Big Data Analytics

Jour 2 — Algorithmes supervisés

François-Marie Giraud

https://www.orsys.fr/

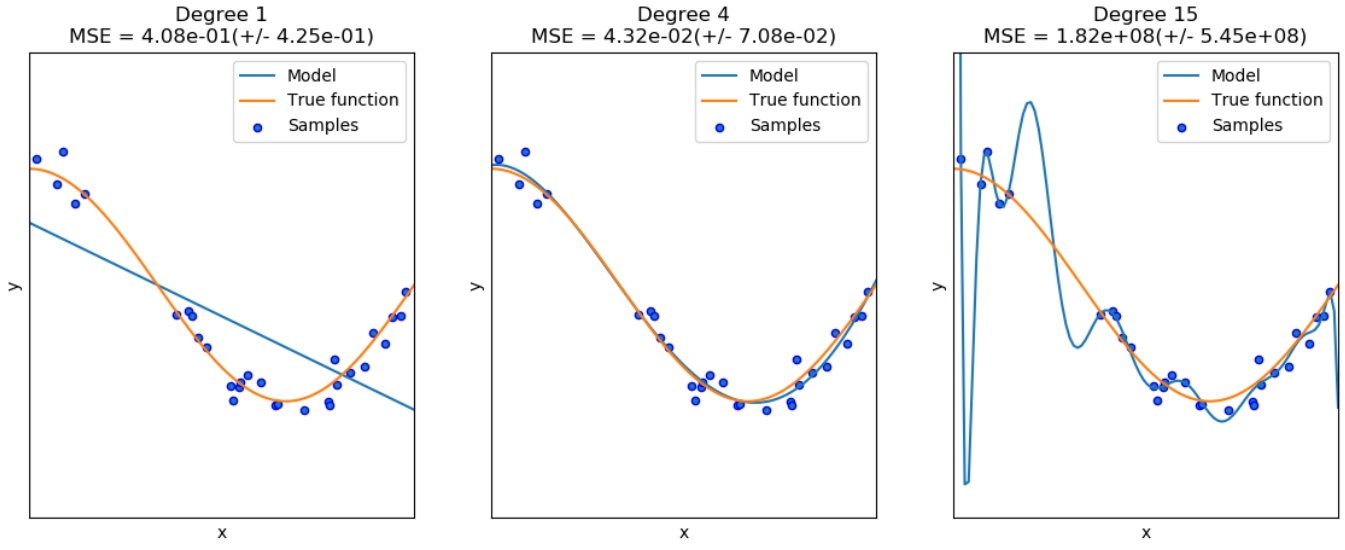
Machine Learning

Machine Learning

Apprentissage

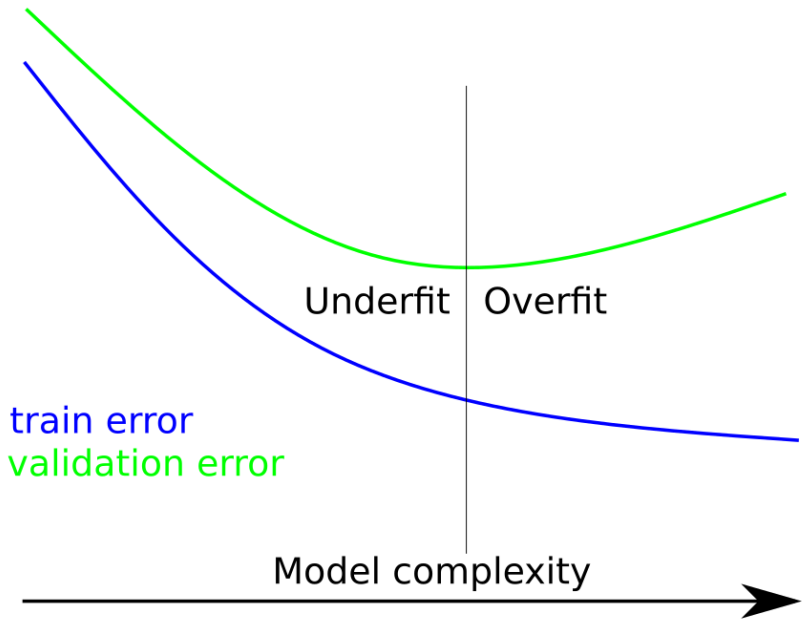
Qualité de l'apprentissage

Entrainement supervisé d'un modèle — overfit



Problème : trop minimiser la perte n'est pas bon !

Qualité de l'apprentissage



→ Minimiser la perte sur un ensemble de validation

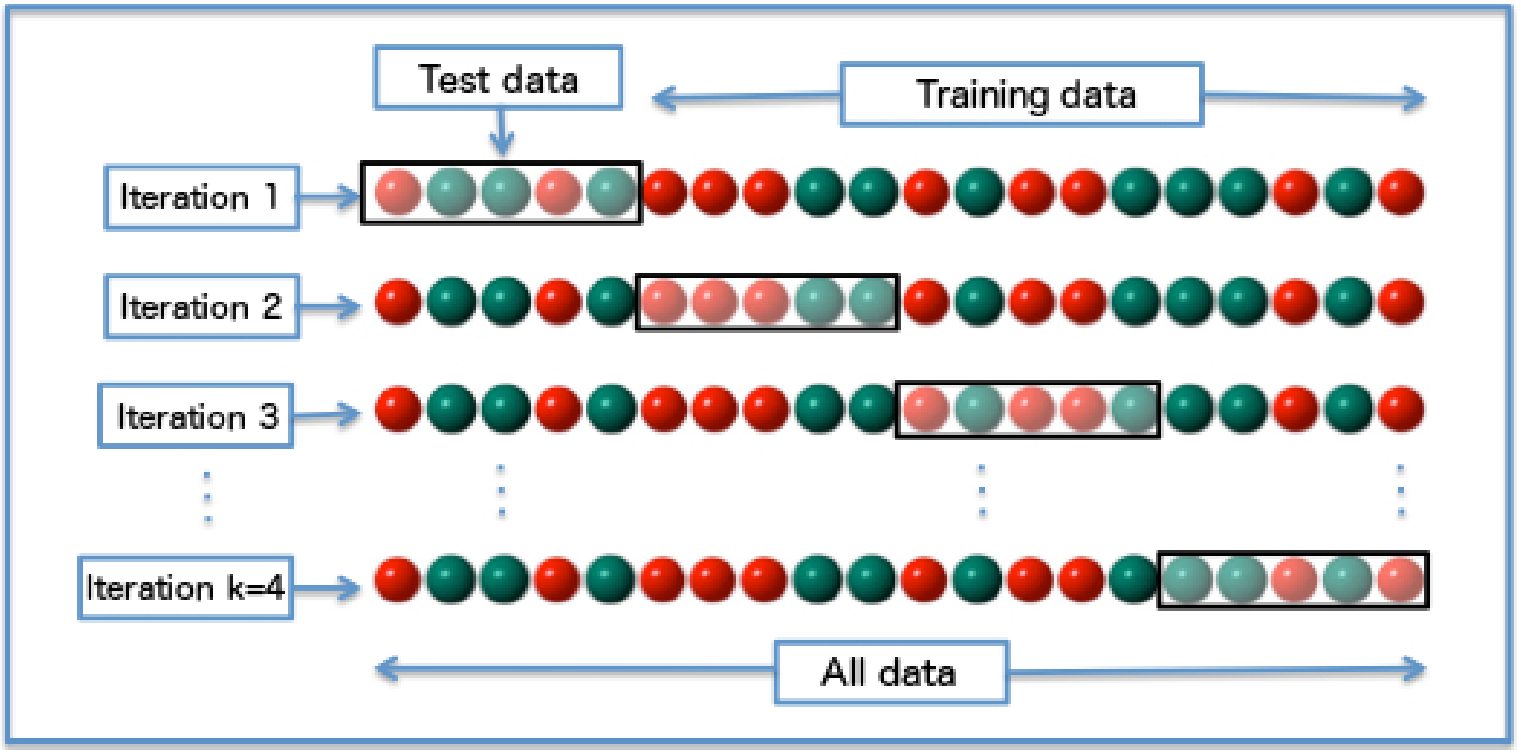
Séparation des données

- ensemble d'entrainement
- ensemble de validation pour mesurer la généralisation
- ensemble de test (pour éviter le biais statistique)

$\rightarrow$ Split 60/20/20 habituel.

Cross-validation

Pour « perdre » moins de données et mieux tester la généralisation :



Ici, 4-fold cross-validation.

Machine Learning

Bonnes Pratiques

Reproductibilité

- extrêmement importante pour compléter les analyses après les retours business
- ensemble de bonnes pratiques d'ingénierie

Reproductibilité

- garder une trace exacte du préprocessing
- de préférence utiliser des notebooks
- faire attention au random (utiliser des seeds)
- définir les datasets utilisés, dates comprises
- garder une trace de l'environnement

Reproductibilité

- garder une trace exacte du préprocessing
- de préférence utiliser des notebooks
- faire attention au random (utiliser des seeds)
- définir les datasets utilisés, dates comprises
- garder une trace de l'environnement

Reproductibilité

- garder une trace exacte du préprocessing
- de préférence utiliser des notebooks
- faire attention au random (utiliser des seeds)
- définir les datasets utilisés, dates comprises
- garder une trace de l'environnement

Reproductibilité

- garder une trace exacte du préprocessing
- de préférence utiliser des notebooks
- faire attention au random (utiliser des seeds)
- définir les datasets utilisés, dates comprises
- garder une trace de l'environnement

Reproductibilité

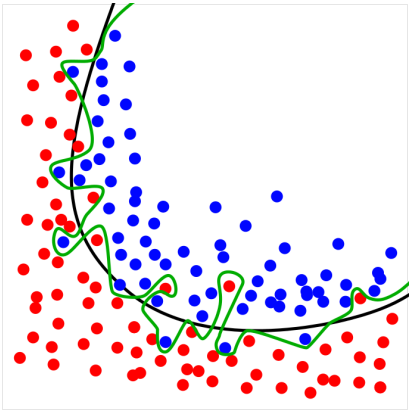
- garder une trace exacte du préprocessing
- de préférence utiliser des notebooks
- faire attention au random (utiliser des seeds)
- définir les datasets utilisés, dates comprises
- garder une trace de l'environnement

Régularisation

Régularisation

$\approx$

empêcher le surapprentissage



Techniques variées en fonction du modèle :

- Pénalisation de la norme des paramètres
- Bruitage
- Dropout
- ...

Optimisation des méta-paramètres

Méta-paramètres : paramètres non appris par le modèle.

Exemples

Forme Nombre de couches ? De quelles tailles ? ...

Optimisation SGD, AdaBoost, Adam, ...

Régularisation Pénalisation de la Norme des paramètres dans la loss, bruitage, dropout, ...

Optimisation par recherche aléatoire ou processus gaussien.

Apprentissage supervisé

Apprentissage supervisé

Probabilités

Rappels - Probabilités

Rappels :

- Une variable aléatoire
  $A \in \mathbb{R}$
- Probabilité
  $0 \leq P(A \in [a_1\ a_2]) \leq 1$
- Probabilité conditionnelle
  $P(\ A > 0 \mid B < -3\ )$
- Évènements indépendants
  $P(A|B) = P(A)$ et $P(B|A) = P(B)$
- Probabilité jointe
  $P(A, B) = P(B|A) * P(A)$
  $P(A, B) = P(A|B) * P(B)$
- A et B Indépendants
  $\iff P(A, B) = P(A) * P(B)$

Théorème de Bayes

$$P(A|B) = \frac{P(B|A).P(A)}{P(B)}$$

Apprentissage supervisé

Naive Bayes

Introduction

Prenons un exemple :

Soit une base de donnée de fruits contenant uniquement des bananes, oranges et courgettes.
Chaque élément possède des caractéristiques couleur, taille, sucré.
Appliquer Naive Bayes, c'est chercher le maximum de vraisemblance d'un élèments dont on ne connait pas la nature mais dont on connait les caractéristiques.
On cherche donc quelle est la plus grande probabilité :

- $P(\text{banane} \mid \text{jaune}, \text{long}, \text{sucré})$
- $P(\text{orange} \mid \text{jaune}, \text{long}, \text{sucré})$
- $P(\text{tomate} \mid \text{jaune}, \text{long}, \text{sucré})$

Calcul

Naive = toutes les variables sont considérée indépendantes, donc :

$$P(\text{banane} \mid \text{jaune}, \text{long}, \text{sucré}) =$$

$$\frac{P(\text{jaune}|\text{banane}) \times P(\text{long}|\text{banane}) \times P(\text{sucré}|\text{banane}) \times P(\text{banane})}{P(\text{jaune}) \times P(\text{long}) \times P(\text{sucré})}$$

Pour estimer les différentes probabilités, on « compte » dans notre base de donnée de fruits :

$$P(\text{sucré} \mid \text{banane}) = \frac{|\text{banane} \wedge \text{sucré}|}{|\text{banane}|}$$

Apprentissage supervisé

Regression Linéaire

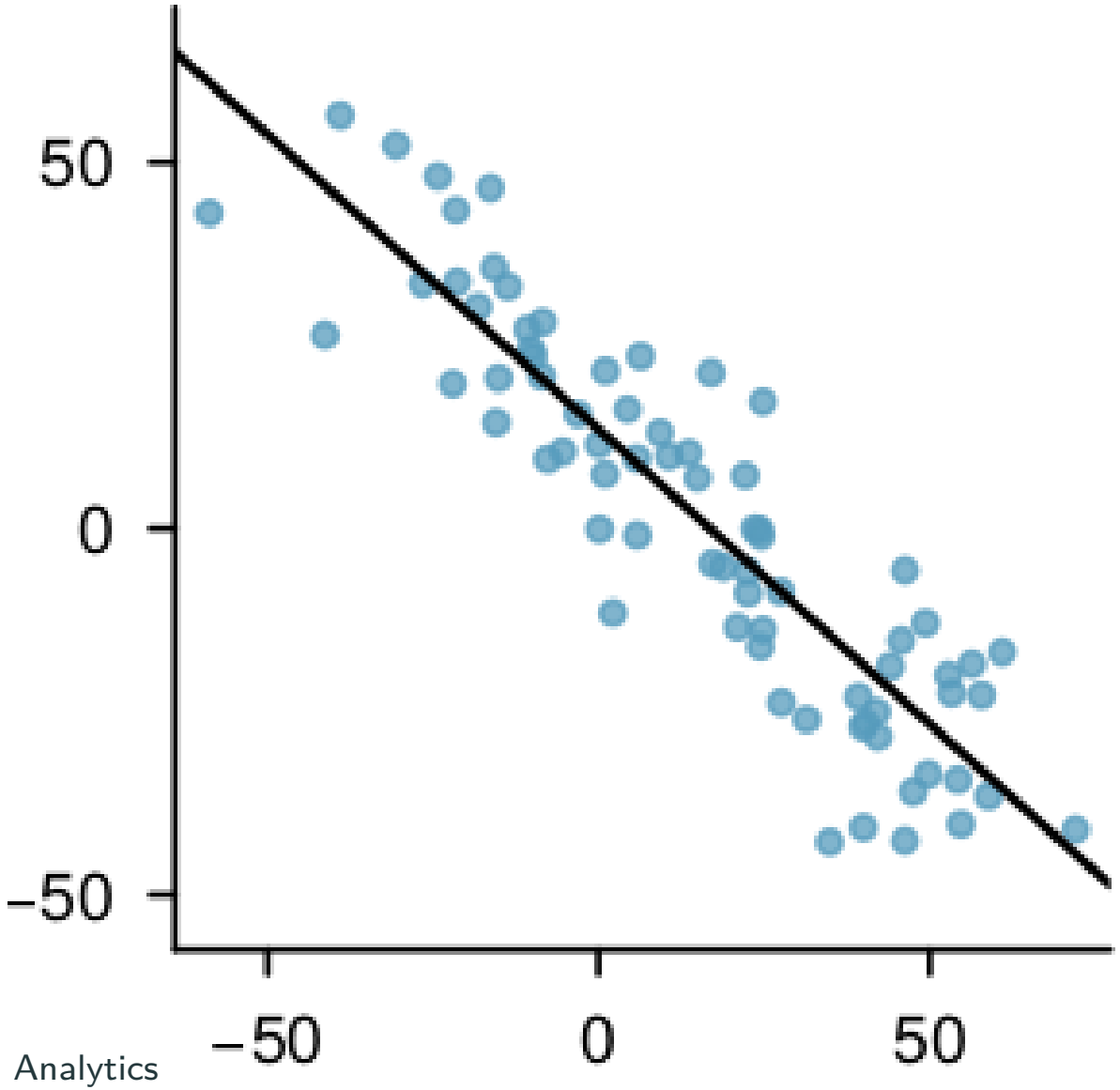
Définition du problème

Soit $\{(x_i, y_i)\}_{i \in \mathbb{R}}$ un ensemble de données tel que $\forall i, x_i \in \mathbb{R}$ et $y_i \in \mathbb{R}$

Trouver $\phi^*(x_i) = y_i^*$ telle que

$\forall i, y_i^* - y_i \to 0$
sous la contrainte que ϕ^* soit une fonction linéaire (affine)

Visualisation



Apprentissage supervisé

Fonction de Coût/Erreur

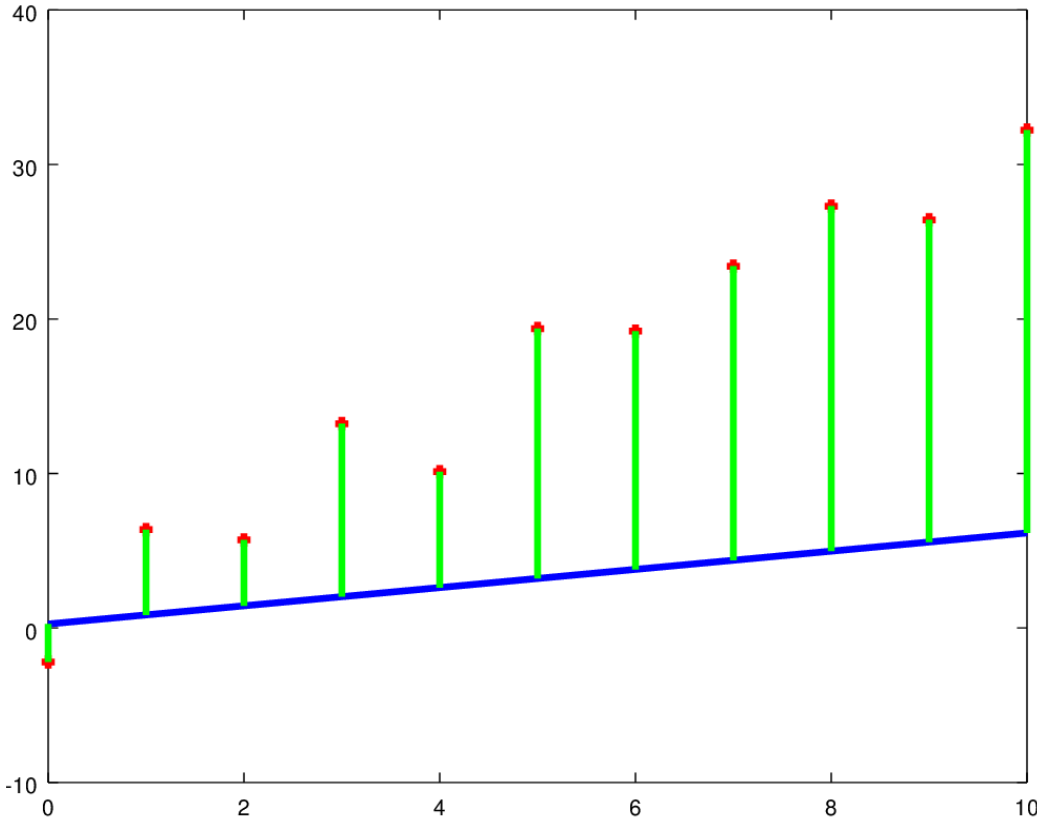
Fonction de Coût/Erreur

Erreur moyenne :

$$\frac{1}{n} \sum_{i=[1..n]} \sqrt{(\hat{y}_i - y_i)^2}$$

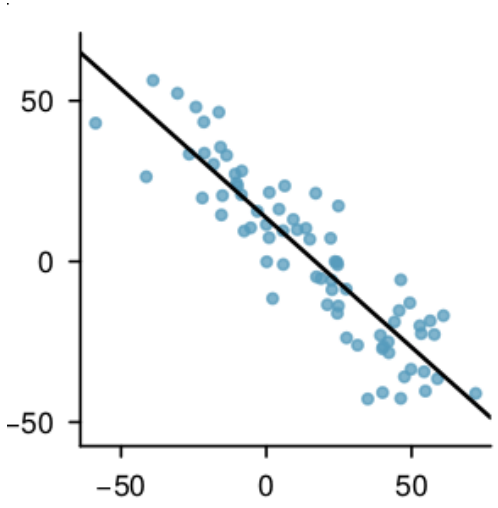
Critère des moindres carrés (M

$$\frac{1}{n} \sum_{i=[1..n]} (\hat{y}_i - y_i)^2$$



Fonction de Coût/Erreur

$$y = a.x + b$$



$$E_\Omega = \frac{1}{2n} \sum_{i=[1..n]} (\hat{y}_i - y_i)^2$$

Apprentissage supervisé

Optimisation

Optimisation

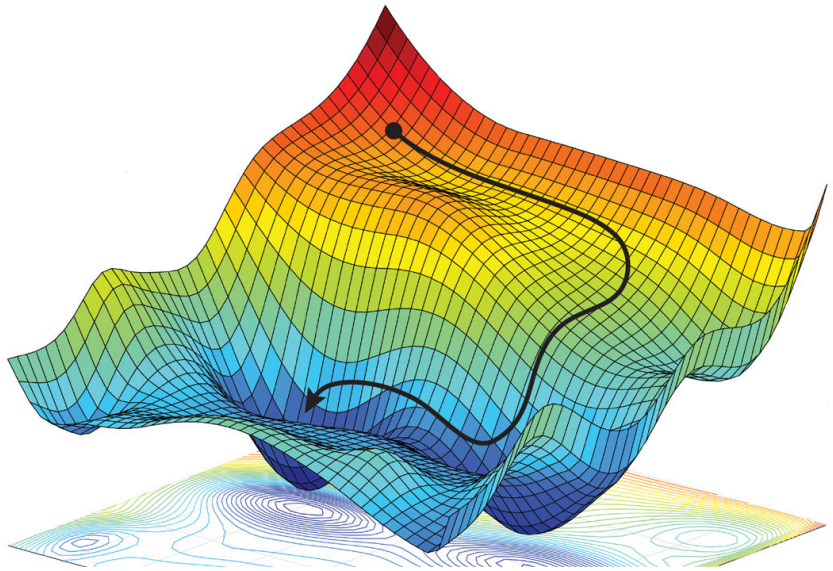
Calcul du gradient de l'erreur par rapport aux paramètres :

$$\frac{\partial Err}{\partial w_i}$$

Mise à jour :

$$w_i' = w_i - \gamma * grad$$

où : $0 < \gamma < 1$ (learning rate)



Optimisation

1 - initialisation aléatoire du modèle

2 - Tant que(critère arret $==$ 0)

- Selection aléatoire d'un **batch** de données
- **Forward** : Passe avant du **batch** dans le modèle
- Calcul de l'erreur par rapport aux sorties attendues
- **Backward** : Rétropropagation du gradient de l'erreur en fonction des paramètres dans le modèle (mise à jour du modèle)
- Calcul critère arret

Principe

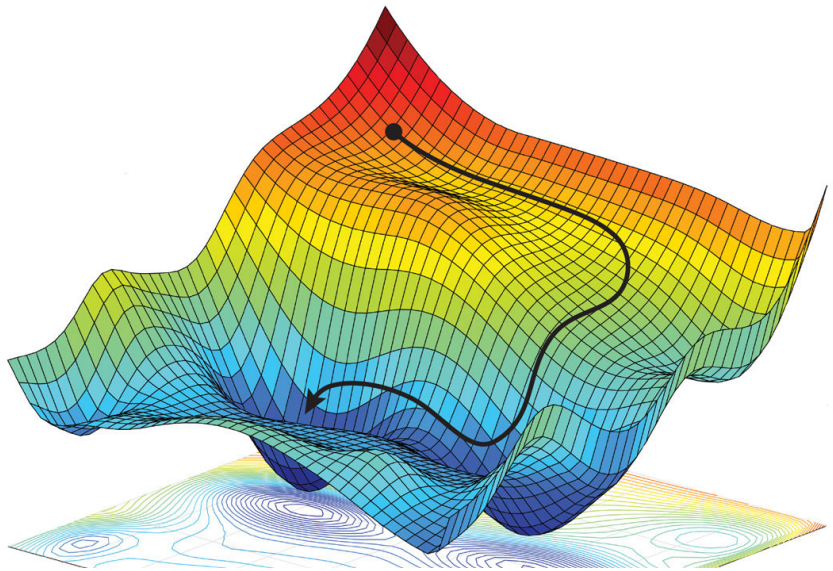
Calcul du gradient de l'erreur par rapport aux paramètres :

$$\frac{\partial Err}{\partial w_i}$$

Mise à jour :

$$w_i' = w_i - \gamma * grad$$

où : $0 < \gamma < 1$ (learning rate)



Algorithme

1. Initialisation aléatoire du modèle
2. Tant qu'aucun critère d'arrêt n'est satisfait :
   - Selection aléatoire d'un **batch** de données
   - **Forward** : Passe avant du **batch** dans le modèle
   - Calcul de l'erreur par rapport aux sorties attendues
   - **Backward** : Rétropropagation du gradient de l'erreur en fonction des paramètres dans le modèle (mise à jour du modèle)
   - Calcul des critères d'arrêt

Algorithme

1. Initialisation aléatoire du modèle
2. Tant qu'aucun critère d'arrêt n'est satisfait :
   - Selection aléatoire d'un **batch** de données
   - **Forward** : Passe avant du **batch** dans le modèle
   - Calcul de l'erreur par rapport aux sorties attendues
   - **Backward** : Rétropropagation du gradient de l'erreur en fonction des paramètres dans le modèle (mise à jour du modèle)
   - Calcul des critères d'arrêt

Algorithme

1. Initialisation aléatoire du modèle
2. Tant qu'aucun critère d'arrêt n'est satisfait :
   - Selection aléatoire d'un **batch** de données
   - **Forward** : Passe avant du **batch** dans le modèle
   - Calcul de l'erreur par rapport aux sorties attendues
   - **Backward** : Rétropropagation du gradient de l'erreur en fonction des paramètres dans le modèle (mise à jour du modèle)
   - Calcul des critères d'arrêt

Algorithme

1. Initialisation aléatoire du modèle
2. Tant qu'aucun critère d'arrêt n'est satisfait :
   - Selection aléatoire d'un **batch** de données
   - **Forward** : Passe avant du **batch** dans le modèle
   - Calcul de l'erreur par rapport aux sorties attendues
   - **Backward** : Rétropropagation du gradient de l'erreur en fonction des paramètres dans le modèle (mise à jour du modèle)
   - Calcul des critères d'arrêt

Algorithme

1. Initialisation aléatoire du modèle
2. Tant qu'aucun critère d'arrêt n'est satisfait :
   - Selection aléatoire d'un **batch** de données
   - **Forward** : Passe avant du **batch** dans le modèle
   - Calcul de l'erreur par rapport aux sorties attendues
   - **Backward** : Rétropropagation du gradient de l'erreur en fonction des paramètres dans le modèle (mise à jour du modèle)
   - Calcul des critères d'arrêt

Algorithme

1. Initialisation aléatoire du modèle
2. Tant qu'aucun critère d'arrêt n'est satisfait :
   - Selection aléatoire d'un **batch** de données
   - **Forward** : Passe avant du **batch** dans le modèle
   - Calcul de l'erreur par rapport aux sorties attendues
   - **Backward** : Rétropropagation du gradient de l'erreur en fonction des paramètres dans le modèle (mise à jour du modèle)
   - Calcul des critères d'arrêt

Algorithme

1. Initialisation aléatoire du modèle
2. Tant qu'aucun critère d'arrêt n'est satisfait :
   - Selection aléatoire d'un **batch** de données
   - **Forward** : Passe avant du **batch** dans le modèle
   - Calcul de l'erreur par rapport aux sorties attendues
   - **Backward** : Rétropropagation du gradient de l'erreur en fonction des paramètres dans le modèle (mise à jour du modèle)
   - Calcul des critères d'arrêt

Exemple de dérivation — Régression linéaire

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

$$E_\Omega = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - (ax_i + b))^2$$

...

$$\frac{\partial E_\Omega}{\partial a} = \frac{1}{n}\sum_{i=1}^{n}(ax_i + b - \hat{y}_i)x_i$$

$$\frac{\partial E_\Omega}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}(ax_i + b - \hat{y}_i)$$

$$\boxed{y = ax + b}$$

$$\boxed{U^{2\,\prime} = 2U' \times U}$$

Mise à jour

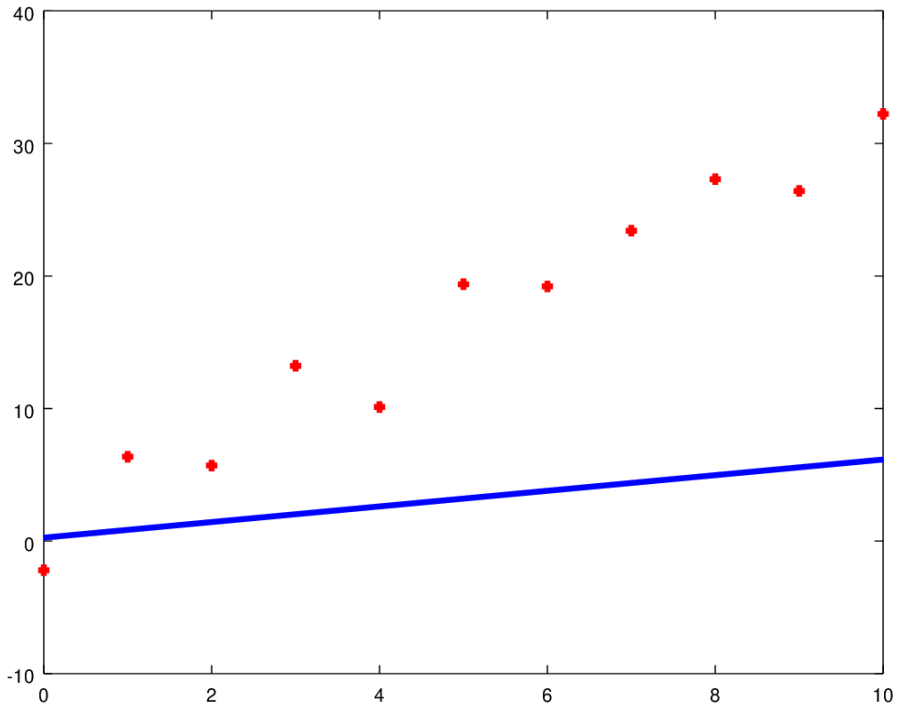
$$a \leftarrow a - \gamma\frac{\partial E_\Omega}{\partial a}$$
$$b \leftarrow b - \gamma\frac{\partial E_\Omega}{\partial b}$$

où $0 < \gamma < 1$ (pas d'apprentissage)

Exemple

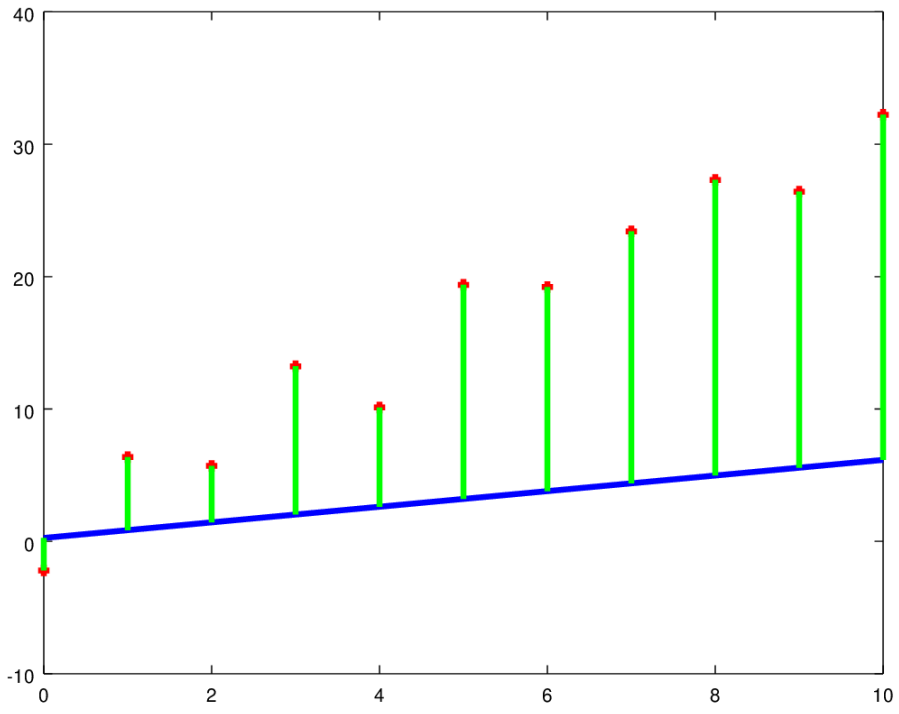
Initialisation au hasard ($\gamma = 0.01$)

- a $= 0.58$ ($\hat{a} = 3.0$)
- b $= 0.25$ ($\hat{b} = 0.5$)



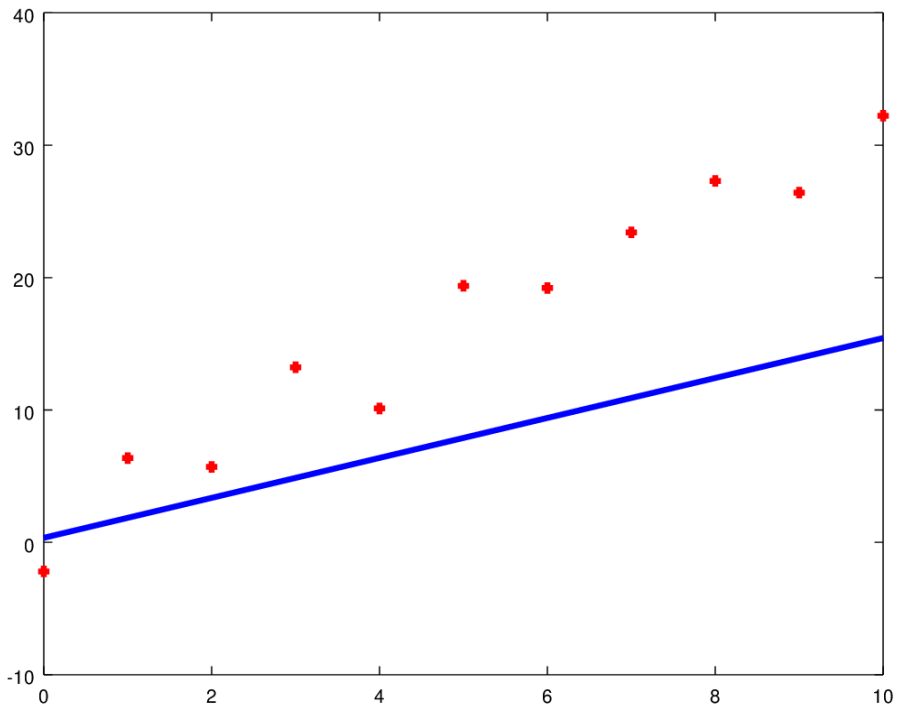
Exemple

- a $= 0.58$ $(\hat{a} = 3.0)$
- b $= 0.25$ $(\hat{b} = 0.5)$



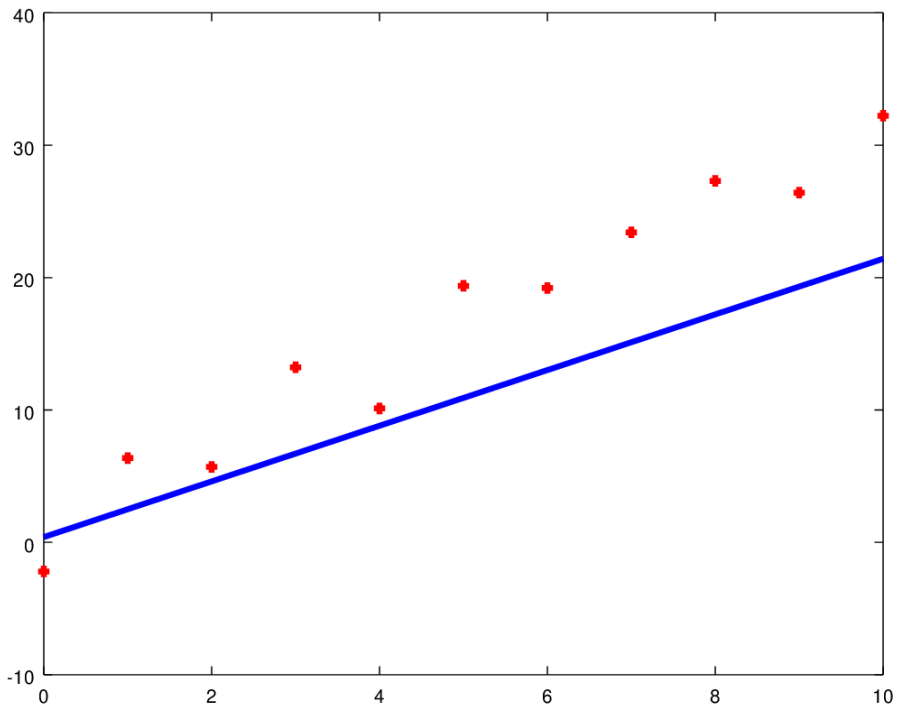
Exemple

- a $= 1.50$ $(\hat{a} = 3.0)$
- b $= 0.35$ $(\hat{b} = 0.5)$



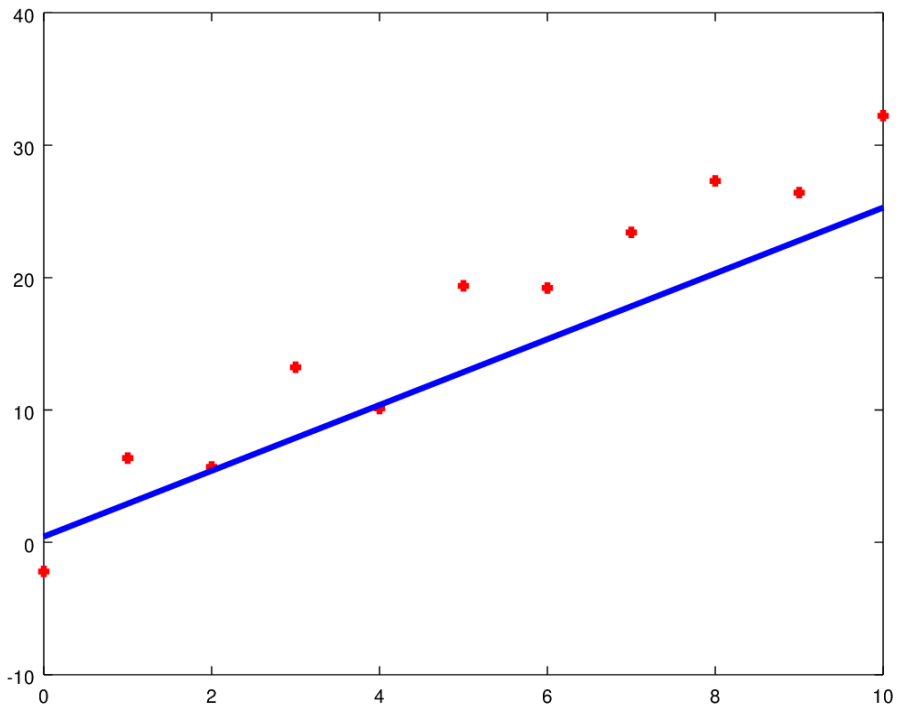
Exemple

- a $= 2.10$ $(\hat{a} = 3.0)$
- b $= 0.40$ $(\hat{b} = 0.5)$



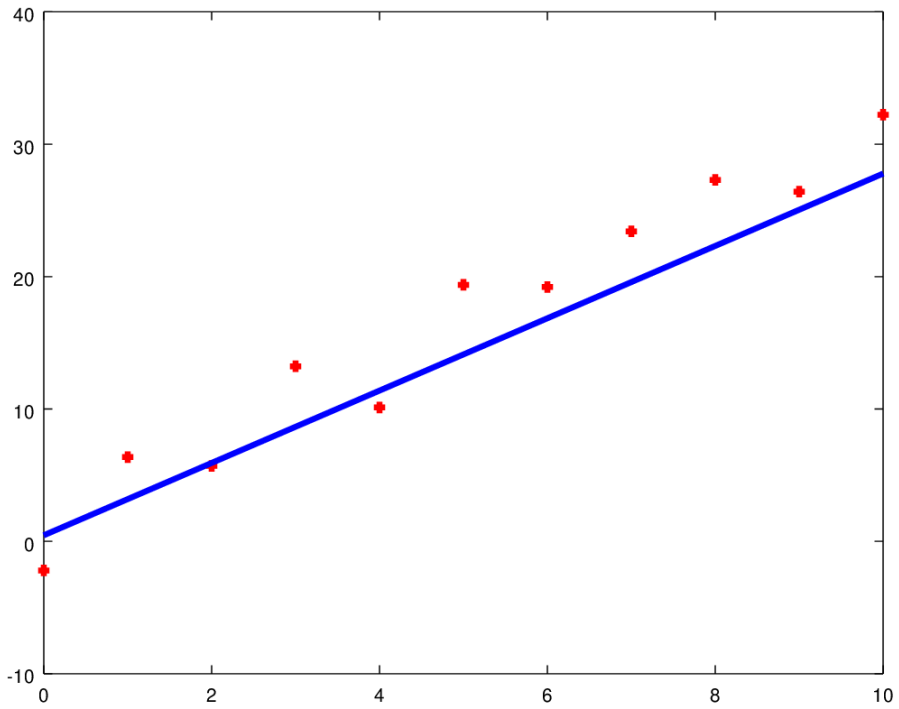
Exemple

- a $= 2.48$ $(\hat{a} = 3.0)$
- b $= 0.43$ $(\hat{b} = 0.5)$



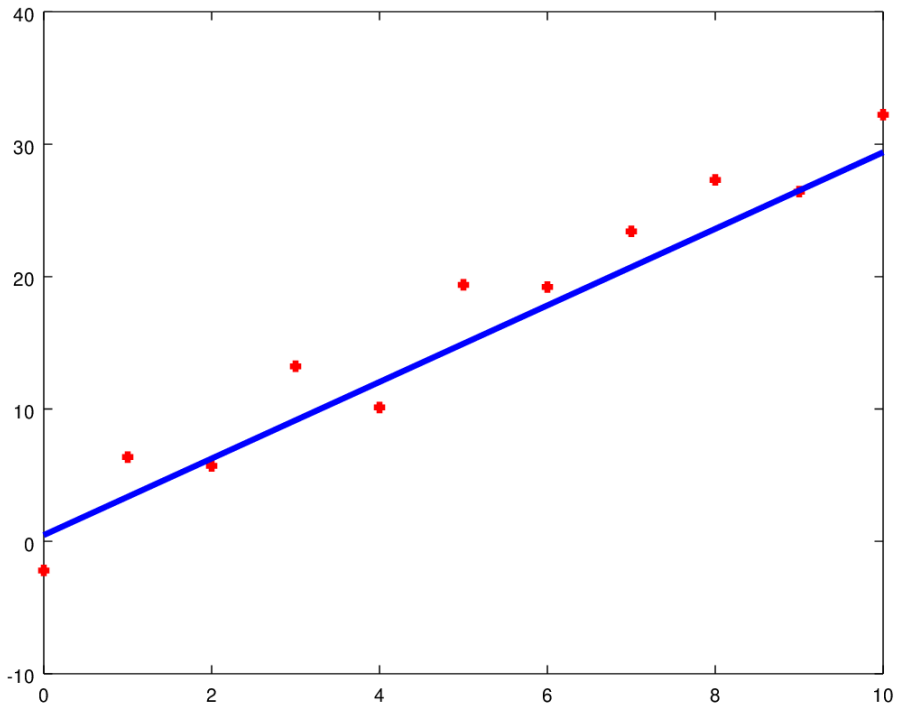
Exemple

- a $= 2.73$ $(\hat{a} = 3.0)$
- b $= 0.46$ $(\hat{b} = 0.5)$



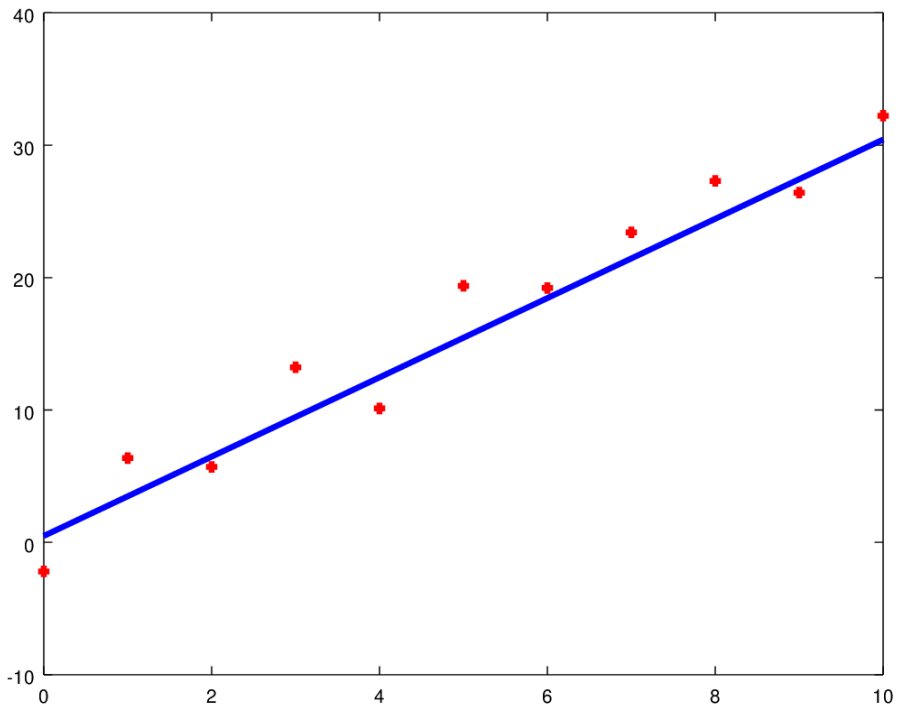
Exemple

- a $= 2.89$ $(\hat{a} = 3.0)$
- b $= 0.47$ $(\hat{b} = 0.5)$



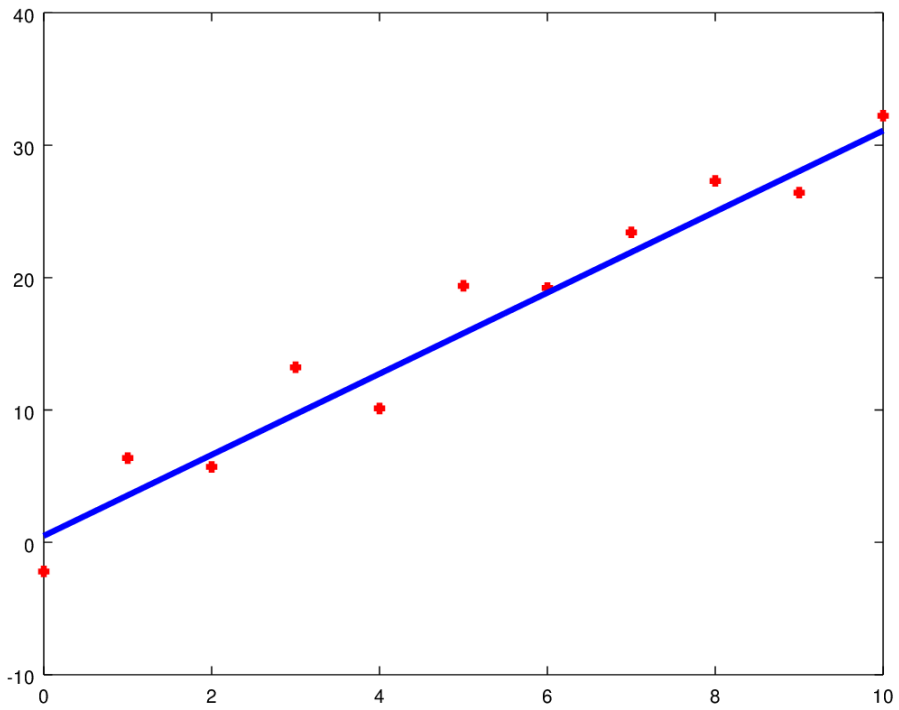
Exemple

- a $= 2.99$ $(\hat{a} = 3.0)$
- b $= 0.48$ $(\hat{b} = 0.5)$



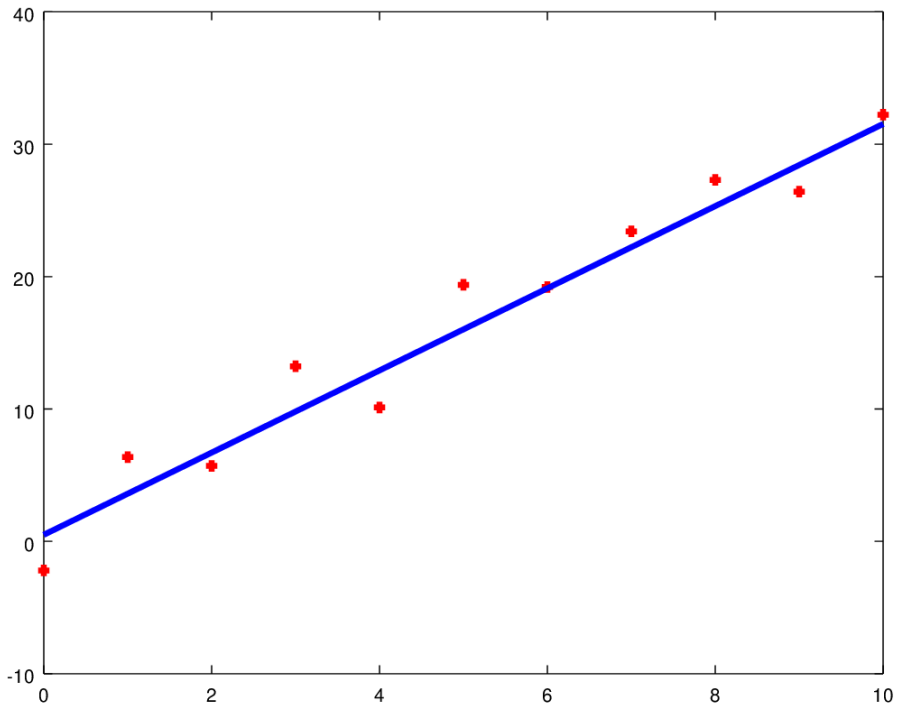
Exemple

- a $= 3.06$ $(\hat{a} = 3.0)$
- b $= 0.49$ $(\hat{b} = 0.5)$



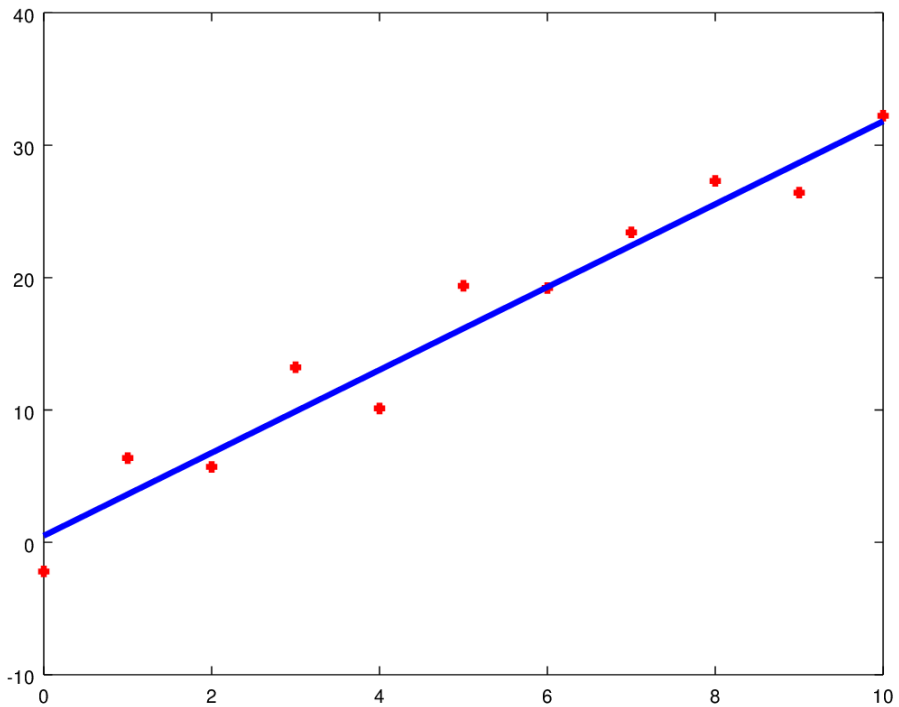
Exemple

- a $= 3.10$ $(\hat{a} = 3.0)$
- b $= 0.49$ $(\hat{b} = 0.5)$



Exemple

- a $= 3.13$ $(\hat{a} = 3.0)$
- b $= 0.50$ $(\hat{b} = 0.5)$

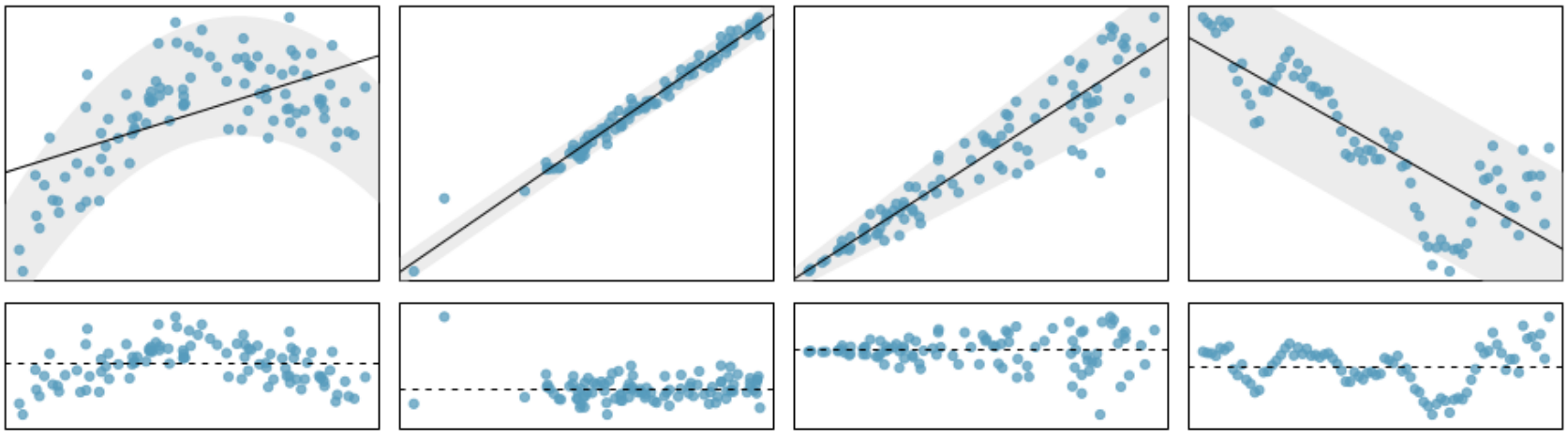


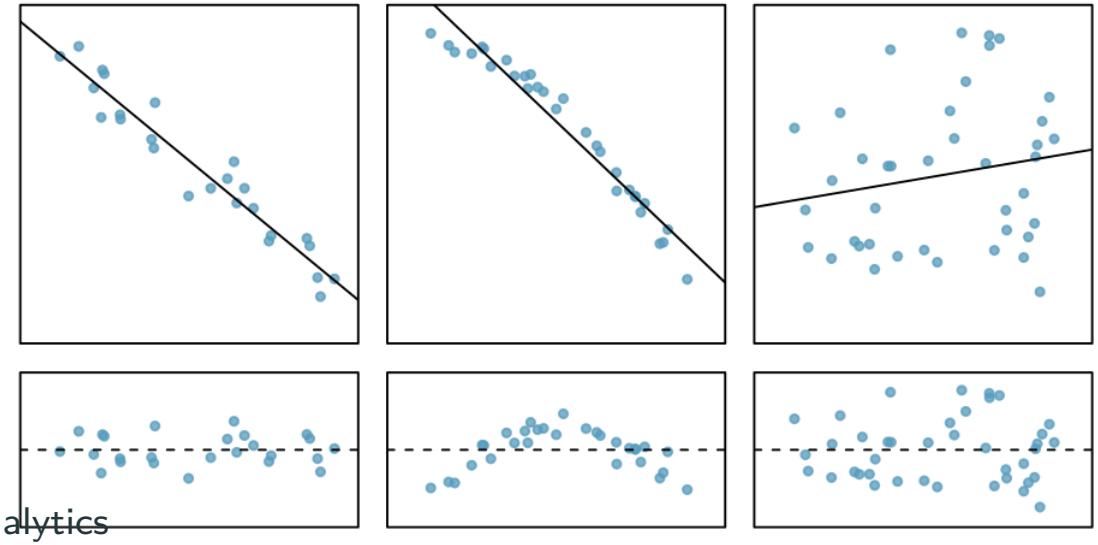
Apprentissage supervisé

Régression Polynomiale

Regression Polynomiale

Limites de la regression linéaire

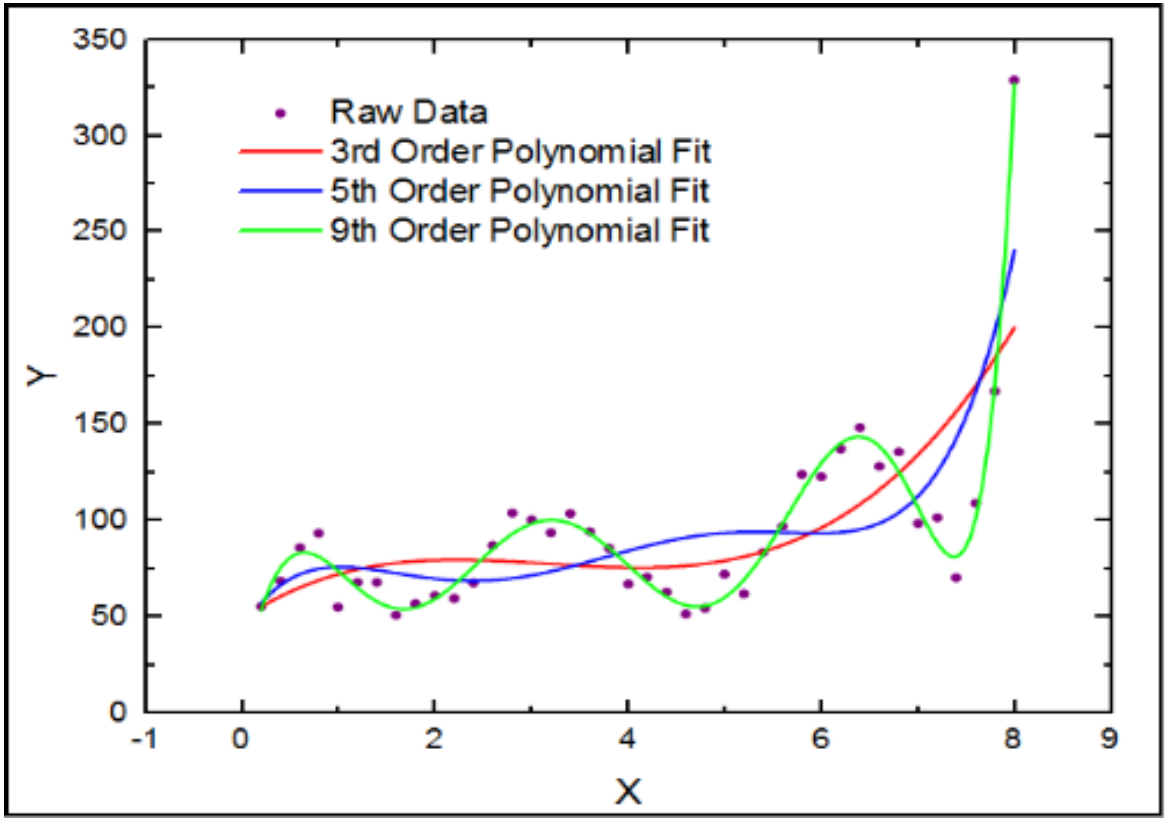




Régression Polynomiale

$$Y = a_0 + a_1 * X + a_2 * X^2 + a_3 * X^3 + ... + a_n * X^n$$

$$Y = \sum_{k \in [0..n]} a_k * X^k$$



Gradient et mise à jour

$$Y = \sum_{k \in [0..n]} a_k * X^k$$

$$E_\Omega = \frac{1}{2n} \sum_{i=[1..n]} (\hat{Y}_i - Y_i)^2$$

$$\frac{\partial E_\Omega}{\partial a_k} = \frac{1}{n} \sum_{i=[1..n]} (Y_i - \hat{Y}_i).X_i^k$$

M.A.J :

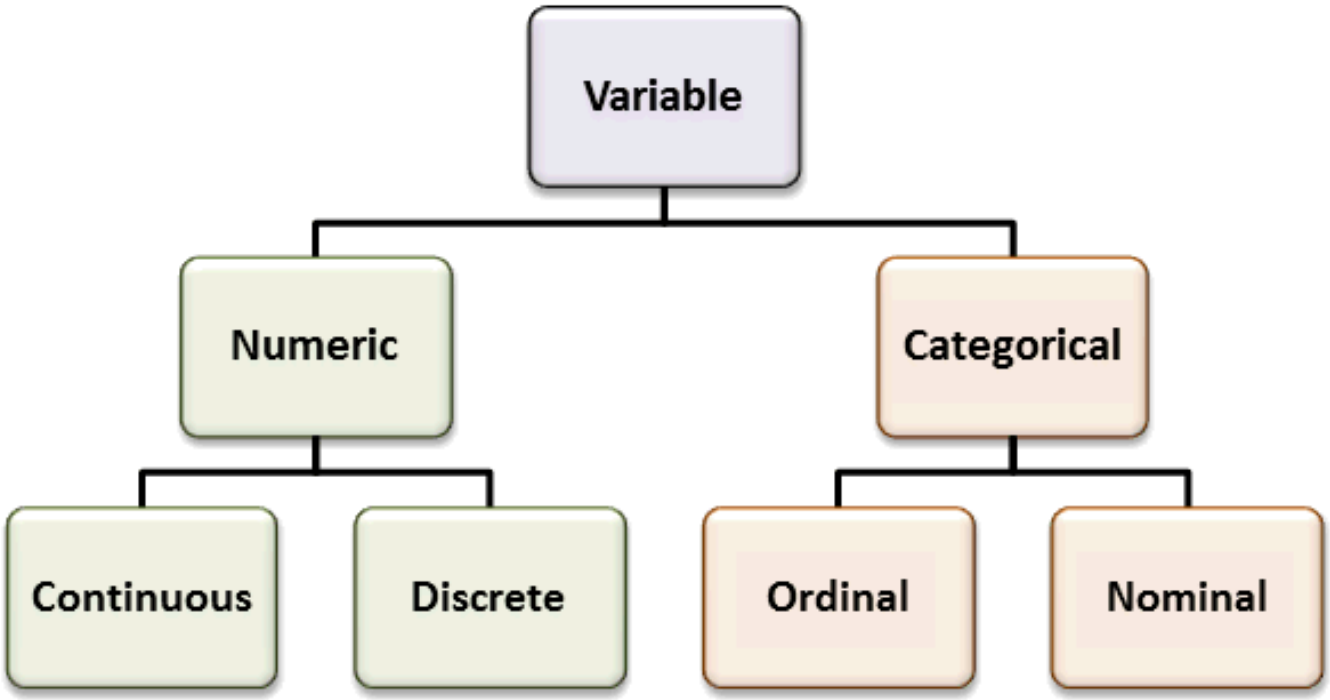
$$a_k \leftarrow a_k - \gamma . \frac{\partial E_\Omega}{\partial a_k}$$

où $1 > \gamma > 0$ (learning rate)

Apprentissage supervisé

Classification

Classification

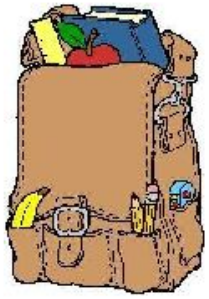


Classification

Classification


001.ak47


002.american-flag


003.backpack


004.baseball-bat


005.baseball-glove


006.basketball-hoop


007.bat


008.bathtub


009.bear


010.beer-mug


011.billiards


012.binoculars

Classification

$\approx$ Distance entre la sortie et la cible ?

Sortie :

0.00	0.10	0.40	0.00	0.00	0.20	0.10	0.00	0.20	0.00

Cible :

0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Classification

Critère des moindre carré (MSE) = 0.12

Sortie :

0.00	0.10	0.40	0.00	0.00	0.20	0.10	0.00	0.20	0.00

Cible :

0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Classification

Critère des moindre carré (MSE) = 0.12

Sortie :

0.12	0.12	0.88	0.12	0.12	0.12	0.12	0.12	0.12	0.12

Cible :

0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Classification

$\Rightarrow$ entropie croisée entre la sortie et la cible :

$$\mathrm{H}(p, q) = -\sum_{x} p(x) \log q(x)$$

- minimale quand sortie $=$ cible
- prend en compte la distribution de la sortie

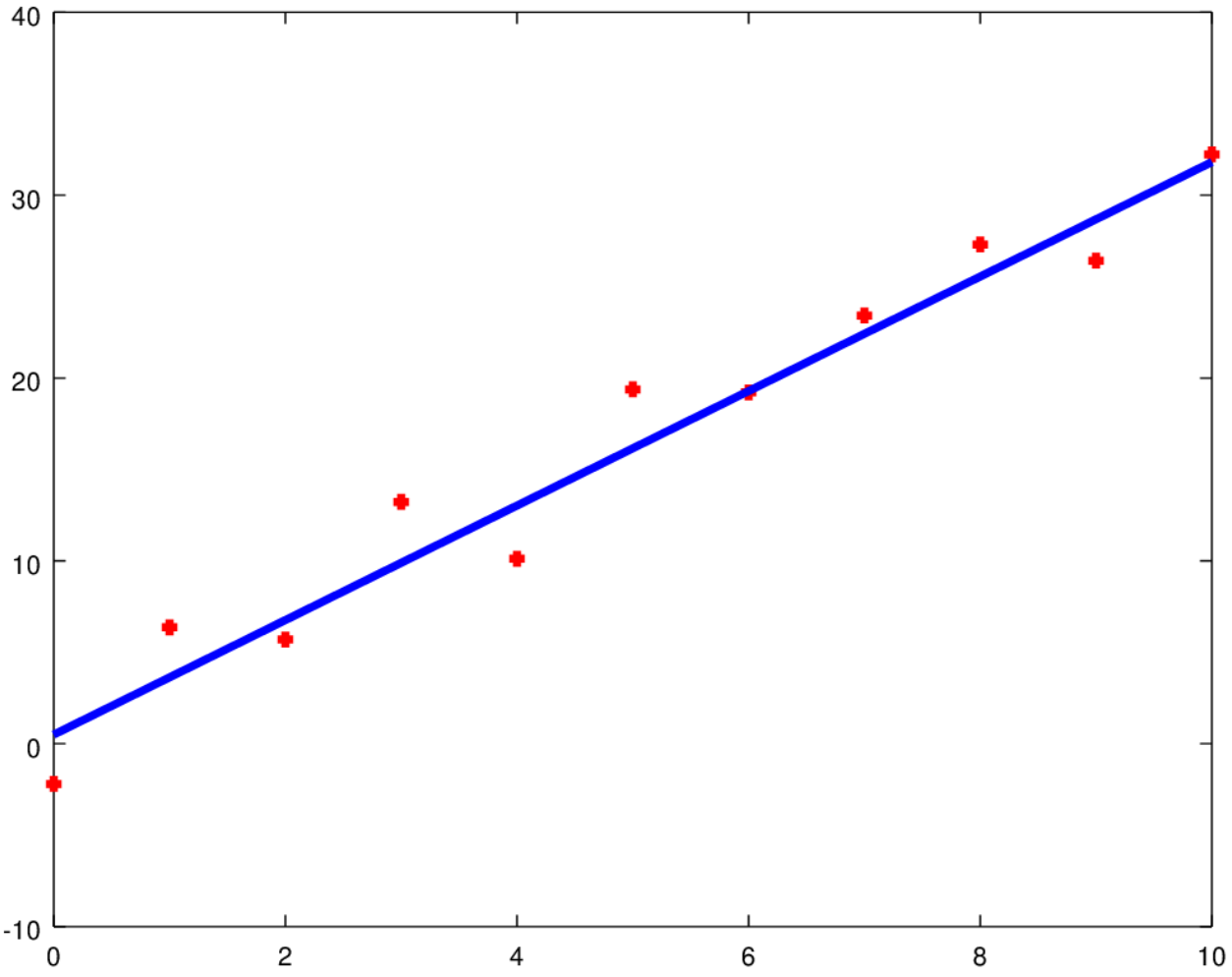
Classification

$\Rightarrow$ entropie croisée entre la sortie et la cible :

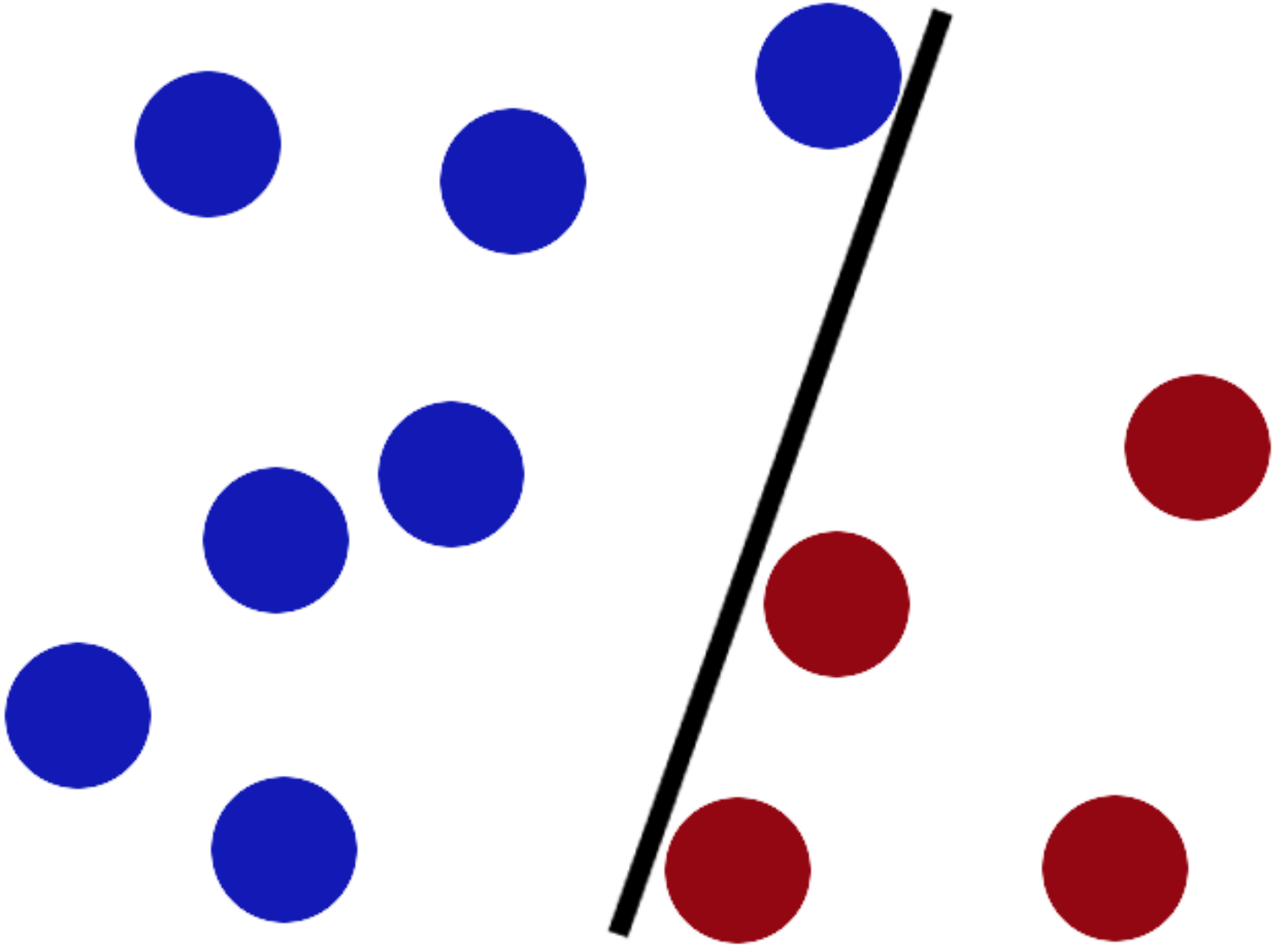
$$\mathrm{H}(p, q) = -\sum_{x} p(x) \log q(x)$$

- minimale quand sortie $=$ cible
- prend en compte la distribution de la sortie

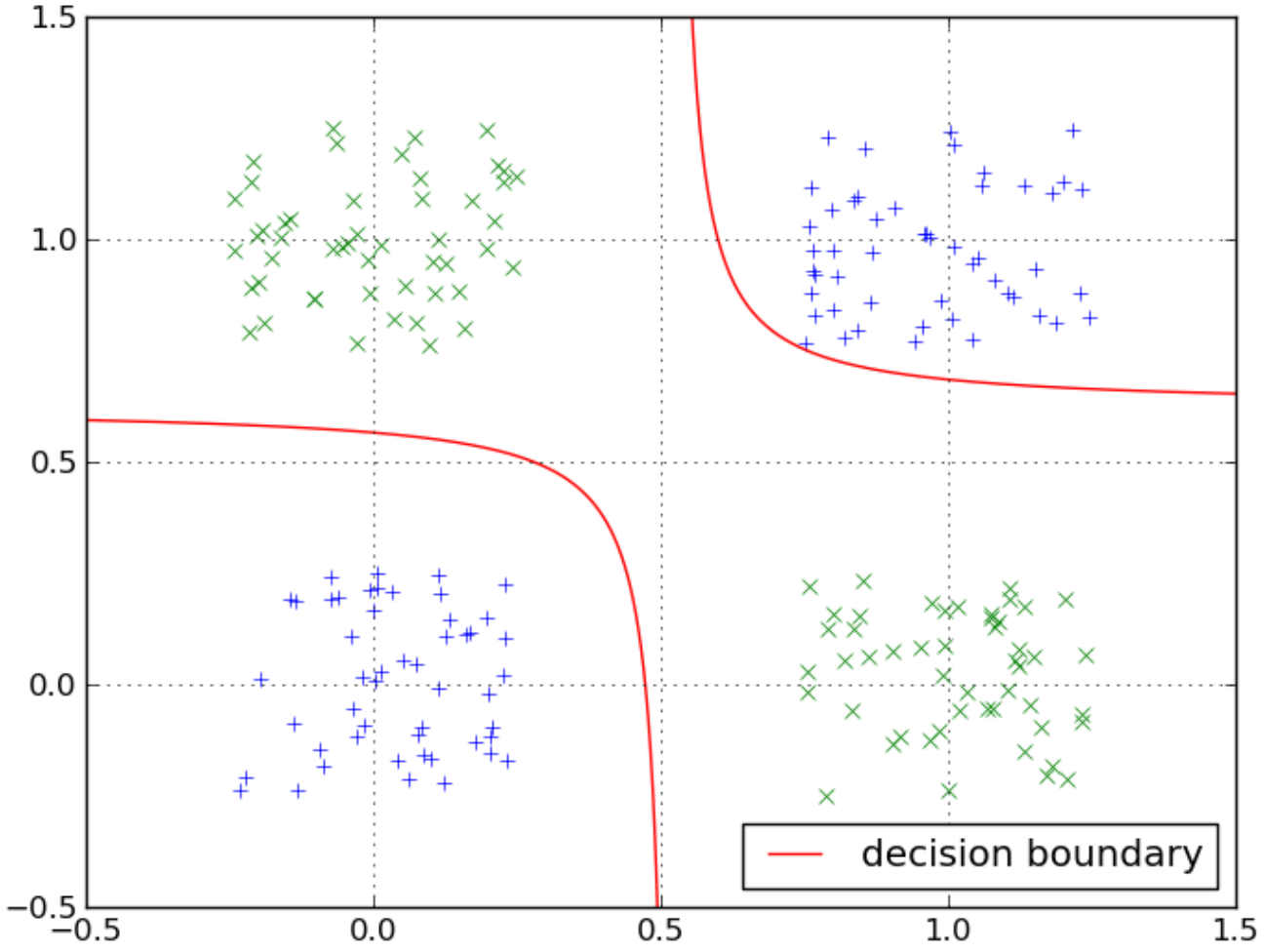
Lien avec la régression linéaire



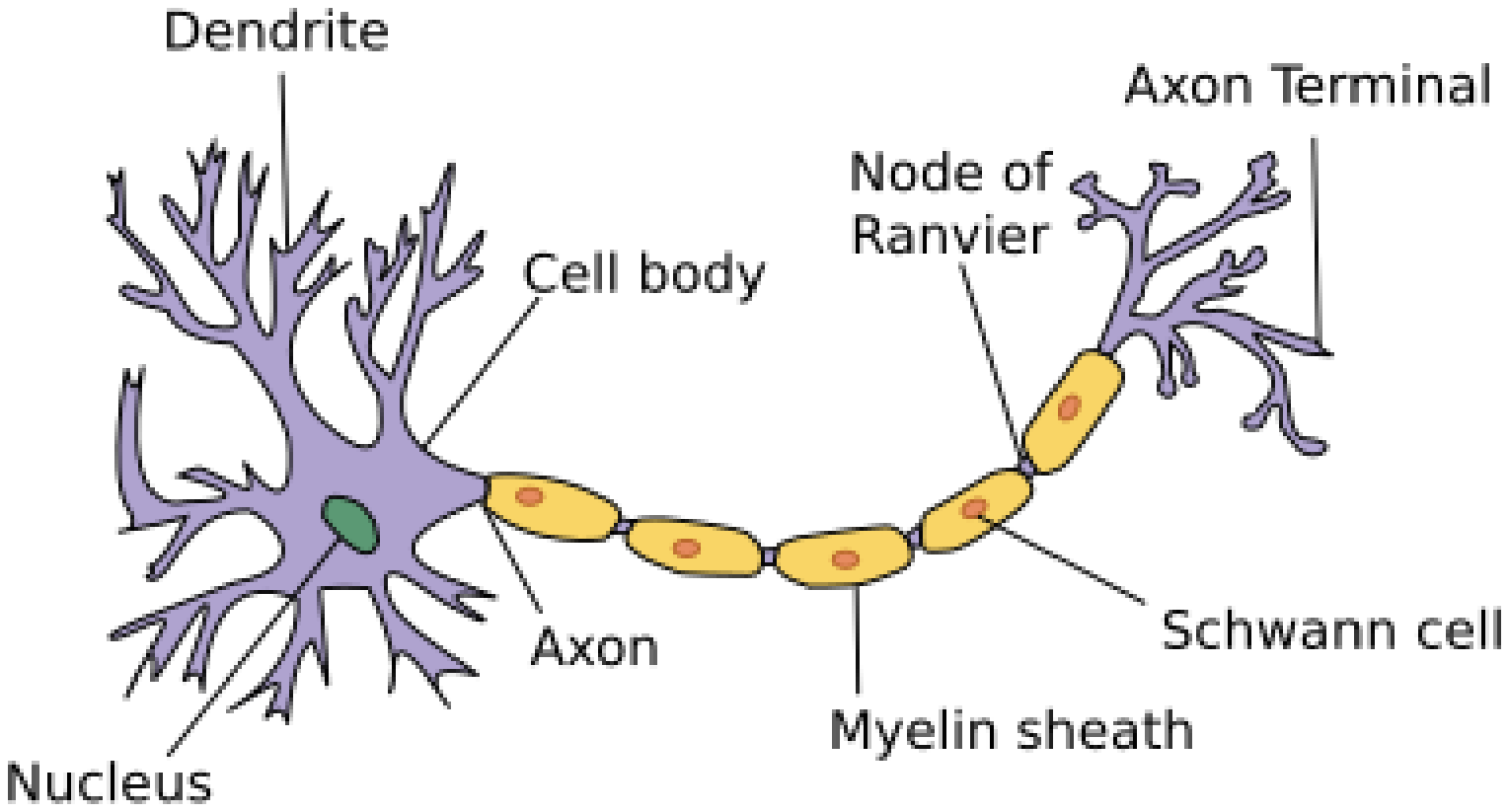
Lien avec la régression linéaire



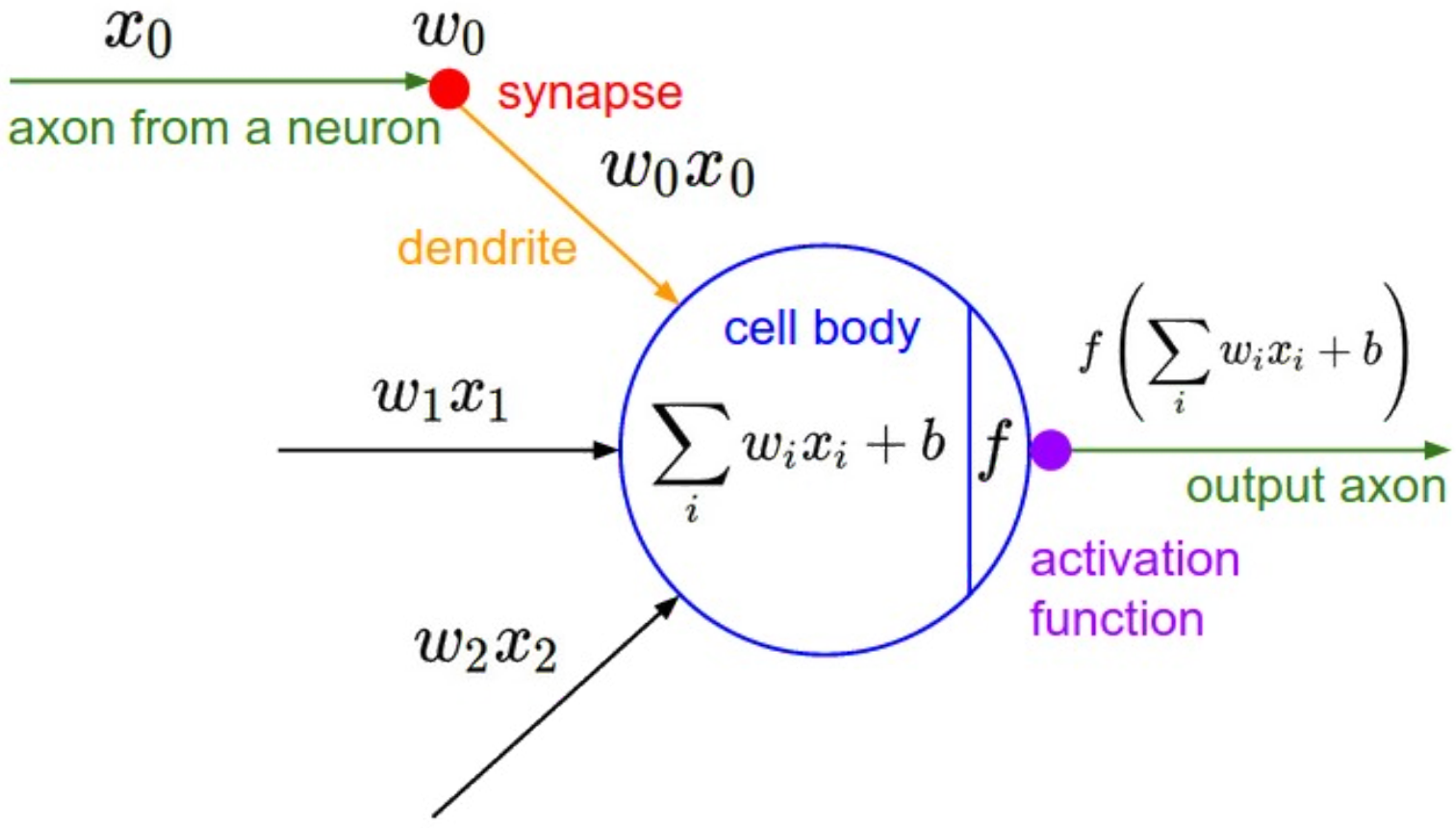
Lien avec la régression linéaire



Lien (ténu) avec la biologie



Modélisation d'un neurone



Modélisation d'un réseau de neurones

Agencement de beaucoup de neurones :

En parallèle Calculent des résultats indépendamment dans la même couche

En série Prennent en entrée les résultats des neurones de la couche précédente

Modélisation d'un réseau de neurones

Agencement de beaucoup de neurones :

En parallèle Calculent des résultats indépendamment dans la même couche

En série Prennent en entrée les résultats des neurones de la couche précédente

Deux types de neurones

On distingue deux types de neurones :

Neurones cachés Neurones des couches intermédiaires. Améliorent l'expressivité du modèle

Neurones de sortie Neurones de la couche finale. Contraints par le type de sortie attendu

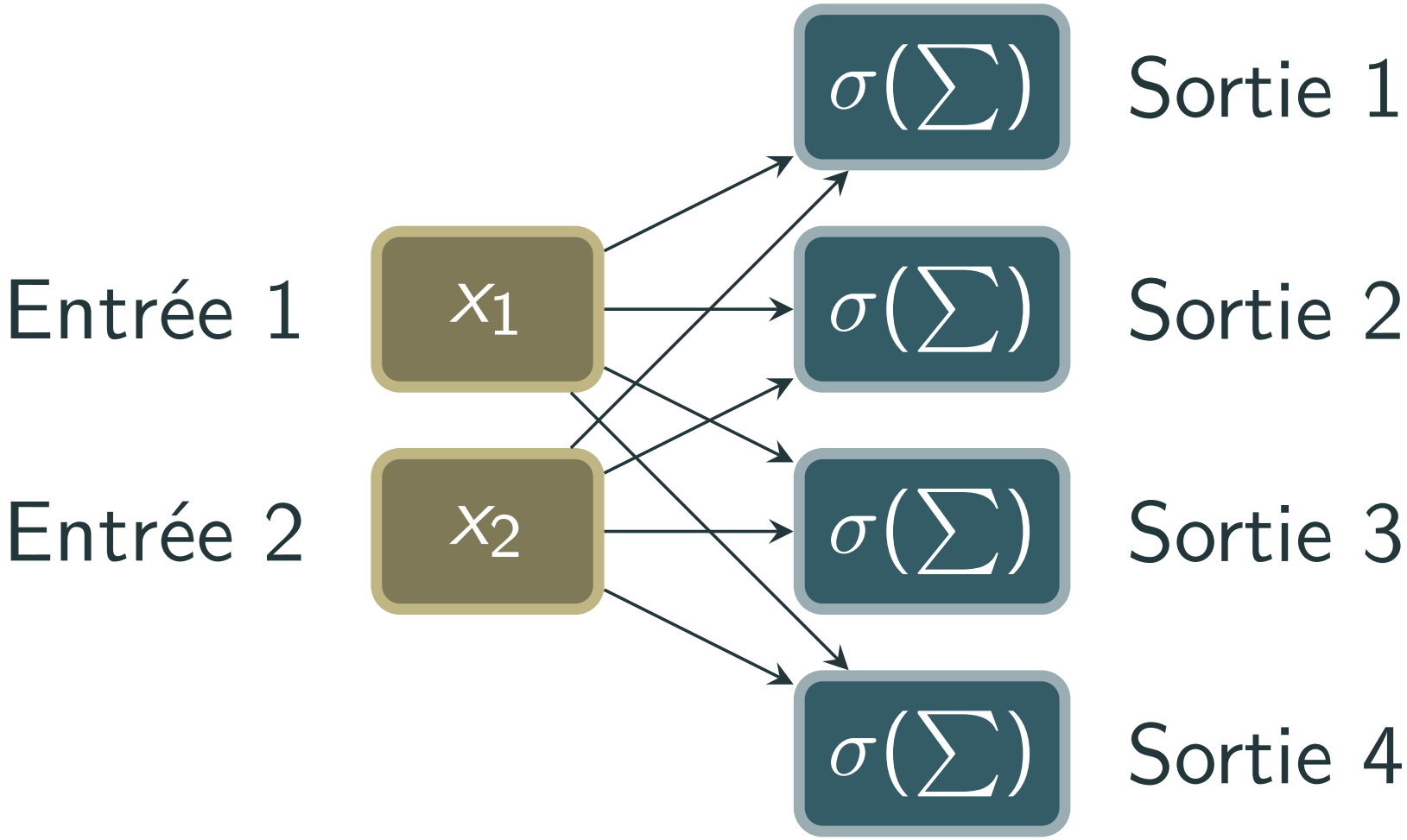
Deux types de neurones

On distingue deux types de neurones :

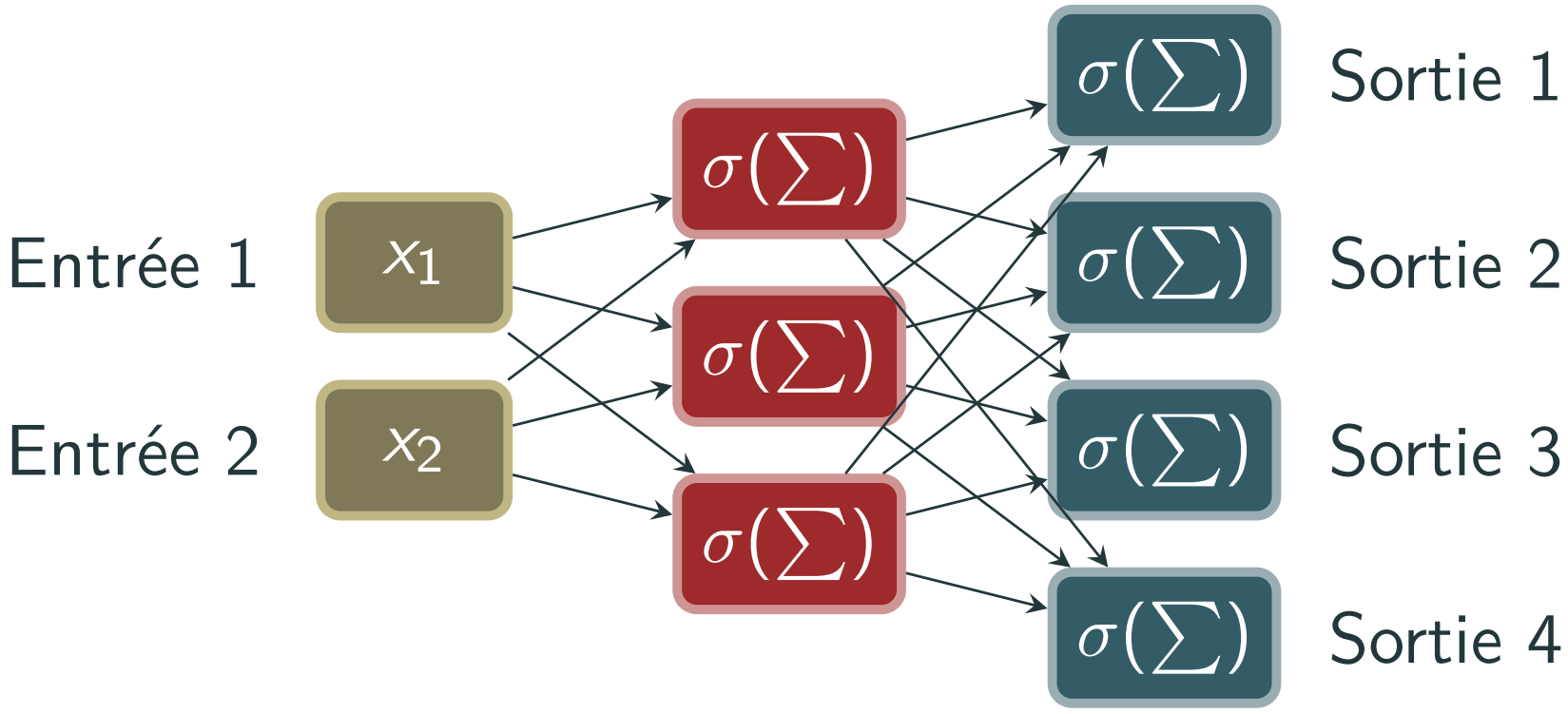
Neurones cachés Neurones des couches intermédiaires. Améliorent l'expressivité du modèle

Neurones de sortie Neurones de la couche finale. Contraints par le type de sortie attendu

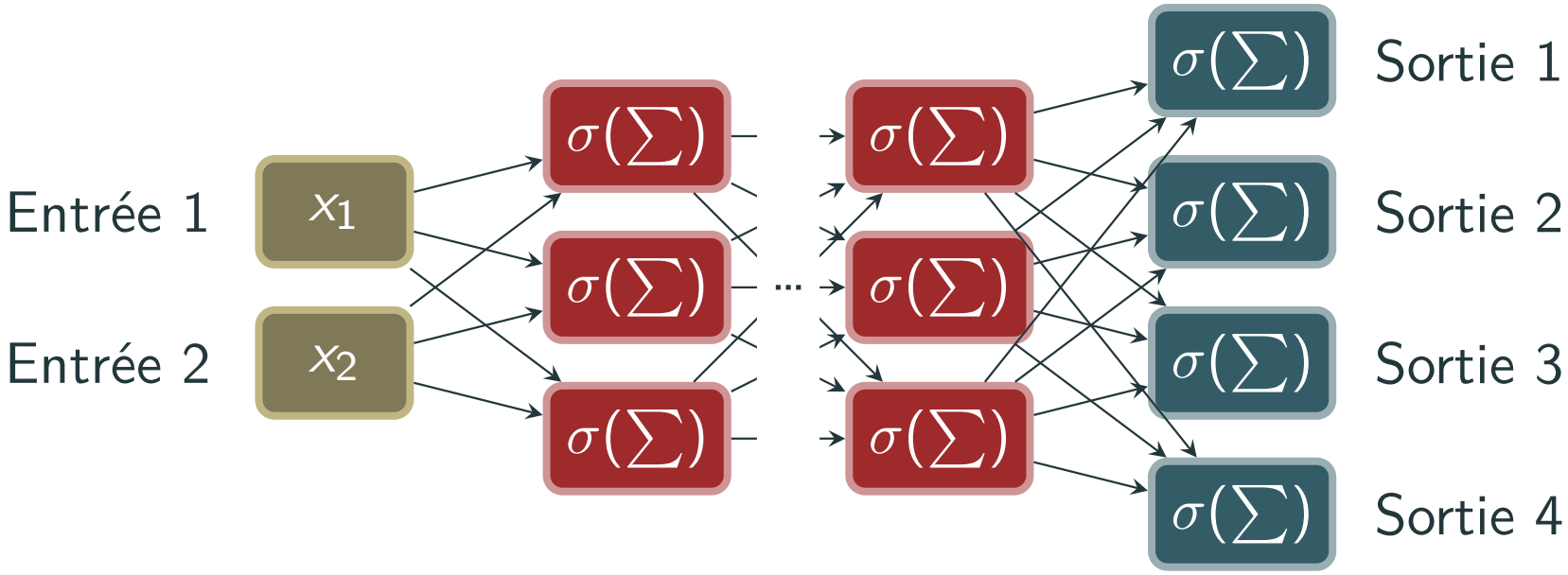
Réseau sans couche cachée



Réseau avec une couche cachée

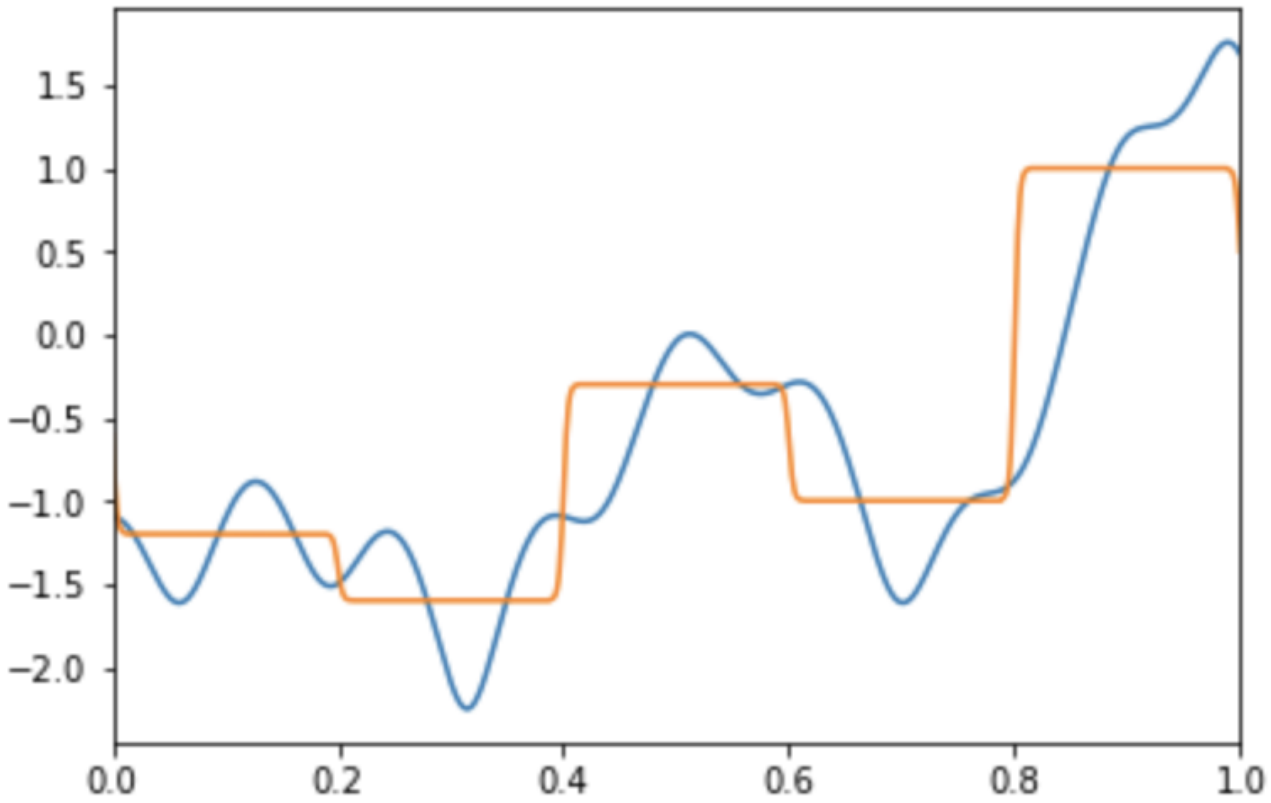


Réseau profond



Un potentiel infini

Kurt Hornik, 1991 : Théorème d'approximation universelle



Démonstration

Visualisation d'un réseau simple

Modélisation matricielle — Échantillon

Représentable sous forme de vecteur à d colonnes correspondant à d caractéristiques :

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_d \end{bmatrix}$$

Voire même de matrice dans le cas d'un batch (groupe d'échantillons) :

$$\mathbf{X} = \begin{bmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,d} \\ X_{2,1} & X_{2,2} & \dots & X_{2,d} \end{bmatrix}$$

Modélisation matricielle — Poids

Représentables sous forme de matrice de poids et de vecteur de biais :

$$\mathbf{W} = \begin{bmatrix} W_{1,1} & W_{1,2} & \dots & W_{1,n} \\ W_{2,1} & W_{2,2} & \dots & W_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ W_{d,1} & W_{d,2} & \dots & W_{d,n} \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} b_1 & b_2 & \dots & b_n \end{bmatrix}$$

Modélisation matricielle complète

$$O = \sigma(\mathbf{X}.\mathbf{W} + \mathbf{b})$$

$$= \sigma\left(\begin{bmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,d} \\ X_{2,1} & X_{2,2} & \dots & X_{2,d} \end{bmatrix} \begin{bmatrix} W_{1,1} & W_{1,2} & \dots & W_{1,n} \\ W_{2,1} & W_{2,2} & \dots & W_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ W_{d,1} & W_{d,2} & \dots & W_{d,n} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 & \dots & b_n \end{bmatrix}\right)$$

$$= \begin{bmatrix} O_{1,1} & O_{1,2} & \dots & O_{1,n} \\ O_{2,1} & O_{2,2} & \dots & O_{2,n} \end{bmatrix}$$

où :

- $\mathbf{X}$ Données en entrée de dimension d
- $\mathbf{W}$ & $\mathbf{b}$ Paramètres à trouver des n neurones de notre modèle
- σ Fonction d'activation
- $\mathbf{O}$ Sortie du réseau

Modélisation matricielle complète

$$O = \sigma(\mathbf{X}.\mathbf{W} + \mathbf{b})$$

$$= \sigma\left(\begin{bmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,d} \\ X_{2,1} & X_{2,2} & \dots & X_{2,d} \end{bmatrix} \begin{bmatrix} W_{1,1} & W_{1,2} & \dots & W_{1,n} \\ W_{2,1} & W_{2,2} & \dots & W_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ W_{d,1} & W_{d,2} & \dots & W_{d,n} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 & \dots & b_n \end{bmatrix}\right)$$

$$= \begin{bmatrix} O_{1,1} & O_{1,2} & \dots & O_{1,n} \\ O_{2,1} & O_{2,2} & \dots & O_{2,n} \end{bmatrix}$$

où :

X Données en entrée de dimension d

W & b Paramètres à trouver des n neurones de notre modèle

σ Fonction d'activation

O Sortie du réseau

Modélisation matricielle complète

$$
\begin{aligned}
O &= \sigma(\mathbf{X}.\mathbf{W} + \mathbf{b}) \\
&= \sigma\left(\begin{bmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,d} \\ X_{2,1} & X_{2,2} & \dots & X_{2,d} \end{bmatrix} \begin{bmatrix} W_{1,1} & W_{1,2} & \dots & W_{1,n} \\ W_{2,1} & W_{2,2} & \dots & W_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ W_{d,1} & W_{d,2} & \dots & W_{d,n} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 & \dots & b_n \end{bmatrix}\right) \\
&= \begin{bmatrix} O_{1,1} & O_{1,2} & \dots & O_{1,n} \\ O_{2,1} & O_{2,2} & \dots & O_{2,n} \end{bmatrix}
\end{aligned}
$$

où :

- $\mathbf{X}$ Données en entrée de dimension d
- $\mathbf{W}$ & $\mathbf{b}$ Paramètres à trouver des n neurones de notre modèle
- σ Fonction d'activation
- $\mathbf{O}$ Sortie du réseau

Modélisation matricielle complète

$$O = \sigma(\mathbf{X}.\mathbf{W} + \mathbf{b})$$

$$= \sigma\left(\begin{bmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,d} \\ X_{2,1} & X_{2,2} & \dots & X_{2,d} \end{bmatrix} \begin{bmatrix} W_{1,1} & W_{1,2} & \dots & W_{1,n} \\ W_{2,1} & W_{2,2} & \dots & W_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ W_{d,1} & W_{d,2} & \dots & W_{d,n} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 & \dots & b_n \end{bmatrix}\right)$$

$$= \begin{bmatrix} O_{1,1} & O_{1,2} & \dots & O_{1,n} \\ O_{2,1} & O_{2,2} & \dots & O_{2,n} \end{bmatrix}$$

où :

- **X** Données en entrée de dimension d
- **W & b** Paramètres à trouver des n neurones de notre modèle
- σ Fonction d'activation
- **O** Sortie du réseau

Fonctions d'activation — Critères de choix

- Propriétés mathématiques (conservation du gradient)
- Propriétés d'apprentissage (éviter la création de poids morts)
- Rapidité de calcul
- Intervalle de sortie pour la dernière couche

Fonctions d'activation — Critères de choix

- Propriétés mathématiques (conservation du gradient)
- Propriétés d'apprentissage (éviter la création de poids morts)
- Rapidité de calcul
- Intervalle de sortie pour la dernière couche

Fonctions d'activation — Critères de choix

- Propriétés mathématiques (conservation du gradient)
- Propriétés d'apprentissage (éviter la création de poids morts)
- Rapidité de calcul
- Intervalle de sortie pour la dernière couche

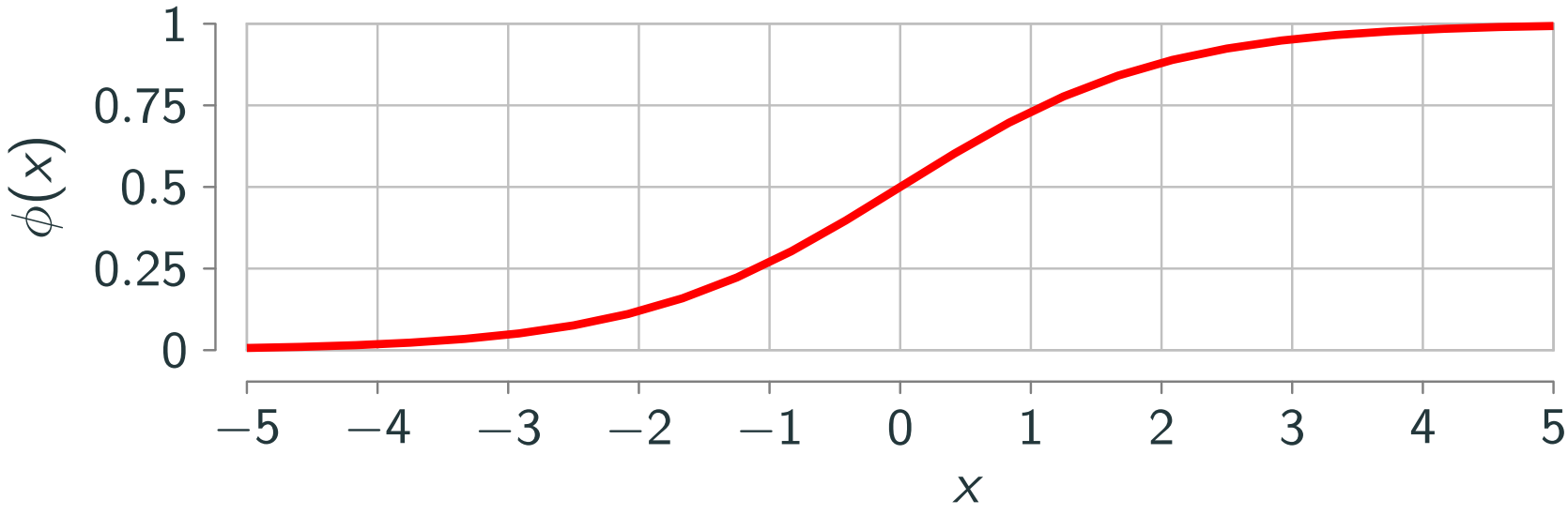
Fonctions d'activation — Critères de choix

- Propriétés mathématiques (conservation du gradient)
- Propriétés d'apprentissage (éviter la création de poids morts)
- Rapidité de calcul
- Intervalle de sortie pour la dernière couche

Fonctions d'activation — Les plus classiques

- Sigmoïde
- Tanh
- Softmax
- ReLU
- ...

Fonctions d'activation — Sigmoïde



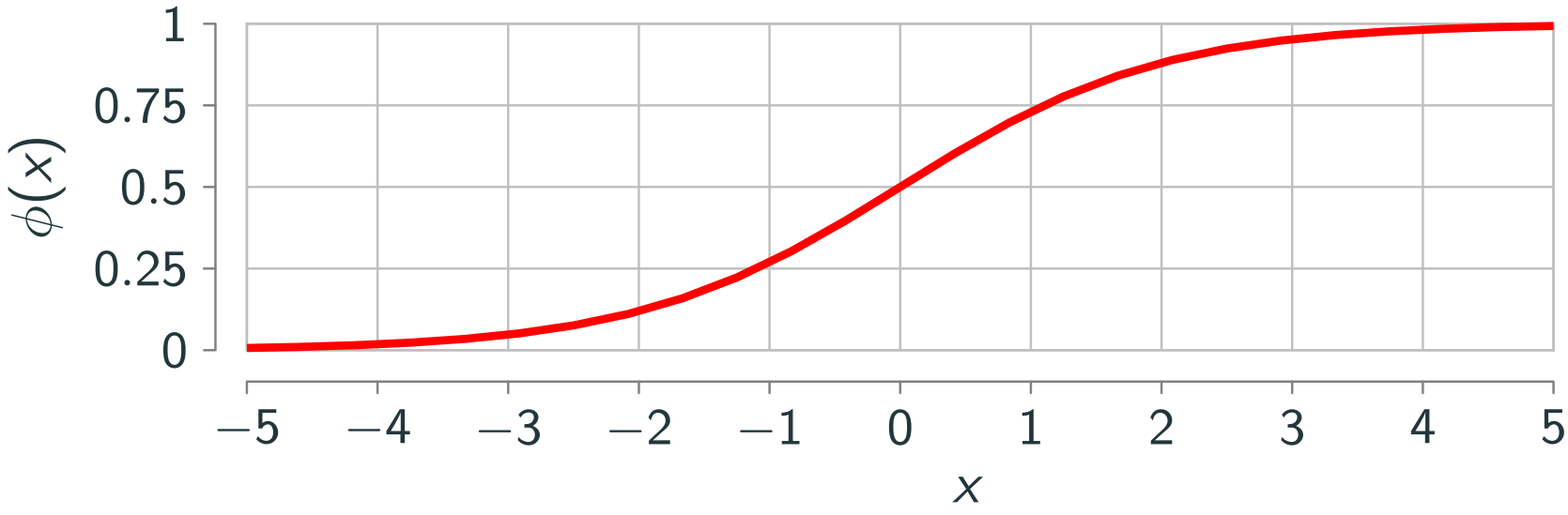
Définition

$$\phi(x) = \frac{1}{1 + e^{-x}}$$

Dérivée

$$\phi'(x) = \phi(x)(1 - \phi(x))$$

Fonctions d'activation — Sigmoïde



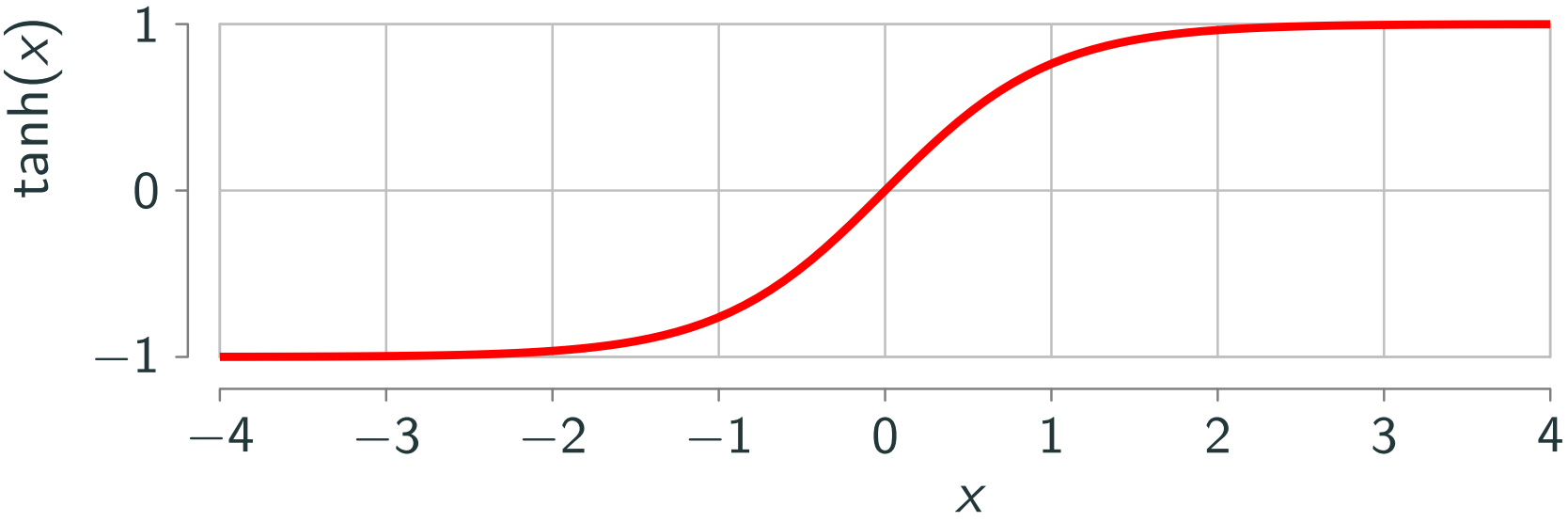
Définition

$$\phi(x) = \frac{1}{1 + e^{-x}}$$

Dérivée

$$\phi'(x) = \phi(x)(1 - \phi(x))$$

Fonctions d'activation — Tangente hyperbolique



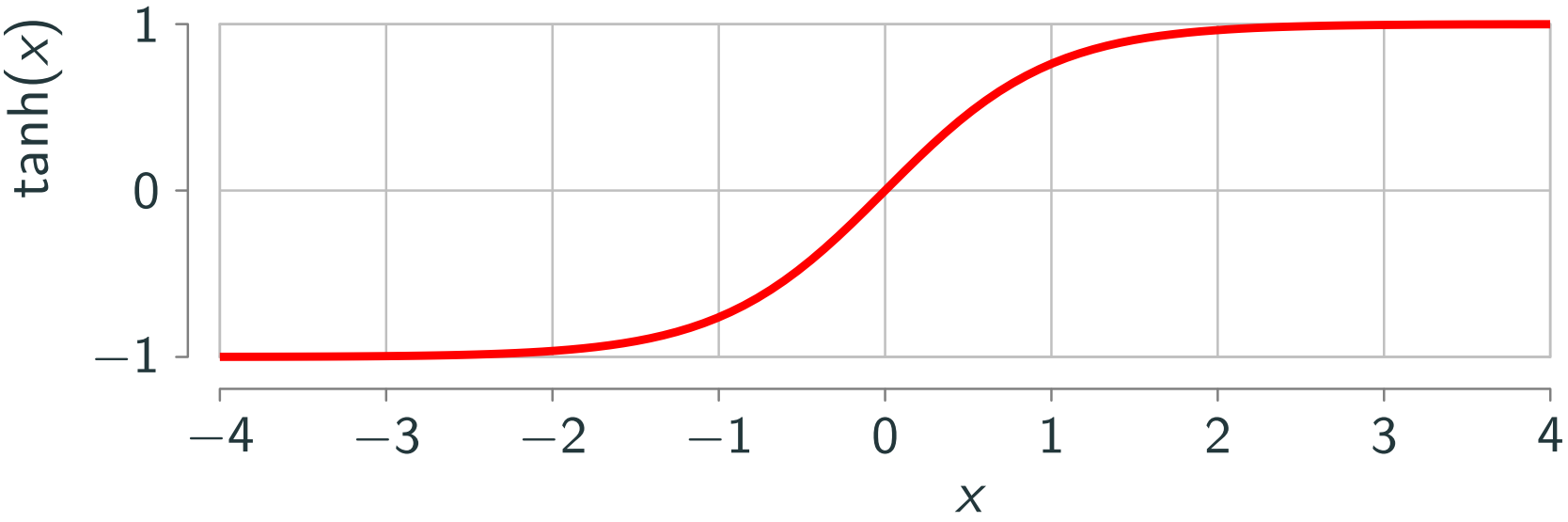
Définition

$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

Dérivée

$$\tanh'(x) = 1 - \tanh^2(x)$$

Fonctions d'activation — Tangente hyperbolique



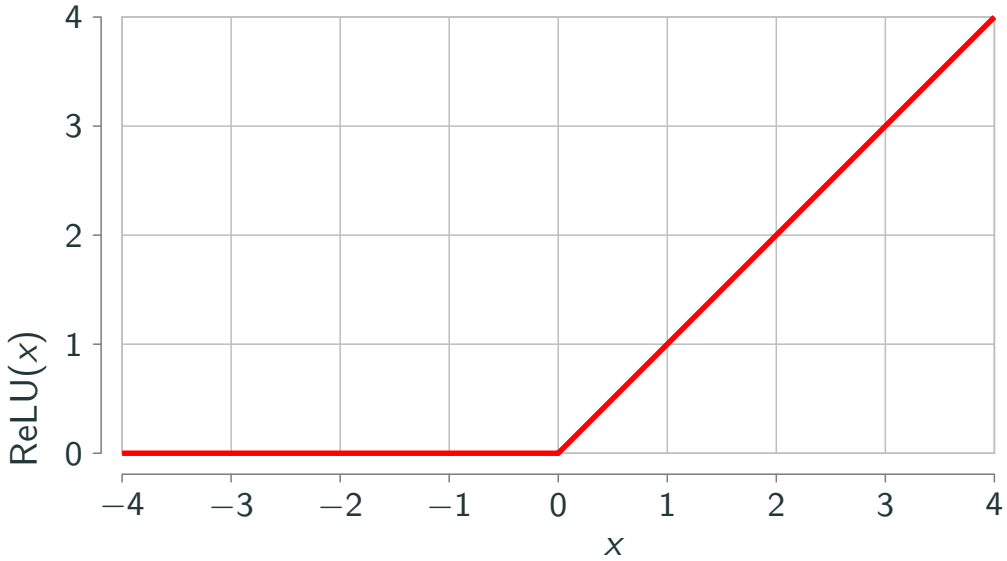
Définition

$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

Dérivée

$$\tanh'(x) = 1 - \tanh^2(x)$$

Fonctions d'activation — ReLU



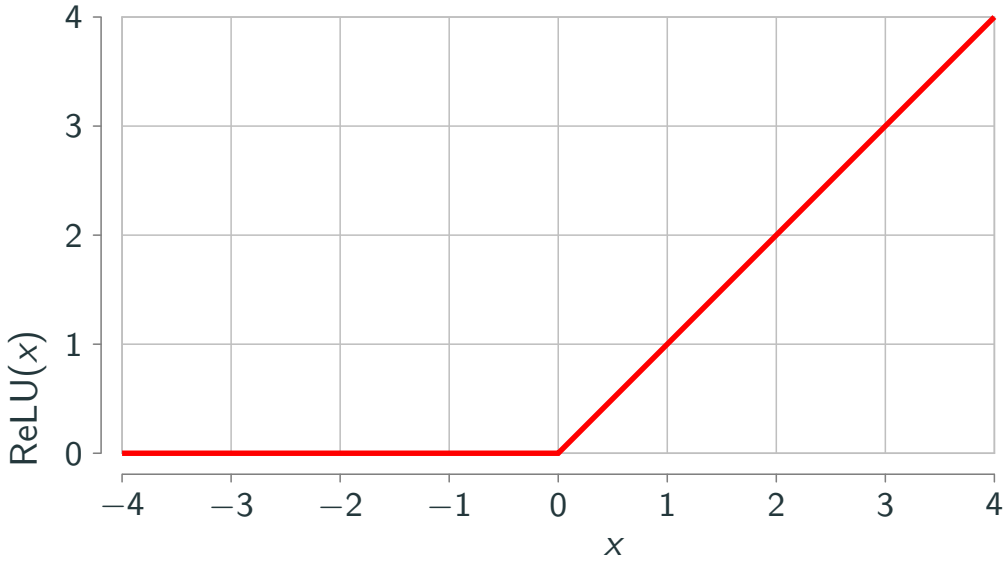
Définition

$$\mathrm{ReLU}(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

Dérivée

$$\mathrm{ReLU}'(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$$

Fonctions d'activation — ReLU



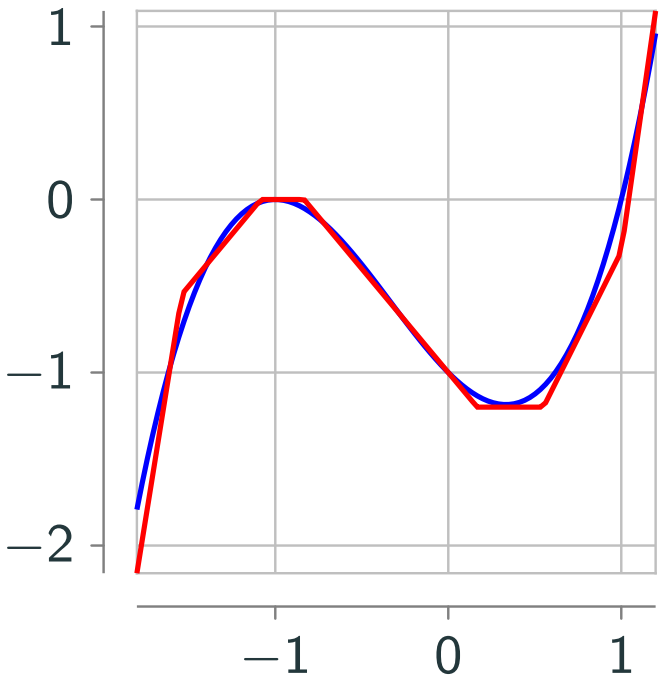
Définition

$$\mathrm{ReLU}(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

Dérivée

$$\mathrm{ReLU}'(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$$

Fonctions d'activation — Approximation d'une fonction



$-n_1 - n_2 - n_3 + n_4 + n_5 + n_6$

$x^3 + x^2 - x - 1$

$n_1 = \text{ReLU}(-5x - 7.7)$

$n_2 = \text{ReLU}(-1.2x - 1.3)$

$n_3 = \text{ReLU}(1.2x - 1)$

$n_4 = \text{ReLU}(1.2x - 0.2)$

$n_5 = \text{ReLU}(2x - 1.1)$

$n_6 = \text{ReLU}(5x - 5)$

Fonctions d'activation — Softmax

Définition

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{k=1}^{n} \exp(x_k)}$$

Propriété

$$\sum_{i=1}^{n} \text{softmax}(x_i) = 1$$

Gradient

$$\frac{\partial \, \text{softmax}(x_i)}{\partial x_j} = \begin{cases} \text{softmax}(x_i)(1 - \text{softmax}(x_j)) & i = j \\ -\text{softmax}(x_i)\,\text{softmax}(x_j) & i \neq j \end{cases}$$

Fonctions d'activation — Softmax

Définition

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{k=1}^{n} \exp(x_k)}$$

Propriété

$$\sum_{i=1}^{n} \text{softmax}(x_i) = 1$$

Gradient

$$\frac{\partial \, \text{softmax}(x_i)}{\partial x_j} = \begin{cases} \text{softmax}(x_i)(1 - \text{softmax}(x_j)) & i = j \\ -\text{softmax}(x_i)\,\text{softmax}(x_j) & i \neq j \end{cases}$$

Fonctions d'activation — Softmax

Définition

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{k=1}^{n} \exp(x_k)}$$

Propriété

$$\sum_{i=1}^{n} \text{softmax}(x_i) = 1$$

Gradient

$$\frac{\partial \, \text{softmax}(x_i)}{\partial x_j} = \begin{cases} \text{softmax}(x_i)(1 - \text{softmax}(x_j)) & i = j \\ -\,\text{softmax}(x_i)\,\text{softmax}(x_j) & i \neq j \end{cases}$$

Apprentissage supervisé

Régression Logistique

Introduction

Équivalent de la régression linéaire mais quand la sortie est binaire.
L'approche est bayésienne, basée sur une étude statistique.

Le cas à 2 classes

On s'intéresse à faire une régression de l'*"évidence"* de la variable aléatoire cible :

$$\ln \frac{P(1|X)}{1-P(1|X)} = b_0 + b_1 x_1 + ... + b_d x_d$$

où

- $X = [x_1, ..., x_d]$ un exemple de la base
- $B = [b_1, ..., b_d]$ l'ensemble des paramètres de notre modèle
- $P(1|X)$ est la probabilité que X soit de classe 1

Les coéfficients B sont alors estimés par descente de gradient

Multinomial logistic Regression

Problème à K classes :

- (K-1) prédicteurs binaire en utilisant une classe 'pivot'
- Maximum de vraisemblance pour la prédiction finale

Avez-vous des questions ?

Apprentissage supervisé

Travaux Pratiques : Régression Linéaire

Instructions — Régression linéaire

Regression Linéaire - TP

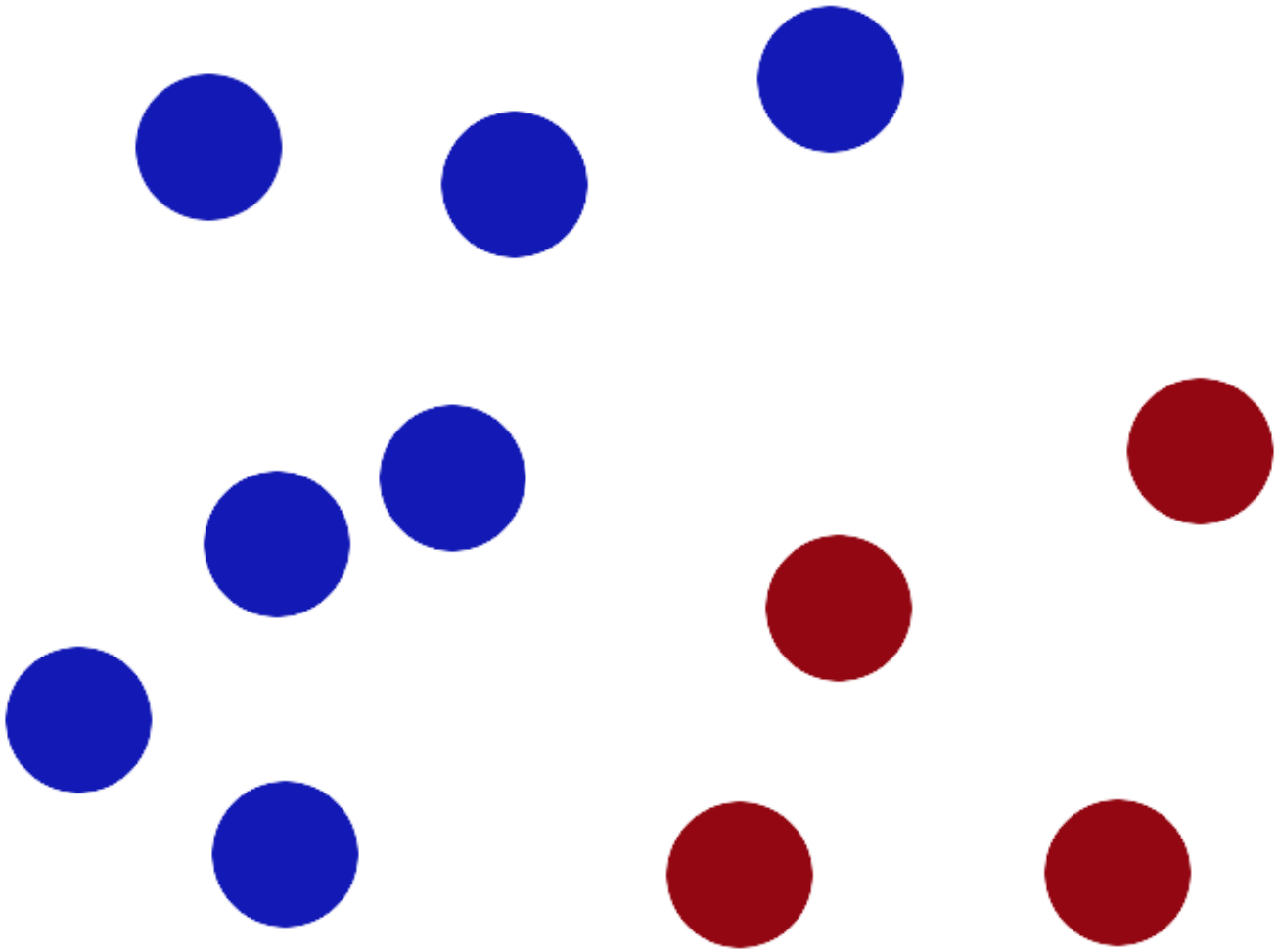
Instructions — Régression polynomiale

Régression Polynomiale - Tutoriel

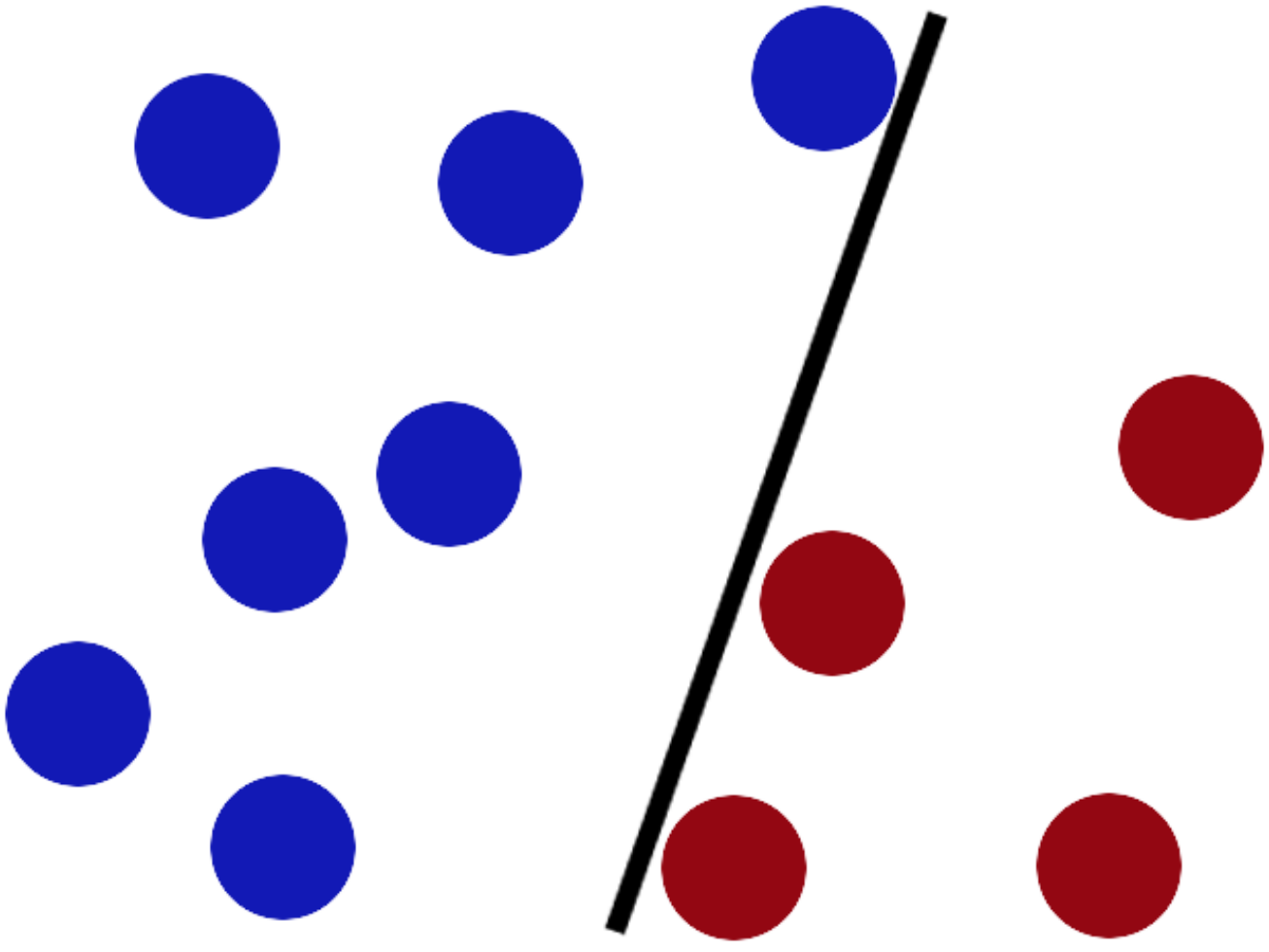
Apprentissage supervisé

Support Vector Machine

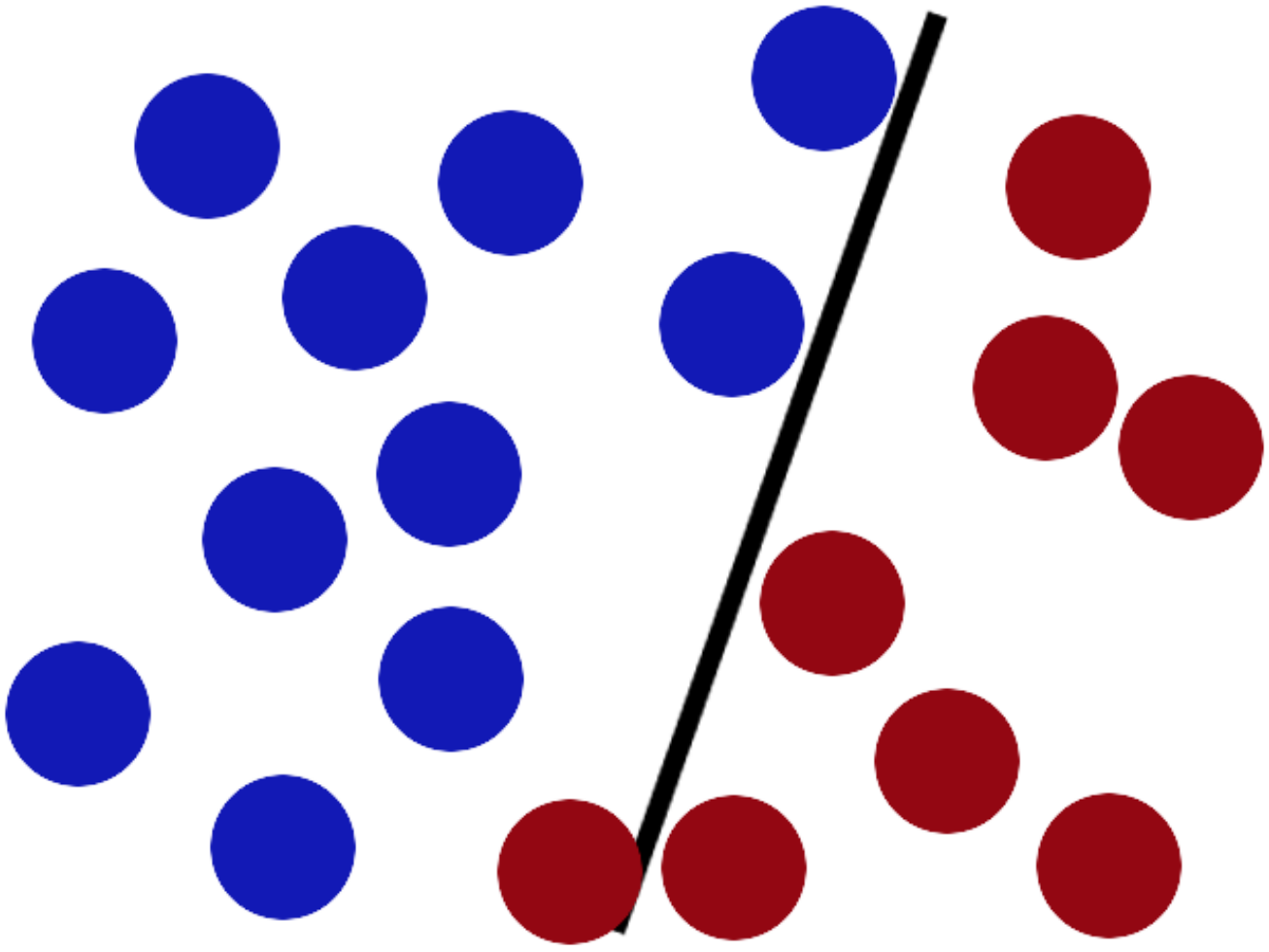
Support Vector Machine



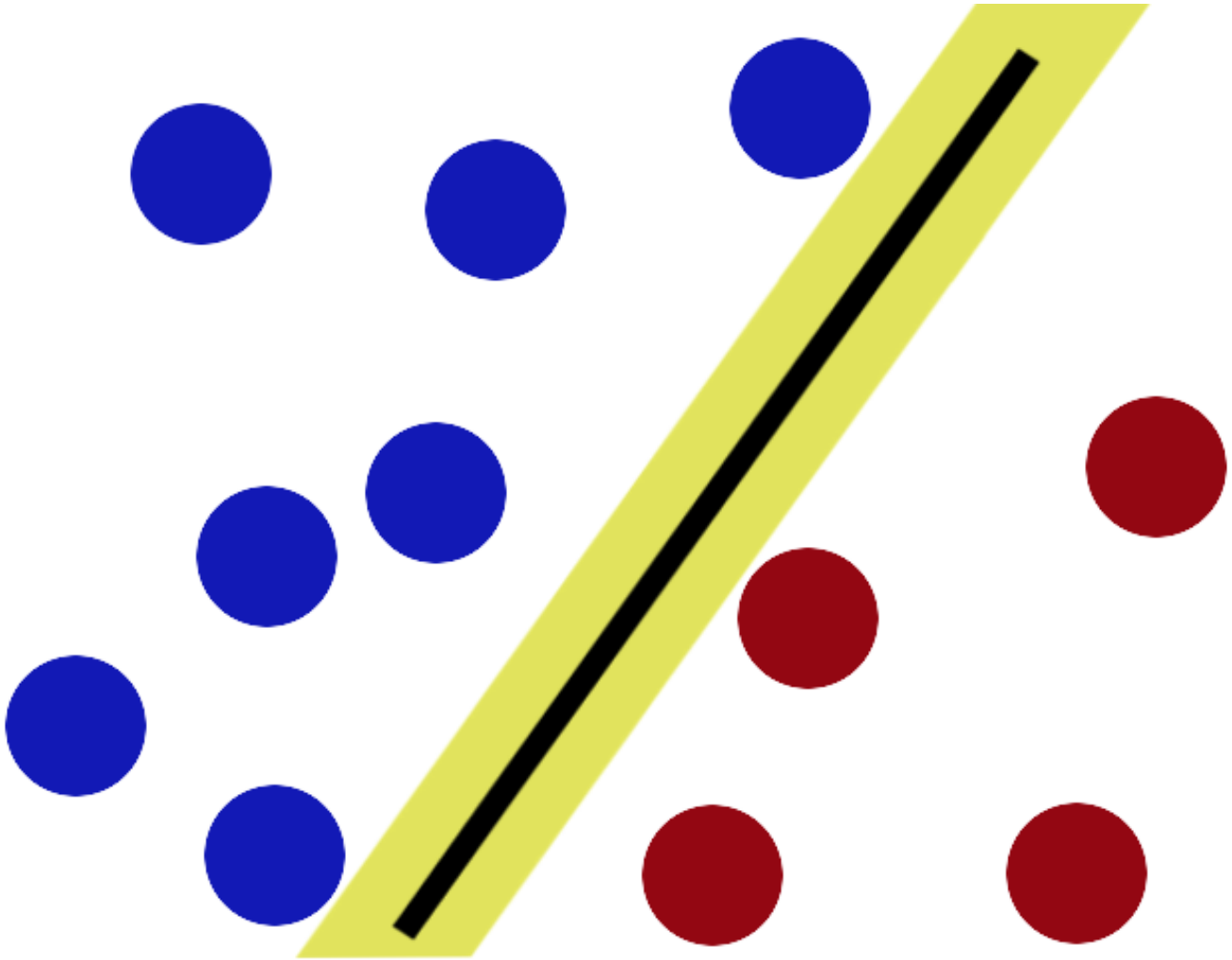
Support Vector Machine



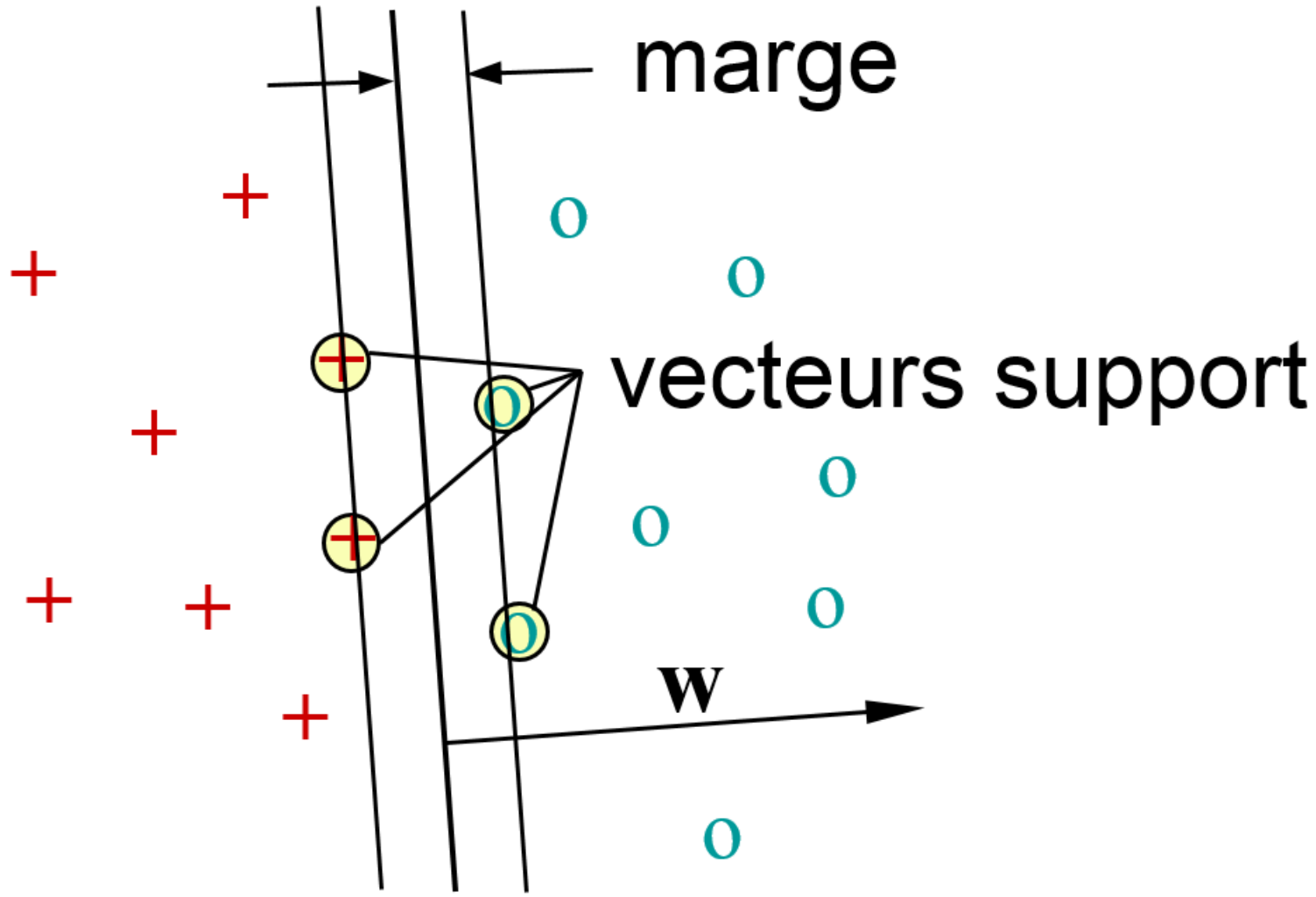
Support Vector Machine



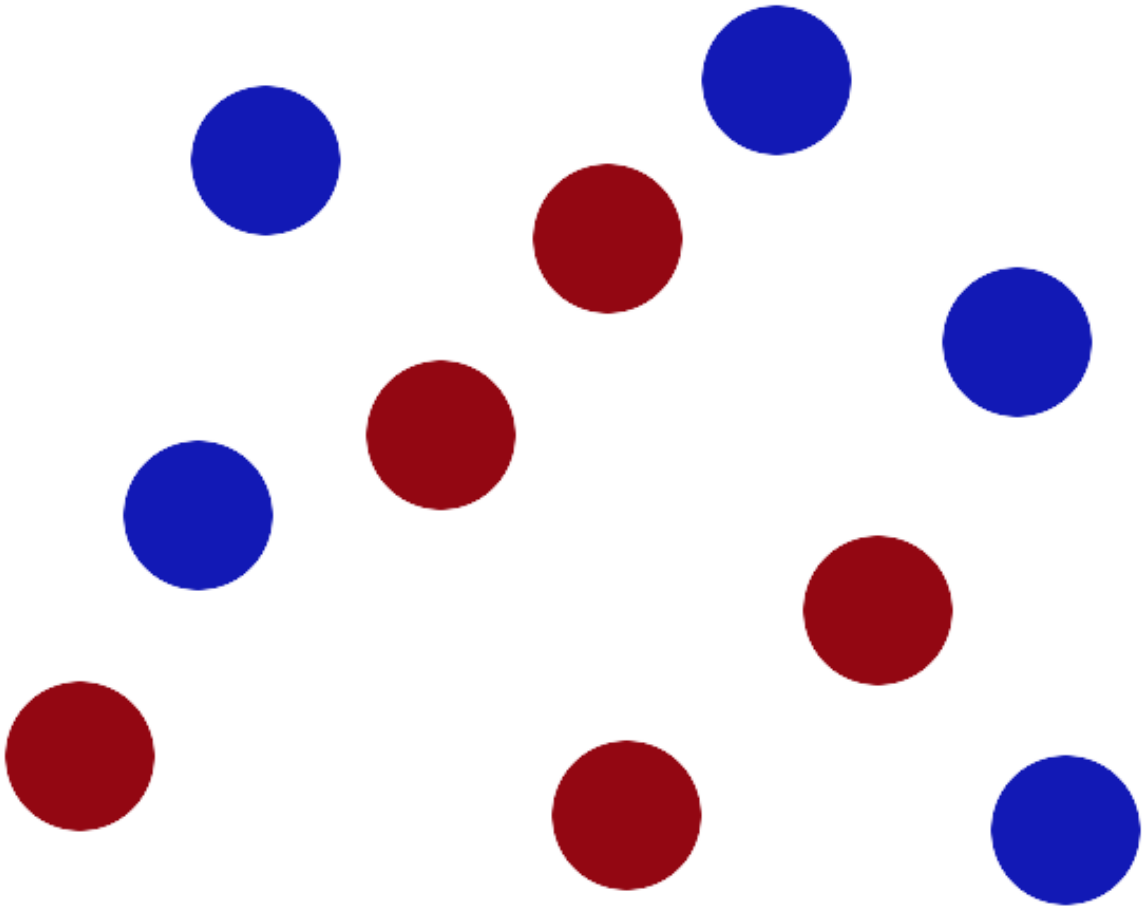
Support Vector Machine



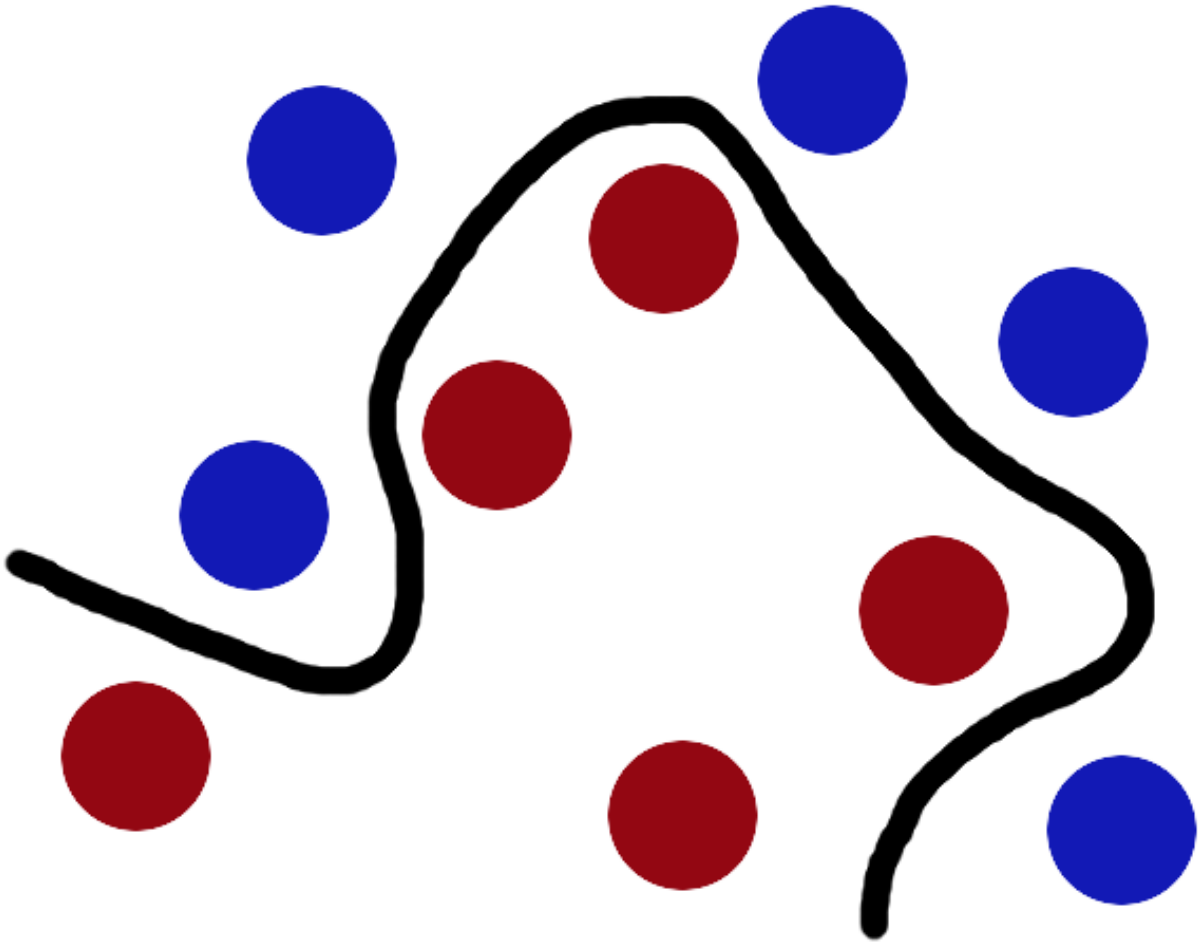
Support Vector Machine



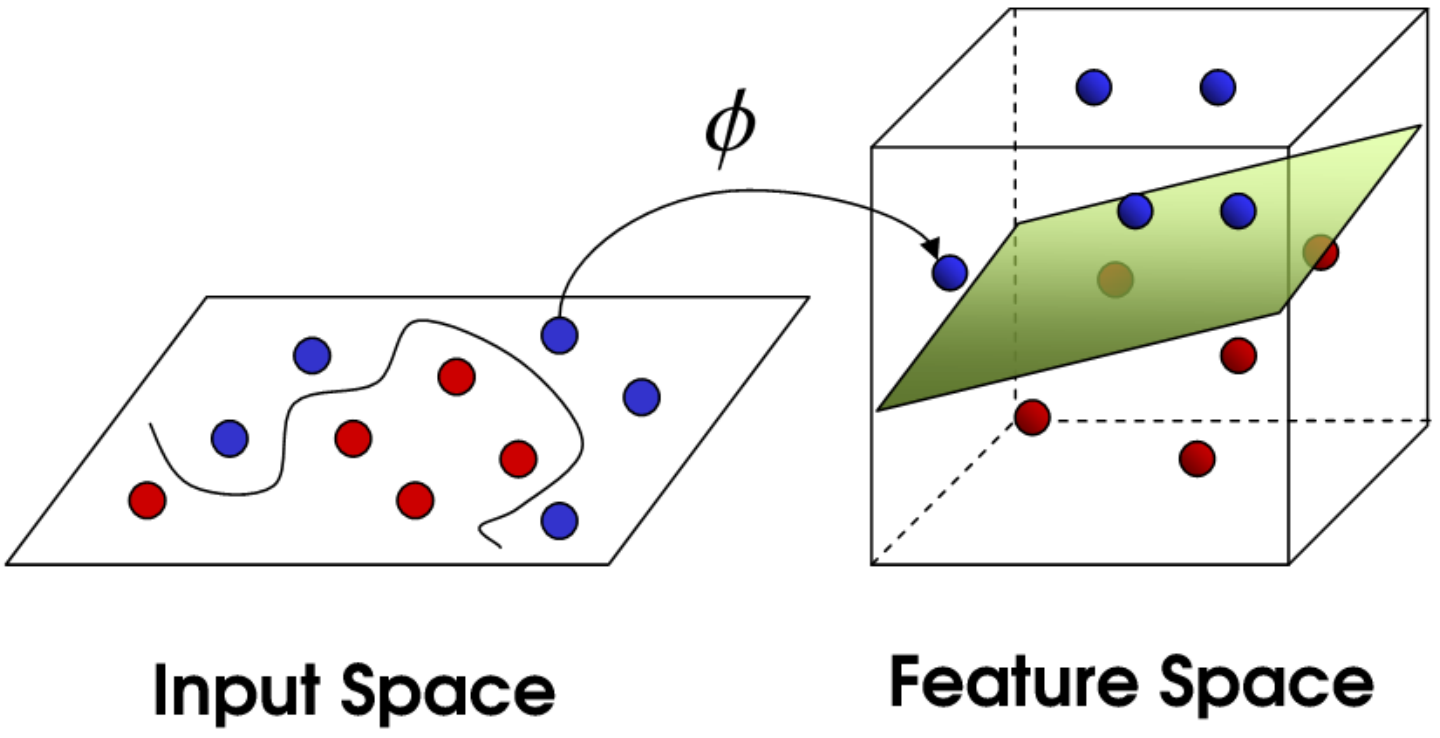
Support Vector Machine



Support Vector Machine



Support Vector Machine



Support Vector Machine

Généralisation à un problème de régression logistique à $K > 2$ classes :

- One Vs All : K modèles. Agréagation par meilleur score.
- One Vs One : $\frac{K(K-1)}{2}$ modèles. Vote majoritaire.

Apprentissage supervisé

Démo Sklearn

SVM

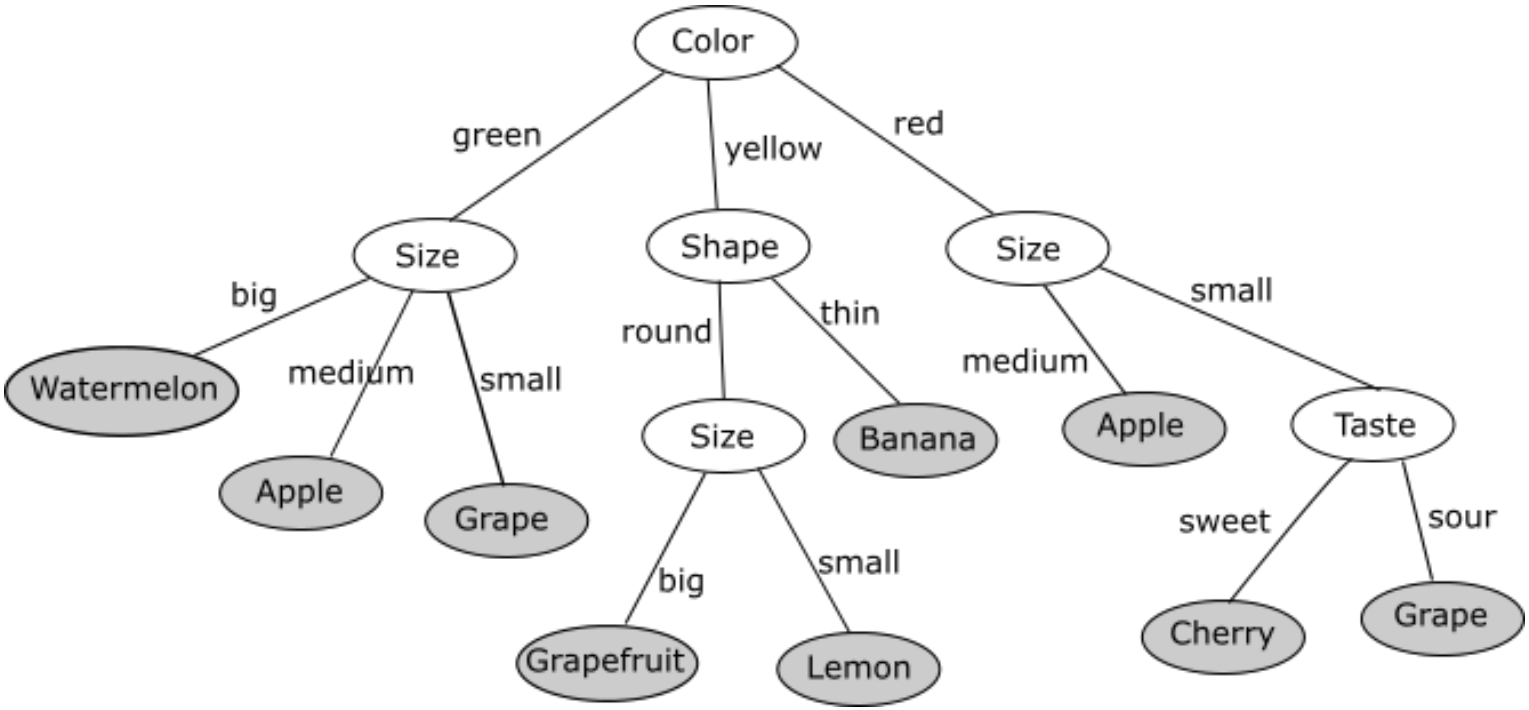
SVM Tutoriel

Apprentissage supervisé

Arbres de décision

Introduction

Modèle de classification ou regréssion qui classe un input dans une de ses feuilles pour rendre sa prédiction :



Avantages

Les arbres de décision

- gèrent les inputs numériques comme catégoriels
- ne nécessitent pas que la variable d'output soit normalement distribuée (regression linéaire)
- sont interprétables
- sont très rapides durant l'inférence
- ne nécessitent pas de normalisation des données
- leur apprentissage est hautement parallèlisable

→ Couteau-suisse du machine learning tabulaire.

Avantages

Les arbres de décision

- gèrent les inputs numériques comme catégoriels
- ne nécessitent pas que la variable d'output soit normalement distribuée (regression linéaire)
- sont interprétables
- sont très rapides durant l'inférence
- ne nécessitent pas de normalisation des données
- leur apprentissage est hautement parallèlisable

→ Couteau-suisse du machine learning tabulaire.

Avantages

Les arbres de décision

- gèrent les inputs numériques comme catégoriels
- ne nécessitent pas que la variable d'output soit normalement distribuée (regression linéaire)
- sont interprétables
- sont très rapides durant l'inférence
- ne nécessitent pas de normalisation des données
- leur apprentissage est hautement parallèlisable

→ Couteau-suisse du machine learning tabulaire.

Avantages

Les arbres de décision

- gèrent les inputs numériques comme catégoriels
- ne nécessitent pas que la variable d'output soit normalement distribuée (regression linéaire)
- sont interprétables
- sont très rapides durant l'inférence
- ne nécessitent pas de normalisation des données
- leur apprentissage est hautement parallèlisable

→ Couteau-suisse du machine learning tabulaire.

Avantages

Les arbres de décision

- gèrent les inputs numériques comme catégoriels
- ne nécessitent pas que la variable d'output soit normalement distribuée (regression linéaire)
- sont interprétables
- sont très rapides durant l'inférence
- ne nécessitent pas de normalisation des données
- leur apprentissage est hautement parallèlisable

→ Couteau-suisse du machine learning tabulaire.

Avantages

Les arbres de décision

- gèrent les inputs numériques comme catégoriels
- ne nécessitent pas que la variable d'output soit normalement distribuée (regression linéaire)
- sont interprétables
- sont très rapides durant l'inférence
- ne nécessitent pas de normalisation des données
- leur apprentissage est hautement parallèlisable

→ Couteau-suisse du machine learning tabulaire.

Avantages

Les arbres de décision

- gèrent les inputs numériques comme catégoriels
- ne nécessitent pas que la variable d'output soit normalement distribuée (regression linéaire)
- sont interprétables
- sont très rapides durant l'inférence
- ne nécessitent pas de normalisation des données
- leur apprentissage est hautement parallèlisable

$\rightarrow$ Couteau-suisse du machine learning tabulaire.

Désavantages

- peuvent overfit les données, mais l'ensembling résoud ce problème
- sont sensibles aux déséquilibres de classe

→ Si les classes ne sont pas équilibrées, peut-être les resampler.

Désavantages

- peuvent overfit les données, mais l'ensembling résoud ce problème
- sont sensibles aux déséquilibres de classe

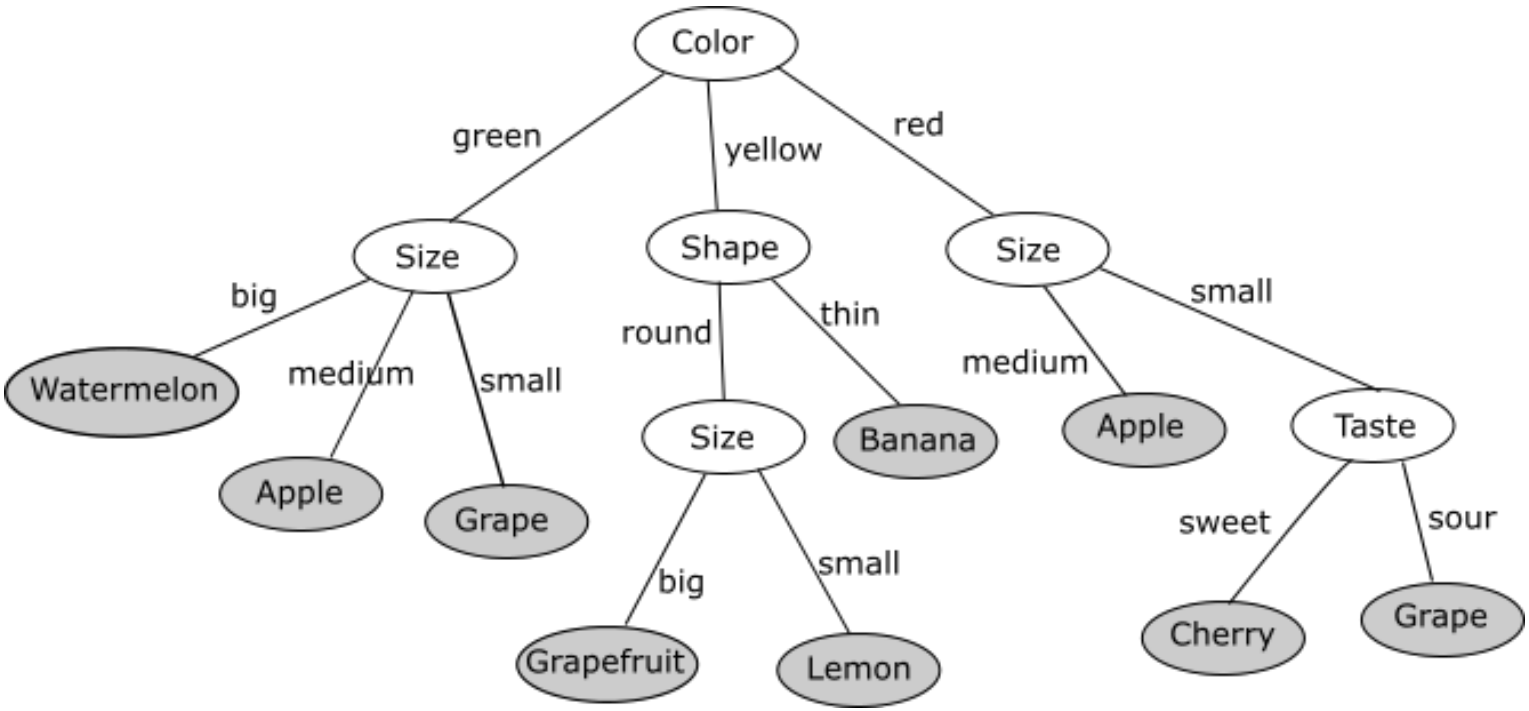
→ Si les classes ne sont pas équilibrées, peut-être les resampler.

Désavantages

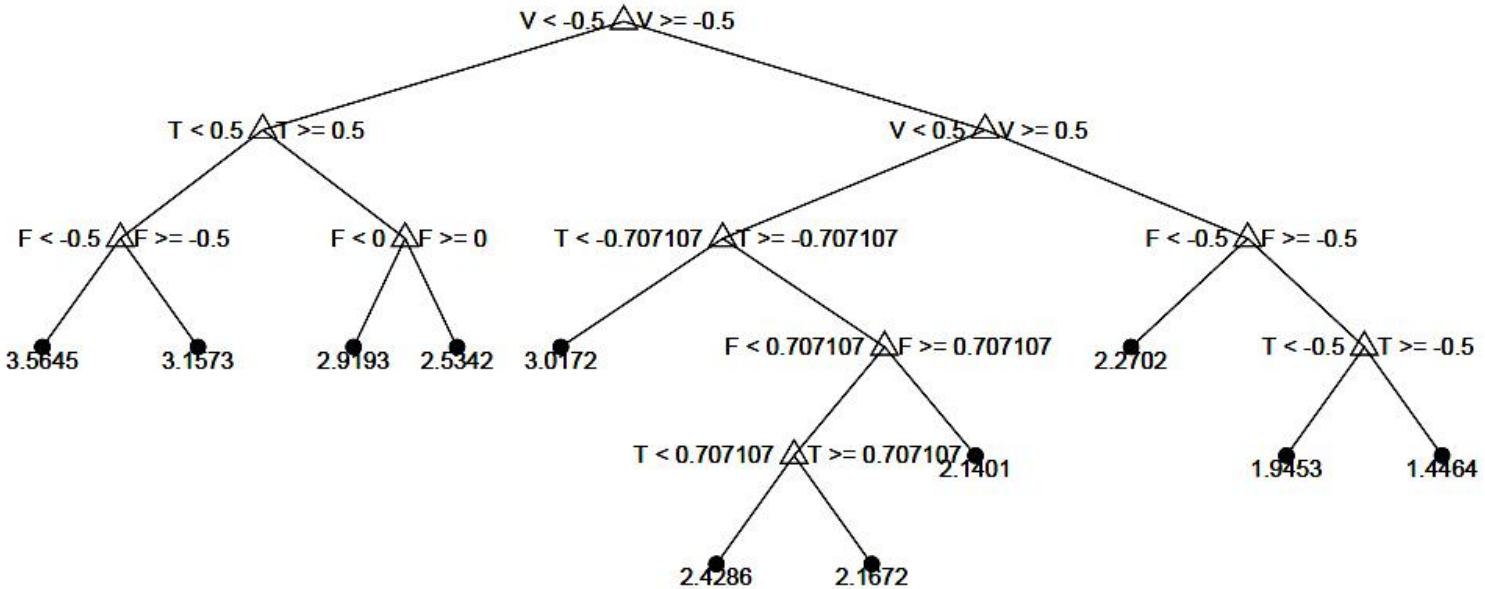
- peuvent overfit les données, mais l'ensembling résoud ce problème
- sont sensibles aux déséquilibres de classe

$\rightarrow$ Si les classes ne sont pas équilibrées, peut-être les resampler.

Arbres de classification



Arbres de régression



Apprendre un arbre de décision

Approche « top-down », procédure récursive :

- créer un nœud de départ qui contient toutes les instances du training set
- tant qu'il reste des nœuds non-traités :
  - choisir un nœud non traité
  - si le nœud remplit des conditions de feuille finale, ne rien faire
  - sinon, créer deux branches à partir du nœud non traité pour répartir les instances dans deux nouveaux nœuds

Conditions de feuilles finales : contient n_{min} éléments, est déjà à profondeur p_{max}, splitterait sans décroître assez l'entropie...

Apprendre un arbre de décision

Approche « top-down », procédure récursive :

- créer un nœud de départ qui contient toutes les instances du training set
- tant qu'il reste des nœuds non-traités :
  - choisir un nœud non traité
  - si le nœud remplit des conditions de feuille finale, ne rien faire
  - sinon, créer deux branches à partir du nœud non traité pour répartir les instances dans deux nouveaux nœuds

Conditions de feuilles finales : contient n_{min} éléments, est déjà à profondeur p_{max}, splitterait sans décroître assez l'entropie...

Apprendre un arbre de décision

Approche « top-down », procédure récursive :

- créer un nœud de départ qui contient toutes les instances du training set
- tant qu'il reste des nœuds non-traités :
  - choisir un nœud non traité
  - si le nœud remplit des conditions de feuille finale, ne rien faire
  - sinon, créer deux branches à partir du nœud non traité pour répartir les instances dans deux nouveaux nœuds

Conditions de feuilles finales : contient n_{min} éléments, est déjà à profondeur p_{max}, splitterait sans décroître assez l'entropie...

Apprendre un arbre de décision

Approche « top-down », procédure récursive :

- créer un nœud de départ qui contient toutes les instances du training set
- tant qu'il reste des nœuds non-traités :
  - choisir un nœud non traité
  - si le nœud remplit des conditions de feuille finale, ne rien faire
  - sinon, créer deux branches à partir du nœud non traité pour répartir les instances dans deux nouveaux nœuds

Conditions de feuilles finales : contient n_{min} éléments, est déjà à profondeur p_{max}, splitterait sans décroître assez l'entropie...

Apprendre un arbre de décision

Approche « top-down », procédure récursive :

- créer un nœud de départ qui contient toutes les instances du training set
- tant qu'il reste des nœuds non-traités :
  - choisir un nœud non traité
  - si le nœud remplit des conditions de feuille finale, ne rien faire
  - sinon, créer deux branches à partir du nœud non traité pour répartir les instances dans deux nouveaux nœuds

Conditions de feuilles finales : contient n_{min} éléments, est déjà à profondeur p_{max}, splitterait sans décroître assez l'entropie...

Apprendre un arbre de décision

Approche « top-down », procédure récursive :

- créer un nœud de départ qui contient toutes les instances du training set
- tant qu'il reste des nœuds non-traités :
  - choisir un nœud non traité
  - si le nœud remplit des conditions de feuille finale, ne rien faire
  - sinon, créer deux branches à partir du nœud non traité pour répartir les instances dans deux nouveaux nœuds

Conditions de feuilles finales : contient n_{min} éléments, est déjà à profondeur p_{max}, splitterait sans décroître assez l'entropie...

Décision rendue

En fonction de la tâche, une fois arrivé dans la feuille de fin :

Classification classe majoritaire

Régression moyenne des valeurs cibles

Splits possibles

Splits possibles d'une feature donnée :

Catégorielle chaque catégorie vs le reste

Ordinale/Continue milieu de chaque valeur ou quantiles

Évaluation de la qualité d'un split

En fonction de la tâche :

Régression coût si on rendait la moyenne des instances comme résultat

$$Loss = \sum |\hat{y} - y| \approx variance$$

Classification Entropie de Shannon :

$$Loss = -\sum_{x \in X} P_x * \log_2(P_x)$$

$= 0 \Rightarrow$ il n'y a pas d'incertitude
maximale quand on a une distribution uniforme

Exemple — démarrage

ID, jardinage, jeux vidéos, chapeaux, âge

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 13 \\ 2 & 0 & 1 & 0 & 14 \\ 3 & 0 & 1 & 0 & 15 \\ 4 & 1 & 1 & 1 & 25 \\ 5 & 0 & 1 & 1 & 35 \\ 6 & 1 & 0 & 0 & 49 \\ 7 & 1 & 1 & 1 & 68 \\ 8 & 1 & 0 & 0 & 71 \\ 9 & 1 & 0 & 1 & 73 \end{bmatrix}$$

Première étape : création du nœud de départ

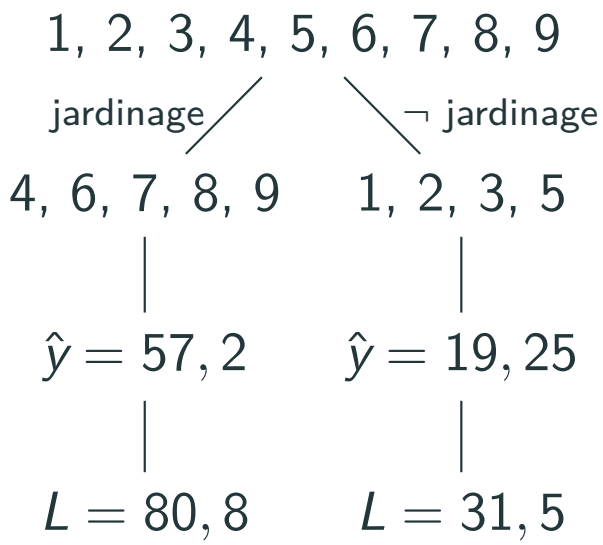
1, 2, 3, 4, 5, 6, 7, 8, 9

Exemple — split

ID, jardinage, jeux vidéos, chapeaux, âge

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 13 \\ 2 & 0 & 1 & 0 & 14 \\ 3 & 0 & 1 & 0 & 15 \\ 4 & 1 & 1 & 1 & 25 \\ 5 & 0 & 1 & 1 & 35 \\ 6 & 1 & 0 & 0 & 49 \\ 7 & 1 & 1 & 1 & 68 \\ 8 & 1 & 0 & 0 & 71 \\ 9 & 1 & 0 & 1 & 73 \end{bmatrix}$$

Split du premier nœud. Il faut tester 3 splits. Split sur jardinage :



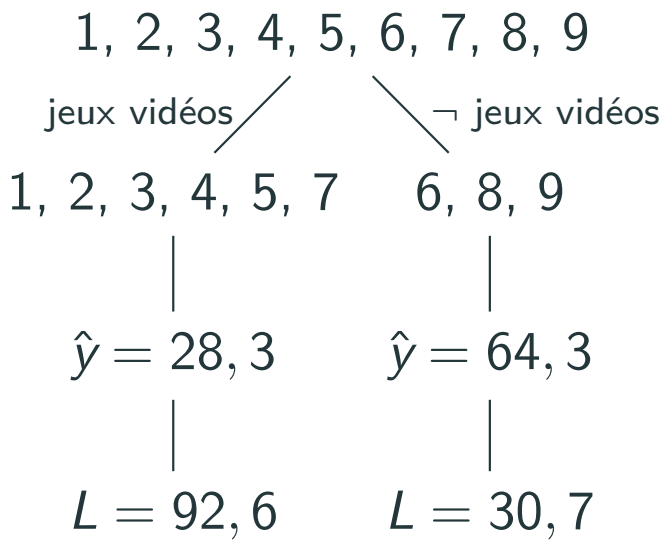
Loss totale : $122,3$

Exemple — split

ID, jardinage, jeux vidéos, chapeaux, âge

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 13 \\ 2 & 0 & 1 & 0 & 14 \\ 3 & 0 & 1 & 0 & 15 \\ 4 & 1 & 1 & 1 & 25 \\ 5 & 0 & 1 & 1 & 35 \\ 6 & 1 & 0 & 0 & 49 \\ 7 & 1 & 1 & 1 & 68 \\ 8 & 1 & 0 & 0 & 71 \\ 9 & 1 & 0 & 1 & 73 \end{bmatrix}$$

Split du premier nœud. Il faut tester 3 splits. Split sur jeux vidéos :



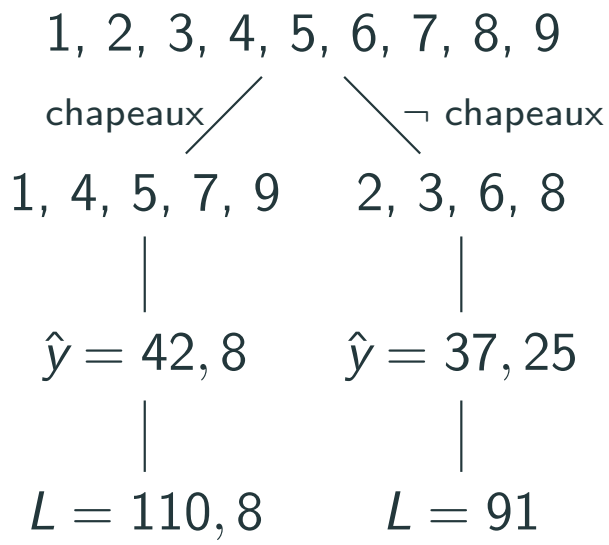
Loss totale : $123, 3$

Exemple — split

ID, jardinage, jeux vidéos, chapeaux, âge

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 13 \\ 2 & 0 & 1 & 0 & 14 \\ 3 & 0 & 1 & 0 & 15 \\ 4 & 1 & 1 & 1 & 25 \\ 5 & 0 & 1 & 1 & 35 \\ 6 & 1 & 0 & 0 & 49 \\ 7 & 1 & 1 & 1 & 68 \\ 8 & 1 & 0 & 0 & 71 \\ 9 & 1 & 0 & 1 & 73 \end{bmatrix}$$

Split du premier nœud. Il faut tester 3 splits. Split sur chapeaux :



Loss totale : $201,8$

Exemple — split

ID, jardinage, jeux vidéos, chapeaux, âge

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 13 \\ 2 & 0 & 1 & 0 & 14 \\ 3 & 0 & 1 & 0 & 15 \\ 4 & 1 & 1 & 1 & 25 \\ 5 & 0 & 1 & 1 & 35 \\ 6 & 1 & 0 & 0 & 49 \\ 7 & 1 & 1 & 1 & 68 \\ 8 & 1 & 0 & 0 & 71 \\ 9 & 1 & 0 & 1 & 73 \end{bmatrix}$$

122,3 jardinage

123,3 jeux vidéos

201,8 chapeaux

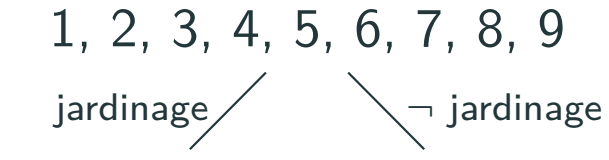
$\rightarrow$ On split donc sur jardinage

Exemple — split

ID, jardinage, jeux vidéos, chapeaux, âge

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 13 \\ 2 & 0 & 1 & 0 & 14 \\ 3 & 0 & 1 & 0 & 15 \\ 4 & 1 & 1 & 1 & 25 \\ 5 & 0 & 1 & 1 & 35 \\ 6 & 1 & 0 & 0 & 49 \\ 7 & 1 & 1 & 1 & 68 \\ 8 & 1 & 0 & 0 & 71 \\ 9 & 1 & 0 & 1 & 73 \end{bmatrix}$$

Résultat après le premier split :



À vous de jouer !

Limiter l'overfit

Fait par :

- la profondeur maximum
- le nombre minimum d'instances dans chaque feuille
- une baisse d'entropie maximale à chaque split
- le nombre minimum d'instances pour split
- le pruning

Apprentissage supervisé

Random Forest

Introduction

- les arbres de décision overfit facilement
- ils sont rapides à apprendre
- en combiner beaucoup est faisable et réduit la variance

$\rightarrow$ création d'une forêt (ensemble d'arbres) aléatoire

But

Produire des arbres décorrélés et moyenner leurs prédictions pour réduire la variance.

Outil 1 — bagging (row sampling)

Boostrap **agg**regat**ing** (Bagging) :

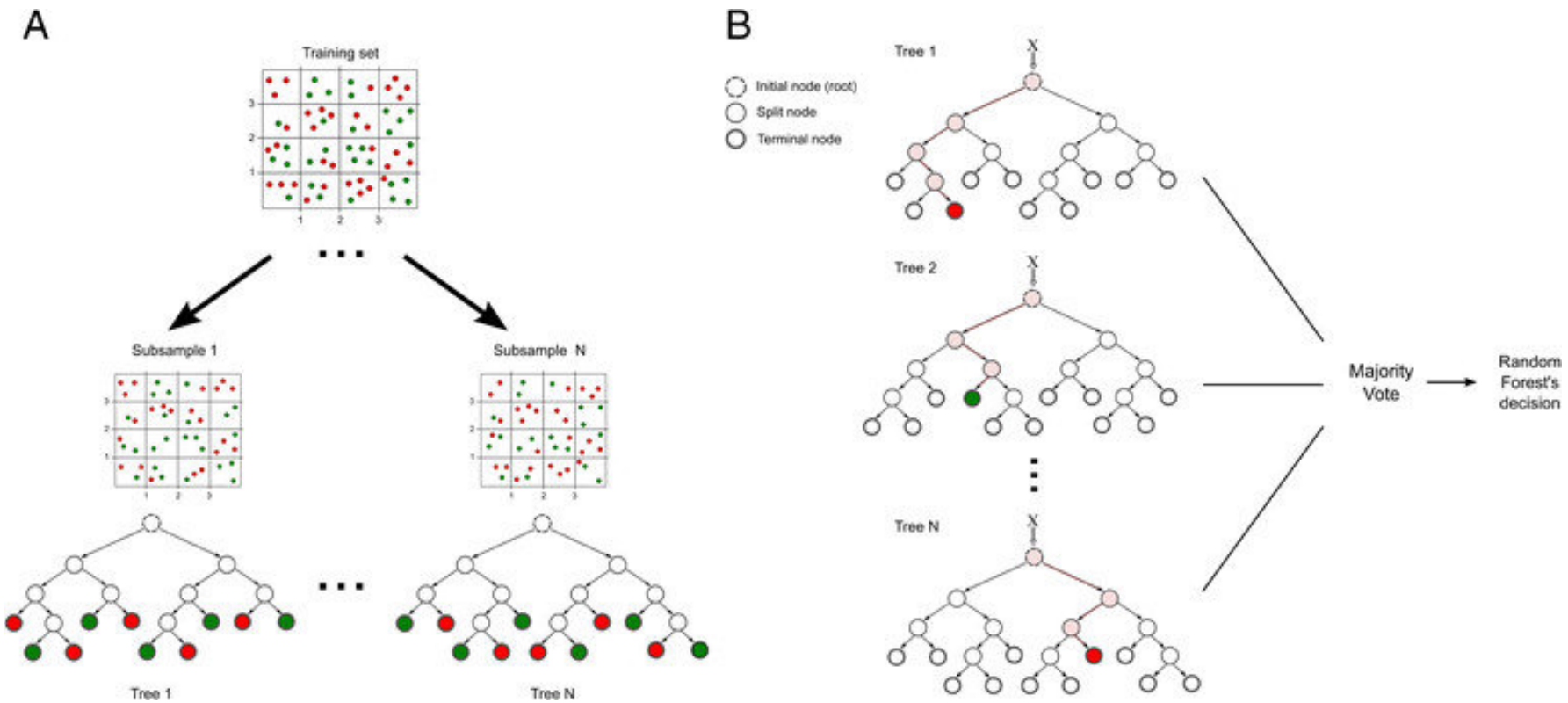
- tirer un échantillon du dataset avec replacement
- entraîner un arbre sur cet échantillon
- répéter B fois

Le bagging s'appelle aussi row sampling.

Outil 2 — random subspace method (column sampling)

- à chaque split, considérer seulement un sous-ensemble des features
- valeurs conseillées :
  - classification : $|\sqrt{m}|$ features par split
  - regréssion : $|\frac{m}{3}|$ features par split, 5 exemples par node minimum

Random Forest



Random Forest

- Pas de sur-apprentissage en augmentant le nombre d'arbres
- Une fois appris, le modèle est très rapide

Conclusion

- les arbres sont interprétables, rapides à entraîner, combinables.
- random forest combine des arbres faibles en un prédicteur versatile

Avez-vous des questions ?

Apprentissage supervisé

Démo Sklearn

Random Forest

Random Forest - Tutoriel

Apprentissage supervisé

Travaux Pratiques : Random Forest

Instructions

Régression avec Random Forest