# Introduction au Machine learning
# Classification supervisée

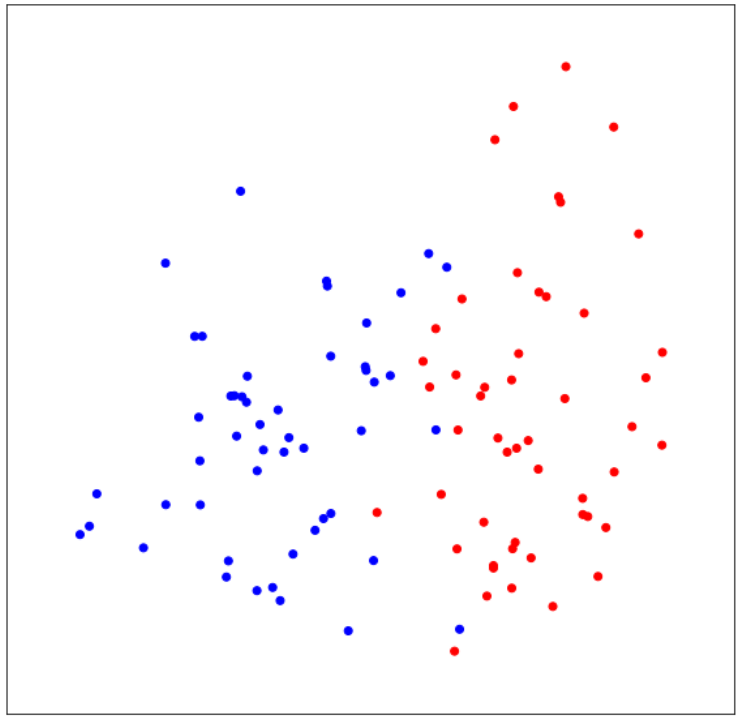
Agathe Guilloux

# Introduction

# Spam detection


- Données : emails
- Input : email
- Output : Spam or No Spam

# Classification binaire : toy datasets



- But : retrouver la classe
- Input : 2 predicteurs
- Output : classe

# Classification multi-classes

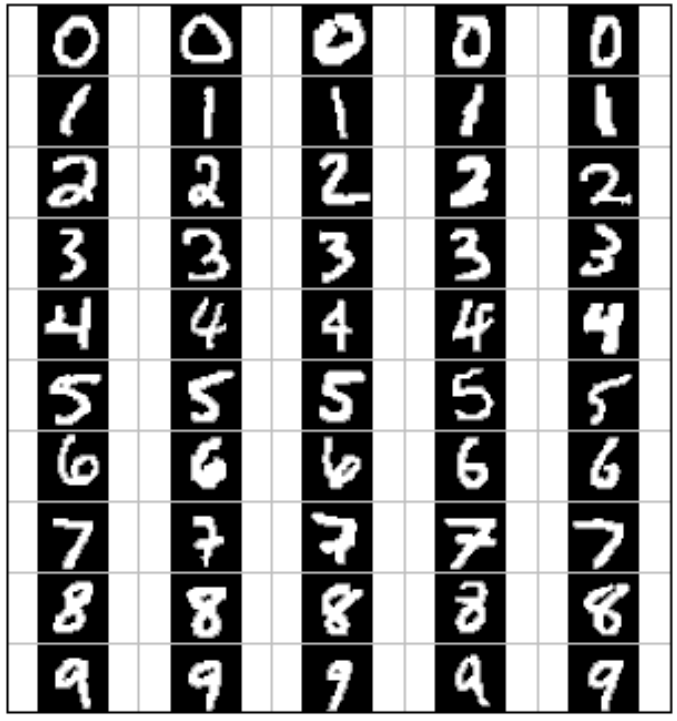
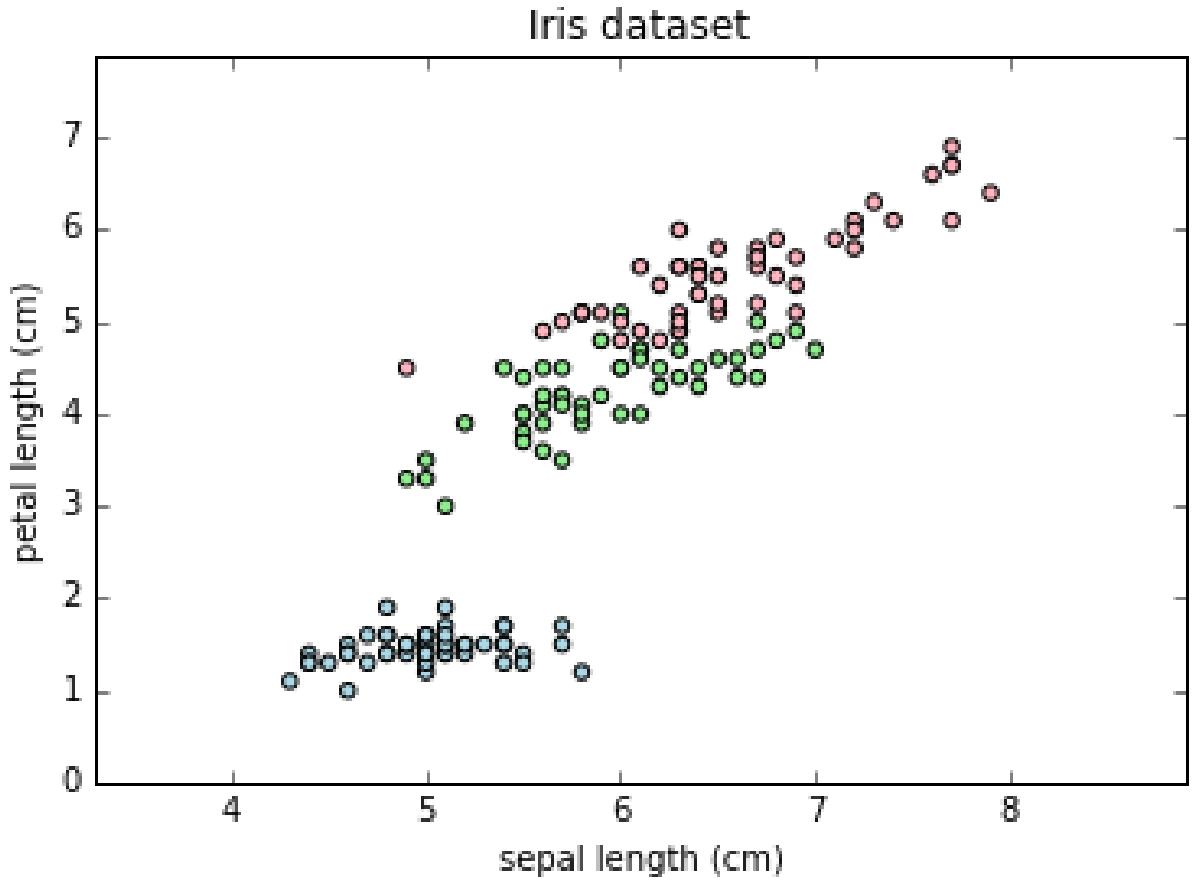


Figure 1: Jeu de données MNIST

- Lire un code postal sur une enveloppe.
- But : assigner un chiffre à une image.
- Input : image.
- Output : chiffre correspondant.

# Classification multi-classes : Iris dataset




- But : retrouver la classe
- Input : 2 predicteurs
- Output : classe

# Le problème de classification binaire

On a des données d'apprentissage (learning data) pour des individus $i = 1, \ldots, n$. Pour chaque individu $i$ :

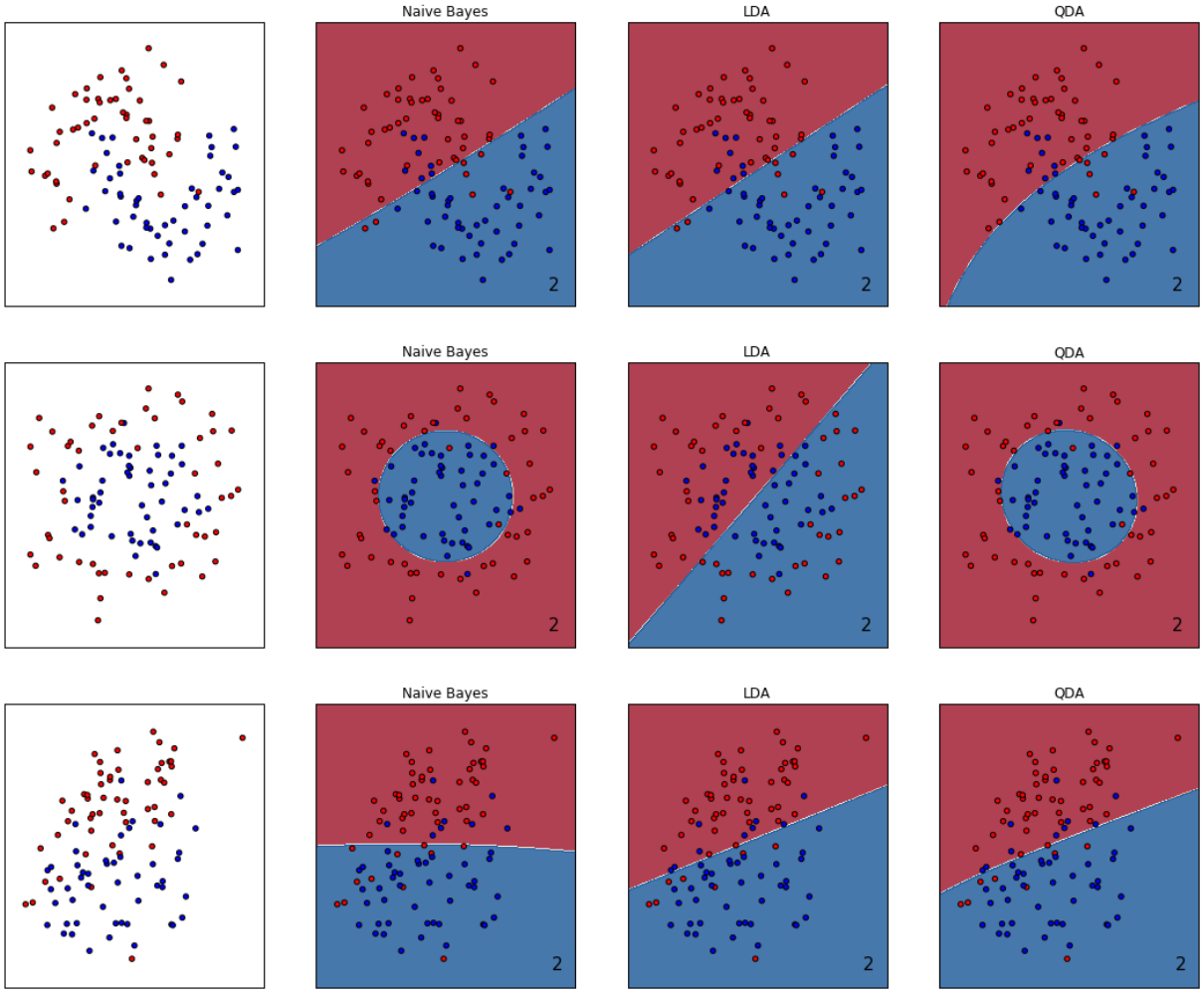
- on a un vecteur de covariables (features) $X_i \in \mathcal{X} \subset \mathbb{R}^d$
- la valeur de son label $Y_i \in \{-1, 1\}$.
- on suppose que les couples $(X_i, Y_i)$ sont des copies i.i.d. de $(X, Y)$ de loi inconnue.

**But**

- On veut pour un nouveau vecteur $X_+$ de features prédire la valeur du label $Y_+$ par $\hat{Y}_+ \in \{-1, 1\}$
- Pour cela, on utilise les données d'apprentissage $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ pour construire un **classifieur** $\hat{c}$ de telle sorte que

$$\hat{Y}_+ = \hat{c}(X_+).$$

# Classification binaire : toy datasets



# Le problème de classification multi-classes

On a des données d'apprentissage (learning data) pour des individus $i = 1, \ldots, n$. Pour chaque individu $i$ :

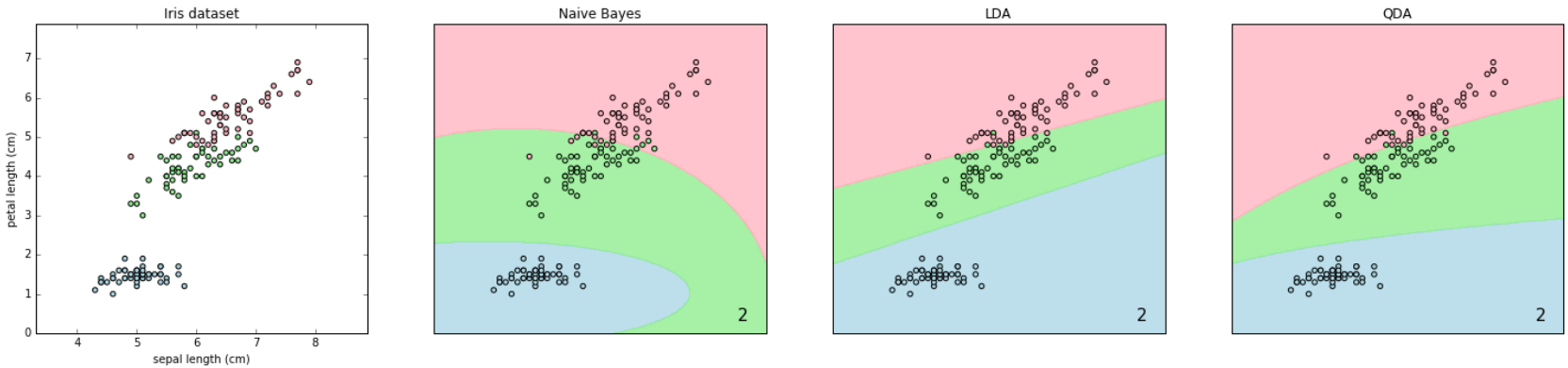
- on a un vecteur de covariables (features) $X_i \in \mathbb{R}^d$
- la valeur de son label $Y_i \in \mathcal{C} = \{1, \ldots, K\}$.
- on suppose que les couples $(X_i, Y_i)$ sont des copies i.i.d. de $(X, Y)$ de loi inconnue.

**But**

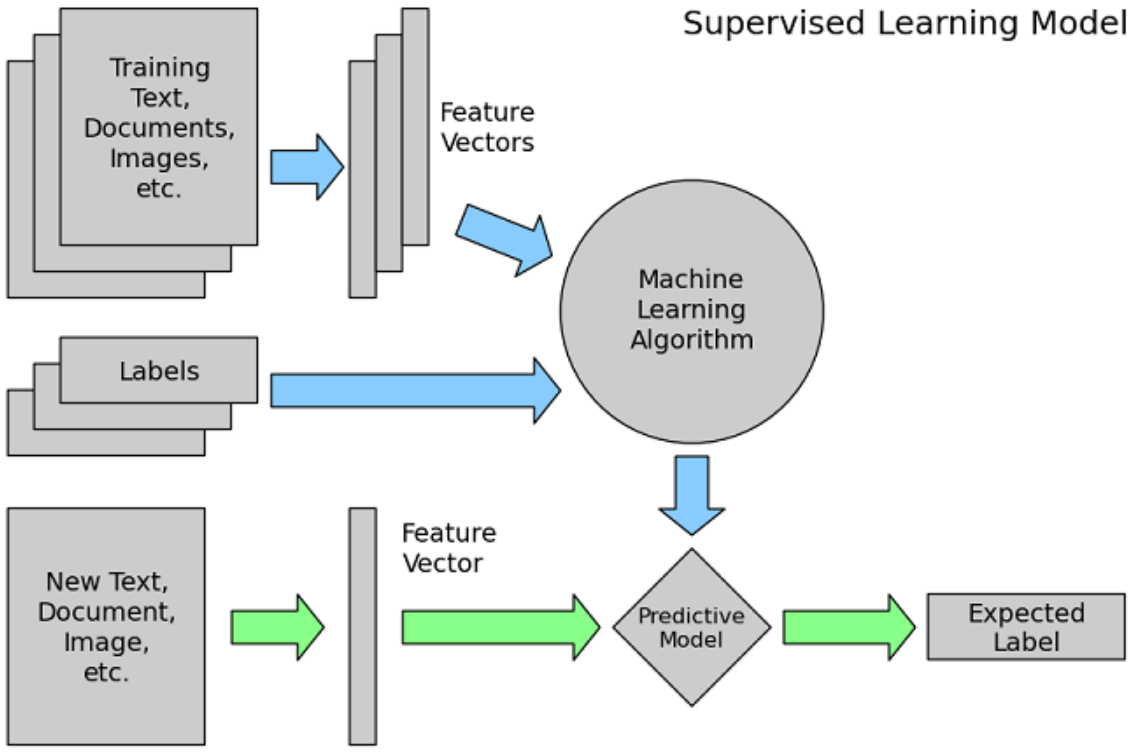
- On veut pour un nouveau vecteur $X_+$ de features prédire la valeur du label $Y_+$ par $\hat{Y}_+ \in \mathcal{C} = \{1, \ldots, K\}$
- Pour cela, on utilise les données d'apprentissage $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ pour construire un **classifieur** $\hat{c}$ de telle sorte que

$$\hat{Y}_+ = \hat{c}(X_+).$$

# Classification multi-classes : Iris dataset



# Apprentissage statistique supervisé




- Input : covariables, variables explicatives, features $X = (X^1, \ldots, X^d)$
- Ouput : variable à expliquer, variable dépendante, réponse, label $Y$

# Approche probabiliste / statistique

# Approche probabiliste / statistique en classification binaire

- Pour construire le classifieur $\hat{c}$, on construit des estimateurs $\hat{p}_1(x)$ et $\hat{p}_{-1}(x)$ de

$$p_1(x) = \mathbb{P}(Y = 1 | X = x) \quad \text{et} \quad p_{-1}(x) = 1 - p_1(x)$$

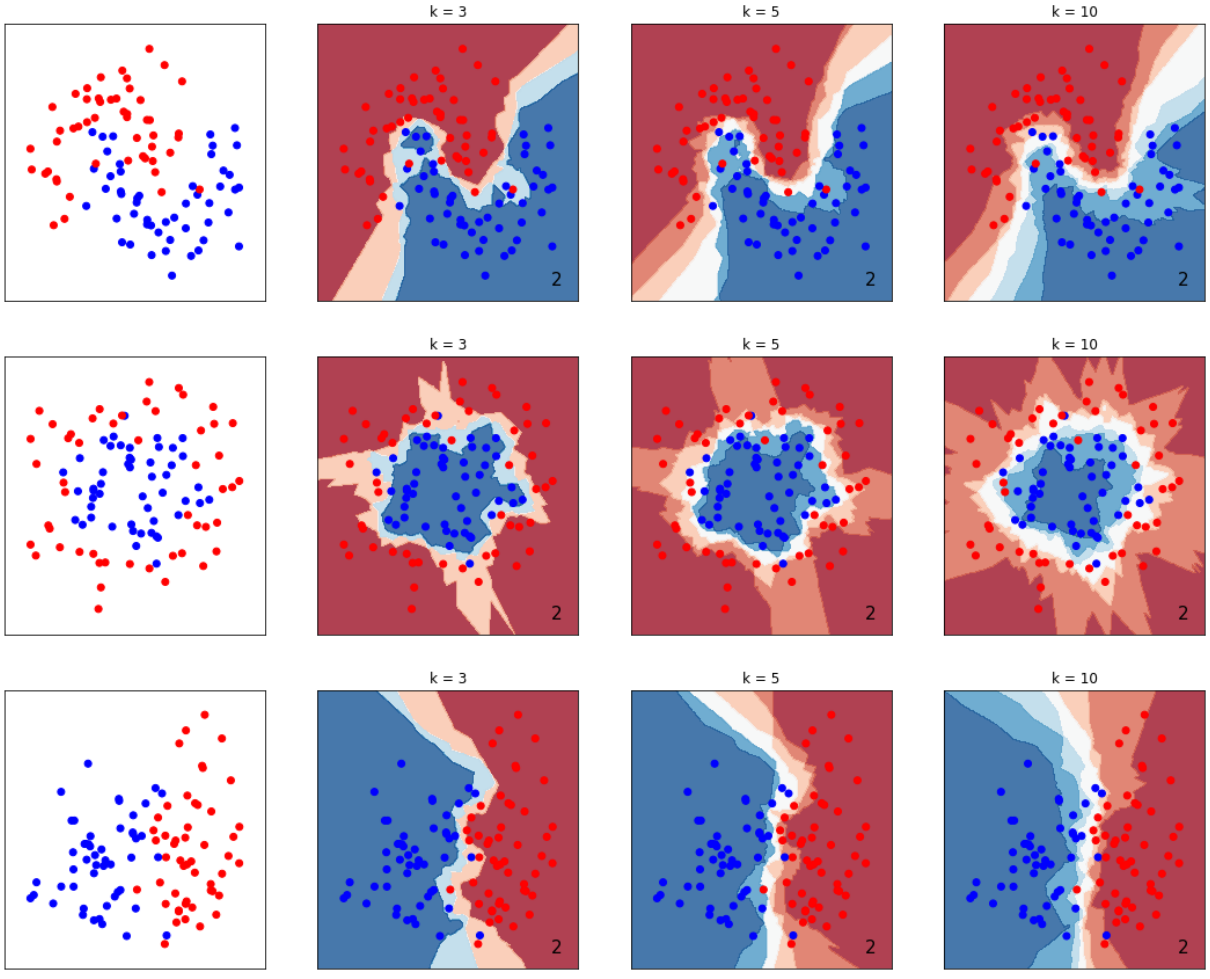
- en modélisant la loi de $Y|X$.
- Puis, conditionnellement à $X_+ = x$, on classifie en utilisant la règle

$$\hat{Y}_+ = \hat{c}(x) = \begin{cases} 1 & \text{si } \hat{p}_1(x) \geq s \\ -1 & \text{sinon} \end{cases}$$

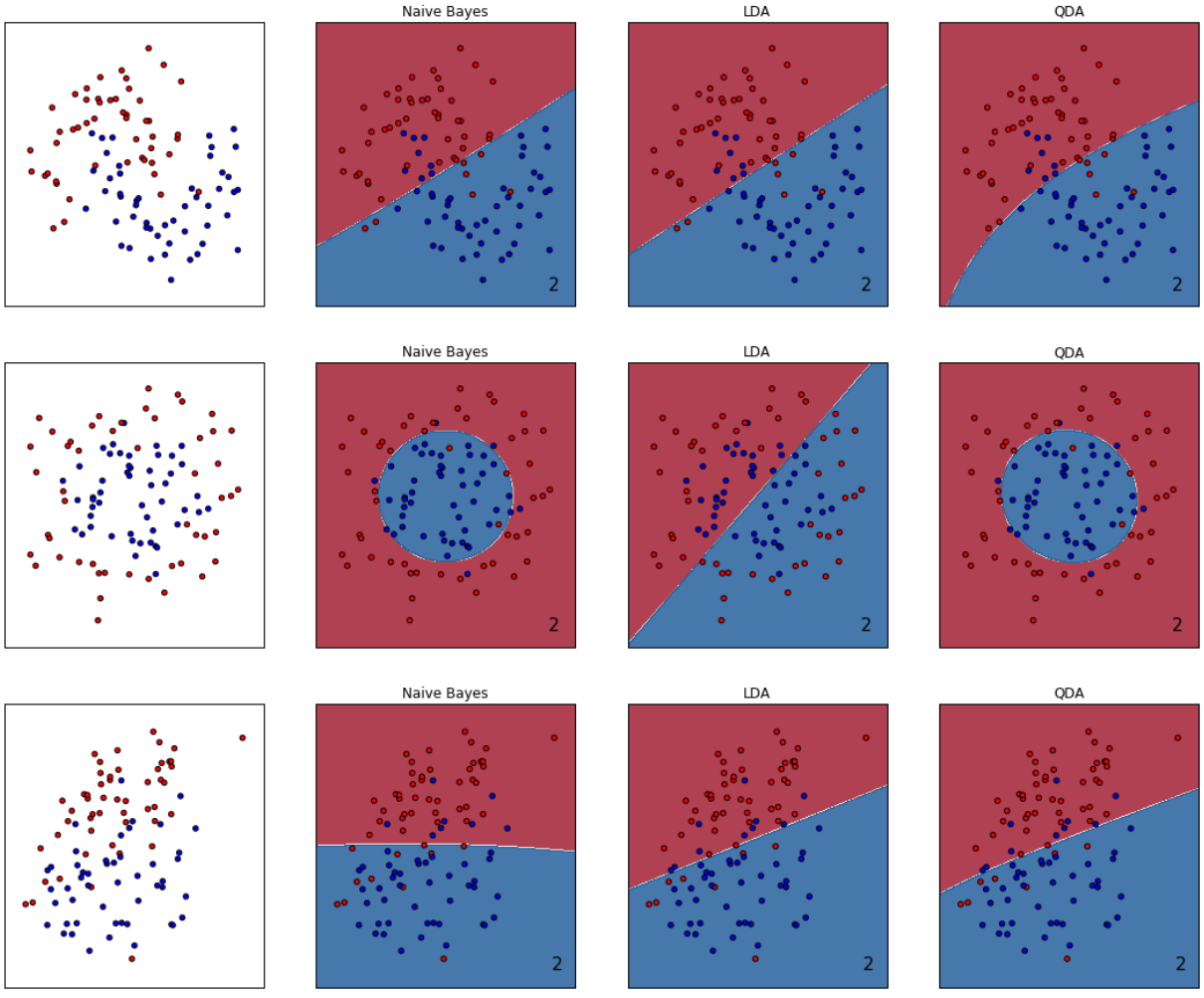
pour un seuil $s \in (0, 1)$.

- Si on choisit $s = 1/2$, cela revient à classifier suivant la plus grande valeur entre $\hat{p}_1(x)$ et $\hat{p}_{-1}(x)$ (on retient cette règle dans la suite).

# Classification binaire : toy datasets



# Classification binaire : toy datasets



# Classifieur bayésien (1)

## Formule de Bayes

Nous savons que

$$\begin{aligned} p_y(x) = \mathbb{P}(Y = y | X = x) &= \frac{f_y(x)\mathbb{P}(Y = y)}{f(x)} \\ &= \frac{f_y(x)\mathbb{P}(Y = y)}{\sum_{y'=-1,1} f_{y'}(x)\mathbb{P}(Y = y')} \\ &\propto f_y(x)\mathbb{P}(Y = y), \end{aligned}$$

où $f$ est la densité jointe de $X$ et $f_y$ est la densité de $X$ condionnellement à $Y = y$.

Donc si on estime les loi de $X|Y$ et de $Y$, on a un estimateur de celle de $Y|X$.

# Classifieur bayésien (2)

## Classifieur bayésien

On construit un classifieur grâce à la formule de Bayes

- en modélisant la loi $X|Y$ puis en l'estimant
- et en estimant la loi marginale de $Y$.

On aura donc

$$\hat{p}_y(x) = \widehat{\mathbb{P}}(Y = y|X = x) \propto \hat{f}_y(x)\widehat{\mathbb{P}}(Y = y)$$

puis

$$\begin{aligned}\hat{Y}_+ = \hat{c}(x) &= \begin{cases} 1 & \text{si } \hat{p}_1(x) \geq \hat{p}_{-1}(x) \\ -1 & \text{sinon} \end{cases} \\ &= \begin{cases} 1 & \text{si } \hat{f}_1(x)\widehat{\mathbb{P}}(Y = 1) \geq \hat{f}_{-1}(x)\widehat{\mathbb{P}}(Y = -1) \\ -1 & \text{sinon} \end{cases} \\ &= \text{argmax}_{y=-1,1}\hat{f}_y(x)\widehat{\mathbb{P}}(Y = y).\end{aligned}$$

# Maximum a posteriori en classification binaire

Si, conditionnellement à $X_+ = x$, on classifie en utilisant la règle

$$\hat{Y}_+ = \hat{c}(x) = \begin{cases} 1 & \text{si } \hat{p}_1(x) \geq \hat{p}_{-1}(x) \\ -1 & \text{sinon ,} \end{cases}$$

c'est équivalent à utiliser une fonction discriminante.

## Fonction discriminante

$$\hat{\delta}_y(x) = \log \widehat{\mathbb{P}}(X = x | Y = y) + \log \widehat{\mathbb{P}}(Y = y)$$

pour $y = 1, -1$ et de classifier suivant sa valeur pour chaque $y$.

# Classification multi-classes et maximum a posteriori

- On modélise la distribution de $Y|X$
- On construit des estimateurs $\hat{p}_k(x)$ pour $k \in \mathcal{C}$ tels que

$$\sum_{k \in \mathcal{C}} \hat{p}_k(x) = 1$$

- Conditionnellement à $X_+ = x$, on classifie en utilisant la règle

$$\hat{Y}_+ = \text{argmax}_{k \in \mathcal{C}} \hat{p}_k(x).$$

On peut alors définir des fonctions discriminantes pour $k \in \mathcal{C}$

$$\hat{\delta}_k(x) = \log \widehat{\mathbb{P}}(X = x | Y = k) + \log \widehat{\mathbb{P}}(Y = k).$$

et décider de classifier avec la règle

$$\hat{Y}_+ = \text{argmax}_{k \in \mathcal{C}} \hat{\delta}_k(x).$$

## Remarque

- On peut choisir plusieurs modèles pour la loi de $X|Y$ qui donnent des classifieurs différents.
- Le plus simple est le "Naive Bayes"
- Nous verrons aussi l'analyse discriminante linéaire (Linear discriminant analysis - LDA) et quadratique (Quadratic discriminant Analysis -QDA)

# Naive Bayes

- On veut un modèle pour la loi de $X|Y$.
- Le plus simple est de considérer que les features $X^j$ $(j = 1, \ldots, d)$ sont indépendantes conditionnellement à $Y$.
- C'est équivalent à supposer que la densité conditionnelle $f_y(x)$ de $X|Y = y$ est donnée par

$$\prod_{j=1}^{d} f_y^j(x^j)$$

où $f_y^j$ est la densité de $X^j|Y = y$.

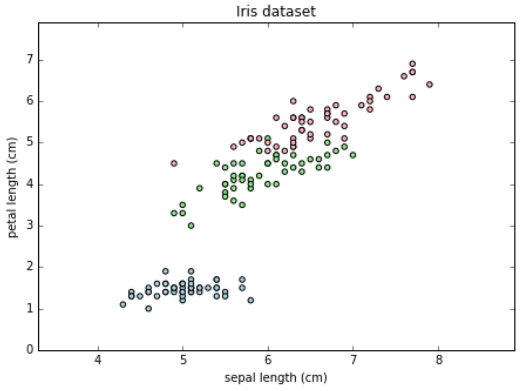
# Quelle loi pour $X^j | Y = y$ ?

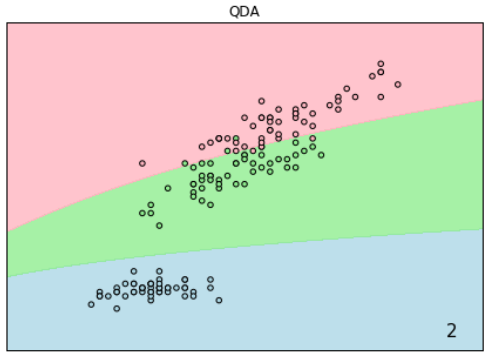
- Si $X^j$ est une variable continue, on choisit la loi normale

$$X^j | Y = y \sim \mathcal{N}(\mu_{j,y}, \sigma^2_{j,y}),$$

  dont les paramètres $\mu_{j,k}$ et $\sigma^2_{j,k}$ sont estimés par maximum de vraisemblance.
  Il suffit alors d'estimer les $\mu_{j,y}, \sigma^2_{j,y}$ pour tous les $j = 1, \ldots, d$ et les $y \in \mathcal{C}$.
- Si $X^j$ est une variable discrète, on choisit la loi de Bernoulli ou la loi multinomiale.

# Exemple sur le dataset `Iris`



# Analyse discriminante

# Analyse discriminante

Supposons que

$$X|Y = y \sim \mathcal{N}(\mu_y, \Sigma_y).$$

On rappelle que la densité de la loi $\mathcal{N}(\mu, \Sigma)$ est donnée par

$$f(x) = \frac{1}{(2\pi)^{d/2}\sqrt{\det \Sigma}} \exp\Big( -\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\Big).$$

# Estimation

On estime les paramètres inconnus par maximum de vraisemblance. Donc pour $y \in \mathcal{C}$, on pose

$$I_y = \{i = 1, \ldots, n \ : \ Y_i = y\} \quad n_y = |I_y|$$

et

$$\widehat{\mathbb{P}}(Y = y) = \frac{n_y}{n}, \quad \hat{\mu}_y = \frac{1}{n_y} \sum_{i \in I_y} X_i,$$

$$\hat{\Sigma}_y = \frac{1}{n_y} \sum_{i \in I_y} (X_i - \hat{\mu}_y)(X_i - \hat{\mu}_y)^\top.$$

Ce sont simplement les proportions, moyennes et variances de chaque sous-groupe de données défini par la valeur du label.

# Fonctions discriminantes

Dans ce cas, les fonctions discriminantes sont

$$\begin{aligned}\hat{\delta}_y(x) &= \log \widehat{\mathbb{P}}(X = x | Y = y) + \log \widehat{\mathbb{P}}(Y = y) \\ &= -\frac{1}{2}(x - \hat{\mu}_y)^\top \hat{\Sigma}_y^{-1}(x - \hat{\mu}_y) - \frac{d}{2}\ln(2\pi) \\ &\quad - \frac{1}{2}\log\det\hat{\Sigma}_y + \log \widehat{\mathbb{P}}(Y = y)\end{aligned}$$

# Linear Discriminant Analysis (LDA)

Supposons que $\Sigma = \Sigma_y$ pour tout $y \in \mathcal{C}$ (cela revient à supposer qu'il y a la même structure de corrélation dans chaque groupe)

## Linear Discriminant Analysis (LDA)

Les frontières de décision sont linéaires et les régions ont la forme (pour la classification binaire) $\langle x, w \rangle \geq c$ avec

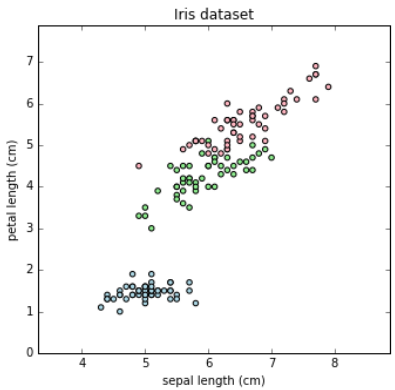
$$w = \hat{\Sigma}^{-1}(\hat{\mu}_1 - \hat{\mu}_{-1})$$

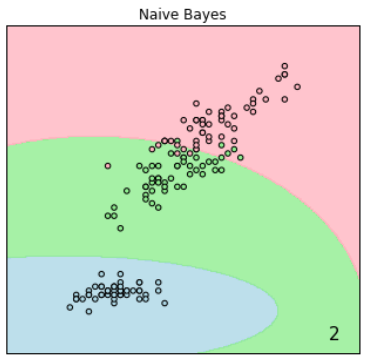
$$c = \frac{1}{2}(\langle \hat{\mu}_1, \hat{\Sigma}^{-1}\hat{\mu}_1 \rangle - \langle \hat{\mu}_{-1}, \hat{\Sigma}^{-1}\hat{\mu}_{-1} \rangle) + \log\left(\frac{\widehat{\mathbb{P}}(Y=1)}{\widehat{\mathbb{P}}(Y=-1)}\right).$$

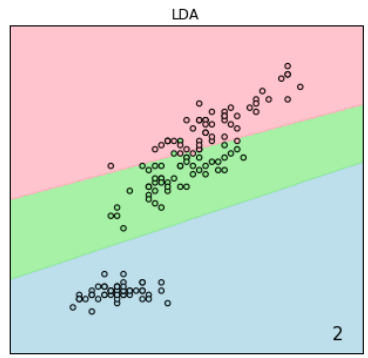
# Quadratic Discriminant Analysis (QDA)

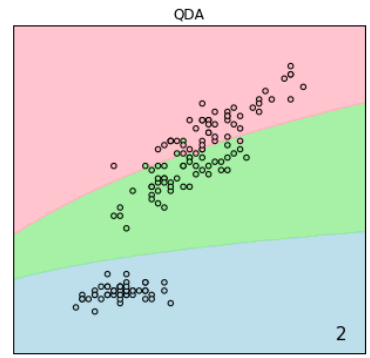
On n'assume plus que $\Sigma = \Sigma_y$ pour tout $y \in \mathcal{C}$. Dans ce cas, les frontières sont quadratiques.

# Exemple sur le dataset `Iris`



# Classifieur constants sur une partition

# Rappel sur la classification

On a pour $i = 1, \ldots, n$

- $X_i \in \mathbb{R}^d$
- $Y_i \in \mathcal{C} = \{-1, 1\}$ ou $\{1, \ldots, K\}$.

**But**

- On veut pour un nouveau vecteur $X_+$ de features prédire la valeur du label $Y_+$ par $\hat{Y}_+ \in \mathcal{C} = \{1, \ldots, K\}$
- Pour cela, on utilise les données d'apprentissage $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ pour construire un **classifieur** $\hat{c}$ de telle sorte que

$$\hat{Y}_+ = \hat{c}(X_+).$$

# Classifieur constants sur une partition

On va considérer

- une partition $\mathcal{A} = \{A_1, \ldots, A_M\}$ de $\mathcal{X}$ (qui peut dépendre des données)
- l'ensemble $\mathcal{F}_{\mathcal{A}}$ des fonctions constantes sur $\mathcal{A}$
- la perte 0/1 $\ell(y, y') = \mathbb{1}_{yy' \leq 0}$

on cherche alors un classifieur $\hat{c}$ qui vérifie

$$\hat{c}_{\mathcal{A}} = \operatorname*{argmin}_{c \in \mathcal{F}_{\mathcal{A}}} \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, c(X_i)) = \operatorname*{argmin}_{c \in \mathcal{F}_{\mathcal{A}}} \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{Y_i c(X_i) \leq 0}.$$

# Vote majoritaire

En classification binaire, on sait alors que $\hat{c}_{\mathcal{A}}$ vérifie pour $x \in A_m$

$$\hat{c}_{\mathcal{A}}(x) = \begin{cases} 1 & \text{si } \#\{i : X_i \in A_m, Y_i = 1\} > \#\{i : X_i \in A_m, Y_i = -1\} \\ -1 & \text{sinon} \end{cases}$$

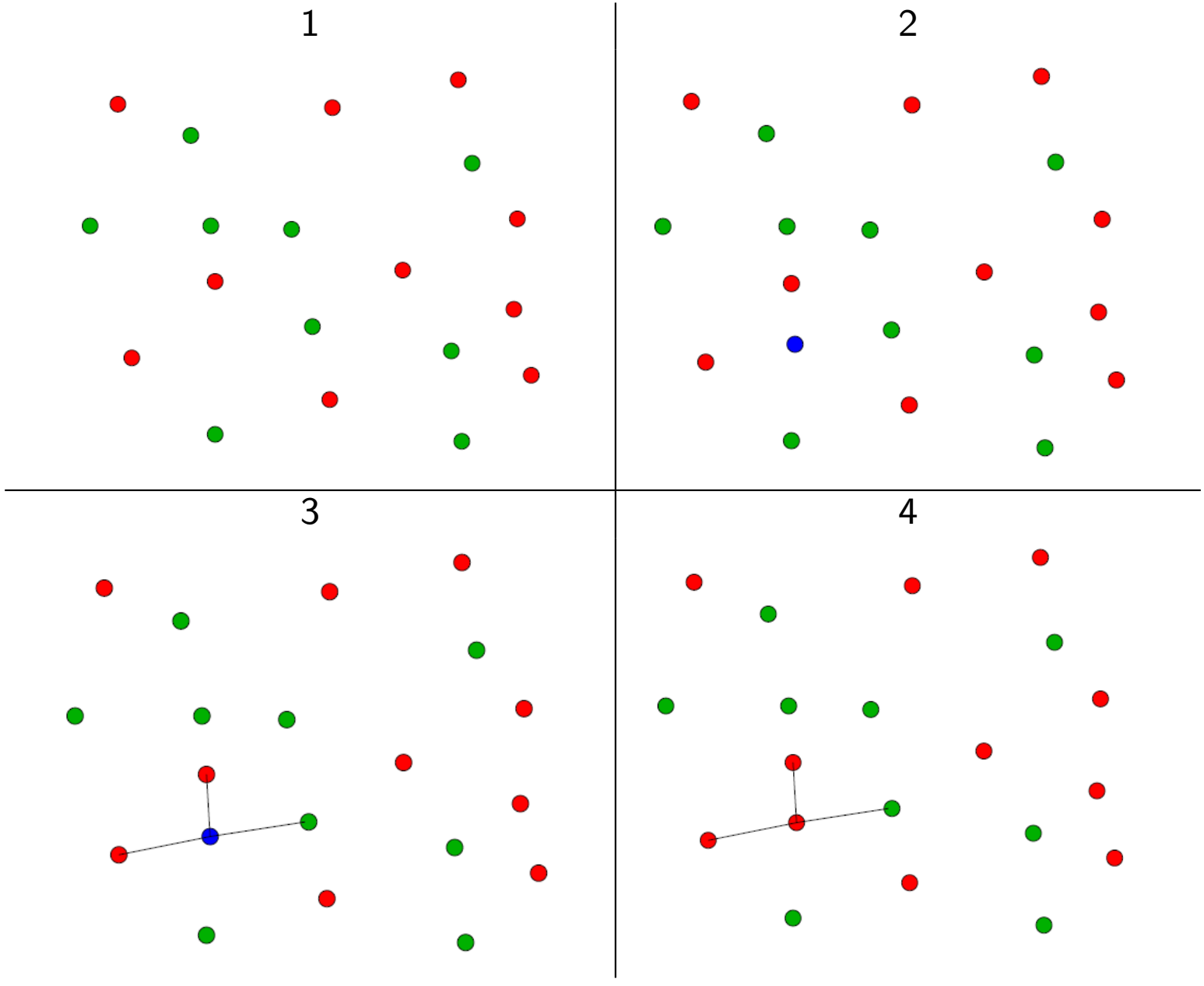
$$= \begin{cases} 1 & \text{si } \bar{Y}_{A_m} > 0 \\ -1 & \text{sinon} \end{cases}$$

En classification multi-classes, on posera pour $x \in A_m$

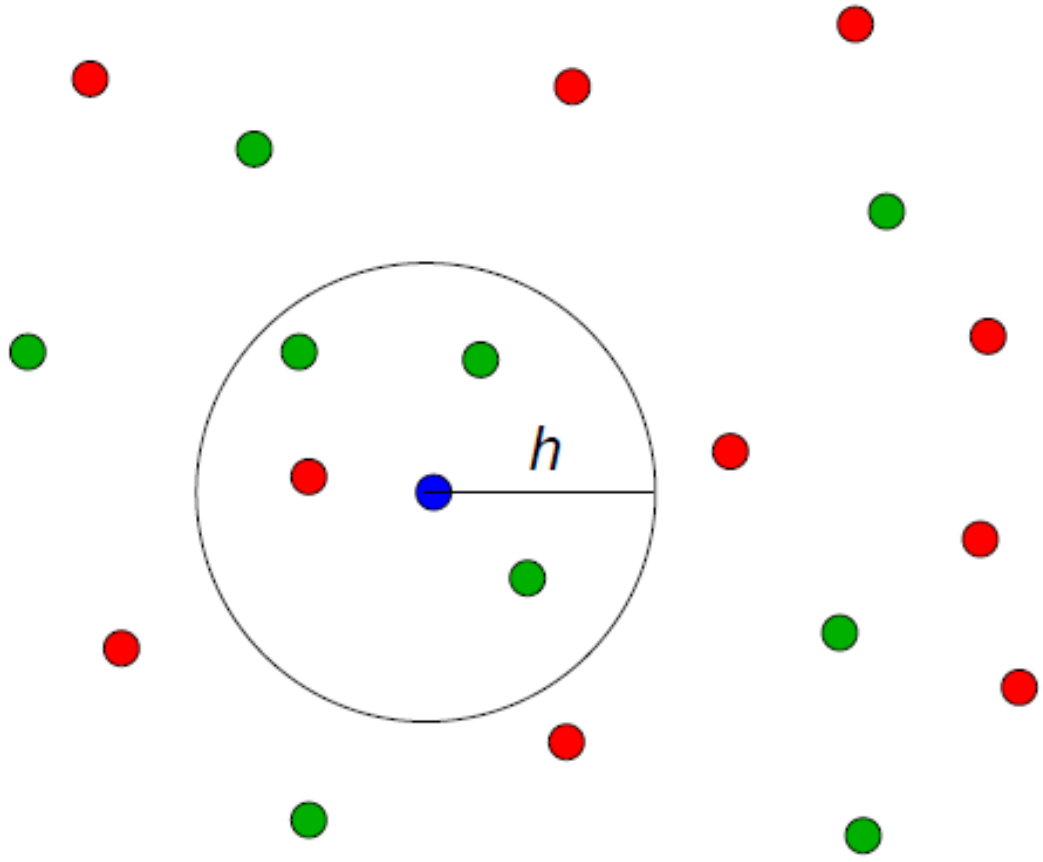
$$\hat{c}_{\mathcal{A}}(x) = \underset{k \in \{1,\dots,K\}}{\text{argmax}} \ \#\{i : X_i \in A_m, Y_i = k\}$$

Il reste à choisir la partition $\mathcal{A} = \{A_1, \dots, A_M\}$ de $\mathcal{X}$ !

# Exemple: $k$ plus proches voisins (avec $k = 3$)



# Exemple: $k$ plus proches voisins (avec $k = 4$)



## $k$ plus proches voisins

On considère l'ensemble $\mathcal{I}_x$ composé des $k$ indices de $\{1, \ldots, n\}$ pour lesquels les distances $\|x - X_i\|$ sont minimales.

On pose

$$\hat{c}(x) = \underset{k \in \{1,\ldots,K\}}{\text{argmax}} \ \#\{i \in \mathcal{I}_x, Y_i = k\}.$$

- En pratique, il faut choisir la distance
- et $k$ !!

# Partition

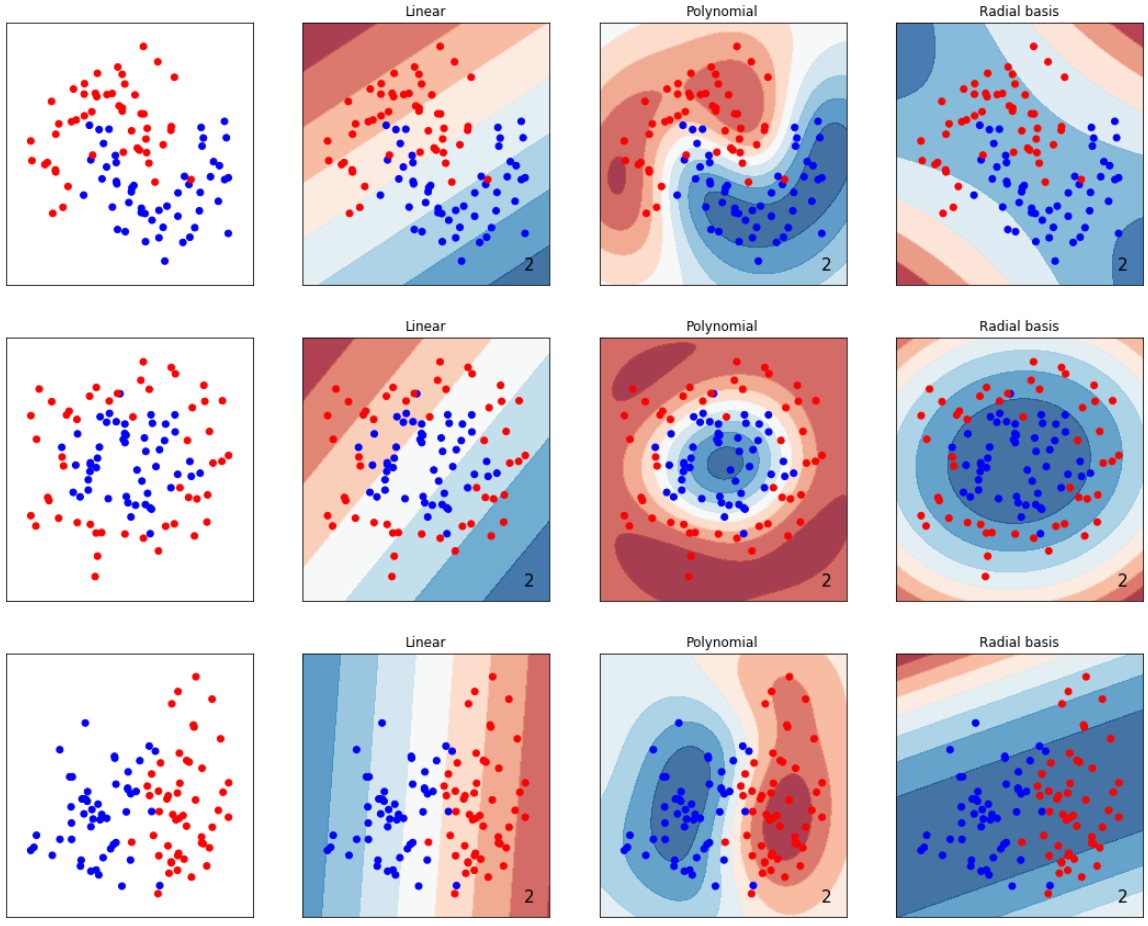
On remarque que $\mathcal{I}_x$ appartient à l'ensemble $\{\phi^1, \ldots, \phi^M\}$ des combinaisons de $k$ éléments parmi $n$ avec

$$M = \binom{n}{k}.$$

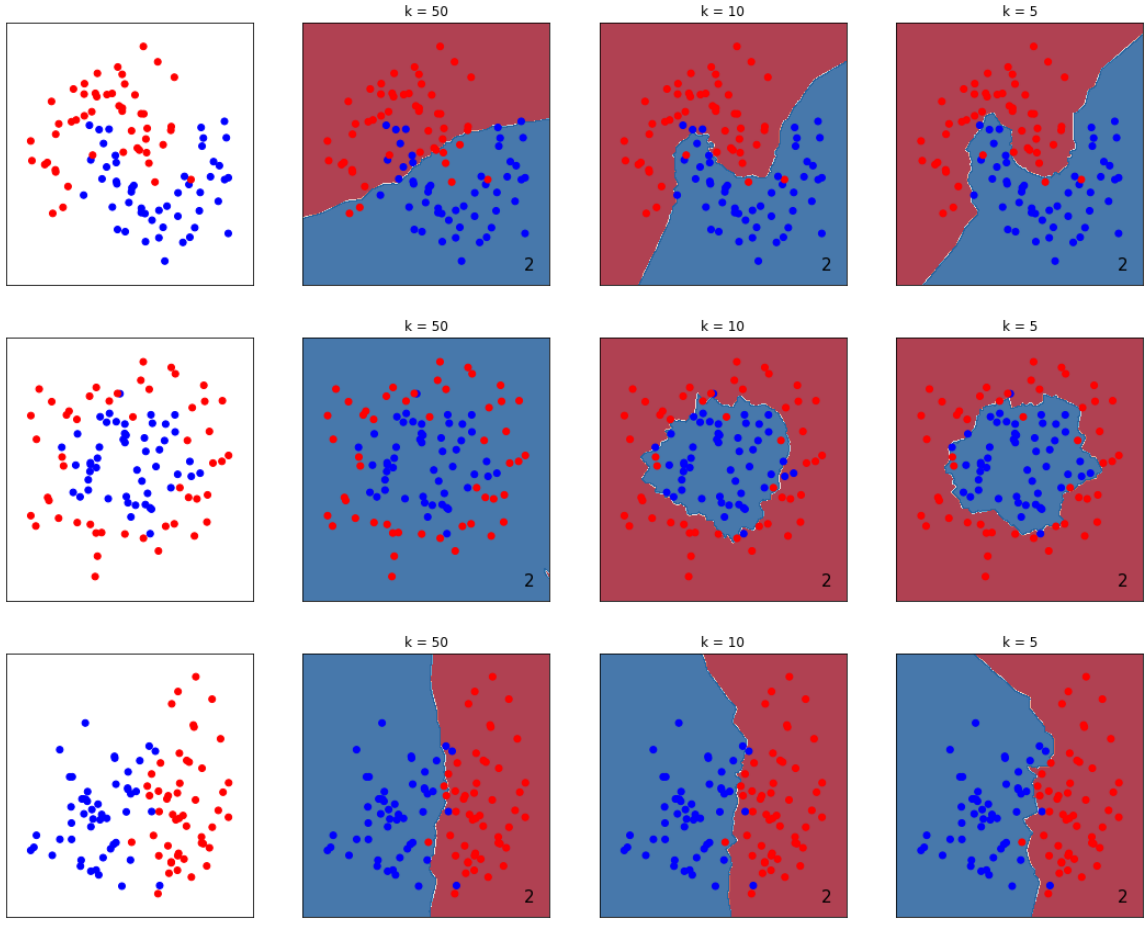
On peut donc poser

$$A_m = \{x \in \mathcal{X}, \mathcal{I}_x = \phi^m\}.$$

# $k$-NN


Linear
Polynomial
Radial basis
Linear
Polynomial
Radial basis
Linear
Polynomial
Radial basis

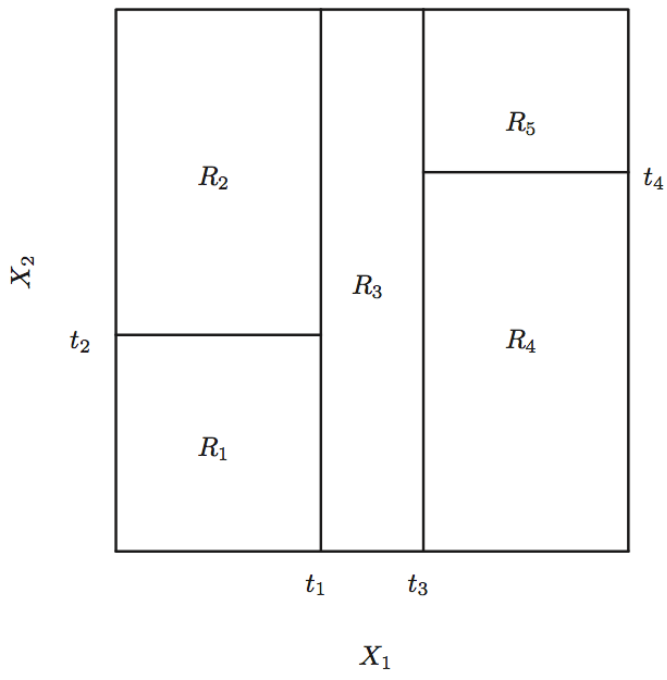
# *k*-NN

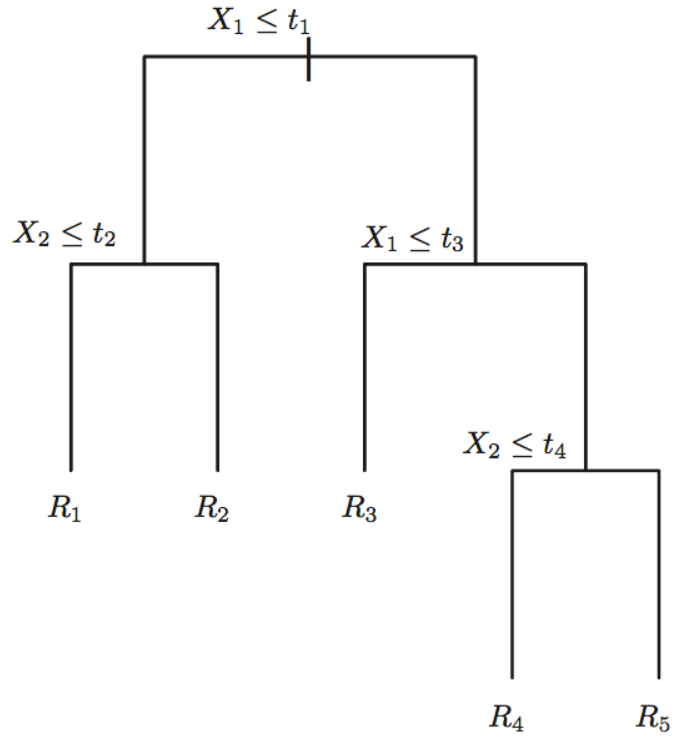


# Construction de l'arbre

Approche "top-bottom"

- On commence par une région qui contient toutes les données
- On coupe récursivement les régions par rapport à une variable et une valeur de cette variable



## Heuristique:

On veut choisir la valeur du “split” de telle sorte que les deux nouvelles régions sont les plus **homogènes** possible...

L'**homogénéité** peut se définir par différents critères

- la variance empirique
- l'indice de Gini
- l'entropie.

# Arbre de classification à partir de l'indice de Gini

On coupe une région $R$ en deux parties $R_-$ and $R_+$. Pour chaque variable $j = 1, \ldots, p$ et chaque valeur de "split" $t$, on définit

$$R_-(j,t) = \{x \in R : x^j < t\} \quad \text{et} \quad R_+(j,t) = \{x \in R : x^j \geq t\}.$$

on cherche $j$ et $t$ qui minimisent

$$\text{Gini}(R_-) + \text{Gini}(R_+)$$

where

$$\text{Gini}(R) = \frac{1}{|\{i, X_i \in R\}|} \sum_{k \in \mathcal{C}} \hat{p}_{R,k}(1 - \hat{p}_{R,k})$$

où $\hat{p}_{R,k}$ est la proportion d'observations avec le label $k$ dans l'ensemble des $\{i, X_i \in R\}$.

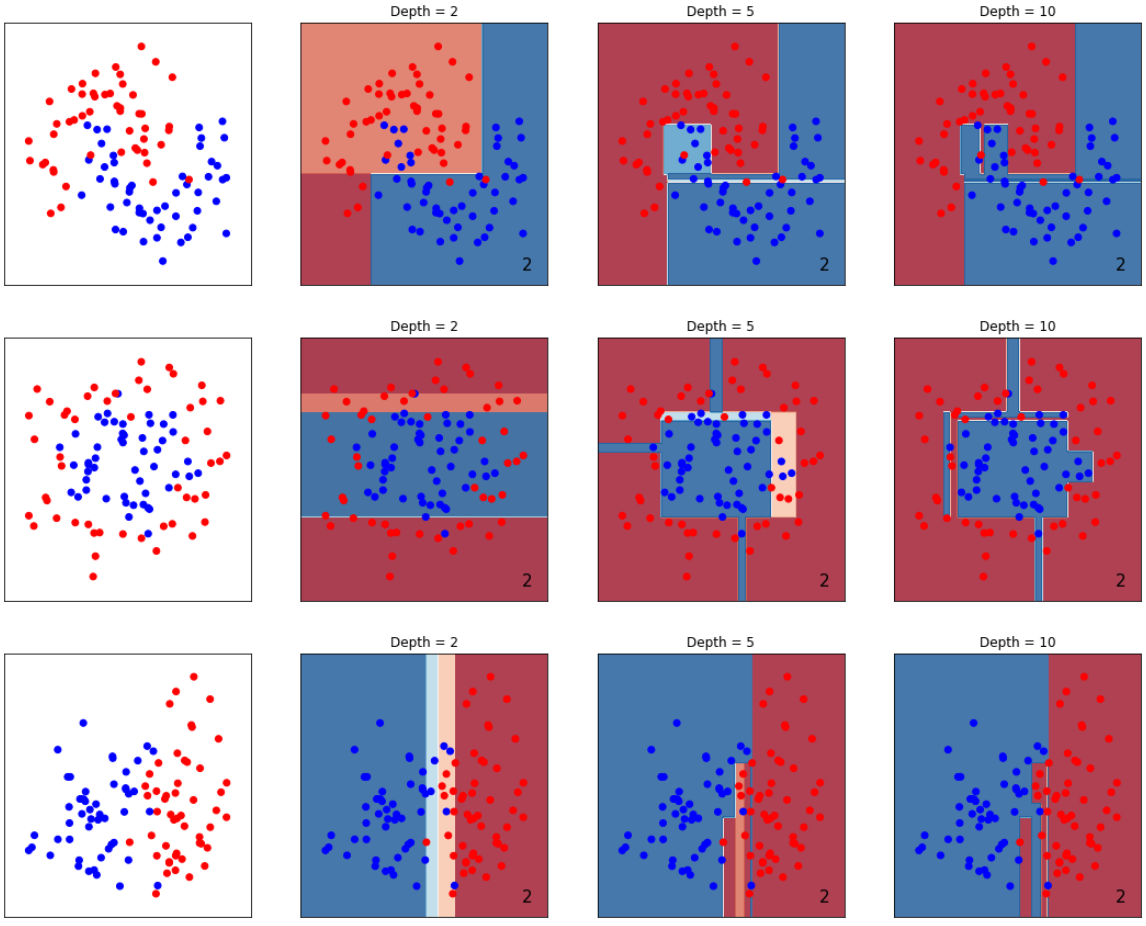
# Algorithmes CART, C.4.5

- l'algorithme CART utilise l'indice de Gini
- l'algorithme C.4.5 (pas implementé dans `sklearn`) utilise l'entropie

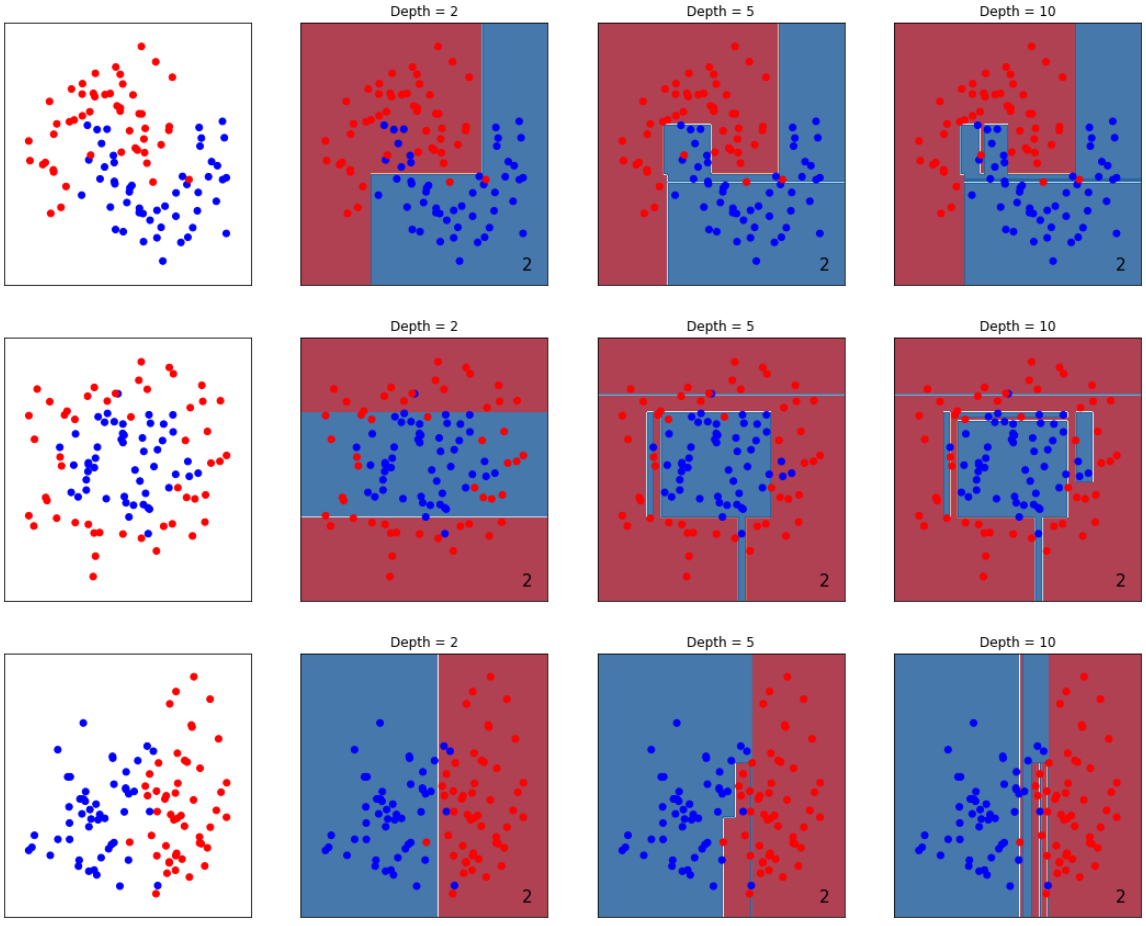
$$E(R) = -\sum_{k \in \mathcal{C}} \hat{p}_{R,k} \log(\hat{p}_{R,k})$$

- il y a d'autres critères possibles ($\chi^2$, etc)

# CART


Depth = 2
Depth = 5
Depth = 10
Depth = 2
Depth = 5
Depth = 10
Depth = 2
Depth = 5
Depth = 10

# CART


Depth = 2
Depth = 5
Depth = 10
Depth = 2
Depth = 5
Depth = 10
Depth = 2
Depth = 5
Depth = 10

# Règles d'arrêt et algorithmes dérivés

## Règles d'arrêt

On arrête l'algorithme quand

- l'arbre a atteint une taille maximale (fixée à l'avance)
- le nombre de feuilles atteint une valeur maximale (fixée à l'avance)
- quand les effectifs des noeuds terminaux atteignent une valeur minimale (fixée à l'avance)

## En pratique

En pratique, ce sont des algorithmes instables et qui sur-apprennent, on les utilisent dans des algorithmes plus complexes qui "mélangent" des arbres

- les forêts alétoires (random forests)
- le boosting.

# Ensemble methods

# Le boosting : weak learners

## Weak learners

- Considérons un ensemble de "weak learners" $\mathcal{H}$
- Chaque learner $h : \mathbb{R}^d \to \{-1, 1\}$ est un learner très simple
- Chaque learner simple est à peine meilleur que celui appris avec les $Y_i$ (moyenne)

## Exemples de weak learners

- Pour la régression : arbres de régression simple de faible profondeur, modèle linéaire à quelques variables
- Pour la classification : arbre de décision simple de faible profondeur, modèle logistique à quelques variables.

# Principe du boosting

## Un "strong learner"

On combine additivement des weak learners

$$g^{(B)}(x) = \sum_{b=1}^{B} \eta^{(b)} h^{(b)}(x)$$

avec $\eta^{(b)} \geq 0$ pour espérer en obtenir un meilleur. $g^{(b)}$ est un learner dans le sens où on $\hat{Y}_i = g^{(b)}(X_i)$.

- Chaque $b = 1, \ldots, B$ est un pas/itération de boosting
- Le boosting fait partie des **ensemble methods** puisqu'il combine des learners faibles pour en fabriquer un meilleur.

Pour aller plus loin : voir Schapire 1999 et Friedman 2001

# Minimisation de l'erreur, méthodes basées sur l'optimisation

# Erreur empirique / erreur de généralisation

On se donne une fonction classifiante déterministe $c : \mathbb{R}^d \in \mathcal{C}$, on définit $\mathcal{L}$ la loi inconnue des couples $(X_i, Y_i)$ et

## Erreur empirique ou erreur visible

$$R_n(c) = \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, c(X_i)).$$

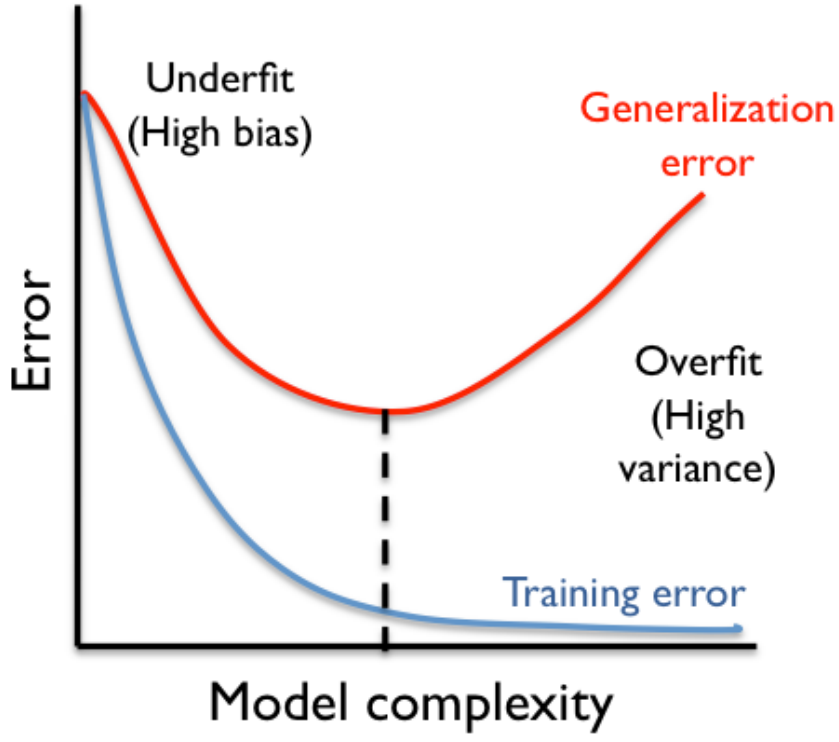
## Erreur de généralisation

$$R(c) = \mathbb{E}_{\mathcal{L}} \left( \ell(Y_+, c(X_+)) \right)$$

où $(X_+, Y_+)$ est un couple indépendant de $\mathcal{D}_n$

- En classification, on prend souvent $\ell(y, y^{'}) = \mathbb{1}_{y \neq y'}$, dans ce cas $1 - R_n(c)$ est appelé “accuracy” de $c$.

# Sur-apprentissage




## Comportement de l'erreur

- L'erreur d'apprentissage décroît avec la complexité du modèle.
- L'erreur de généralisation (sur le test) a un comportement très différent.

# Remarques

- On a
$$R(c) = \mathbb{E}_{\mathcal{L}}\big(R_n(c)\big).$$
- On voudrait retrouver
$$c^\star = \operatorname*{argmin}_c R(c)$$

Mais on se restreint le plus souvent à un sous-ensemble $\mathcal{G}$ (par exemple les fonctions constantes sur une partition $\mathcal{A}$)

$$c_{\mathcal{G}}^{\text{oracle}} = \operatorname*{argmin}_{c\in\mathcal{G}} R(c)$$

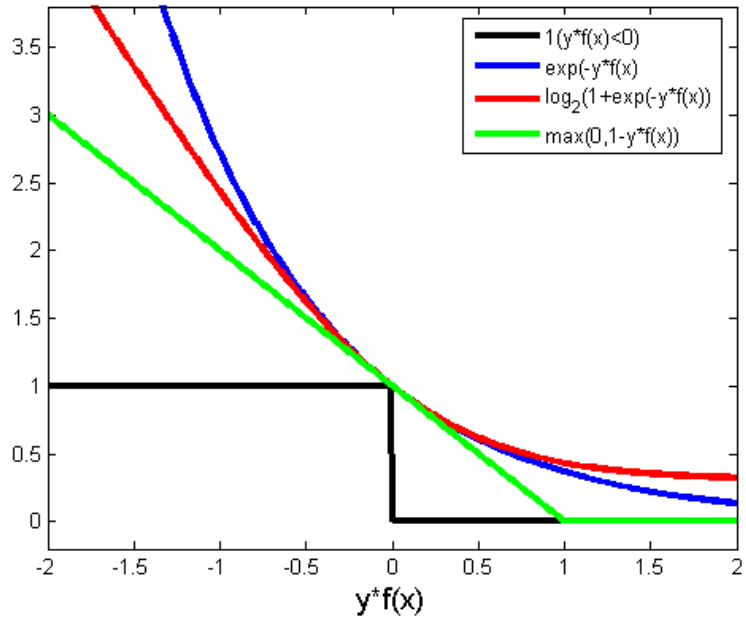
puis, comme la loi $\mathcal{L}$ est inconnue, on remplace $R$ par $R_n$

$$\hat{c}_{\mathcal{G}} = \operatorname*{argmin}_{c\in\mathcal{G}} R_n(c).$$

On a bien sûr
$$R(\hat{c}_{\mathcal{G}}) \geq R(c_{\mathcal{G}}^{\text{oracle}}) \geq R(c^\star).$$

# Autres fonctions de perte classiques en classification binaire

- Logistic loss (régression logistique) $\ell(y, f(x)) = \log(1 + \exp(-yf(x)))$
- Hinge loss (SVM), $\ell(y, f(x)) = (1 - yf(x))_+$
- Quadratic hinge loss (SVM), $\ell(y, f(x)) = \frac{1}{2}(1 - yf(x))_+^2$
- Huber loss $\ell(y, f(x)) = -4yy' \mathbb{1}_{yf(x)<-1} + (1 - yf(x))_+^2 \mathbb{1}_{yf(x)\geq -1}$




- Toutes ces pertes peuvent être vues comme des approximations convexe de la perte 0/1 $\ell(y, y') = \mathbb{1}_{yy' \leq 0}$

# Cross Validation


- **Very simple idea:** use a second learning/verification set to compute a verification error.
- Sufficient to remove the dependency issue!

# Cross Validation

- Use $(1-\epsilon)n$ observations to train and $\epsilon n$ to verify!
- Validation for a learning set of size $(1-\epsilon) \times n$ instead of $n$!
- Unstable error estimate if $\epsilon n$ is too small ?

- Most classical variations:
  - Leave One Out,
  - $V$-fold cross validation.

# Réseaux de neurones

https://playground.tensorflow.org/

# References I


Jerome H Friedman. "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics* (2001), pp. 1189–1232.

Robert E Schapire. "A brief introduction to boosting". In: *Ijcai*. Vol. 99. 1999, pp. 1401–1406.