# Machine learning

Redha Moulla

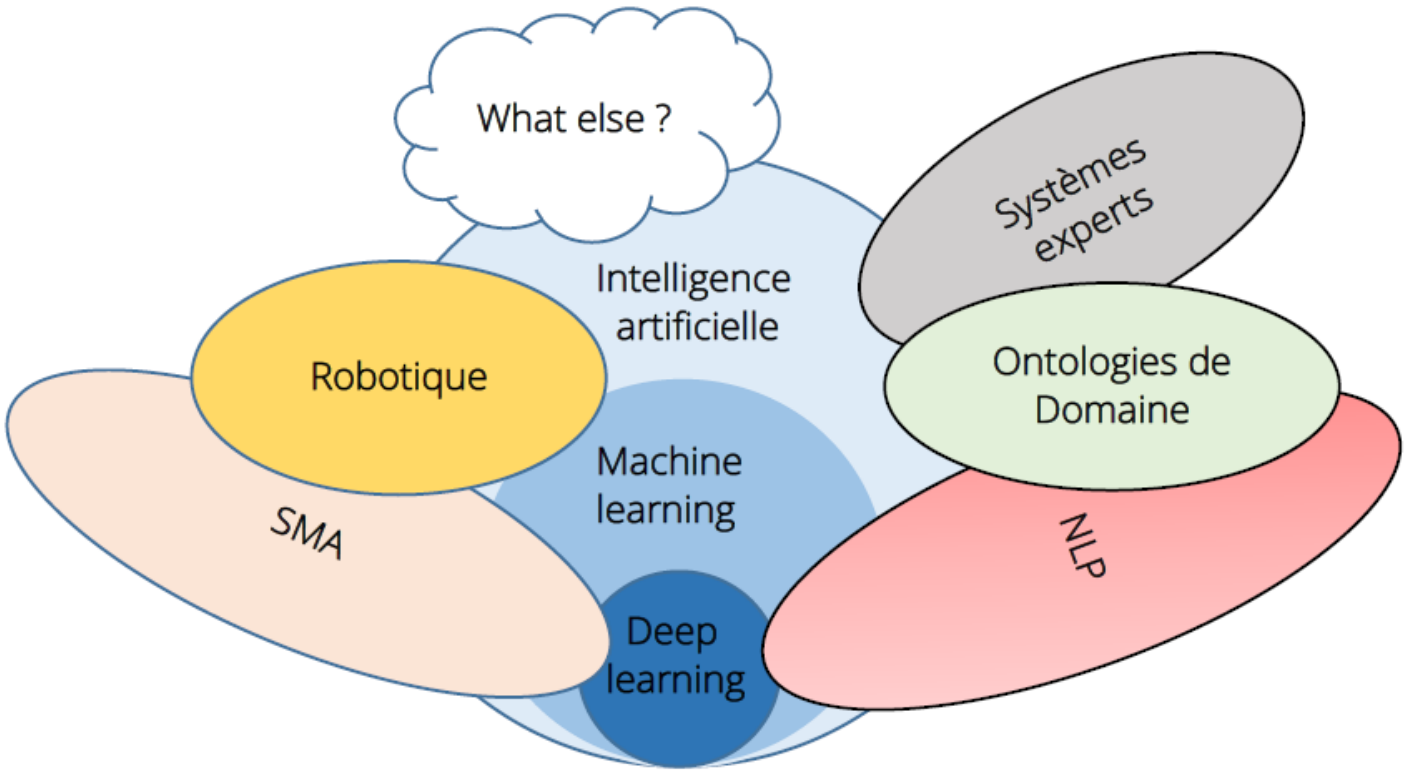
Ecole Supérieure Européenne de Management

Automne 2023

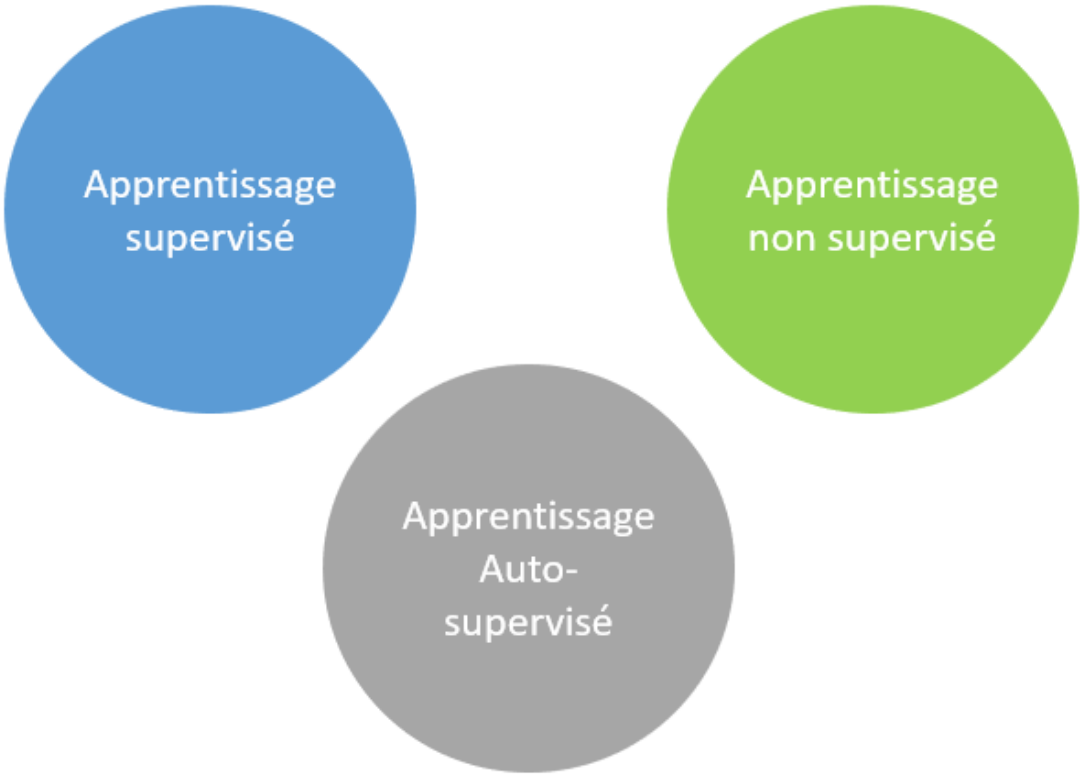
# Plan

- Principes du machine learning
- Apprintissage supervisé : modèles linéaires
- Apprentissage supervisé : modèles non linéaires
- Apprentissage non supervisé
- Introduction au deep learning

# Qu'est-ce que l'intelligence artificielle ?



# Typologies d'apprentissage automatique



# Apprentissage supervisé

# Formalisation du problème

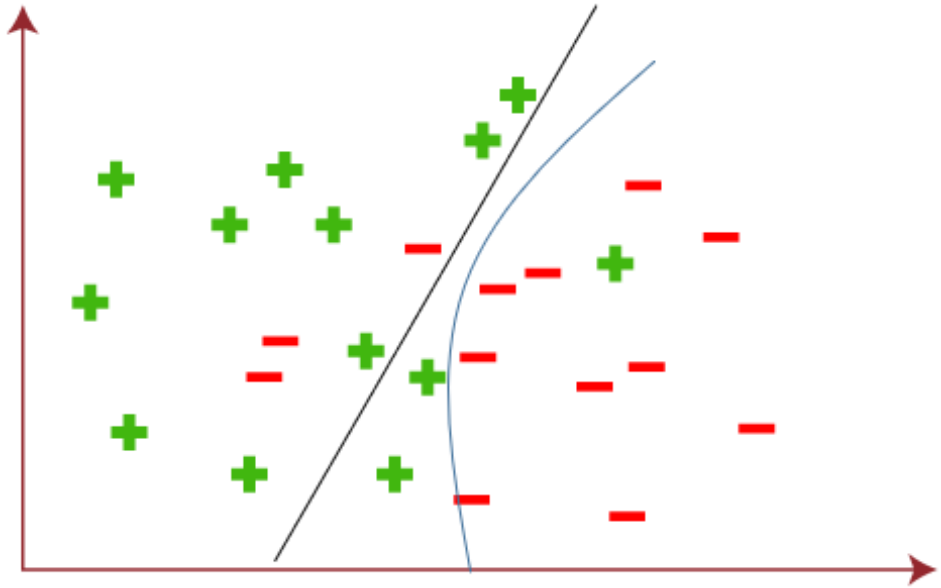
Soient les observations $x_1, x_2, ..., x_n \in \chi$, où $\chi$ est l'espace des observations.
Soient $y_1, y_2, ..., y_n \in \mathcal{Y}$ les labels correspondants.
L'apprentissage supervisé consiste à chercher une fonction $f$ telle que $f(x_i) \approx y_i$ pour $i \in 1, 2, ..., n$.

On parle de :

- Régression quand $\mathcal{Y} = R$
- Classification quand $\mathcal{Y} = 1, 2, ..., C$ où $C \geq 2$

# Espace des solutions

Comment choisir la fonction $f$ dans l'ensemble des solutions $\mathcal{F}$ ?



- A priori sur la classe du modèle (biais inductif)
- Minimisation du risque empirique

# Biais inductif

Le biais inductif est l'ensemble des a priori (hypothèses) qui déterminent la classe de modèles à laquelle appartient la fonction $f$. Ces hypothèses traduisent les connaissances ou les biais de la personne qui construit le modèle.

Exemple de biais inductif :

- Hypothèse de linéarité
- Hypothèse de proches voisins
- Hypothèse de maximum de marge
- Etc.

# Minimisation du risque empirique 1/2

Une fois la classe $\mathcal{C}$ est choisie, la fonction $f \in C$ est déterminée en minimisant une fonction coût (erreur).

### Definition

La fonction coût $L : \mathcal{Y} \times \mathcal{Y} \rightarrow R$ mesure l'erreur entre les prédictions $f(x)$ et les vraies valeurs de $y$.

### Definition

Etant donné un ensemble de $n$ observations étiquetées $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, le risque empirique d'une fonction $h$ est défini comme la moyenne emprique des erreurs associées aux prédictions $y_1, y_2, ..., y_n$.

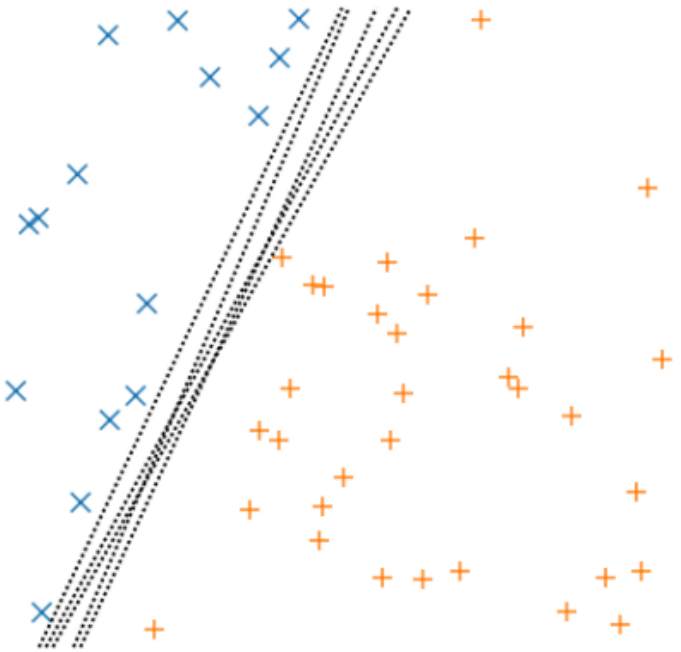
$$R_n(h) = \frac{1}{n} \sum_{i=1}^{n} L(h(x_i), y_i)$$

# Minimisation du risque empirique 2/2

La fonction $f$ est alors déterminée en minimisant le risque empirique $R_n$ :

$$f = \arg\min_{h \in \mathcal{C}} \frac{1}{n} \sum_{i=1}^{n} L(h(x_i), y_i)$$

**Remarque :** Il est important de noter que la minimisation du risque empirique ne garantit pas l'exitence d'une solution unique. En effet, le problème de l'apprentissage supervisé est généralement mal posé.



# Fonctions de coût

Le choix de la fonction de coût dépend de la nature du problème et des données. On distingue deux types de fonctions de coût.

**Régression**

- L'erreur quadratique : $L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$
- L'erreur absolue : $L(\hat{y}, y) = |\hat{y} - y|$

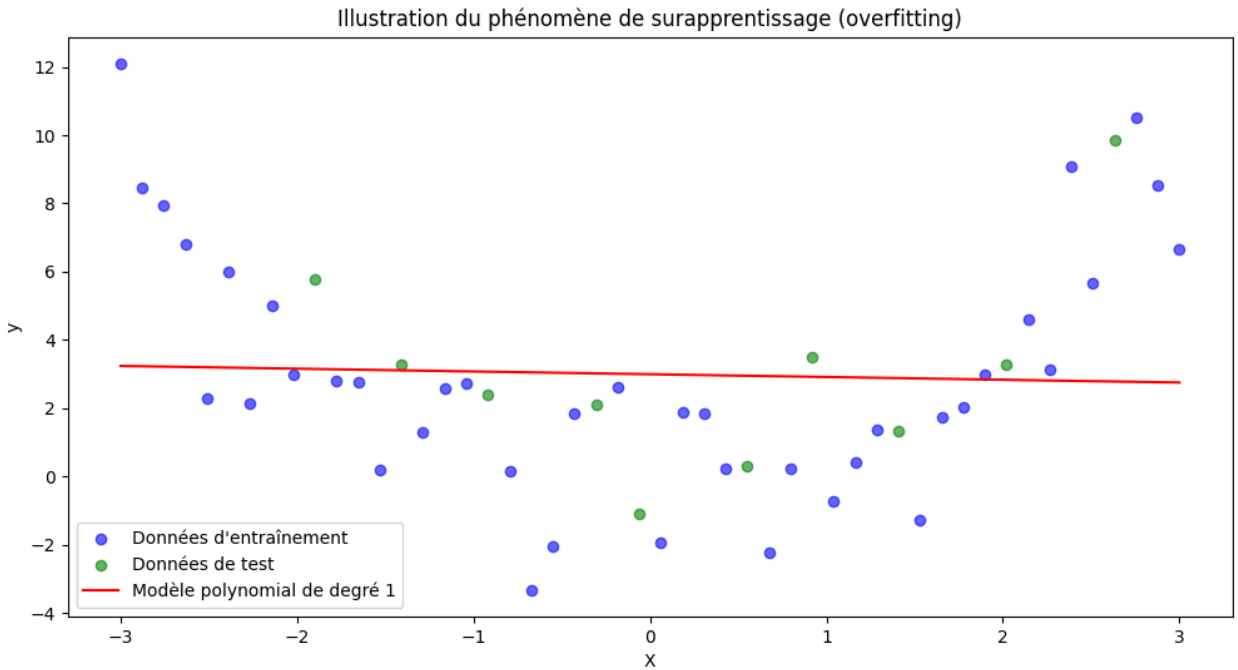
**Classification**

- Entropie croisée : $L(\hat{y}, y) = -ylog(\hat{y}) + (1 - y)log(1 - \hat{y})$
- L'erreur Hinge : $L(\hat{y}, y) = max(0, 1 - \hat{y}y)$

# Sous-apprentissage

## Definition

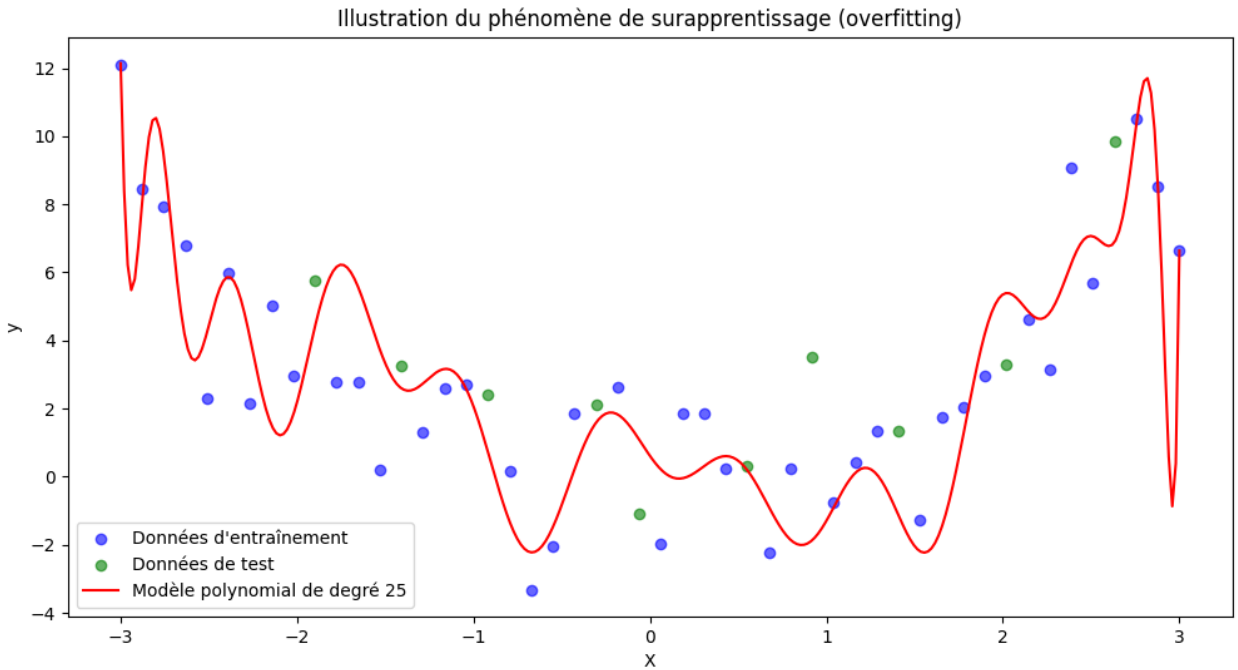
On dit qu'un modèle de machine learning est en régime de sous-apprentissage (underfitting) lorsqu'il n'arrive pas à capturer la complexité (l'information) présente dans le jeu de données d'entraînement.



# Sur-apprentissage

## Definition

On dit qu'un modèle de machine learning est en régime de sur-apprentissage (overfitting) lorsqu'il n'arrive pas à généraliser à des données non encore observées, i.e. lorsqu'il est trop adapté aux données d'entraînement.



# Compromis biais-variance

Soit un ensemble de $n$ observations $x_1, x_2, ..., x_n$ avec les labels correspondants $y_1, y_2, ..., y_n$ tels que : $y_i = f(x_i) + \epsilon_i$, où $\epsilon$ a une moyenne nulle et une variance $\sigma^2$.
On peut montrer que :

$$E\Big((f(x) - y)^2\Big) = Biais\Big(f(x)\Big) + Var\Big(f(x)\Big) + \sigma^2$$

où $Biais\Big(f(x)\Big) = E(f(x) - y)^2$

Pour remédier au problème de sur-apprentissage, on applique généralement ce que l'on appelle une régularisation.

# Métriques de performance : régression

On dispose d'un certain nombre de métriques pour évaluer les performances des modèles de machine learning. Celles-ci peuvent être divisées en deux catégories.

**Régression**

- L'erreur quadratique moyenne (MSE) : elle est définie comme la moyenne des carrés des écarts entre les prédictions et les valeurs observées.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2$$

- La racine carrée de l'erreur quadratique moyenne (RMSE) :

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2}$$

# Métriques de performance : classification 1/2

**Accuracy** : L'accuracy est la matrique de base qui permet d'évaluer les peerformance d'un modèle de classification. Elle est définie comme :

$$\text{Accuracy} = \frac{\text{Nombre de prédictions correctes}}{\text{Nombre total de prédictions}}$$

**Matrice de confusion** : La matrice de confusion est une représentation permettant d'offrir plus de finesse par rapport à l'accuracy, notamment quand le jeu de données est déséquilibré (présence de classes majoritaires). Elle compare les prédictions du modèle avec les valeurs réelles et est structurée comme suit :

| | | **Valeur Prédite** | |
|---|---|---|---|
| | | Positif | Négatif |
| **Valeur Réelle** | Positif | Vrai Positif (VP) | Faux Négatif (FN) |
| | Négatif | Faux Positif (FP) | Vrai Négatif (VN) |

# Métriques de performance : classification 1/2

A partir de la matrice de confusion, on peut dériver d'autres métriques :

- Précision : elle est définie comme la proportion des prédictions correctes parmi toutes les prédictions positives :

$$\text{Précision} = \frac{VP}{VP + FP}$$

- Rappel (recall) : il représente la proportion des vrais positifs correctement prédits par le modèle.

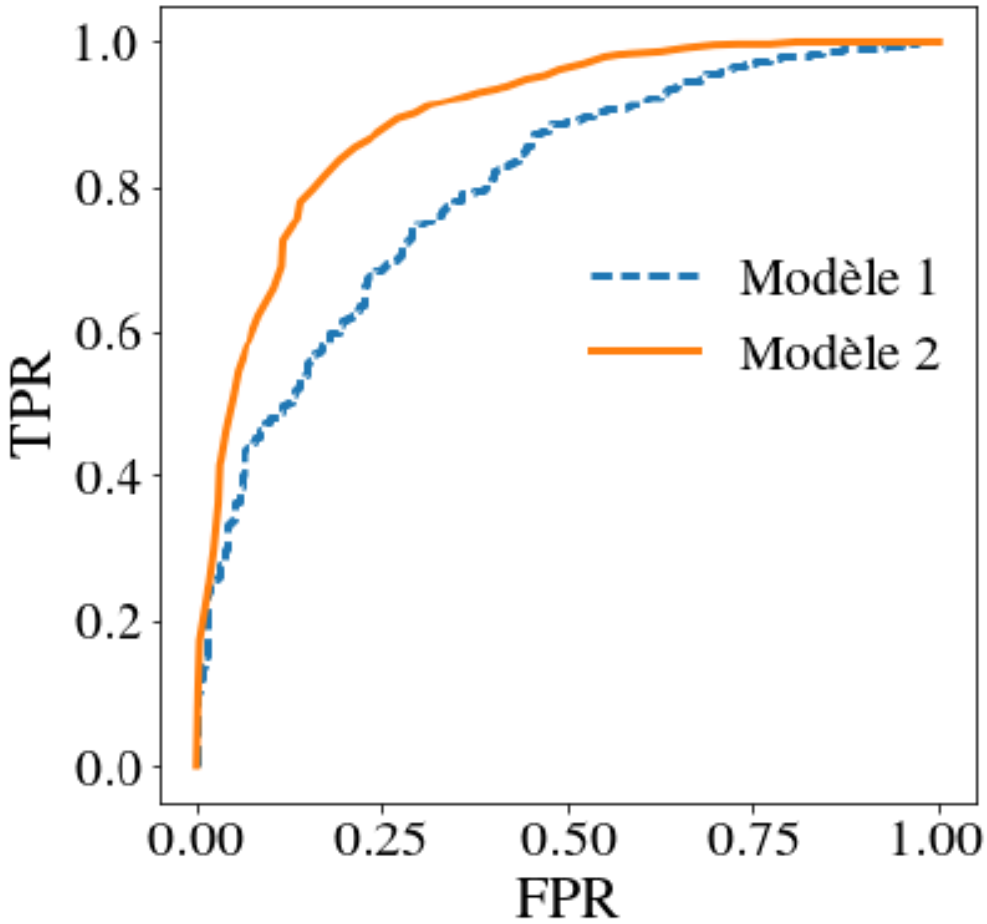
$$\text{Rappel} = \frac{VP}{VP + FN}$$

- Score F1 (F1-score) : Le score F1 est défini comme la moyenne harmonique de la précision et du rappel.

$$\text{Score F1} = 2\frac{Précisin \times Rappel}{Précision + Rappel}$$

# Métriques de performance : courbe ROC

- **ourbe ROC (Receiver-Operator Characteristic)** : Elle décrit l'évolution de la proportion des vrais positifs en fonction de celle des faux positifs.



# Sélection de modèle

Pour sélectionner le modèle le plus pertinent par rapport à une métrique donnée, on applique la méthodologie suivante :

- On partitionne le jeu de données disponible en trois parties : un jeu d'entraînement, un jeu de validation et un jeu de test.
- On entraîne $M$ modèles sur le jeu d'entraînement.
- On évalue les performances respectives des $M$ modèles sur le jeu de validation et on sélectionne le meilleur.
- Le modèle sélectionné est ensuite évalué sur le jeu de test. Idéalement, le jeu de test est ainsi utilisé une seule fois.

# Validation croisée K-fold

La validation croisée est une méthode plus robuste pour évaluer les performances des modèles. Il y a deux manières d'appliquer une validation croisée : K-fold et Leave-One-Out (LOO).

La validation croisée K-fold s'effectue selon la méthodologie suivante :

- On partitionne le jeu de données $\mathcal{D}$ en $K$ parties ayant approximativement la même taille.
- Pour chaque partie $\mathcal{D}_k$, on entraîne le modèle sur l'ensemble des données restantes $\bigcup_{i \neq k} \mathcal{D}_i$. Ce modèle est ensuite évalué sur $\mathcal{D}_k$.
- On considère la moyennes des performances du modèles sur les différents $\mathcal{D}_k$.

La validation croisée K-fold permet ainsi de prendre en compte la variabilité qu'il peut y avoir dans les données. Par ailleurs, on peut également avoir une estimation de la variance du modèle.

# Validation croisée Leave-One-Out

La validation croisée Leave-One-Out est un cas particulier de la validation croisée k-fold où le nombre de parties (folds) est également au nombre d'observations (K = n).

La validation croisée LOO s'effectue selon la méthodologie suivante :

- On partitionne le jeu de données $\mathcal{D}$ en $n$ parties ayant chacune $n-1$ observations.
- On entraîne le modèle sur chacune des parties $\mathcal{D}$ (ayant n-1 observations) et on l'évalue sur l'observation restante.
- On considère la moyennes des performances du modèles sur les différents $\mathcal{D}_k$.

La validation croisée LOO présente l'avantage que les modèles sont entraînés sur des jeux de données d'une taille plus grande. Par ailleurs, le nombre de modèles obtenus est plus grand également, ce qui peut plaider pour davantage de robustesse. Cependant, ces modèles sont entraînés sur des jeux de données plus similaires les uns par rapport aux autres. D'autre part, les jeux de test étant composés d'une seule observation, les performances risquent de présenter variabilité plus grande.

# Bootstrap

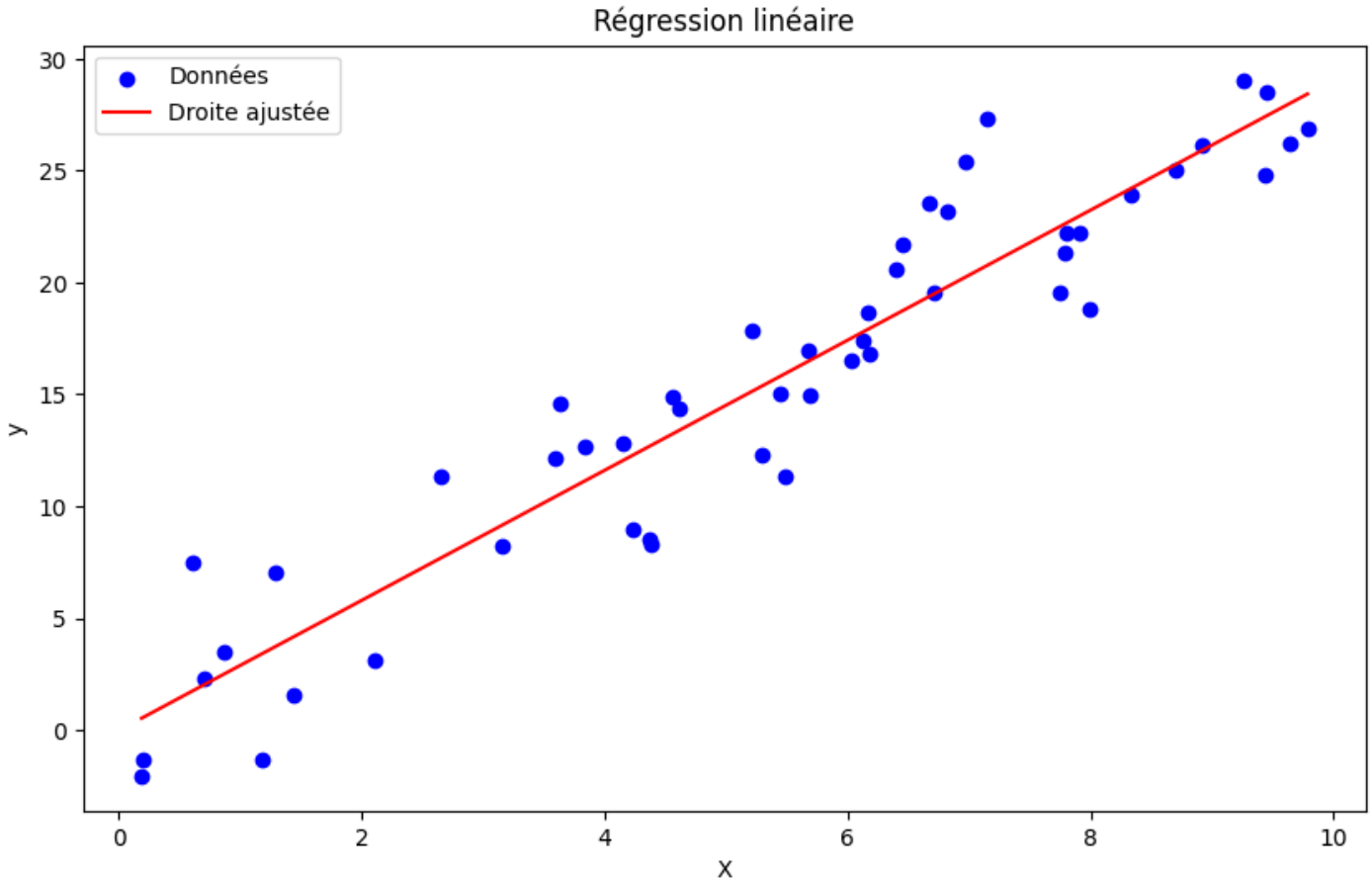
Le Bootstrap est une technique d'évaluation de modèle qui consiste à partitionner le jeu de données $\mathcal{D}$ en $K$ échantillons $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_K$. Chaque jeu de données est obtenu en tirant $n$ observations avec remise.

La performance du modèle est alors obtenue en moyennant sur ses différentes performances sur les $K$ jeux de données.

# Régression linéaire simple

Soit un ensemble de $n$ obeservations $x_1, x_2, ..., x_n$ avec les labels correspondants $y_1, y_2, ..., y_n$, on cherche le modèle linéaire qui ajuste le mieux ces données.

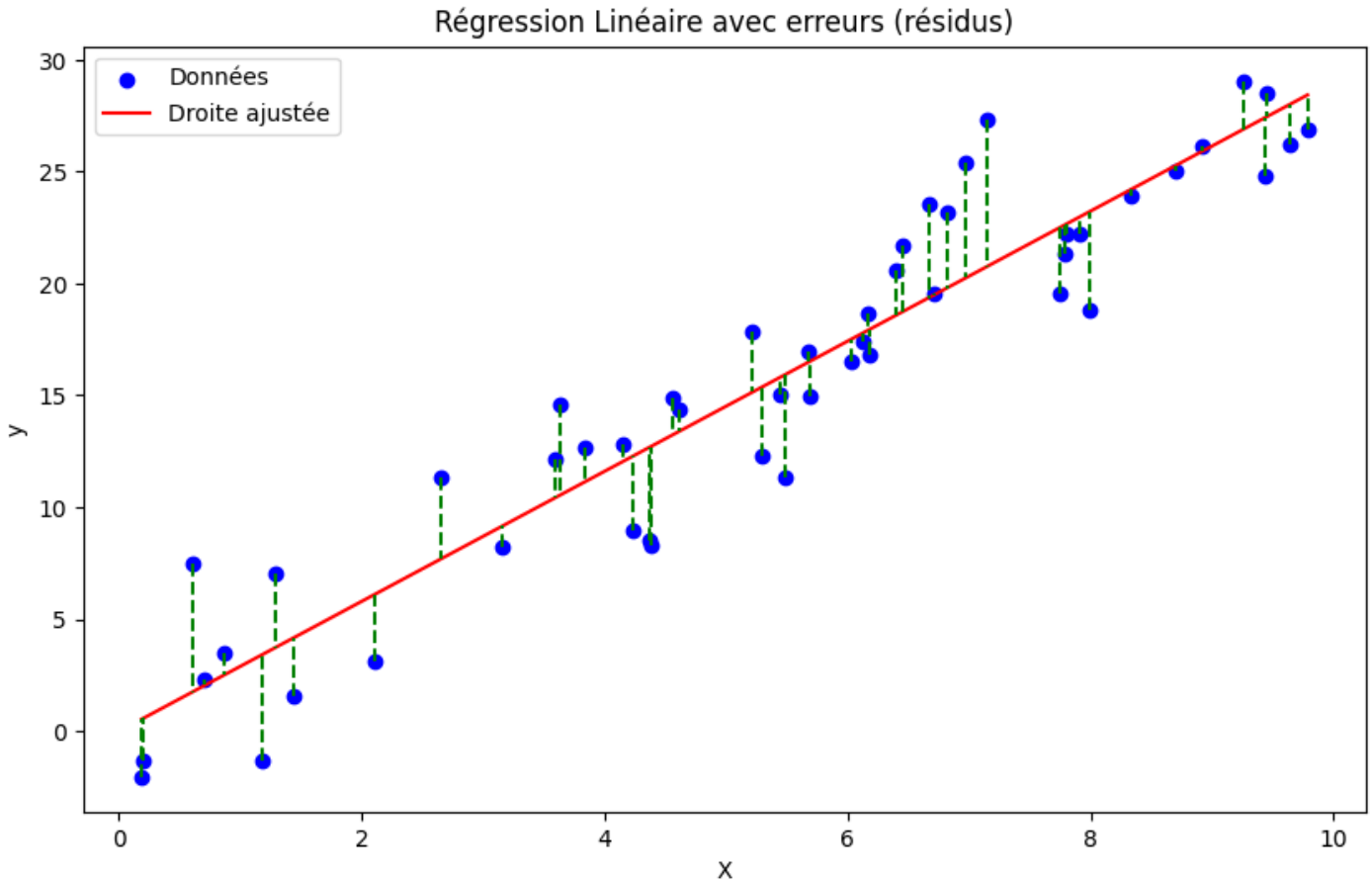
$$\hat{y} = \beta_0 + \beta_1 x$$



# Résidus

## Definition

Soit $\hat{y}_i = \beta_0 + \beta_1 x_i$ la $i$ ième prédiction du modèle. Le $i$ ième résidu, noté $e_i$, est alors défini comme l'erreur de prédiction sur $i$ ième observation : $e_i = y_i - \hat{y}_i$.



# Méthode des moindres carrés

On considère la somme des carrés des résidus, notée $RSS$ :

$$RSS = e_1^2 + e_2^2 + ... + e_n^2$$

L'approche par moindre carrés consiste à estimer les coefficients $\beta_0$ et $\beta_1$ en minimisant la $RSS$. Autrement dit :

$$\underset{\beta_0,\beta_1}{\arg\min} \left( \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2 \right)$$

Le problème peut être résolu d'une manière analytique. On obtient :

$$\begin{aligned} \beta_1 &= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \\ \beta_0 &= \bar{y} - \beta_1 \bar{x} \end{aligned} \tag{1}$$

# Régression linéaire multiple

On considère $n$ observations $X^1, X^2, ..., X^n$ où chaque observation $X^i$ est désormais un vecteur ayant $p$ composantes ($p$ variables explicatives).

$$X^i = \begin{pmatrix} x_1^i \\ x_2^i \\ \vdots \\ x_p^i \end{pmatrix}$$

La régression linéaire s'écrit alors :

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$$

Les coefficiens $\beta_0, \beta_1, \ldots, \beta_p$ sont déterminés par la méthode des moindres carrés :

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\arg\min} \sum_{i=1}^{n} \left( y^i - (\beta_0 + \sum_{j=1}^{p} \beta_j x_j^i) \right)^2$$

# L'équation normale

La régression linéaire peut être écrite sous une forme vectorielle :

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta}$$

où $X$ est est appelée matrice de design.

$$\mathbf{X} = \begin{pmatrix} 1 & x_1^1 & \dots & x_p^1 \\ \vdots & \vdots & \dots & \vdots \\ 1 & x_1^n & \dots & x_p^n \end{pmatrix}$$

La somme des carrés des résidus est donnée par :

$$RSS = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

On peut montrer que si la matrice de design $X$ est de rang plein, alors :

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

# Régression polynomiale

La régression linéaire peut prendre en compte les dépendances non linéaires entre les variables explicatives $x_1, x_2, .., x_p$ et la variable expliquée $y$. Lorsque cette dépendance prend la forme d'un polynome de degré $d$, la régression linéaire s'écrit alors :

$$\hat{y} = \beta_{00} + \sum_{j=1}^{p} \beta_{ij} x_j + \sum_{j=1}^{p} \beta_{ij} x_j^2 + ... + \sum_{j=1}^{p} \beta_{ij} x_j^d$$

Les coefficients $\beta_{ij}$ peuvent être de la même manière, avec la méthode des moindres carrés.

**Remarque :** On peut utiliser n'importe quelle fonction non linéaire pour transformer les variables explicatives ($cos$, $ln$, etc.). Le modèle reste tout de même linéaire (linéarité par rapport aux coefficients).

# Choix de modèle

Il y a plusieurs manières d'évaluer la pertinence d'un modèle de régression linéaire. Le coefficient coefficient de détermination $R^2$ mesure l'ajustement du modèle. Il est donné par :

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

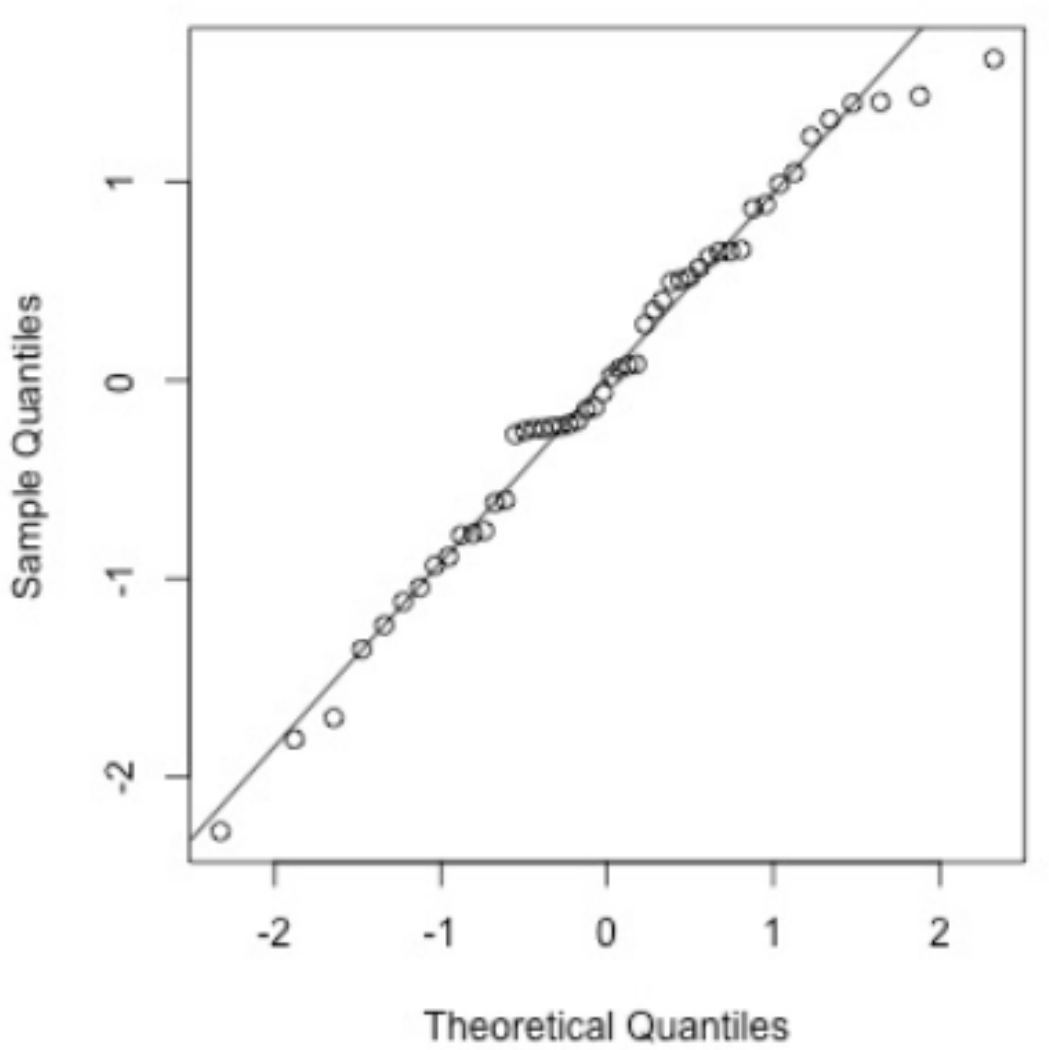
Pour une régression linéaire multiple, on préférera cependant le coefficient de détermination ajusté $R_a^2$.

$$R_a^2 = 1 - \frac{n-1}{n-k-1} * (1 - R^2)$$

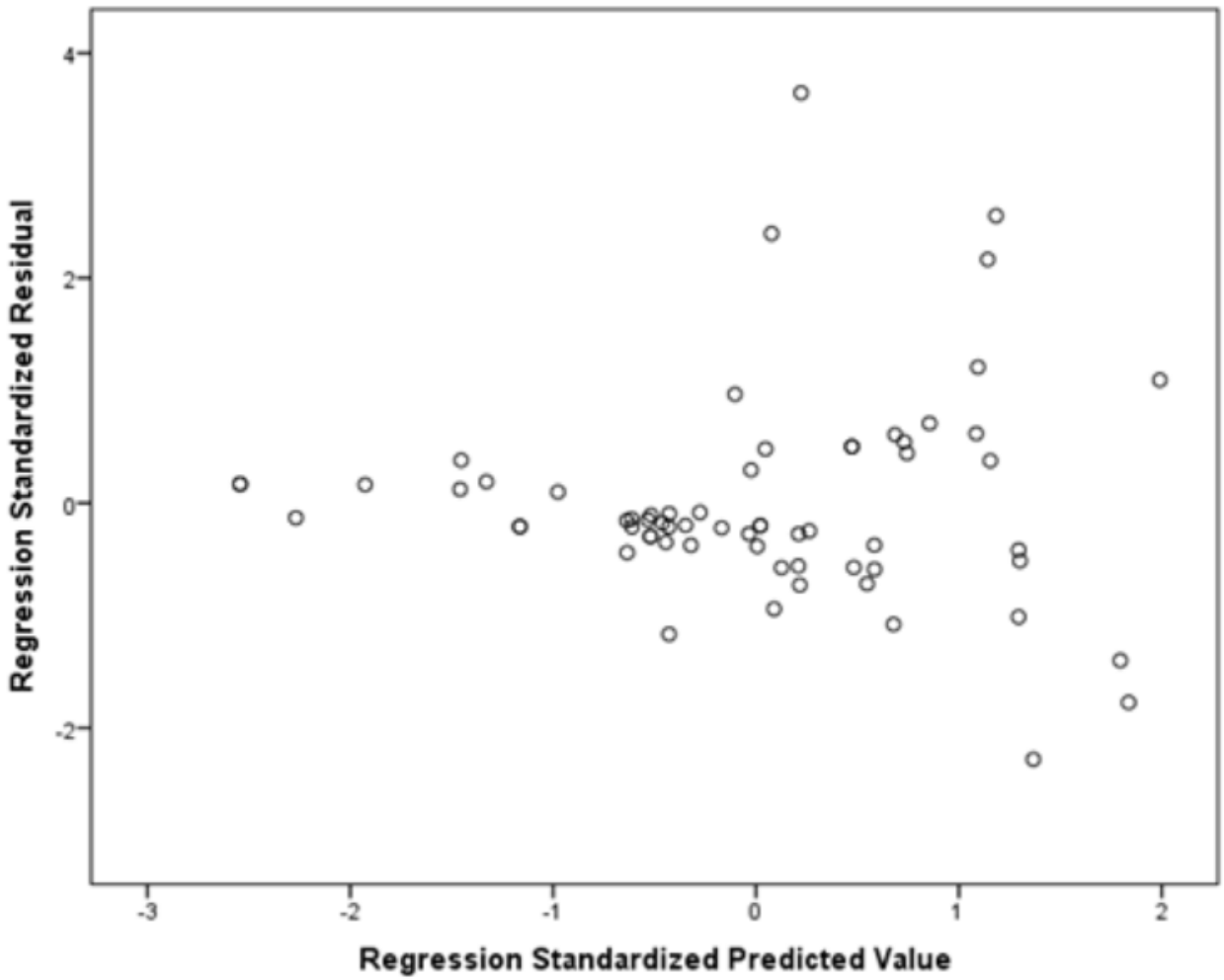
où $n$ est le nombre d'aboservations et $k$ le nombre de variables.

Des critères comme le AIC et le BIC sont également utilisés pour sélectionner un modèle.

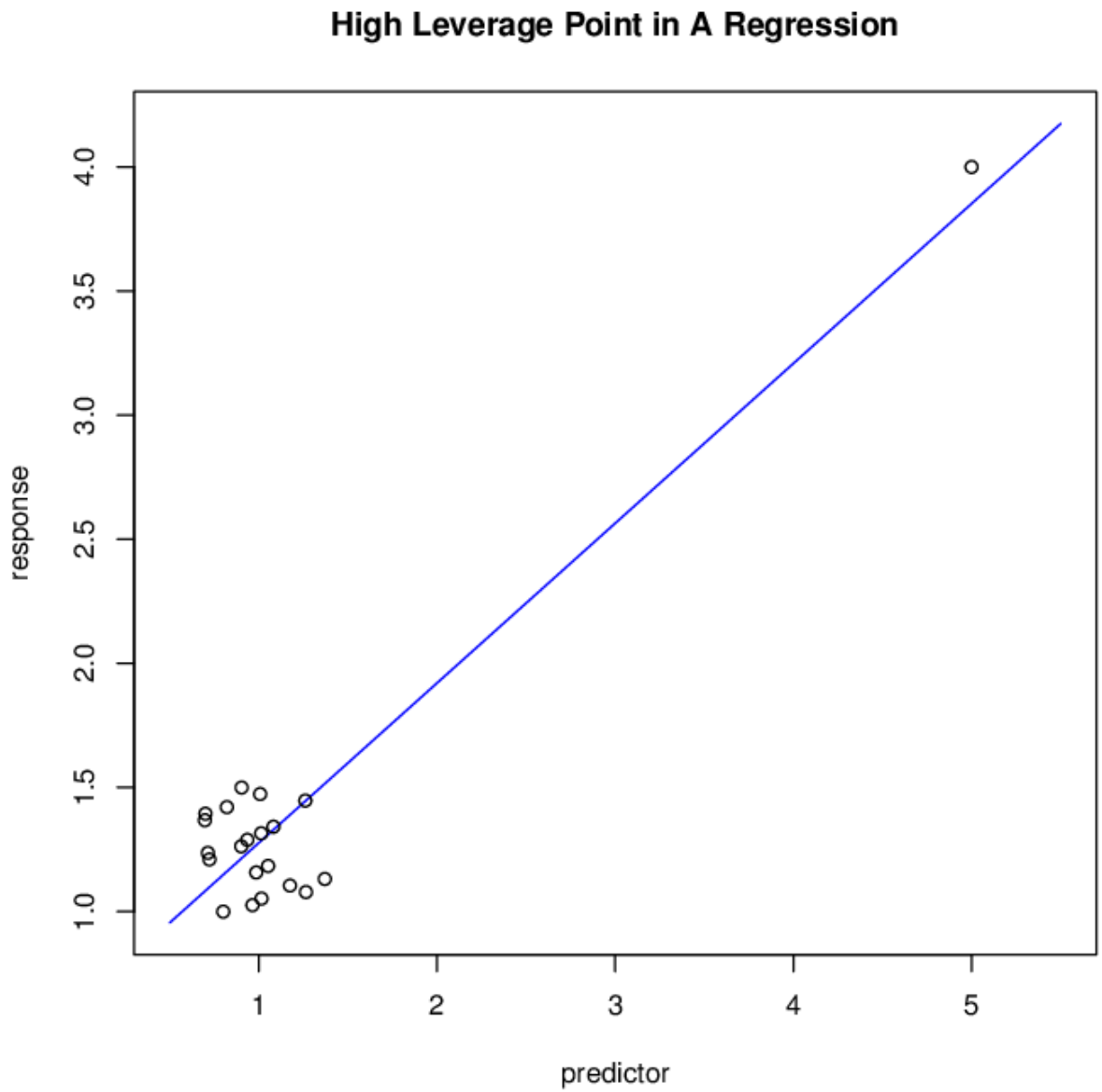
# Diagnostic de la régression linéaire : normalité des résidus



# Diagnostic de la régression linéaire : Homoscédasticité



# Diagnostic de la régression linéaire : effet levier



# Régression Ridge

La régularisation Ridge, dite également régularisation $L_2$, permet de remédier au problème de surapprentissage en imposant une contraite sur les coefficients $\beta$ lors de la minimisation du risque empirique.

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\arg\min} \left( \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{i=1}^{p} x_{ij}\beta_j)^2 \right)$$

sous la contrainte :

$$\sum_{j=1}^{p} \beta_j^2 \leq Cte$$

Le problème peut être écrit d'une manière plus compacte :

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\arg\min} \left( \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{i=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right)$$

où le paramètre $\lambda$ contrôle la force la régularisation. Plus $\lambda$ est grand, plus le modèle est régularisé.

# Formulation vectorielle de la régression Ridge

La somme des carrés des résidus associée à la régression Ridge s'écrit de la manière suivante :

$$RSS = (\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta}$$

Les coefficient $\vec{beta}$ peuvent alors être calculés en minimisant la $RSS$.

$$\underset{\beta_0,\beta_1,..,\beta_p}{\arg\min}\ (\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta}$$

On peut alors montrer que :

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

où $\mathbf{I}$ est la matrice identité de dimension $p \times p$.

# Formulation vectorielle de la régression Ridge

La matrice de design $X$ de dimensions $n \times n$ peut être décomposée en valeurs singulières (SVD) de la manière suivante :

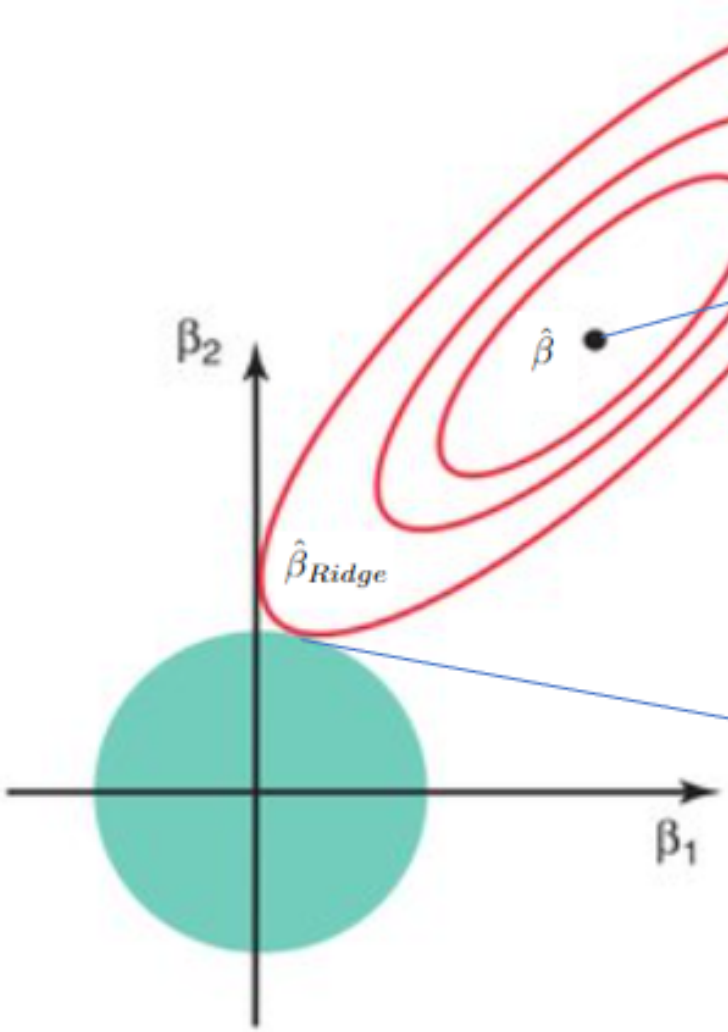
$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$$

où $\mathbf{U}$ et $\mathbf{V}$ sont des matrices orthoganales de dimensions respectives $n \times p$ et $p \times p$.
On peut montrer que :

$$\begin{aligned} \mathbf{X}\boldsymbol{\beta} &= \mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\boldsymbol{y} \\ &= \sum_{j=1}^{p} u_j \frac{d_j^2}{d_j^2 + \lambda} u_j^T \vec{y} \end{aligned} \tag{2}$$

On peut notrer que le paramètre de régularisation $\lambda$ tend à réduire l'influence des variables explicatives associées à une faible valeur singulière.

# Interprétation géométrique de régression Ridge




$$\arg\min_{\beta_0,\beta_1,\ldots,\beta_p} \left( \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{i=1}^{p} x_{ij}\beta_j)^2 \right)$$

$$\arg\min_{\beta_0,\beta_1,\ldots,\beta_p} \left( \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{i=1}^{p} x_{ij}\beta_j)^2 \right)$$

$$\sum_{j=1}^{p} \beta_j^2 \leq Cte$$

# Régression Lasso

La régularisation Lasso, dite également régularisation $L_1$, force les coefficients associés à des variables explicatives ayant une moindre importance vers zéro. C'est ainsi une technique de réduction de la dimensionalité du problème pour avoir un modèle avec peu de variable (plus simple et plus explicable).

La régression Lasso est formalisée de la manière suivante :

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\arg\min} \left( \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{i=1}^{p} x_{ij}\beta_j)^2 \right)$$

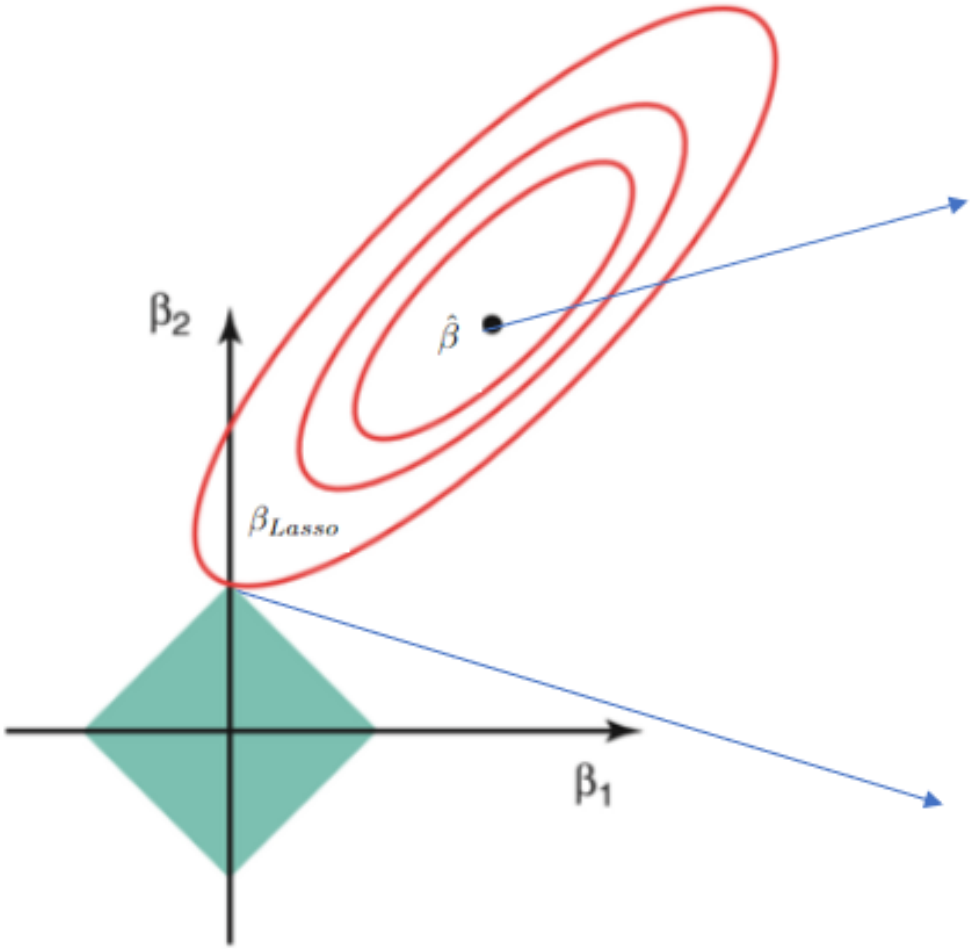
sous la contrainte :

$$\sum_{j=1}^{p} |\beta_j| \leq Cte$$

Où

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\arg\min} \left( \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{i=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right)$$

# Interprétation géométrique de régression Lasso




$$\arg\min_{\beta_0,\beta_1,\ldots,\beta_p} \left( \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{i=1}^{p} x_{ij}\beta_j)^2 \right)$$

$$\arg\min_{\beta_0,\beta_1,\ldots,\beta_p} \left( \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{i=1}^{p} x_{ij}\beta_j)^2 \right)$$

$$\sum_{j=1}^{p} |\beta_j| \leq Cte$$

# Elastic net

Elastic net combine les deux approches, Ridge et Lasso, pondérées avec un parramètre $\alpha \in [0, 1]$.

Le problème s'écrit ainsi :

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\arg\min} \left( \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{i=1}^{p} x_{ij}\beta_j)^2 \right)$$

sous la contrainte :

$$\sum_{j=1}^{p} (1-\alpha)|\beta_j| + \alpha\beta_j^2 \leq Cte$$

Ou

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\arg\min} \left( \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{i=1}^{p} x_{ij}\beta_j)^2 + \lambda( \sum_{j=1}^{p} (1-\alpha)|\beta_j| + \alpha\beta_j^2) \right)$$

# Régression logistique : formulation du problème

On cherche à modéliser la probabilité conditionnelle $Pr(Y = 1|X = \vec{x})$ comme une fonction de $\vec{x}$.

$$Pr(Y = 1|X = \vec{x}) = f(\vec{x})$$

On veut par ailleurs que :

- La fonction $f$ prenne ses valeurs dans $[0, 1]$.
- La fonction $f$ varie lorsque sa valeur est proche de 0 ou de 1.

# Régression logistique : fonction logit

**Definition**

On appelle la fonction logit la transformation définie :

$$[0,1] \to R$$
$$p \mapsto \ln \frac{p}{1-p}$$

La régression logistique s'écrit alors :

$$\ln \frac{p(\mathbf{x})}{1-p(\mathbf{x})} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p$$

Après quelques simplifications :

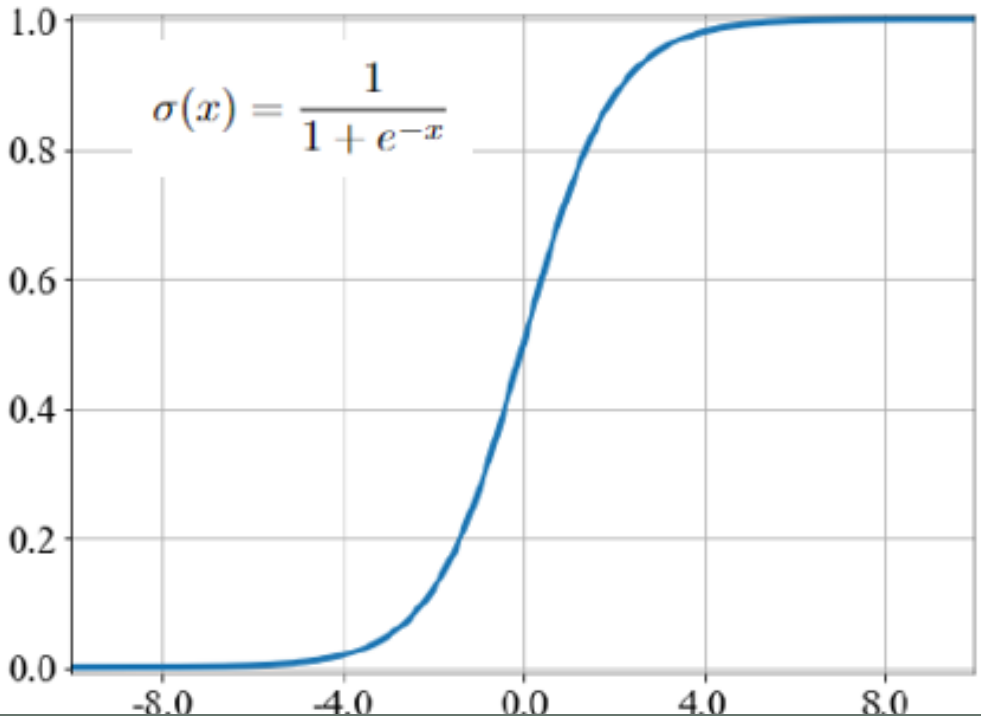
$$p(\mathbf{x}) = \frac{1}{1+e^{-(\boldsymbol{\beta}^T \mathbf{x})}}$$

# Fonction logistique

## Definition

La fonction logistique, notée $\sigma$, est définie :

$$\begin{aligned} \sigma : & R \to [0,1] \\ & x \mapsto \frac{1}{1+e^{-x}} \end{aligned} \tag{3}$$



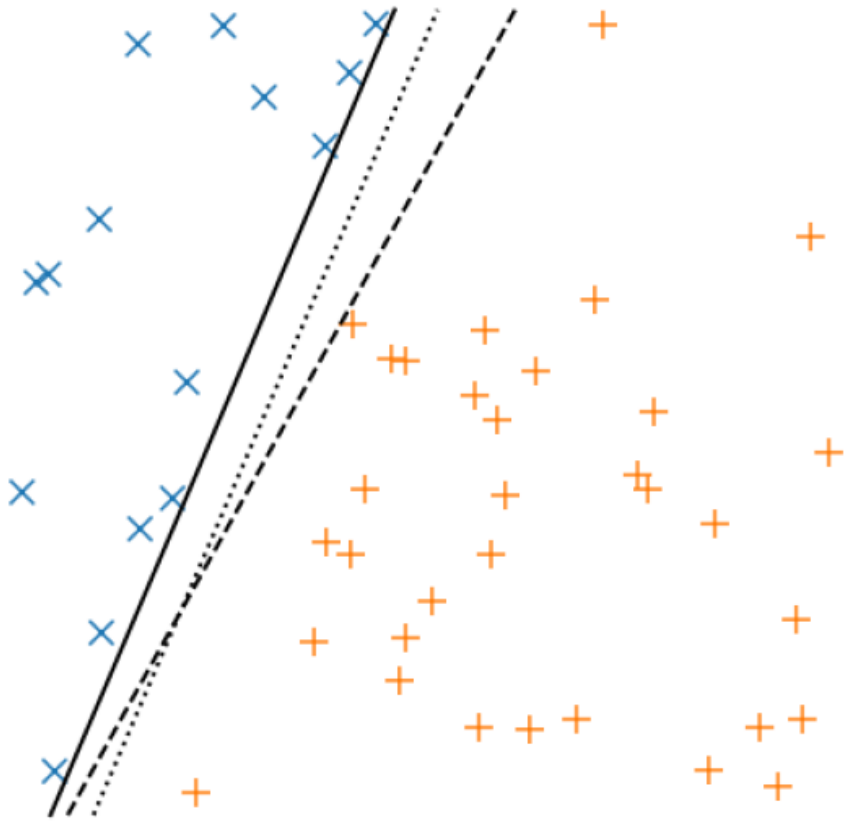
# Calcul des coefficients de la régression logistique

Les coefficients de la régression logistique peuvent être calculés en minimisant le risque empirique par rapport à une fonction de coût sous forme d'entropie croisée :

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\arg\min} \left( -\sum_{i=1}^{n} y_i \ln(\hat{y}_i) + (1-y_i)\ln(1-\hat{y}_i) \right)$$

# Machines à vecteurs de support (SVM)

Considérons un problème de classification binaire. On recherche l'hyperplan séparateur qui maximise la marge $\gamma$ entre les deux classes. La marge $\gamma$ étant définie comme la distance entre cet hyperplan et les observations les plus proches.

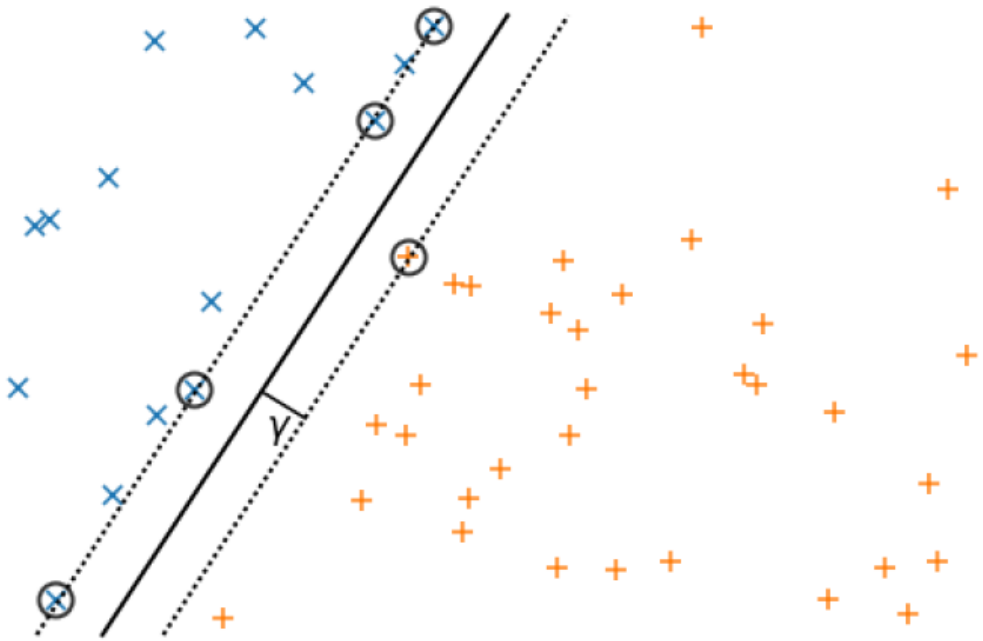


# Position du problème

Soient $n$ observations $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ et $n$ poids associés $w_1, w_2, \ldots, w_n$.

On cherche la fonction $(w^T x + b)$ telle que lorsque $h(x) \geq 1$ alors $x$ appartient à la classe 1 et lorsque $h(x) \leq -1$ alors $x$ appartient à la classe 0.

L'hyperlan d'équation $h(x) = 0$ est ainsi le plan séparateur pour les deux classes.



# Formulation primale

La distance entre un point $x_k$ et sa projection sur le plan séparateur est donnée par :

$$\gamma_k = y_k \frac{w^T x_k + b}{\|w\|}$$

Les poids $w$ et le biais $b$ peuvent être calculés en résolvant le problème d'optimisation suivant :

$$\arg\max_{w,b} \left( \frac{1}{\|w\|} \min_k \left( y_k (w_k^T x_k + b) \right) \right)$$

En pratique, il est plus simple de résoudre le problème équivalent :

$$\arg\min \frac{1}{2} \|w\|^2$$

sous la contrainte :

$$y_k (w^T x_k + b) \geq 1$$

# Formulation duale 1/2

Le problème d'optimisation peut être résolu à l'aide des multiplicateurs de Lagrange $\alpha$. Le lagrangien s'écrit :

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{k=1}^{n} \alpha_k \Big( y_k(w^T x + b) \Big)$$

En minimisant le lagrangien par rapport à $w$ et $b$

$$\begin{aligned} \nabla_w L(w, b, \alpha) &= 0 \\ \frac{\partial L(w, b, \alpha)}{\partial b} &= 0 \end{aligned} \tag{4}$$

on obtient :

$$\begin{cases} \sum_{k=1}^{n} \alpha_k y_k x_k = w \\ \sum_{k=1}^{n} \alpha_k y_k = 0 \end{cases}$$

# Formulation duale 2/2

En injectant les équations précédentes dans la formule du lagrangien, on obtient :

$$L(w, b, \alpha) = \sum_{k=1}^{n} \alpha_k - \frac{1}{2} \sum_{k,j}^{n} y_k y_j \alpha_k \alpha_j x_k^T x_j$$

Le problème duale s'écrit ainsi :

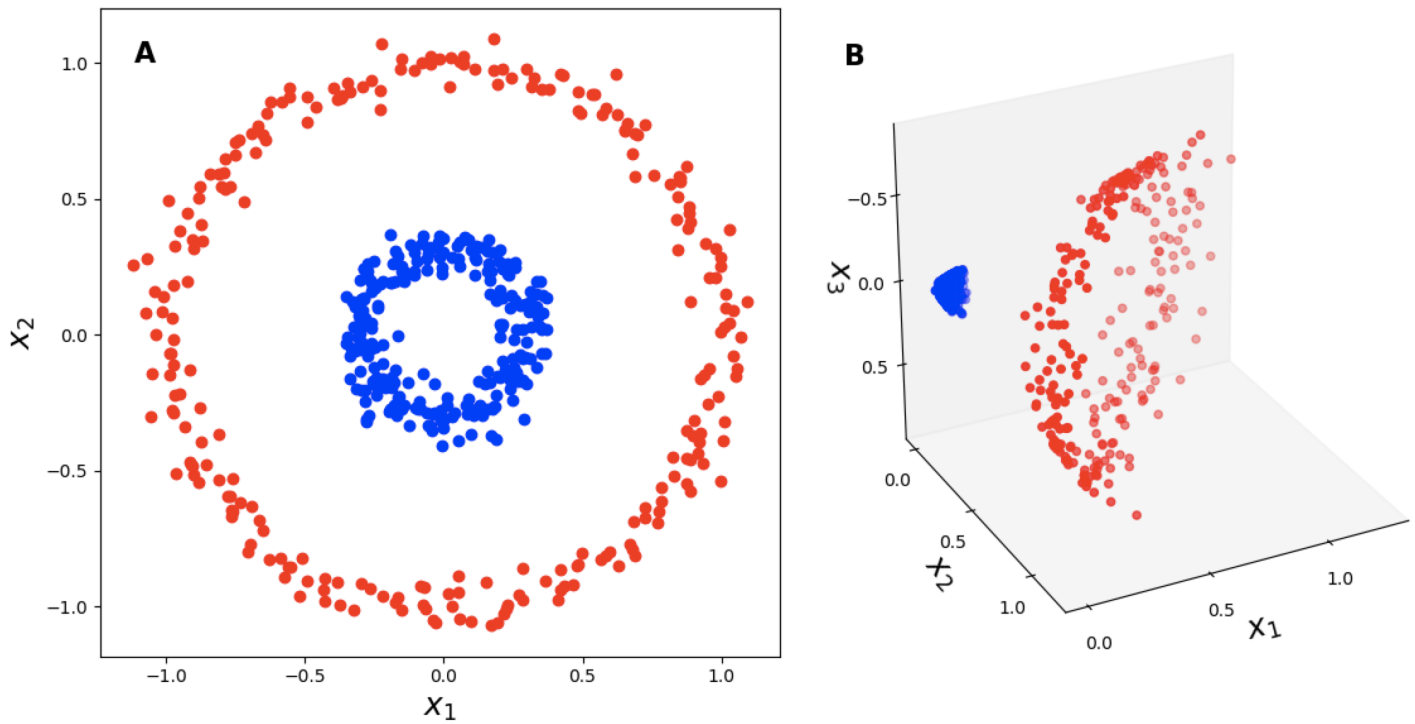
$$\arg\max_{\alpha} \left( \sum_{k=1}^{n} \alpha_k - \frac{1}{2} \sum_{k,j}^{n} y_k y_j \alpha_k \alpha_j x_k^T x_j \right)$$

sous les contraintes

$$\begin{cases} \alpha_k \geq 0, \ k = 1, 2, \ldots, n \\ \sum_{k=1}^{n} \alpha_k y_k = 0 \end{cases}$$

# SVM à noyau

On considère un problème de classification non linéaire. L'astuce du noyau consiste à augmenter la dimension du problème, pour le résoudre ensuite avec un séparateur linéaire dans le nouvel espace.



# Théorème de Mercer

## Definition

On appelle noyau la fonction $K$ définie dans un espace de Hilbert qui associe à deux vecteurs $x, z$ le produit scalaire suivant :

$$K(x, z) = \Phi(x) \cdot \Phi(z)$$

où $\Phi$ est généralement une fonction non linéaire.

## Théorème

*Soit la fonction $K : R^p \times R^p \mapsto R$. Alors $k$ est un noyau de Mercer seulement et seulement si, pour tout $x \in x_1, x_2, \ldots, x_n$, la matrice correspondante est symétrique et semi-définie positive.*

# Formulation duale des SVM à noyau

Le problème dual pour les SVM à noyau s'écrit alors :

$$\arg\max_{\alpha} \left( W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j K(x_i, x_j \right)$$

sous les contraintes

$$\begin{cases} \alpha_k \geq 0, \ k = 1, 2, \ldots, n \\ \sum_{k=1}^{n} \alpha_k y_k = 0 \end{cases}$$

où la fonction noyau est donné par :

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

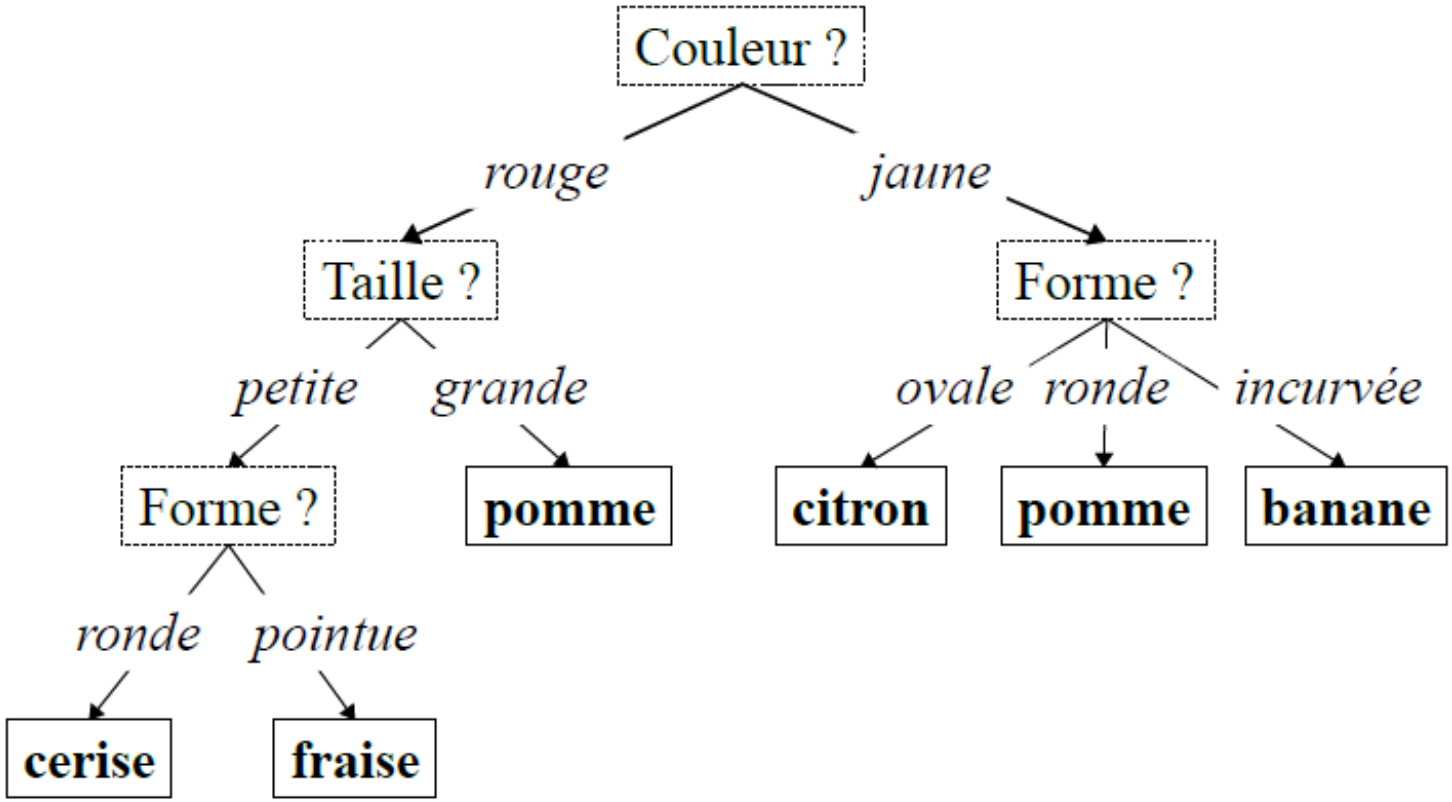
# Exemples de noyaux

Il existe une variété de noyaux pour les SVM. Les plus utilisés étant les noyaux gaussien et polynomial.

- Noyau gaussien : $k(x, z) = \exp(-\frac{\|x-z\|^2)}{2\sigma^2}$
- Noyau polynomial : $K(x, z) = (x^T z + 1)^d$

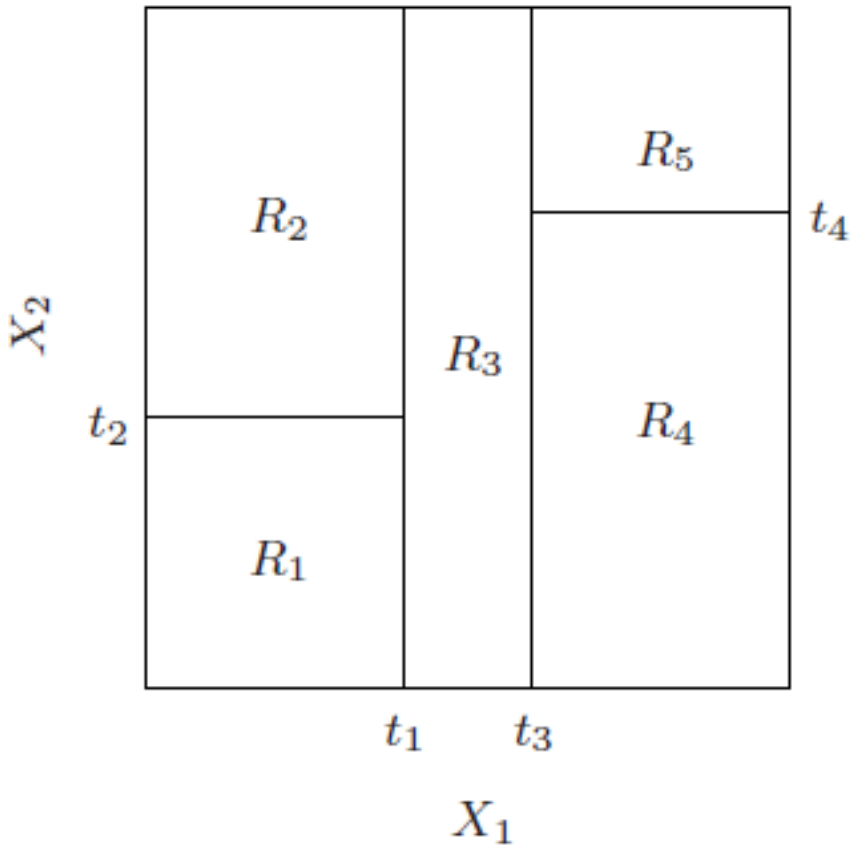
# Arbre de décision

Les arbres de décisions sont des modèles dont le processus de décision est hiérarchique et prend la forme d'un arbre.



# Entraînement des arbres de décision

Les arbres de décisions sont généralement entraînés à l'aide de la technique CART (Classification And Regression Trees).
Etant donné un ensemble d'observations $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, les arbres de décision partitionnent cet espace en plusieurs régions $R_1, R_2, \ldots, R_m$.



# Régression

Pour une régression, l'approche adoptée pour entraîner un arbre de décision consiste à chercher la meilleure variable $x_j$ et le meilleur point de partionnement $s$ par rapport à cette variable en considérant le problème d'optimisation suivant :

$$\arg\min_{j,s} \Big( \sum_{x_i \in R_l(j,s)} (y_i - y_l(j,s))^2 + \sum_{x_i \in R_r(j,s)} (y_i - y_r(j,s))^2 \Big)$$

où $y_l(j,s)$ et $y_r(j,s)$ sont les moyennes des labels associées à chacune des deux régions formées par le partitionnement.

Il est important ici de noter que nous considérons comme critère d'optimisation la somme des erreurs quadratiques :

$$\sum_{i=n}^{n} (y_i - f(x_i))^2$$

# Classification

L'approche pour la classification est similaire à celle pour la régression ; la fonction de coût quadratique est remplacée cette fois par une fonction mesurant l'impureté au niveau des feuilles (le degré d'hétérogénéité des labels).
On considère donc le problème d'optimisation suivant :

$$\arg\min_{j,s} \Big( \sum_{x_i \in R_l(j,s)} \big( \frac{|R_l(j,s)|}{n} Imp(R_l(j,s)) + \frac{|R_r(j,s)|}{n} Imp(R_r(j,s)) \big)$$

On définit la proportion d'exemples d'entraînement qui appartiennent à la classe $c$ par :

$$p_{mk} = \frac{1}{|R_m|} \sum_{x_i \in R_m} \delta(y_i, k)$$

Différentes fonctions mesurant l'impureté peuvent être considérées :

- Erreur de classification : $1 - p_{mk}$
- Indice de Gini : $\sum_{k=1}^{K} p_{mk}(1 - p_{mk})$
- Entropie croisée : $-\sum_{k=1}^{K} p_{mk} \ln(p_{mk})$

# Elagage de arbres de décisions

Les arbres de décision peuvent sur-apprendre très facilement. Ainsi, pour limiter le phénomène de sur-apprentissage, les modèles peuvent être simplifiés par élagage (pruning). On considère deux types de pruning :

- Elagage a priori : la complexité des arbres est limitée par construction en introduisant un critère de complexité :

$$C_\lambda(T) = \sum_{m=1}^{|T|} \frac{Imp(R_m)}{|R_m|} + \lambda|T|$$

où $|T|$ et $|R_m|$ sont respectivement le nombre de régions de l'arbre et le cardinal de la région $R_m$.

- Elagage a posteriori : les arbres sont élagués après construction, en supprimant des sous-arbres et en les remplaçant par des feuilles (en fusionnant des régions).

# Méthodes ensemblistes (bagging)

Le bagging vise à améliorer les performances du modèles en termes de robustesse (réduction de la variance). Il repose sur deux principes :

- Les predictions issues d'un ensemble de modèles, même faibles, sont plus robustes que celles issues d'un seul modèle.
- L'échantillonnage aléatoire améliore la robustesse du modèle.

Soit un ensemble de $n$ observations $\mathcal{D} = (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n$. Le bagging consiste à construire un modèles prédictif selon les étapes suivantes :

- On construit $B$ échantillons en bootstrap (tirage avec remise) à partir de $\mathcal{D}$.
- On entraîne autant de modèles sur chaque échantillon.
- On combine les prédictions issues des $B$ modèles (vote majoritaire pour la classification et calcul de la moyenne pour la régression).

# Forêts aléatoires (random forests)

La technique des forêts aléatoires (random forests) consiste à appliquer une approche de type bagging sur les arbres de décision.

L'algorithme random forests suit cette procédure :

- Tirer par bootstrap $B$ échantillons de tailles $n$ à partir de l'ensemble $D$.
- Pour chaque échantillon tiré, construire un arbre en répétant les étapes suivantes jusqu'à atteindre $n_min$.
  - Tirer d'une manière aléatoire $m$ variables parmi les $p$ variables.
  - Sélectionner la meilleure variable avec le meilleur point de partitionnement.
  - partitionner le noeud en deux sous-branches.
- Agréger les arbres construits.

  Les prédictions sont agrégées selon qu'il s'agisse de régression ou de classification :

- Régression : moyenne $f^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$
- Classification : vote majorataire.

# Remarques

- L'algorithme de random forests intègre nativement une forme de validation croisée. Les performances mesurées sur $\bigcup_{b_i \neq b_k}$ (out of bag ou OOB) sont souvent proches de celles que l'on pourrait mesurer avec une validation croisée.
- Le nombre de variable tirées pour chaque noeud est généralement donné par $\sqrt{p}$ pour la classification et $\frac{p}{3}$ pour la régression. Cet hyperparamètre dépend cependant du problème considéré.
- Lorsque le nombre de vairable est élevé alors que le nombre de variables réellement partinente est faible, la probabilité que les $p$ variables sélectionnées pour chaque partitionnement incluent des variables pertinentes devient faible, et les performances du modèle en termes de généralisation peuvent se détériorer considérablement.
- L'algorithme random forests permet de restituer des informations sur l'importance des variable (feature importance).

# Boosting

Le boosting est une méthode d'apprentissage ensembliste qui combine plusieurs modèles faibles pour créer un modèle global plus fort. Il se concentre sur la conversion d'apprenants faibles en apprenants forts.

**Principes de Base** :

- Construire séquentiellement des modèles faibles, généralement des arbres de décision.
- Chaque modèle suivant essaie de corriger les erreurs du modèle précédent.
- Les modèles sont pondérés en fonction de leur précision et combinés pour obtenir le modèle final.

**Avantages** :

- Amélioration de l'exactitude et de l'efficacité des prédictions.
- Bonne performance sur des ensembles de données variés.
- Réduction des risques de surapprentissage comparé aux modèles individuels.

# Adaptive boosting (Adaboost)

**Adaptive Boosting (Adaboost)** : Méthode itérative pour améliorer un ensemble de modèles faibles.

**Algorithme** :

1. Initialisez les poids des observations : $D_1(i) = \frac{1}{n}$.
2. Pour $t = 1, 2, ..., T$ :
   - Entraînez un modèle faible $h_t(x)$.
   - Calculez l'erreur : $\epsilon_t = \sum D_t(i)[y_i \neq h_t(x_i)]$.
   - Calculez le poids du modèle : $\alpha_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$.
   - Mettez à jour les poids : $D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$.
3. Modèle final : $H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right)$.

**Remarques** :

- Adaboost ajuste les poids des observations pour se concentrer sur les erreurs difficiles.
- Les modèles sont combinés en fonction de leur précision.

# Gradient Boosting

**Gradient Boosting** : Construction itérative d'un modèle additif en minimisant une fonction de perte.

**Processus Itératif** :

1. Commencez avec un modèle initial : $f_0(x)$.
2. Pour $t = 1, 2, ..., T$ :
   - Calculez les résidus : $r_{ti} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{t-1}(x)}$.
   - Entraînez un modèle faible $h_t(x)$ sur $r_{ti}$.
   - Trouvez $\gamma_t$ qui minimise $L(y_i, f_{t-1}(x_i) + \gamma h_t(x_i))$.
   - Mettez à jour : $f_t(x) = f_{t-1}(x) + \gamma_t h_t(x)$.
3. Modèle final : $f_T(x)$.

**Remarques** :

- Chaque modèle successif corrige les erreurs du modèle précédent.
- Les arbres de décision sont généralement utilisés comme modèles faibles.

# Apprentissage non supervisé

# Apprentissage non supervisé

Dans l'apprentissage non supervisé, on considère $n$ observations sans labels. On s'intéresse fondamentalement à la probabilité jointe de ces observations.

On peut distinguer deux grandes catégories d'apprentissage non supervisé :

- Clustering (partionnement) : cela consiste à patitionner les $n$ observations en $K$ groupes pertinents (généralement le critère de pertinence a une signification d'un point de vue métier).
- Réduction de dimension : il s'agit de trouver une représentation des données originelles dans un nouvelles espace de plus petite dimension. Cela peut être effectué à différentes fins : visualisation des données, compression des données, amélioration des performances du modèles (modèles plus robuste, plus explicable, etc.).

# Méthode des k-moyennes (k-means)

Soit un ensemble de $n$ observations $(x_i, x_2, \ldots, x_n$. La méthode des k-means vise à partitionner cet ensembe selon $k$ groupes en minimisant la variance globale à l'intérieur des clusters. Plus formellement, le problème des k-means s'écrit :

$$\underset{\mathcal{C}_1,\mathcal{C}_2,\ldots,\mathcal{C}_K}{\arg\min} \sum_{k=1}^{K} \sum_{x \in \mathcal{C}_k} \|x - u_k\|^2$$

Une telle optimisation est cependant difficile et très coûteuse en temps.

# Algorithme de Lloyd

Considérons $n$ observations $x_1, x_2, \ldots, x_n$ et un nombre déterminé $K$ de clusters, l'algorithme de Lloyd consiste à exécuter les étapes suivantes :

- On choisit aléatoirement $k$ centroïdes parmi les $n$ observations.
- On affecte chaque observation $x_i$ au cluster dont le centroïde est plus proche.

$$k(x_i) = \underset{k=1,2,\ldots,K}{\arg\min} \|x_i - u_k\|$$

- recalculer les controïdes de chaque cluster avec la nouvelle configuration.

$$\mu_k = \frac{1}{\mathcal{C}_k} \sum_{x_i \in \mathcal{C}_{\|}} x_i$$

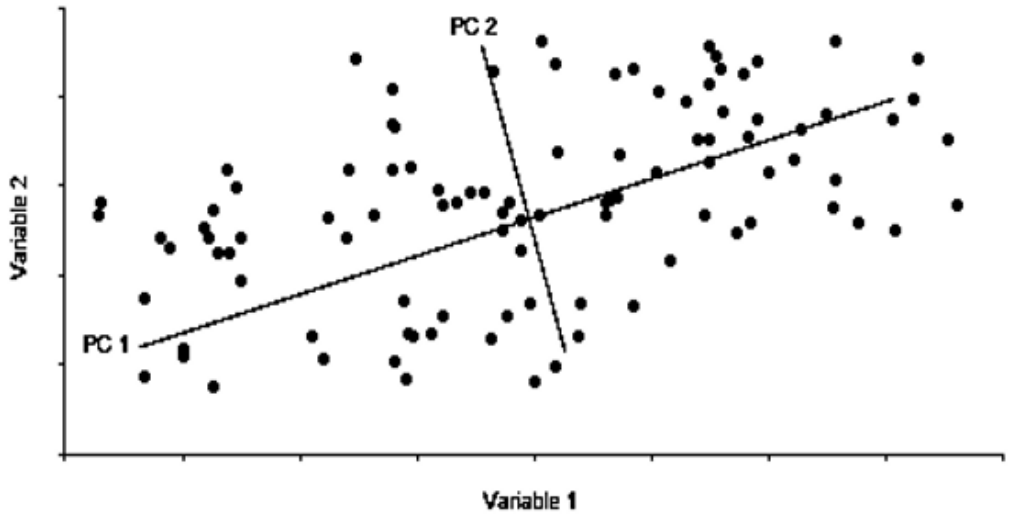
- On répète la procédure jusqu'à convergence (jusqu'à ce que les centroïdes soient stables).

# Remarques

- L'algorithme des k-means étant basé sur une distance euclidienne, il est nécessaire de normaliser les données avant de l'exécuter.
- L'algorithme des k-means est très sensible aux données aberrantes (outliers). Il faut donc considérer les données d'une manière attentive. Cependant, cela permet également d'utiliser l'algorithme des k-means pour la détection automatique des outliers.
- Les centroïdes étant initialisés d'une manière aléatoire, les clusters obtenus ne sont pas stables ; les clusters peuvent changer d'une exécution à l'autre. Il existe cependant une variante plus stable, appelée k-means++, qui permet de sélectionner les centroïdes d'une manière semi-aléatoire.
- Il est possible de partitionner les données avec une métrique plus générale que la distance euclidienne. On peut définir un algorithme k-means à noyau sur un espace de Hilbert pour aller au-delà de la métrique euclidienne.

# Analyse en Composantes Principales (ACP)

L'Analyse en Composantes Principales (ACP) est une technique statistique de réduction de dimensionnalité. Elle transforme les données en un nouveau système de coordonnées où la plus grande variance est capturée sur les premiers axes, appelés composantes principales.



# Formulation mathématique de l'ACP

Soit $X$ une matrice de données de dimension $n \times p$ (n observations, p variables), centrée (moyenne nulle). L'ACP cherche à trouver les vecteurs propres et les valeurs propres de la matrice de covariance $C = \frac{1}{n-1} X^T X$.

Les composantes principales sont données par les vecteurs propres $u_k$ de $C$, ordonnés par leurs valeurs propres correspondantes $\lambda_k$ en ordre décroissant.

La k-ième composante principale de l'ensemble de données est donnée par :

$$Z_k = X u_k$$

Les valeurs propres $\lambda_k$ représentent la variance expliquée par chaque composante principale.

# Décomposition en valeurs propres

La matrice de covariance $C$ peut être décomposée comme suit :

$$C = VLV^T$$

où $V = [u_1, u_2, \ldots, u_p]$ est la matrice des vecteurs propres et $L$ est une matrice diagonale des valeurs propres $\lambda_k$.

La contribution de chaque composante principale à la variance totale est donnée par :

$$\frac{\lambda_k}{\sum_{i=1}^{p} \lambda_i}$$

# Interprétation des composantes principales

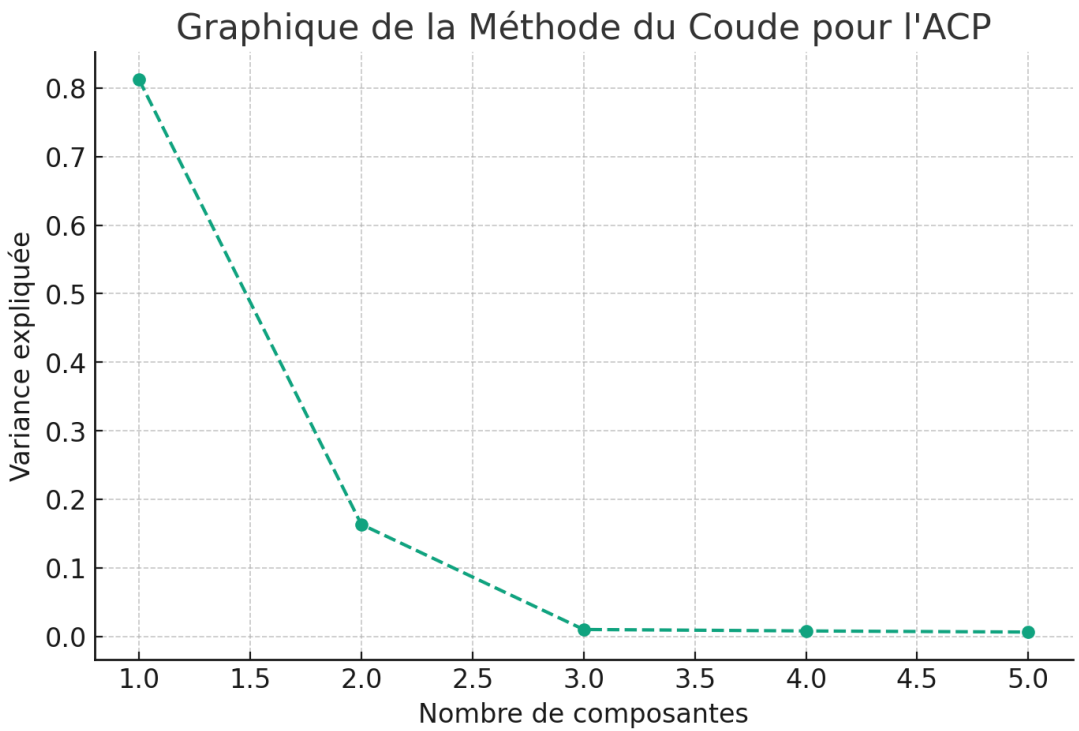
Chaque composante principale est une combinaison linéaire des variables d'origine. L'interprétation des composantes principales peut être réalisée en examinant les coefficients (charges) de ces combinaisons linéaires.

- Plus le coefficient absolu d'une variable est grand dans une composante, plus cette variable contribue à la variance capturée par cette composante.
- La direction et la magnitude des composantes principales peuvent être visualisées sur un biplot.

# Choix du nombre de composantes principales

Le nombre de composantes à retenir est déterminé en fonction du pourcentage de variance totale que l'on souhaite expliquer.

- La méthode du coude (Scree plot) : un graphique montrant la proportion de la variance expliquée en fonction du nombre de composantes.
- Critère de Kaiser : retenir les composantes avec une valeur propre supérieure à 1.



# Réduction de dimensionnalité

L'ACP est souvent utilisée pour réduire la dimensionnalité des données en conservant les composantes qui capturent la majorité de la variance.

- Cela permet une visualisation simplifiée des données en 2D ou 3D.
- La réduction de dimensionnalité peut également aider à améliorer l'efficacité des algorithmes d'apprentissage supervisé.

L'ACP a de nombreuses applications dans différents domaines :

- Analyse de données en biologie, finance, marketing.
- Traitement d'images et de signaux.
- Reconnaissance de motifs et classification.

# Limitations de l'ACP

Bien que l'ACP soit un outil puissant, elle présente certaines limitations :

- Sensibilité aux outliers.
- Difficulté d'interprétation des composantes si les variables sont fortement corrélées.
- La réduction de dimensionnalité peut entraîner une perte d'informations importantes.

# Introduction au deep learning

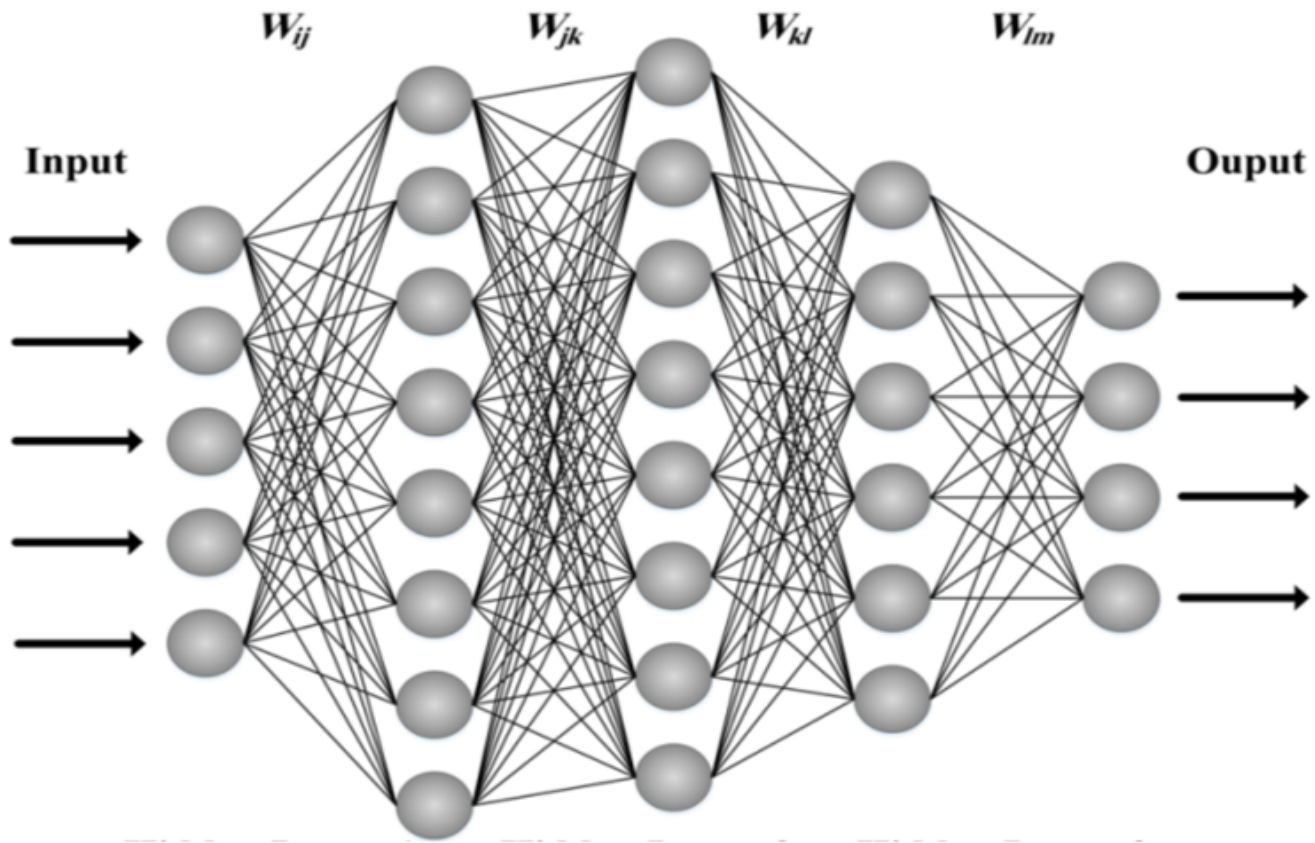
# Introduction aux réseaux de neurones

**Définition** : Les réseaux de neurones sont des modèles computationnels inspirés par le fonctionnement des neurones dans le cerveau humain. Ils sont capables d'apprendre des tâches complexes en modélisant des relations non linéaires entre les entrées et les sorties.

**Caractéristiques** :

- **Extraction automatique des features** : Capacité d'adaptation et d'extraction des features à partir des données sans programmation explicite.
- **Modélisation non linéaire** : Aptitude à capturer des relations complexes dans les données.
- **Flexibilité** : Applicables à un large éventail de tâches et de typologies de données (images, langage naturel, données graphiques, etc.).

# Illustration d'un réseau de neurones classique



# Le neurone artificiel

**Modèle Mathématique** : Chaque neurone artificiel effectue une somme pondérée de ses entrées, ajoute un biais, et passe le résultat à travers une fonction d'activation pour obtenir la sortie.

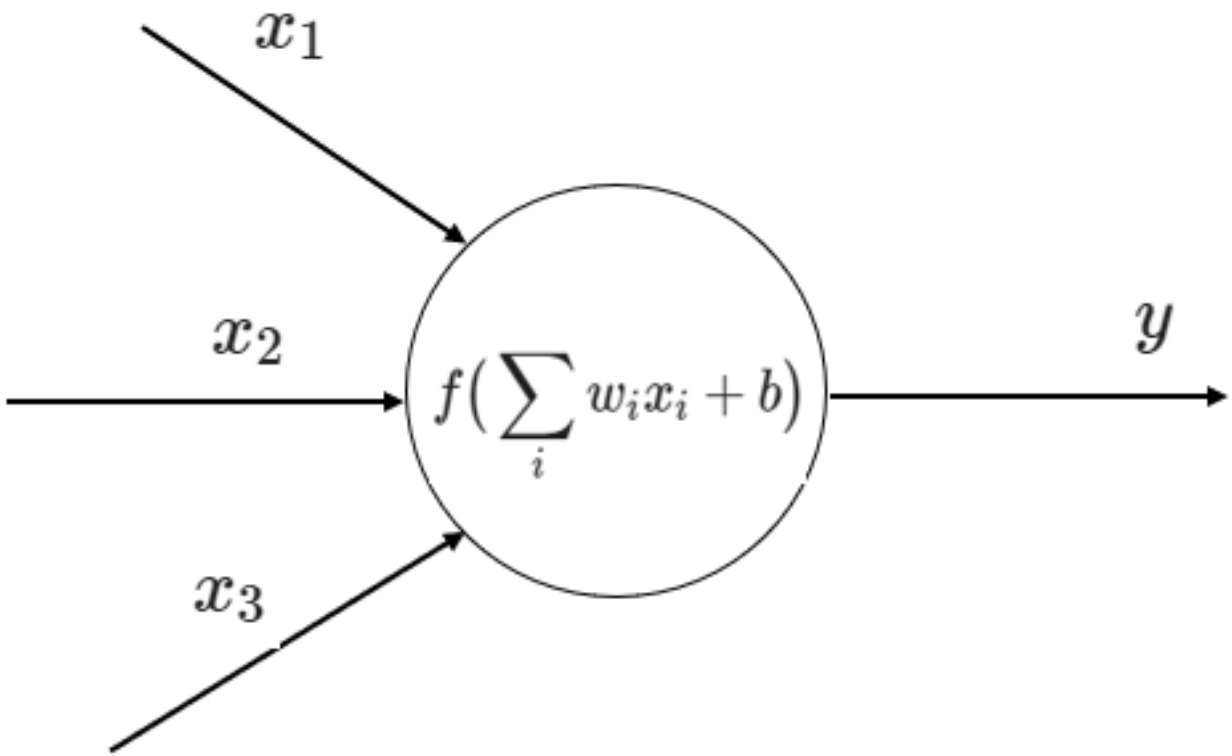
$$z_i = \sum_{j=1}^{m} w_{ij} x_j + b_i$$

$$a_i = f(z_i)$$

où :

- $x_j$ représente l'entrée du neurone,
- $w_{ij}$ est le poids associé à l'entrée $x_j$,
- $b_i$ est le biais du neurone,
- $f$ est la fonction d'activation,
- $z_i$ est le potentiel d'action pré-synaptique,
- $a_i$ est la sortie activée du neurone.

# Illustration d'un neurone aritificiel



# Fonctions d'activation

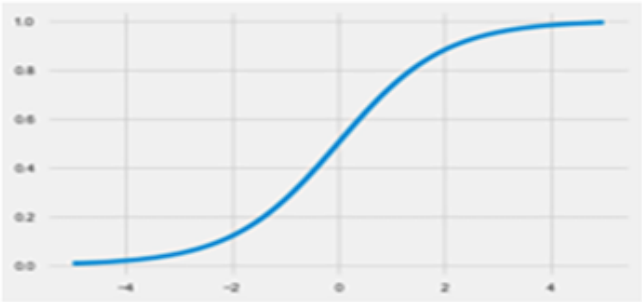
Les fonctions d'activation permettent au modèles d'apprendre des relations plus complexes en cpaturant les non-linéarités dans les données.

- **Sigmoïde** : $\sigma(z) = \frac{1}{1+e^{-z}}$, plage de sortie (0, 1), utilisée pour la probabilité dans la classification binaire.
- **Tangente Hyperbolique (tanh)** : $\tanh(z)$, plage de sortie (-1, 1), version centrée et normalisée de la sigmoïde.
- **ReLU (Unité Linéaire Rectifiée)** : $f(z) = \max(0, z)$, non saturante, favorise la convergence rapide et permet d'éviter le problème de disparition des gradients.
- **Leaky ReLU** : $f(z) = \max(\alpha z, z)$, variante de ReLU qui permet un petit gradient lorsque $z < 0$.
- **Softmax** : Utilisée pour la couche de sortie des problèmes de classification multi-classes.

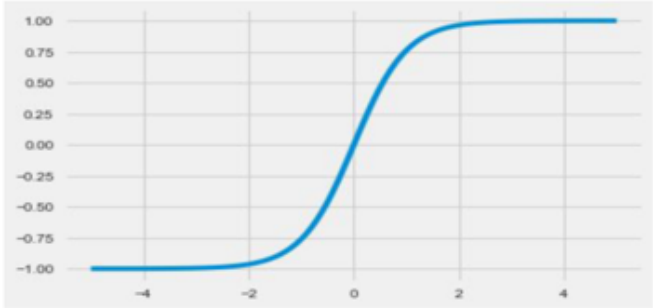
$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

# Illustration des fonctions d'activation

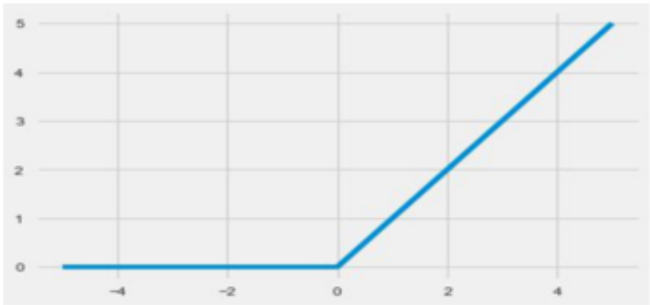
Sigmoïde



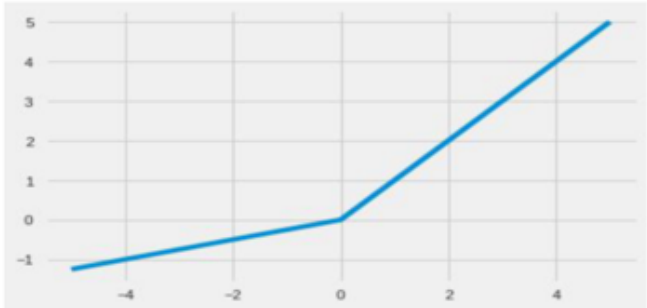

Tanh




ReLU




ReLU paramétrique



# Entraînement d'un réseau de neurones artificiel

**Principe d'Entraînement** : L'entraînement d'un réseau de neurones consiste à ajuster ses poids pour minimiser une fonction de coût qui mesure l'erreur entre les prédictions et les vraies valeurs.

**Descente de gradient stochastique (SGD)** :

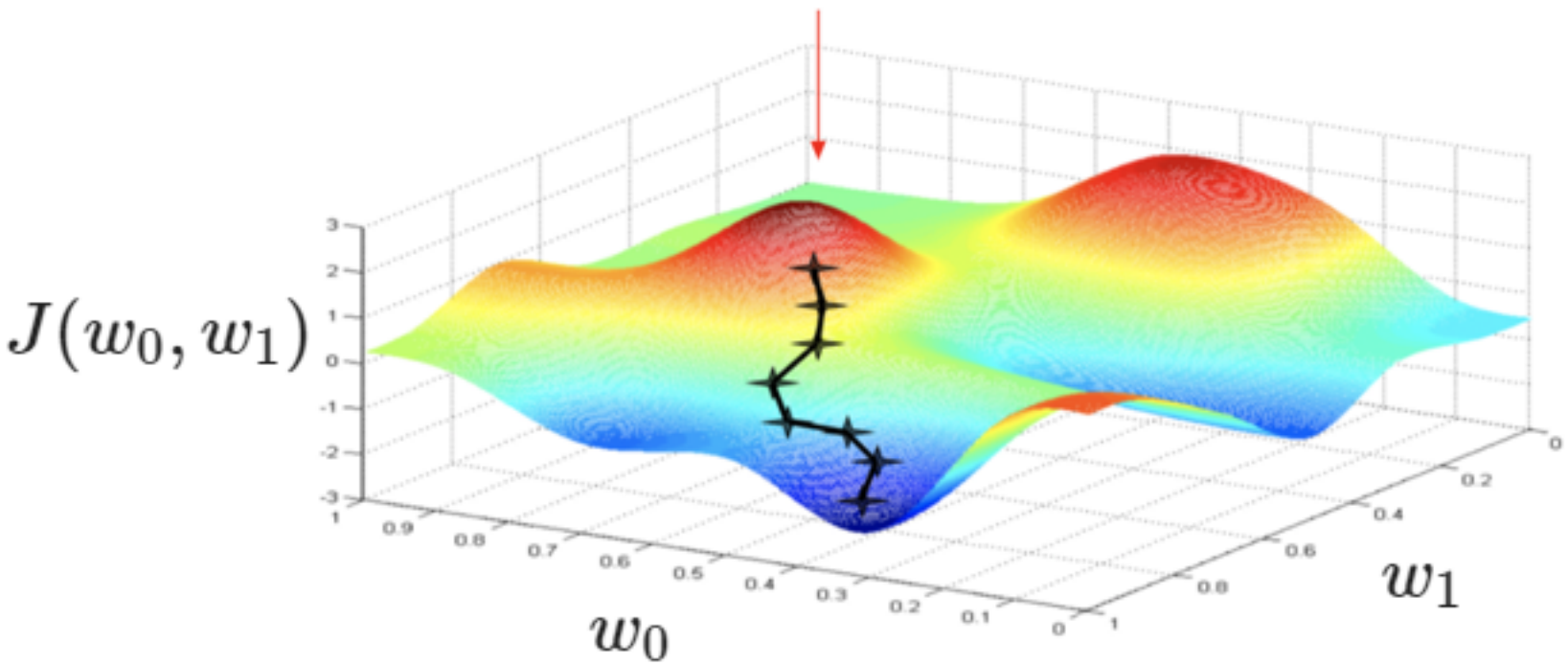
- Méthode d'optimisation utilisée pour mettre à jour les poids du réseau de manière itérative.
- À chaque itération, un sous-ensemble (batch) de données est utilisé pour calculer le gradient de la fonction de coût.
- Les poids sont mis à jour dans la direction opposée du gradient pour réduire l'erreur.

$$w_{new} = w_{old} - \eta \cdot \nabla_w J(w)$$

où :

- $w_{old}$ et $w_{new}$ sont les valeurs des poids avant et après la mise à jour,
- $\eta$ est le taux d'apprentissage,
- $\nabla_w J(w)$ est le gradient de la fonction de coût par rapport aux poids.

# Illustration graphique de la SGD

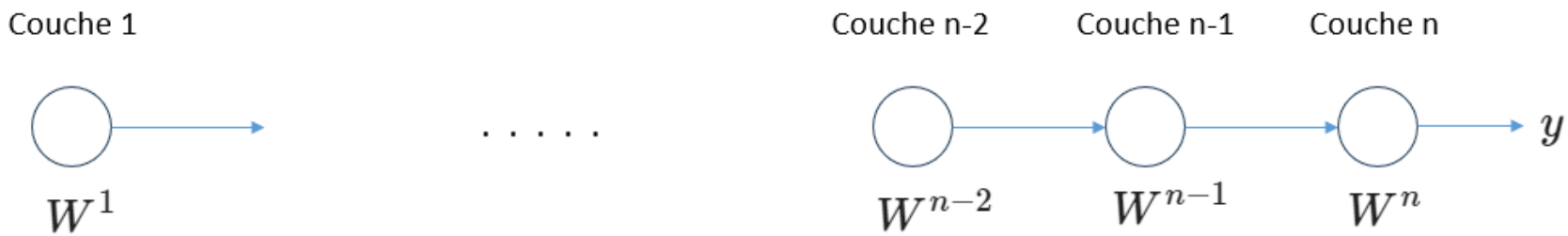


# Entraînement des réseaux de neurones profonds

Dans les réseaux de neurones profonds, l'erreur calculée à la sortie du réseau dépend indirectement des poids des couches cachées. Ce lien indirect rend difficile de savoir comment ajuster ces poids pour réduire l'erreur.

**Implications pour l'Entraînement** :

- **Propagation de l'erreur** : Sans un mécanisme pour propager l'erreur de la sortie vers les couches antérieures, il est impossible de déterminer l'impact de chaque poids sur l'erreur finale.
- **Complexité de l'ajustement des Poids** : Chaque poids dans les couches cachées affecte l'erreur de sortie de manière complexe, nécessitant une méthode précise pour leur ajustement.



# Rétropropagation du gradient (backpropagation)

**Principe de la Rétropropagation** : La rétropropagation est un algorithme clé pour l'entraînement des réseaux de neurones profonds. Elle utilise la règle de la chaîne pour calculer les gradients de la fonction de coût par rapport à tous les poids du réseau.

**Utilisation de la Règle de la Chaîne** : La règle de la chaîne permet de décomposer le gradient de la fonction de coût par rapport à chaque poids en produits de dérivées partielles à travers les couches du réseau.

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial a_k} \cdot \frac{\partial a_k}{\partial z_k} \cdot \frac{\partial z_k}{\partial w_{ij}}$$

où $a_k$ est la sortie activée du neurone, et $z_k$ est son entrée pondérée.

**Étapes de la Rétropropagation** :

1. Propagation avant pour obtenir les activations à travers les couches.
2. Calcul de l'erreur à la couche de sortie.
3. Propagation de l'erreur en arrière en appliquant la règle de la chaîne pour obtenir les gradients.
4. Mise à jour des poids en utilisant les gradients calculés.

# Batch normalization

La Batch normalization est une technique qui consiste à normaliser les entrées de chaque couche à travers un mini-batch. Elle est utilisée pour améliorer la vitesse, la performance et la stabilité des réseaux de neurones profonds.

- Normalise les activations de chaque couche pour avoir une moyenne proche de 0 et une variance proche de 1.
- Réduit le problème des gradients disparus en régulant la distribution des activations.
- Permet d'utiliser des taux d'apprentissage plus élevés et réduit la sensibilité aux initialisations des poids.

**Équations de la batch normalization** :

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mu_{\text{batch}}}{\sqrt{\sigma^2_{\text{batch}} + \epsilon}}$$

$$y^{(k)} = \gamma \hat{x}^{(k)} + \beta$$

où $x^{(k)}$ est l'activation d'entrée, $\mu_{\text{batch}}$ et $\sigma^2_{\text{batch}}$ sont respectivement la moyenne et la variance du mini-batch, et $\gamma$ et $\beta$ sont des paramètres appris.

# Initialisation des poids dans les réseaux de neurones

L'initialisation des poids dans un réseau de neurones a un impact significatif sur la vitesse et la qualité de la convergence lors de l'entraînement. Une mauvaise initialisation peut conduire à des problèmes tels que la disparition ou l'explosion des gradients.

- **Initialisation aléatoire** : Petites valeurs aléatoires pour briser la symétrie et permettre un apprentissage différencié.
- **Initialisation de Xavier/Glorot** : Conçue pour les fonctions d'activation sigmoïde/tanh, ajuste la variance en fonction du nombre de neurones entrants et sortants.
- **Initialisation de He** : Adaptée aux réseaux utilisant ReLU.

**Formules d'Initialisation** :

- Xavier/Glorot : $w \sim \text{Uniform}\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$ ou $\text{Normal}\left(0, \frac{1}{n}\right)$
- He : $w \sim \text{Uniform}\left[-\frac{\sqrt{6}}{\sqrt{n_{\text{in}}+n_{\text{out}}}}, \frac{\sqrt{6}}{\sqrt{n_{\text{in}}+n_{\text{out}}}}\right]$ ou $\text{Normal}\left(0, \frac{2}{n_{\text{in}}}\right)$

où $n$ est le nombre de neurones dans la couche précédente.

# Surapprentissage dans les réseaux de neurones

Le surapprentissage (overfitting) se produit lorsqu'un réseau de neurones apprend trop bien les détails et le bruit des données d'entraînement, au détriment de sa capacité à généraliser sur de nouvelles données.

Le surapprentissage dans les réseaux de neurones peut se produire pour différentes raisons :

- Complexité excessive du modèle
- Manque de diversité dans les données d'entraînement
- Entraînement prolongé

**Stratégies pour atténuer le surapprentissage** :

- **Régularisation** : Techniques de régularisation telles que L1/L2, Dropout, pour limiter la complexité du modèle.
- **Dropout** : "Eteindre" un ensemble de neurones choisis aléatoirement pendant l'entraînement.
- **Early Stopping** : Arrêt de l'entraînement lorsque la performance sur un ensemble de validation cesse de s'améliorer.
- **Augmentation de Données** : Augmenter la variété des données d'entraînement pour améliorer la robustesse du modèle.

# Bibliographie


- Hastie, T., Tibshirani, R., Friedman, J. H., Friedman, J. H. (2009). The elements of statistical learning: data mining, inference, and prediction (Vol. 2, pp. 1-758). New York: springer.
- Russell, S. J. (2010). Artificial intelligence a modern approach. Pearson Education, Inc..
- Ng, A. (2000). CS229 Lecture notes. CS229 Lecture notes, 1(1), 1-3.
- Azencott, C. A. (2019). Introduction au machine learning. Dunod.
- Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep learning. MIT press.
- Vapnik, V. (1999). The nature of statistical learning theory. Springer science business media.
- Cortes, C., Vapnik, V. (1995). Support-vector networks. Machine learning, 20(3), 273-297.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... Amodei, D. (2020). Language models are few-shot learners. Advances in neural information processing systems, 33, 1877-1901.

# Annexe : Multiplicateurs de Lagrange

# Introduction aux Multiplicateurs de Lagrange

**Définition**: Les multiplicateurs de Lagrange sont un outil mathématique qui permet de trouver les optima (maximum ou minimum) d'une fonction sous des contraintes spécifiques. Ils fournissent un cadre puissant pour intégrer les contraintes directement dans le processus d'optimisation.

**Exemples d'Applications**:

- **Économie** : Optimisation des fonctions de coût et de bénéfice, modélisation des contraintes budgétaires.
- **Ingénierie** : Conception optimale sous contraintes physiques ou de ressources.
- **Physique** : Minimisation de l'énergie dans les systèmes physiques, problèmes de mécanique classique.
- **Optimisation des Réseaux** : Gestion du trafic et allocation des ressources dans les réseaux informatiques et de télécommunications.

# Formulation des multiplicateurs de Lagrange

**Principe de Base**: Les multiplicateurs de Lagrange aident à résoudre des problèmes d'optimisation de la forme :

$$\max_x f(x) \text{ sous la contrainte } g(x) = 0$$

où $f$ est la fonction à optimiser et $g$ est la fonction de contrainte.

**Formulation mathématique**: La méthode introduit un multiplicateur $\lambda$, appelé multiplicateur de Lagrange, en considérant la fonction lagrangienne :

$$\mathcal{L}(x, \lambda) = f(x) - \lambda \cdot g(x)$$

L'optimum est trouvé en résolvant le système d'équations :

$$\nabla_x \mathcal{L}(x, \lambda) = 0$$
$$g(x) = 0$$

**Interprétation Géométrique**: Cette méthode peut être interprétée géométriquement comme la recherche de points où les gradients de $f$ et $g$ sont parallèles, signifiant que leur rapport est constant.

# Exemple d'ptimisation avec les multiplicateurs de Lagrange

**Problème**: Maximiser la fonction $f(x, y) = xy$ sous la contrainte $x^2 + y^2 = 1$.

**Fonction Lagrangienne**: La fonction Lagrangienne est définie par :

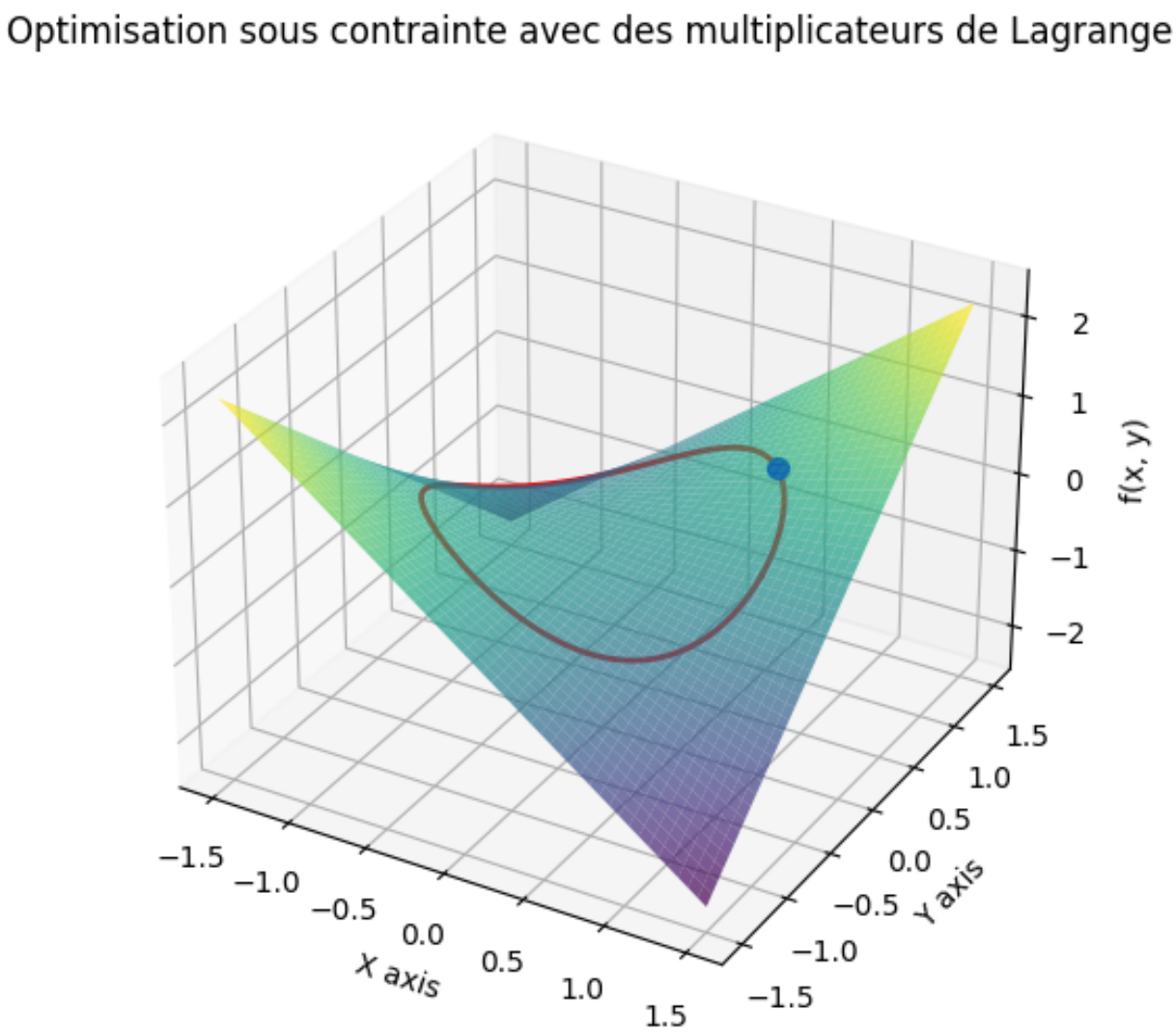
$$\mathcal{L}(x, y, \lambda) = xy - \lambda(x^2 + y^2 - 1)$$

**Résolution**: On trouve les points stationnaires en résolvant le système d'équations suivant :

$$\frac{\partial \mathcal{L}}{\partial x} = y - 2\lambda x = 0$$
$$\frac{\partial \mathcal{L}}{\partial y} = x - 2\lambda y = 0$$
$$\frac{\partial \mathcal{L}}{\partial \lambda} = x^2 + y^2 - 1 = 0$$

**Solution**: En résolvant ce système, on obtient les points d'optimum potentiel et les valeurs correspondantes de $\lambda$. Par exemple, une solution est $(x, y, \lambda) = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, \frac{1}{2}\right)$, correspondant à un maximum local.

# Illustration de l'optimisation sous contraintes avec des multiplicateurs de Lagrange



# Merci de votre attention

redha_moulla@yahoo.fr