2} * \frac{2}{2} \right) + \frac{4}{12} * \left( \frac{4}{4} * \frac{0}{4} \right) + \frac{6}{12} * \left( \frac{2}{6} * \frac{4}{6} \right) \right]$$

```
In [18]: 36./12**2 - (6/12.*2/6.*4/6.)
```

```
Out[18]: 0.13888888888888889
```

# TODO

- PCA