Cours d'introduction à la Data Science

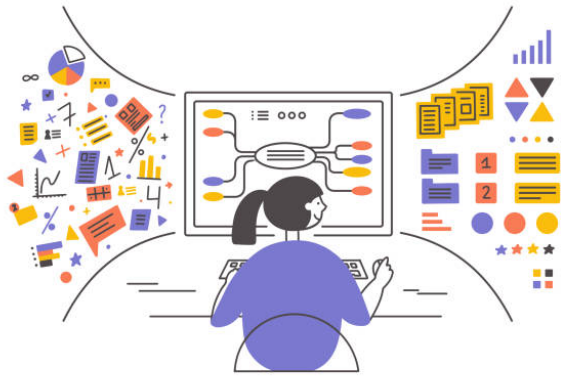
Chargé du cours : Jules DEGILA, PhD, MCF
Assistante : Aurélie KPOZE, PhD

jules.degila@imsp-uac.org, satou.kpoze@imsp-uac.org

16 Décembre 2025

Chapitre 4

Introduction à l'apprentissage automatique



Réaliser, explorer, comparer, évaluer et interpréter les modèles d'apprentissage automatique

Plan du Chapitre 4

1. Définition et concepts fondamentaux de l'apprentissage automatique
2. Le processus général de construction d'un modèle
3. Les principaux types d'apprentissage : supervisé, non supervisé et semi-supervisé
4. Préparation des données pour l'apprentissage automatique
5. Les modèles d'apprentissage supervisé : régression et classification
6. Les modèles d'apprentissage non supervisé : clustering et réduction de dimensions
7. Évaluation et comparaison des modèles
8. Interprétation, limites et bonnes pratiques

Objectifs du Chapitre 4

À la fin du chapitre, vous serez capables de :

- Décrire les concepts fondamentaux de l'apprentissage automatique et ses différentes familles.
- Expliquer les étapes nécessaires pour construire un modèle de machine learning.
- Préparer et transformer un jeu de données pour l'apprentissage (nettoyage, sélection des variables, normalisation...).
- Évaluer la performance d'un modèle à l'aide de métriques adaptées.
- Comparer différents modèles et choisir celui le plus adapté à un problème donné.
- Interpréter les résultats et identifier les limites des modèles utilisés.

Introduction

- Pourquoi l'apprentissage automatique ?
  - Dès les années 1950, les pionniers de l'IA (Turing, Minsky, Samuel) imaginent des machines capables d'apprendre à partir des données.
  - En 1959, Arthur Samuel introduit le terme **machine learning** avec un programme de jeu de dames capable de s'améliorer.
  - Les années 1980–1990 voient l'apparition d'algorithmes majeurs : réseaux de neurones, arbres de décision, SVM.

Introduction

- Les années 2000 marquent l'essor du Big Data et de la puissance de calcul, propulsant l'apprentissage automatique.
- Aujourd'hui, il est au cœur de nombreuses applications : recommandation, vision, langage, santé, finance, transports intelligents, etc.
- Avant : les machines suivaient uniquement des règles explicites, ce qui limitait fortement leur capacité à traiter des situations complexes.
- Avec l'IA : les modèles apprennent à partir des données en s'inspirant du fonctionnement du cerveau humain (réseaux de neurones), ce qui leur permet de découvrir des patterns et de s'adapter.

Introduction

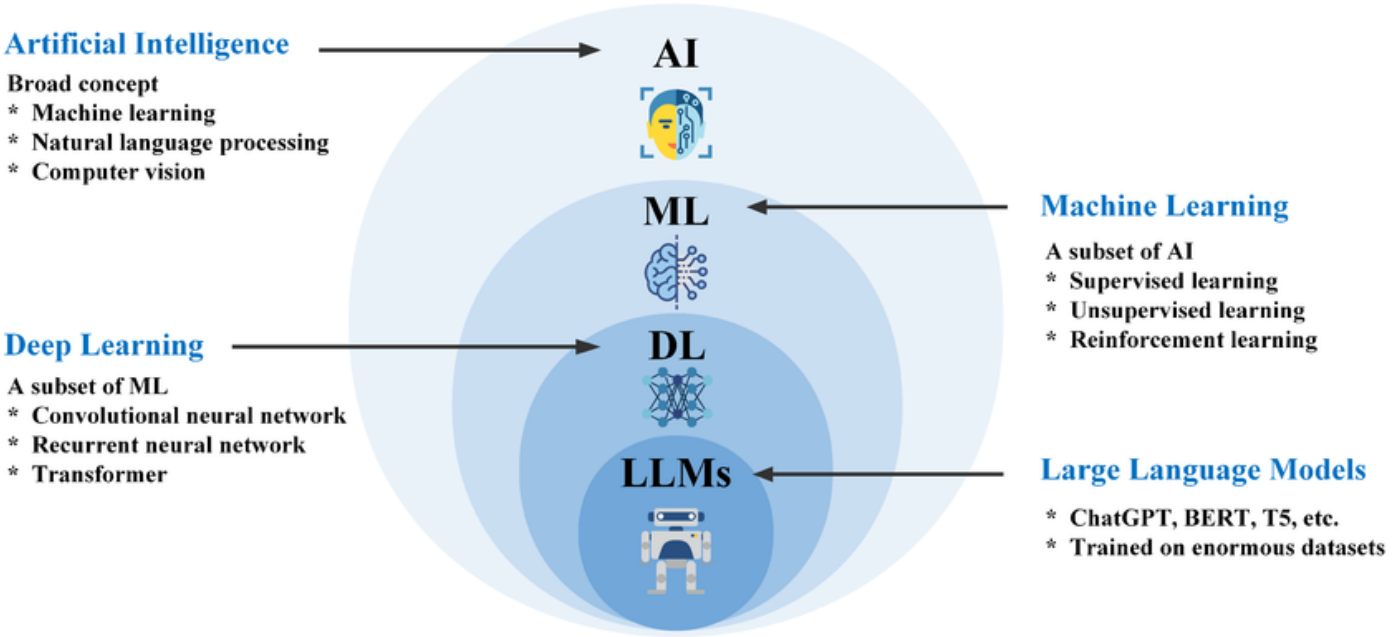


Figure: *

Image source : Yang, Zheng et al. (2025). *AI-Driven Safety and Security for UAVs*.

Définition

- **Définition générale :** L'apprentissage automatique est un ensemble de méthodes permettant à une machine d'apprendre automatiquement à partir des données afin de réaliser une tâche sans être explicitement programmée.

- **Exemple :** À partir d'un historique d'emails (spam / non-spam), un modèle apprend à prédire si un nouvel email est un spam. Aucune règle si... alors... n'est écrite : c'est le modèle qui les découvre.

- **Définition mathématique :** Étant donné un ensemble de données $D = \{(x_i, y_i)\}_{i=1}^{n}$, trouver une fonction f dans une famille de fonctions $\mathcal{F}$ telle que :

$$f = \arg\min_{g \in \mathcal{F}} \mathcal{L}(g(x_i), y_i)$$

où $\mathcal{L}$ est une fonction de perte mesurant l'erreur entre la prédiction et la vérité.

Modèles en Machine Learning

Pourquoi utiliser des modèles ?

- Les données seules ne suffisent pas à faire des prédictions
- Un modèle permet de **capturer des régularités** dans les données
- Il généralise à de **nouvelles données non vues**

Qu'est-ce qu'un modèle ?

- Une **fonction mathématique** paramétrée
- Apprise à partir de données d'entraînement
- Qui relie des **entrées** (features) à des **sorties** (prédictions)

Exemple :

$$y = f(x; \theta)$$

- x : données d'entrée
- θ : paramètres du modèle
- y : prédiction

Concepts Fondamentaux

Types de données et représentation (features, X et y)

- **Types de données**
  - Numériques : revenus, âge, température...
  - Catégorielles : couleur, pays, type de produit...
  - Textuelles : documents, commentaires, tweets...
  - Images, sons ou signaux : photos, audio, capteurs...

- **Représentation des données**
  - Une observation = un individu, un exemple, un cas.
  - Une feature (variable explicative) = une caractéristique mesurée.
  - X : matrice des features (dimensions $n \times p$).
  - y : variable cible (label), connue uniquement en apprentissage supervisé.

Tâches d'apprentissage : supervisé / non-supervisé

- **Apprentissage supervisé**
  - Les données sont annotées : on connaît la cible y.
  - Objectif : apprendre une fonction $f : X \to y$.
  - Exemples :
    - Classification : spam / non-spam, diagnostic médical.
    - Régression : prédiction de prix, estimation d'une quantité.

- **Apprentissage non supervisé**
  - Les données **ne** sont **pas annotées** : pas de cible y.
  - Objectif : découvrir une structure cachée dans les données.
  - Exemples :
    - Clustering : grouper des clients par comportement.
    - Réduction de dimension : simplifier des données complexes (ACP).

Fonctions de perte

- **Définition** Une fonction de perte mesure l'écart entre la prédiction du modèle $\hat{y}$ et la valeur réelle y.

- **Objectif** Trouver les paramètres du modèle qui **minimisent** cette erreur :

$$\theta^* = \arg\min_{\theta} \ \mathcal{L}(y, \hat{y})$$

- **Rôle clé**
  - Guide l'apprentissage.
  - Permet la comparaison de modèles.
  - Impacte la vitesse et la stabilité de l'optimisation.

Régression

- **Définition** Tâche d'apprentissage supervisé où la variable cible y est numérique et continue.
- **Objectif** Apprendre une fonction $f : X \to \mathbb{R}$ pour prédire une valeur réelle.
- Pour y parvenir, le modèle minimise l'erreur entre la valeur prédite $\hat{y}$ et la valeur réelle y à l'aide d'une fonction de perte.
- **Exemples d'applications**
  - Prédiction du prix d'une maison
  - Prévisions météorologiques
  - Estimation d'un score ou d'une quantité
- **Modèles courants**
  - Régression linéaire, polynomiale
  - Arbres de décision, forêts aléatoires
  - SVM régressif, réseaux de neurones

Fonctions de perte pour la régression

- **Erreur quadratique moyenne (MSE)**

$$\mathcal{L}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Sensible aux valeurs extrêmes (outliers).

- **Erreur absolue moyenne (MAE)**

$$\mathcal{L}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

Plus robuste aux outliers.

- **Erreur quadratique root (RMSE)**

$$\text{RMSE} = \sqrt{\text{MSE}}$$

Interprétation plus intuitive (unités de y).

Principaux algorithmes de régression (1/2)

- **Régression linéaire**
  - Modèle simple et interprétable.
  - Paramètres : coefficients, biais.
- **Régression polynomiale**
  - Étend la régression linéaire avec des termes non linéaires.
  - Paramètres : coefficients des termes polynomiaux.
- **k-NN régressif**
  - Prédit la moyenne (ou médiane) des voisins proches.
  - Paramètres : k, métrique de distance.

Principaux algorithmes de régression (2/2)

- **Arbres de décision / Forêts aléatoires**
  - Modèles flexibles captant les non-linéarités.
  - Paramètres : profondeur, feuilles, nombre d'arbres.
- **SVM pour la régression (SVR)**
  - Trouve une fonction plate dans un tube de tolérance ϵ.
  - Paramètres : ϵ, C, noyau.
- **Réseaux de neurones**
  - Adaptés aux relations complexes et données volumineuses.
  - Paramètres : couches, poids, biais, activation.

Classification

- **Définition** Tâche d'apprentissage supervisé où la variable cible y prend un nombre fini de classes.
- **Objectif** Apprendre une fonction $f : X \to \{1, 2, \ldots, K\}$ pour prédire la classe d'un nouvel exemple.
- **Exemples d'applications**
  - Détection de spam (spam / non spam)
  - Diagnostic médical (malade / non malade)
  - Reconnaissance d'images (chats, chiens...)
- **Modèles courants**
  - k-NN, régression logistique, arbres de décision, SVM
  - Forêts aléatoires, réseaux de neurones

Fonctionnement de la classification

- Les modèles de classification apprennent des **frontières de décision** qui séparent une classe d'une autre, en utilisant des fonctions d'activation (Classification binaire, multi-classes)

- **Fonctions d'activation**:
  - **Sigmoïde** : utilisée en classification binaire pour convertir un score en une probabilité entre 0 et 1.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

  - **Softmax** : utilisée en classification multi-classes pour transformer les scores en probabilités qui somment à 1.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

- **Décision finale** La classe ayant la **plus grande probabilité** est choisie comme prédiction.

Fonctions de perte pour la classification

- **Entropie croisée binaire (Binary Cross-Entropy / Log Loss)** Utilisée en classification binaire. Mesure l'erreur entre la probabilité prédite et la classe réelle (0 ou 1).
- **Entropie croisée catégorielle (Categorical Cross-Entropy)** Utilisée en classification multi-classes. Compare la distribution de probabilités prédite avec la classe réelle encodée en one-hot.
- **Entropie croisée catégorielle clairsemée (Sparse Categorical Cross-Entropy)** Variante de l'entropie croisée multi-classes qui utilise directement les labels sous forme d'entiers (sans one-hot encoding).
- **Hinge Loss** Utilisée en SVM. Pénalise les prédictions qui se trouvent du mauvais côté de la marge de séparation.

Principaux algorithmes de classification (1/2)

- **k-plus proches voisins (k-NN)**
  - Classe selon les k exemples les plus proches.
  - Paramètres : k, métrique de distance.
- **Régression logistique**
  - Modèle linéaire pour prédire une probabilité de classe.
  - Paramètres : poids, biais.
- **Arbres de décision**
  - Divise les données selon les variables les plus informatives.
  - Paramètres : seuils, profondeur, feuilles.

Principaux algorithmes de classification (2/2)

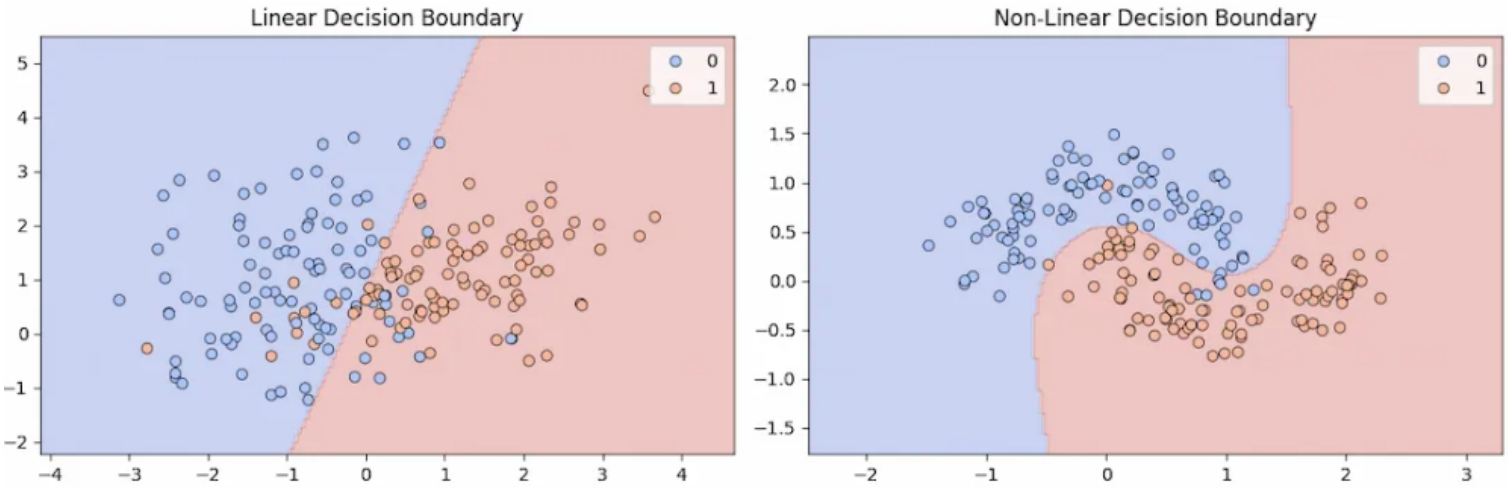
- **Forêts aléatoires (Random Forest)**
  - Ensemble d'arbres entraînés sur des sous-échantillons.
  - Paramètres : nombre d'arbres, profondeur, sous-échantillonnage.
- **Machines à vecteurs de support (SVM)**
  - Trouve la frontière qui maximise la marge entre classes.
  - Paramètres : noyau, C, γ.
- **Réseaux de neurones**
  - Modèles composés de couches et de neurones.
  - Paramètres : poids, biais, nombre de couches, fonctions d'activation.

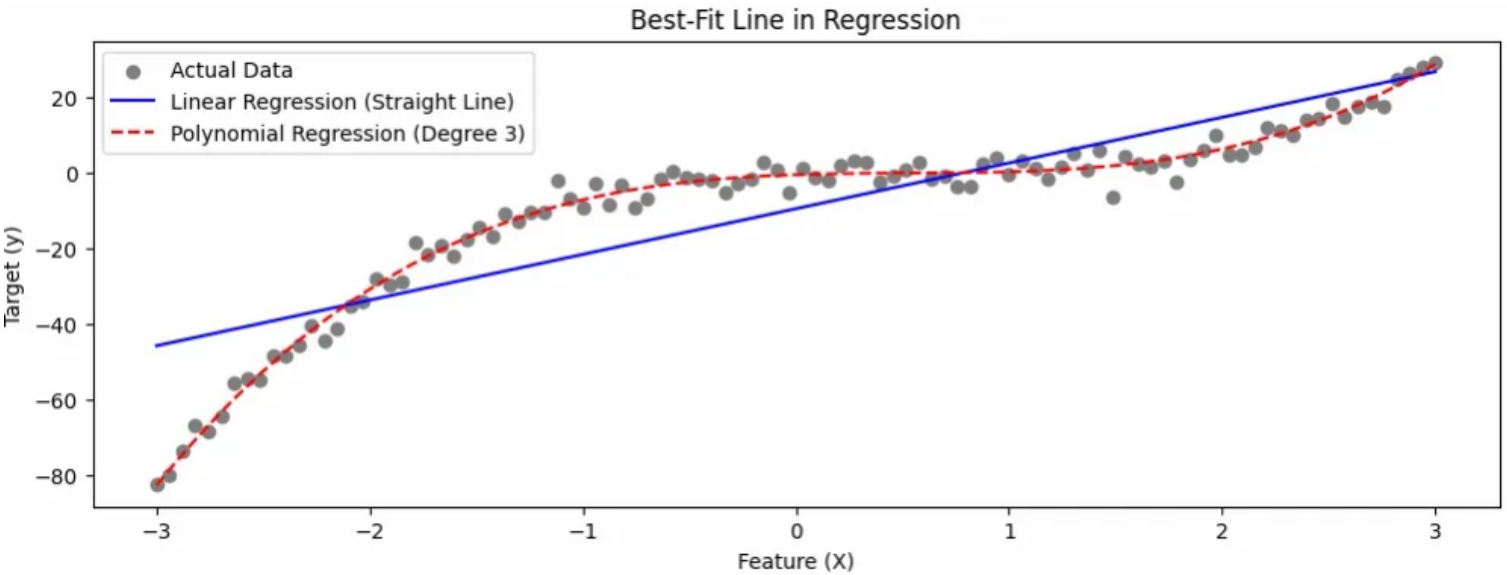
Decision Boundary vs Best-Fit Line

- **Ligne de meilleure approximation (Régression)**
  - Le modèle cherche une ligne ou une courbe qui s'ajuste au mieux aux données continues.
  - L'objectif est de minimiser l'erreur de prédiction (MSE, MAE...).
  - Utilisé lorsque la variable cible est numérique.

- **Frontière de décision (Classification)**
  - Le modèle apprend une frontière qui sépare les observations en classes.
  - Cette frontière peut être linéaire, courbe, ou très complexe selon l'algorithme.
  - Utilisé lorsque la variable cible est catégorielle.

Decision Boundary vs Best-Fit Line





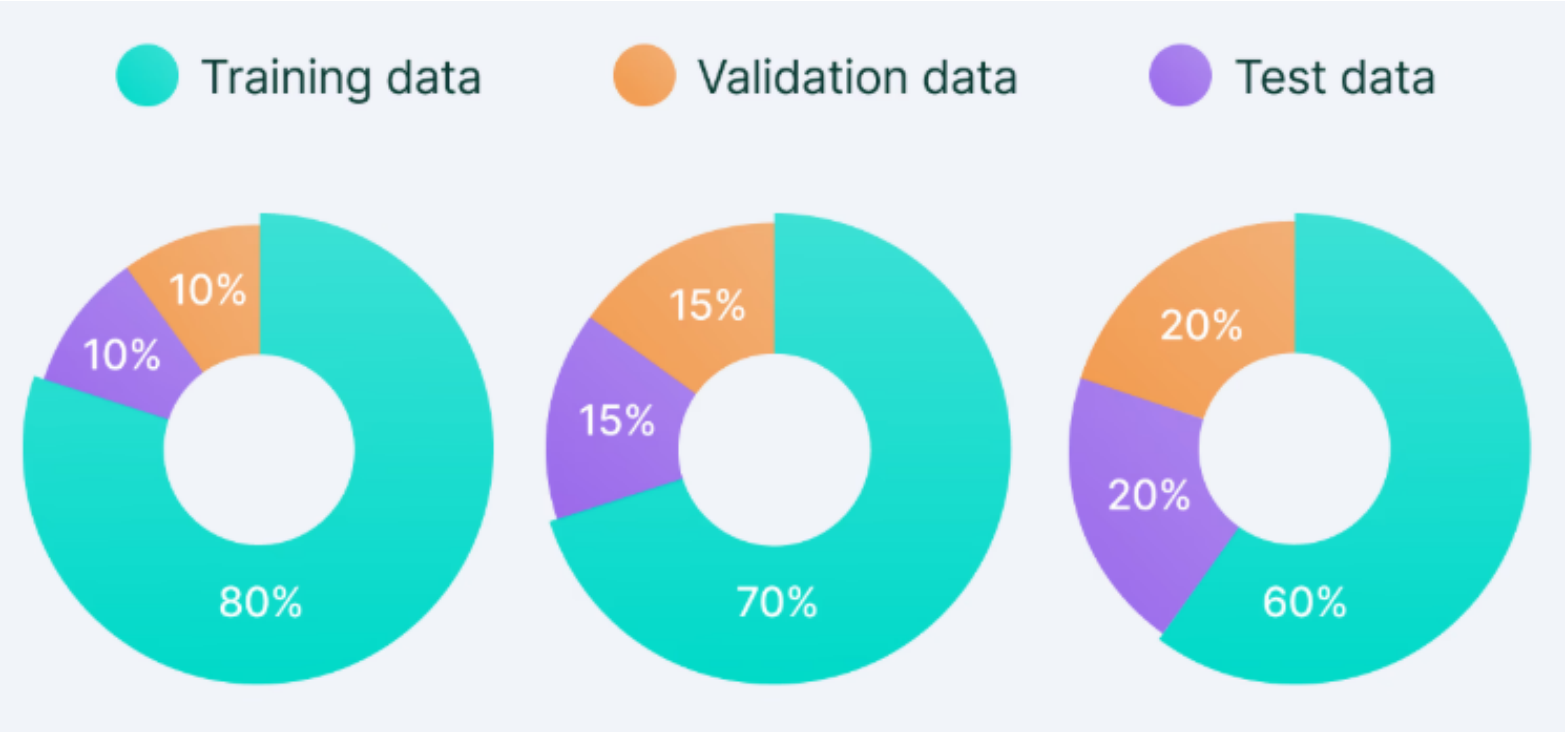
Paramètres d'un modèle

- **Définition** Les paramètres sont les **valeurs internes apprises automatiquement** par le modèle pendant l'entraînement.
- **Rôle** Ils déterminent la façon dont le modèle transforme les entrées en prédictions.
- **Exemples**
  - Régression linéaire : coefficients $w_1, w_2, \ldots$ et biais b
  - Réseaux de neurones : poids et biais de chaque neurone
  - Arbres de décision : seuils de séparation dans les noeuds
  - SVM : vecteurs de support et coefficients associés
- **Caractéristiques**
  - Appris automatiquement par optimisation
  - Dépendent des données
  - Changent à chaque entraînement

Modèles et paramètres

Tâche	Modèles courants	Paramètres appris
Régression	Régression linéaire, régression ridge/lasso, forêts aléatoires, réseaux de neurones	Coefficients (poids), biais, profondeur/feuilles (arbres), poids des neurones
Classification	k-NN, régression logistique, SVM, arbres de décision, forêts aléatoires, réseaux de neurones	Poids, biais, vecteurs de support, seuils/ noeuds des arbres, poids des réseaux
Clustering	k-means, DBSCAN, clustering hiérarchique	Centroides, structure de graphe, distances, dendrogramme

Objectif d'un modèle de Machine Learning



Objectif d'un modèle de Machine Learning

- Un bon modèle doit :
  - apprendre efficacement les motifs des données d'entraînement ;
  - se généraliser à de nouvelles données jamais vues ;
  - éviter de mémoriser les données (sur-apprentissage) ;
  - éviter de trop simplifier les relations (sous-apprentissage).

- Pour évaluer cela, on mesure les performances :
  - sur les données d'entraînement ;
  - sur un jeu de validation ou de test.

Biais et Variance

- **Biais** : erreur due à un modèle trop simple.
  - Le modèle ignore des motifs importants.
  - Mène au **sous-apprentissage**.
  - Exemple : régression linéaire sur une relation non linéaire.

- **Variance** : erreur due à un modèle trop sensible aux données.
  - Le modèle apprend aussi le bruit et les outliers.
  - Mène au **sur-apprentissage**.
  - Exemple : polynôme très complexe suivant tous les points.

Trouver un équilibre entre le biais et la variance

Sur-apprentissage (Overfitting)

- **Définition** Le sur-apprentissage survient lorsque le modèle apprend **trop bien** les données d'entraînement, y compris le bruit, les erreurs et les détails spécifiques.

- **Conséquence** Excellentes performances sur l'entraînement, mais mauvaises performances sur de nouvelles données.

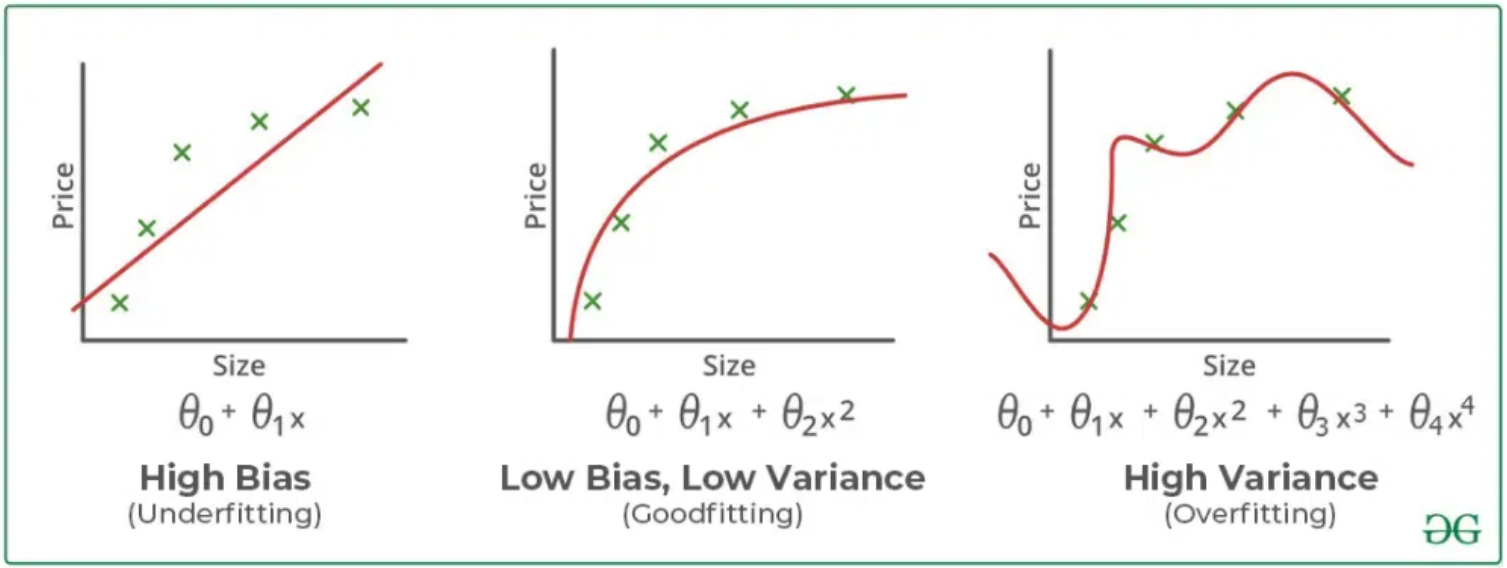
- **Signes d'overfitting**
  - Grande différence entre erreur d'entraînement et erreur de test.
  - Modèle très complexe (trop de paramètres, arbres trop profonds...).

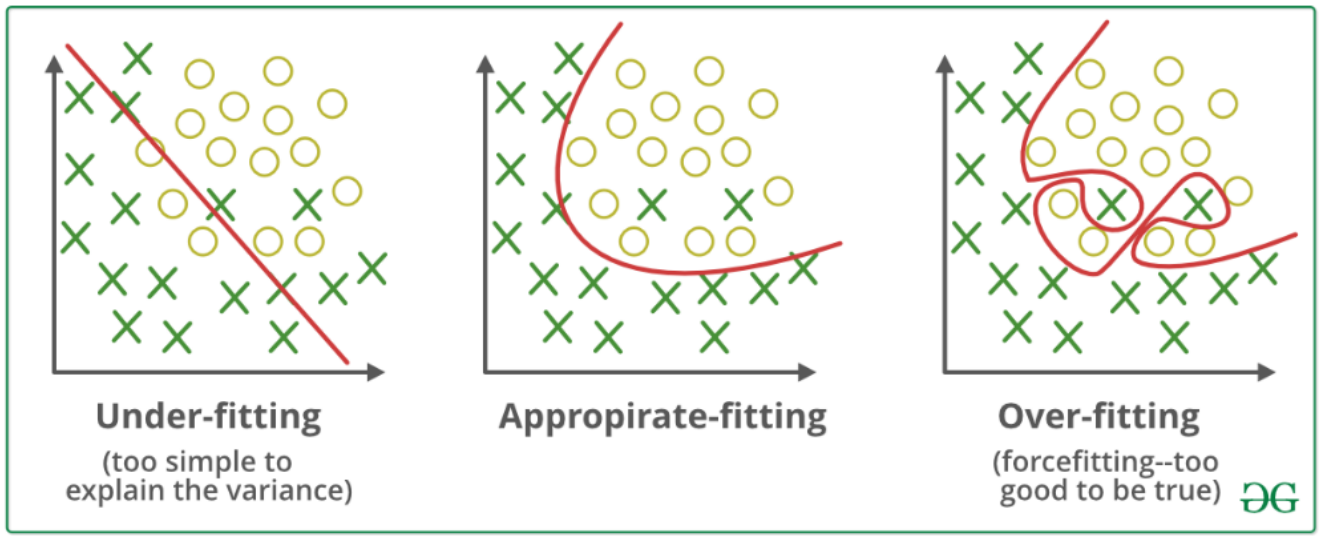
- **Solutions possibles**
  - Régularisation (Lasso, Ridge, Dropout...)
  - Réduire la complexité du modèle
  - Collecter plus de données
  - Utiliser la validation croisée

Sous-apprentissage (Underfitting)

- **Définition** Le sous-apprentissage survient lorsque le modèle est **trop simple** pour capturer les relations dans les données.
- **Conséquence** Mauvaises performances :
  - sur les données d'entraînement
  - et sur les données de test.
- **Signes d'underfitting**
  - Erreurs élevées dès l'entraînement
  - Modèle trop rigide (ex. régression linéaire pour un phénomène non linéaire)
- **Solutions possibles**
  - Utiliser un modèle plus complexe
  - Ajouter des features pertinentes
  - Réduire la régularisation si elle est trop forte

Overfitting vs Underfitting





Régularisation

- **Définition** La régularisation est un ensemble de techniques qui visent à **réduire le sur-apprentissage** en limitant la complexité d'un modèle.
- **Objectif** Améliorer la **généralisation** du modèle sur de nouvelles données.
- **Principe** Ajouter une **pénalité** dans la fonction de coût pour :
  - empêcher les poids de devenir trop grands ;
  - simplifier le modèle ;
  - éviter d'apprendre le bruit.

Régularisation L2 (Ridge)

- **Principe** Ajoute une pénalité basée sur la **norme L2** des coefficients.
- **Fonction de coût**

$$\mathcal{L}(w) = \frac{1}{n}\sum(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} w_j^2$$

- **Effets**
  - Réduit la taille des coefficients ;
  - n'annule pas les variables ;
  - modèle plus stable et plus lisse.
- **Utilisation**
  - Données multicolinéaires ;
  - modèles sensibles aux fluctuations des données.

Régularisation L1 (Lasso)

- **Principe** Ajoute une pénalité basée sur la **norme L1** des coefficients.
- **Fonction de coût**

$$\mathcal{L}(w) = \frac{1}{n}\sum(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |w_j|$$

- **Effets**
  - Certains coefficients deviennent **exactement nuls** ;
  - permet la **sélection automatique de variables**.
- **Utilisation**
  - Modèles avec beaucoup de features ;
  - besoin de parcimonie.

Elastic Net

- **Principe** Combine les pénalités L1 (Lasso) et L2 (Ridge).
- **Fonction de coût**

$$\mathcal{L}(w) = \frac{1}{n}\sum(y_i - \hat{y}_i)^2 + \lambda_1 \sum |w_j| + \lambda_2 \sum w_j^2$$

- **Effets**
  - Sélection de variables (L1) ;
  - Stabilisation des coefficients (L2).
- **Utilisation**
  - Problèmes avec beaucoup de variables corrélées ;
  - Défi “p ¿ n” (plus de variables que d’observations).

Early Stopping

- **Principe** Arrêter l'entraînement avant que le modèle ne commence à trop apprendre le bruit.
- **Observation clé** L'erreur de validation **diminue puis remonte** : on s'arrête à son minimum.

- **Avantages**
  - Simple et très efficace ;
  - évite le sur-apprentissage ;
  - ne nécessite pas de modifier la fonction de coût.

Dropout

- **Principe** Durant l'entraînement, certains neurones sont **désactivés au hasard**.
- **Effets**
  - Le réseau devient plus robuste ;
  - empêche les neurones de trop dépendre les uns des autres ;
  - réduit fortement le sur-apprentissage.
- **Utilisation**
  - Modèles profonds (Deep Learning) ;
  - classification d'images, texte, etc.

Hyperparamètres d'un modèle

- **Définition** Les hyperparamètres sont des **valeurs fixées avant l'entraînement** du modèle. Ils contrôlent la structure, la complexité ou la manière dont le modèle apprend.
- **Rôle** Ils influencent fortement les performances du modèle et son risque de sur/sous-apprentissage.
- **Exemples**
  - Arbres de décision : profondeur maximale, nombre minimal d'échantillons par feuille
  - k-NN : nombre de voisins k
  - Réseaux de neurones : taux d'apprentissage, nombre de couches, taille des couches
  - SVM : C, noyau, γ
  - Lasso/Ridge : coefficient de régularisation λ

Hyperparamètres d'un modèle

- Choisis **avant** l'entraînement
- Ne sont **pas appris** par le modèle
- Ajustement des hyperparamètres (Hyperparameter Tuning) :
  - **Validation croisée** : tester plusieurs valeurs d'hyperparamètres sur différents sous-jeux (folds) pour choisir ceux qui généralisent le mieux.
  - **Grid search** : exploration systématique d'une grille de combinaisons d'hyperparamètres.
  - **Random search** : sélection aléatoire d'hyperparamètres dans un espace donné ; souvent plus efficace que Grid Search.
  - **Optimisation bayésienne** (Bayesian Optimization) : méthode intelligente qui modélise la performance du modèle et choisit les prochains hyperparamètres à tester de manière guidée (basée sur une fonction d'acquisition). Rapide et efficace pour des modèles coûteux à entraîner.

Evaluation

- Un modèle performant doit :
  - apprendre correctement les motifs présents dans les données d'entraînement ;
  - se généraliser à de nouvelles données jamais vues.
- Une bonne évaluation permet de détecter :
  - le sur-apprentissage (overfitting),
  - le sous-apprentissage (underfitting),
  - les mauvais choix de modèles ou d'hyperparamètres.
- On évalue un modèle sur :
  - un jeu d'entraînement,
  - un jeu de validation,
  - un jeu de test.

Evaluation

- **Généralisation** : capacité d'un modèle à fournir de bonnes prédictions sur des données nouvelles.
- Un modèle généralise bien s'il :
  - ne mémorise pas le bruit (évite l'overfitting),
  - ne sous-exploite pas les informations (évite l'underfitting),
  - obtient des performances cohérentes entre train et test.
- Essentielle pour prédire en conditions réelles.

Train, Validation, Test

- **Entraînement (Train)** Sert à ajuster les paramètres du modèle.
- **Validation** Sert à :
  - ajuster les hyperparamètres ;
  - détecter le sur-apprentissage ;
  - comparer différents modèles.
- **Test** Sert uniquement à mesurer la performance finale et la généralisation.

Validation croisée

- Technique pour mieux estimer la performance réelle d'un modèle.
- **Principe du k-fold** :
  - On divise les données en k sous-ensembles ;
  - le modèle est entraîné k fois ;
  - chaque sous-ensemble sert une fois de validation.
- Avantages :
  - meilleure estimation de la généralisation ;
  - réduit le risque de sur-apprentissage ;
  - exploite mieux les données disponibles.

Métriques pour la classification

- **Vrai Positif (VP)** Le modèle prédit **positif** et la réalité est **positive**. Exemple : prédire malade pour une personne réellement malade.
- **Faux Positif (FP)** Le modèle prédit **positif** alors que la réalité est **négative**. Exemple : prédire malade pour une personne en bonne santé.
- **Vrai Négatif (VN)** Le modèle prédit **négatif** et la réalité est **négative**. Exemple : prédire sain pour une personne réellement saine.
- **Faux Négatif (FN)** Le modèle prédit **négatif** alors que la réalité est **positive**. Exemple : prédire sain pour une personne malade.
- Ne peuvent être définies que pour des données dont on connaît la classe réelle (appelée **ground truth**).

Qu'est-ce que la Ground Truth ?

- La **ground truth** (vérité terrain) correspond aux **vraies étiquettes** ou aux **valeurs réelles** associées aux données. Ce sont les réponses correctes servant de référence pour évaluer un modèle.

- **Origine de la ground truth**
  - Annotation humaine experte (médecine, vision, NLP)
  - Mesures physiques (capteurs, laboratoires)
  - Données historiques validées
  - Protocoles de collecte fiables

- **Rôle essentiel**
  - Permet de comparer les prédictions du modèle à la vérité
  - Indispensable pour calculer : VP, FP, VN, FN, A etc.
  - Sans ground truth : impossible d'évaluer correctement un modèle supervisé

Métriques pour la classification

- **Exactitude (Accuracy)**

$$\frac{\text{Nombre de prédictions correctes}}{\text{Nombre total d'exemples}}$$

- **Précision (Precision)**

$$\frac{VP}{VP + FP}$$

- **Rappel (Recall)**

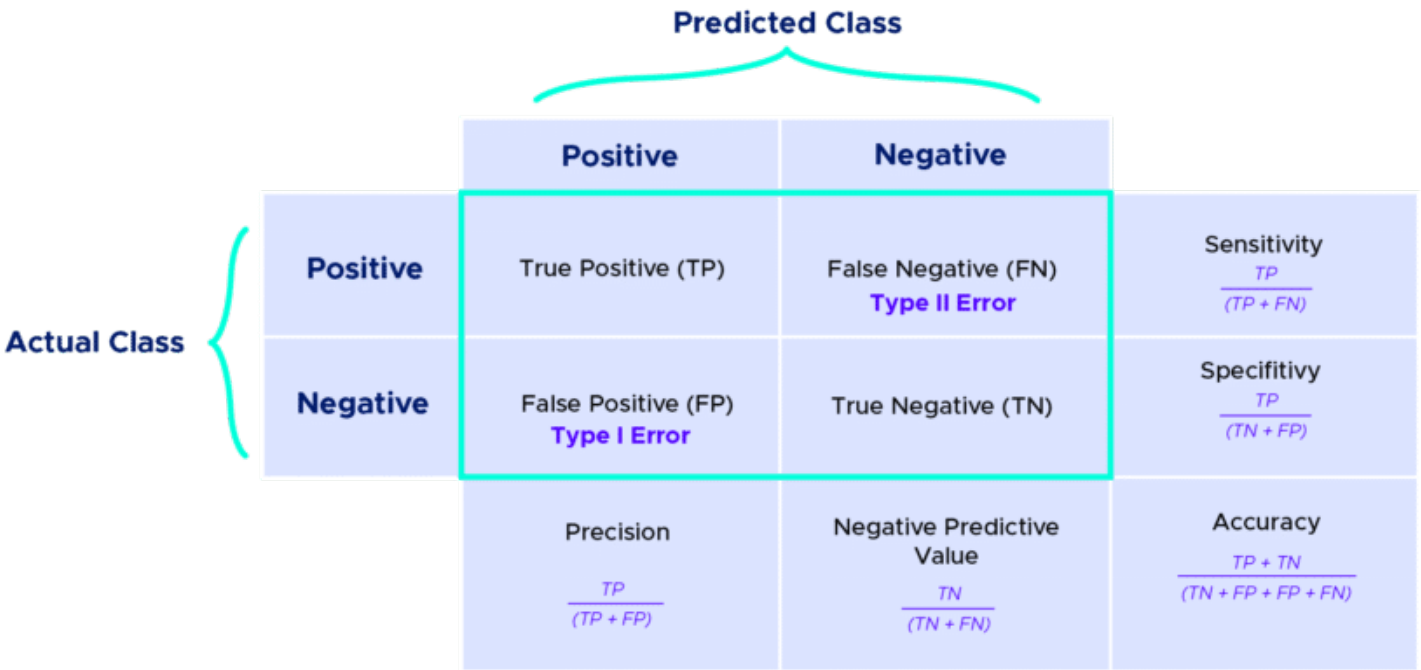
$$\frac{VP}{VP + FN}$$

- **F1-score**

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

Matrice de confusion

- Représente les performances du modèle classe par classe.
- Permet d'identifier :
  - les classes bien reconnues ;
  - les classes confondues ;
  - les biais éventuels.



Modèle d'apprentissage supervisé

Régression Logistique (Classification)

Objectif : comprendre l'intuition, la formule, la décision, l'apprentissage et l'évaluation

Régression Logistique : contexte (Supervisé)

Type d'apprentissage : supervisé (on connaît la vérité terrain y)

But : prédire une classe à partir de variables explicatives X

Cas typique : classification binaire

$$y \in \{0, 1\}$$

- $y = 1$: événement d'intérêt (ex: *spam*, *malade*, *fraude*)
- $y = 0$: non-événement (ex: *non-spam*, *sain*, *normal*)

Exemples d'applications :

- Email : spam vs non-spam
- Santé : malade vs sain
- Banque : fraude vs non-fraude

Pourquoi pas une simple régression linéaire ?

La régression linéaire produit des valeurs sur $\mathbb{R}$:

$$\hat{y} = w^T x + b \quad \Rightarrow \quad \hat{y} \in (-\infty, +\infty)$$

Mais en classification, on veut :

- une probabilité entre 0 et 1 ;
- une décision de classe (0 ou 1).

Idée : garder le score linéaire $z = w^T x + b$, puis le transformer en probabilité avec une fonction d'activation (sigmoïde).

Le modèle : score linéaire + sigmoïde

1) Score linéaire :

$$z = w^T x + b = w_1 x_1 + w_2 x_2 + \cdots + w_p x_p + b$$

2) Transformation en probabilité (Sigmoïde) :

$$\hat{p} = P(y = 1|x) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Interprétation :

- si z est grand et positif $\Rightarrow \hat{p} \approx 1$ (classe 1 très probable)
- si z est grand et négatif $\Rightarrow \hat{p} \approx 0$ (classe 0 très probable)
- si $z = 0 \Rightarrow \hat{p} = 0.5$

Règle de décision (classification)

Une fois la probabilité prédite $\hat{p}$ obtenue :

Décision avec un seuil (par défaut 0.5) :

$$\hat{y} = \begin{cases} 1 & \text{si } \hat{p} \geq 0.5 \\ 0 & \text{si } \hat{p} < 0.5 \end{cases}$$

Remarque importante : le seuil peut changer selon le contexte.

- En santé : on veut souvent un rappel élevé $\Rightarrow$ seuil plus bas
- En spam : on veut éviter de bloquer des emails normaux $\Rightarrow$ seuil plus haut

Frontière de décision (intuition géométrique)

La frontière correspond aux points où le modèle hésite :

$$\hat{p} = 0.5 \quad \Leftrightarrow \quad z = w^T x + b = 0$$

Donc la frontière est :

- une droite en 2D ;
- un plan en 3D ;
- un hyperplan en dimension p.

Message clé : La régression logistique sépare les classes avec une frontière linéaire.

Fonction de perte : Log Loss (entropie croisée binaire)

Pour apprendre w et b, on minimise une erreur adaptée à la classification :

Perte par exemple :

$$\mathcal{L}(y, \hat{p}) = -\left[y \log(\hat{p}) + (1 - y) \log(1 - \hat{p})\right]$$

Intuition :

- si $y = 1$ et $\hat{p}$ est petit $\Rightarrow$ grosse pénalité
- si $y = 0$ et $\hat{p}$ est proche de 1 $\Rightarrow$ grosse pénalité
- très sévère quand le modèle est **confiant mais faux**

Perte moyenne sur n exemples :

$$J(w, b) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, \hat{p}_i)$$

Comment le modèle apprend ? (Descente de gradient)

Objectif :

$$(w^*, b^*) = \arg\min_{w,b} \; J(w, b)$$

Principe (idée) :

- On démarre avec des poids au hasard
- On calcule la perte (erreur)
- On ajuste légèrement w et b pour diminuer la perte
- On répète jusqu'à stabilisation

Mise à jour (forme générale) :

$$w \leftarrow w - \eta \frac{\partial J}{\partial w} \qquad b \leftarrow b - \eta \frac{\partial J}{\partial b}$$

- η : taux d'apprentissage (hyperparamètre)

Régularisation (pour éviter l'overfitting)

On ajoute une pénalité pour favoriser un modèle plus simple :

Ridge (L2) :

$$J_{\text{reg}} = J + \lambda \sum_{j=1}^{p} w_j^2$$

- réduit la taille des coefficients (modèle plus stable)
- garde généralement toutes les variables

Lasso (L1) :

$$J_{\text{reg}} = J + \lambda \sum_{j=1}^{p} |w_j|$$

- peut mettre certains w_j à 0 (sélection automatique de variables)

Attention : λ trop grand $\Rightarrow$ modèle trop simple $\Rightarrow$ underfitting.

Évaluation : quelles métriques utiliser ?

Après entraînement, on prédit sur validation/test puis on calcule :

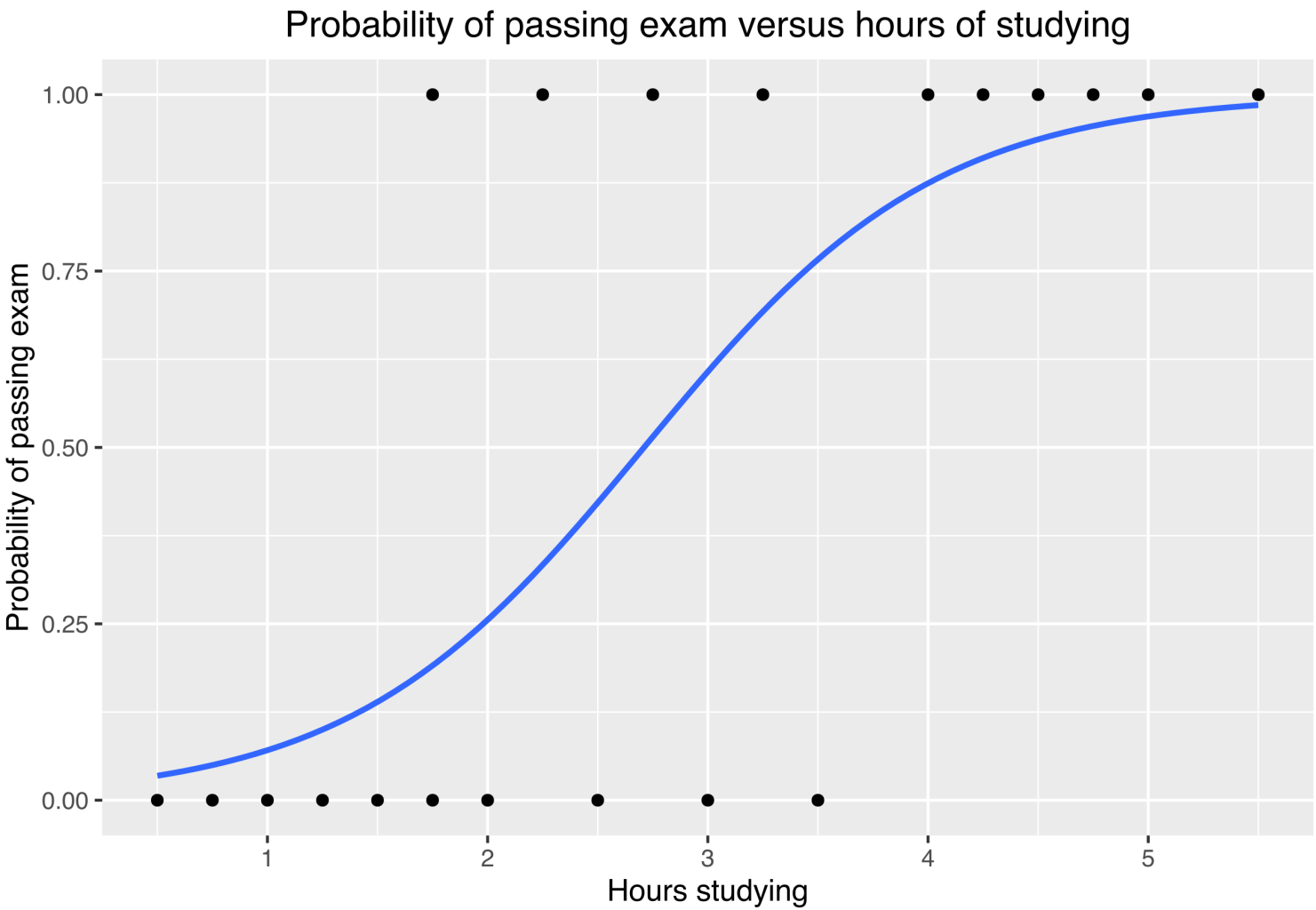
- **Accuracy** : utile si classes équilibrées
- **Precision** : éviter les faux positifs (ex: email normal classé spam)
- **Recall** : éviter de rater des positifs (ex: rater une fraude / maladie)
- **F1-score** : compromis précision / rappel

Bon réflexe : regarder aussi la matrice de confusion pour comprendre les erreurs.

Résumé

- La régression logistique est un modèle supervisé de classification.
- Elle calcule un score $z = w^T x + b$ puis une probabilité $\hat{p} = \sigma(z)$.
- Décision via un seuil (souvent 0.5), frontière de décision linéaire.
- Apprentissage : minimisation de la log loss par descente de gradient.
- Régularisation (L1/L2) pour limiter l'overfitting.

Regression logistique



Modèle d'apprentissage supervisé

Arbre de Décision

Un modèle basé sur des règles simples et interprétables

Arbre de décision : définition

Type : Apprentissage supervisé

Idée principale :

- Le modèle apprend une suite de questions sur les données
- Chaque question divise les données en sous-groupes
- La prédiction finale se fait dans une feuille

Forme générale :

Si (condition 1) alors
si (condition 2) alors classe A
sinon classe B

Structure d'un arbre de décision

- **Racine** : première question (la plus informative)
- **Noeuds internes** : questions intermédiaires
- **Branches** : réponses possibles (oui / non)
- **Feuilles** : prédiction finale

Selon la tâche :

- Classification : la feuille contient une classe
- Régression : la feuille contient une valeur numérique

Comment l'arbre choisit une question ?

À chaque noeud, l'arbre cherche :

- quelle variable utiliser ?
- quel seuil de séparation ?

Objectif : obtenir des sous-groupes les plus homogènes possibles.

Mesures de pureté (classification) :

- Indice de Gini
- Entropie (gain d'information)

Principe :

La meilleure question est celle qui réduit le plus le mélange des classes.

Apprentissage de l'arbre

L'arbre est construit de manière gloutonne :

1. Tester plusieurs questions possibles
2. Choisir la meilleure séparation
3. Répéter sur chaque sous-groupe
4. S'arrêter selon des critères définis

Critères d'arrêt :

- profondeur maximale atteinte
- trop peu d'exemples
- noeud déjà pur

Hyperparamètres importants

- **max_depth** : profondeur maximale de l'arbre
- **min_samples_split** : minimum d'exemples pour diviser
- **min_samples_leaf** : minimum d'exemples dans une feuille

Lien avec la généralisation :

- arbre trop profond $\rightarrow$ overfitting
- arbre trop simple $\rightarrow$ underfitting

Avantages et limites

Avantages :

- Très interprétable
- Facile à expliquer
- Capture des relations non linéaires

Limites :

- Sensible au bruit
- Instable (petit changement $\rightarrow$ arbre différent)

Solution courante : forêts aléatoires, boosting

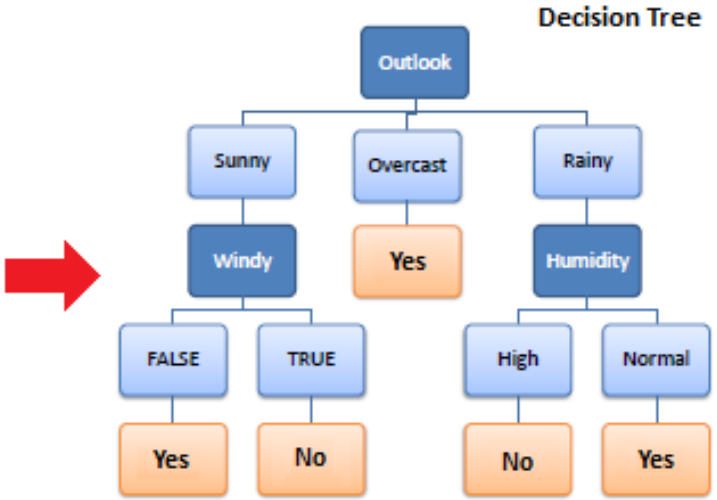
Résumé : Arbre de décision

- Modèle supervisé basé sur des règles
- Prédiction via une suite de décisions
- Complexité contrôlée par la profondeur
- Interprétable mais sensible à l'overfitting

Arbre de décision

Predictors | Target

Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No



Modèle d'apprentissage supervisé

k-plus proches voisins (k-NN)

Apprendre par similarité et voisinage

k-NN : définition et contexte

Type : Apprentissage supervisé

Tâches possibles :

- Classification
- Régression

Principe général :

- Les données d'entraînement sont étiquetées
- Un nouvel exemple est comparé aux exemples connus
- Sa prédiction dépend des exemples les plus proches

Idée clé :

Dis-moi qui sont tes voisins, je te dirai qui tu es.

Pourquoi k-NN est supervisé ?

Les données d'entraînement sont de la forme :

$$(X_i, y_i)$$

- X_i : caractéristiques
- y_i : classe ou valeur réelle connue

Lors de la prédiction :

- on utilise directement les labels des voisins

Conclusion :

Sans labels, k-NN ne peut pas prédire.

Absence de phase d'apprentissage explicite

Contrairement à d'autres modèles :

- pas de paramètres appris
- pas de fonction $f(x; \theta)$

Ce que fait k-NN :

- mémorise les données d'entraînement
- effectue le calcul au moment de la prédiction

Conséquence :

- entraînement rapide
- prédiction coûteuse

Mesure de similarité : la distance

Pour comparer deux points, on utilise une distance.

Distance la plus courante : Euclidienne

$$d(x, z) = \sqrt{\sum_{j=1}^{p} (x_j - z_j)^2}$$

Autres distances possibles :

- Manhattan
- Minkowski
- Cosinus (données textuelles)

Point important :

La notion de "proche" dépend de la distance choisie.

Algorithme k-NN : étapes

Pour un nouvel exemple x_{new} :

1. Calculer la distance entre x_{new} et tous les points d'entraînement
2. Trier les distances
3. Sélectionner les k plus proches voisins
4. Utiliser leurs labels pour prédire

Remarque :

- ces étapes sont répétées pour chaque prédiction

Prédiction en classification

Les k voisins ont des classes connues.

Méthode : vote majoritaire

$$\hat{y} = \text{classe la plus fréquente parmi les voisins}$$

Exemple :

- $k = 5$
- 3 voisins classe A
- 2 voisins classe B

Prédiction : classe A

Prédiction en régression

Les k voisins ont des valeurs numériques.

Méthode la plus simple : moyenne

$$\hat{y} = \frac{1}{k} \sum_{i=1}^{k} y_i$$

Variantes :

- moyenne pondérée par la distance
- médiane (plus robuste aux outliers)

Hyperparamètre clé : le choix de k

Rôle de k :

- contrôle la complexité du modèle

k petit :

- modèle très sensible au bruit
- frontières très complexes
- risque d'overfitting

k grand :

- prédictions trop lissées
- perte de structure locale
- risque d'underfitting

Importance de la normalisation

k-NN repose sur des distances.

Problème :

- une variable à grande échelle domine les autres

Exemple :

- âge (0–100)
- revenu (0–1 000 000)

Solution :

- standardisation ou normalisation des données

Lien avec biais et variance

- k petit $\rightarrow$ faible biais, forte variance
- k grand $\rightarrow$ fort biais, faible variance

Bon choix de k :

- validation croisée
- test de plusieurs valeurs

Avantages et limites de k-NN

Avantages :

- Très simple à comprendre
- Aucun apprentissage complexe
- Bon point de départ pédagogique

Limites :

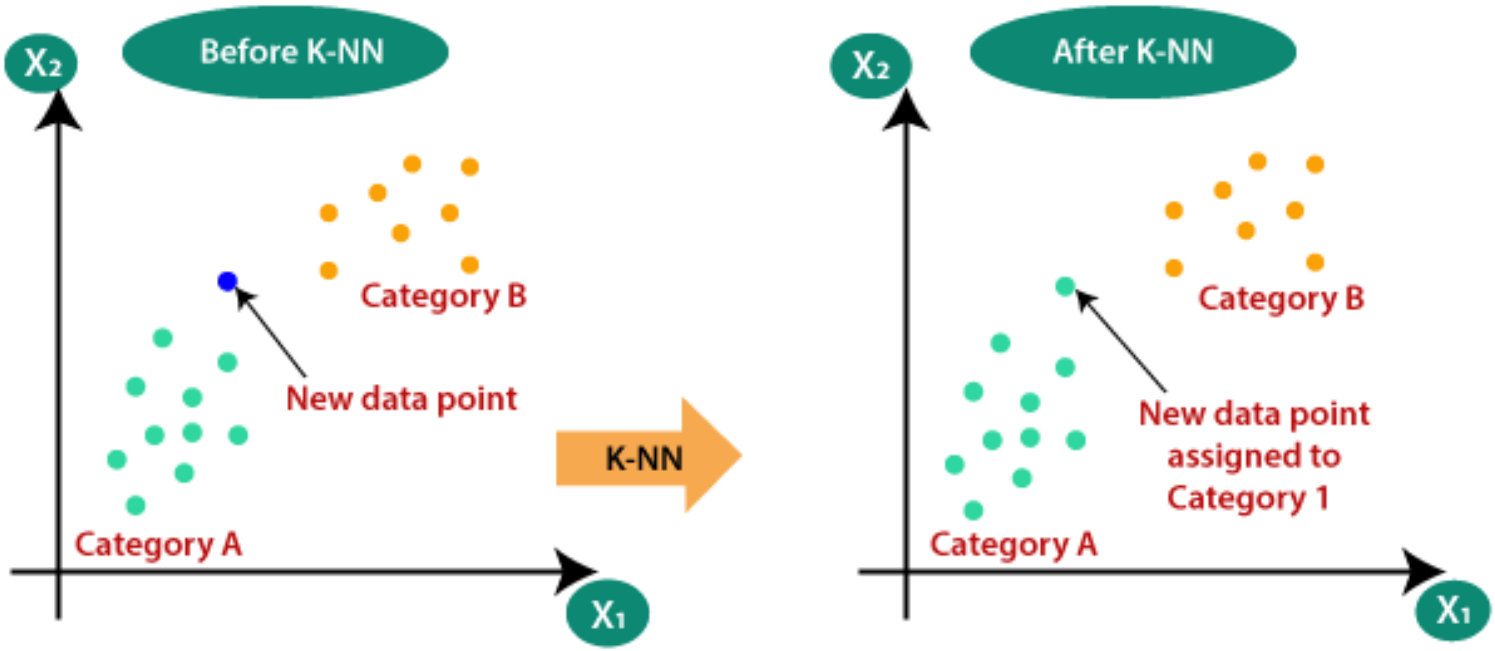
- Coût élevé à la prédiction
- Sensible au bruit
- Performances faibles en haute dimension

Résumé : k-NN

- Algorithme supervisé basé sur la similarité
- Utilise les labels des voisins
- Hyperparamètre clé : k
- Très dépendant de la distance et de la normalisation

Message clé : k-NN apprend par comparaison directe avec les données connues.

KNN


Before K-NN
X2
Category B
New data point
Category A
X1
K-NN
After K-NN
X2
Category B
New data point
assigned to
Category 1
Category A
X1

Modèle d'apprentissage supervisé

Support Vector Machine (SVM)

Séparer les données en maximisant la marge

SVM : contexte et définition

Type : Apprentissage supervisé

Tâches possibles :

- **Classification** (la plus courante)
- **Régression** (SVR)

Objectif principal :

- Trouver une frontière qui sépare les classes
- Cette frontière doit être la plus éloignée possible des données

Idée clé :

SVM ne cherche pas seulement à séparer les classes, mais à les séparer le plus proprement possible.

Données et cadre du problème

On dispose d'un ensemble de données étiquetées :

$$D = \{(x_i, y_i)\}_{i=1}^{n}$$

- $x_i \in \mathbb{R}^p$: vecteur de caractéristiques
- $y_i \in \{-1, +1\}$: classe (notation SVM)

Objectif :

- Trouver une fonction de décision

$$f(x) = w^T x + b$$

- telle que le signe de $f(x)$ donne la classe

Hyperplan de séparation

Hyperplan :

$$w^T x + b = 0$$

Interprétation géométrique :

- en 2D : une droite
- en 3D : un plan
- en dimension p : un hyperplan

Règle de prédiction :

$$\hat{y} = \begin{cases} +1 & \text{si } w^T x + b > 0 \\ -1 & \text{si } w^T x + b < 0 \end{cases}$$

La notion de marge (concept fondamental)

Marge :

- distance entre l'hyperplan et les points les plus proches
- ces points sont appelés vecteurs de support

Pourquoi maximiser la marge ?

- séparation plus robuste
- meilleure généralisation
- moins sensible au bruit

Principe clé de SVM :

Parmi toutes les frontières possibles, SVM choisit celle qui maximise la marge.

Vecteurs de support

Vecteurs de support :

- points les plus proches de la frontière
- seuls points qui définissent la solution

Propriété importante :

- les autres points (loin de la frontière) n'influencent pas le modèle

Conséquence :

- modèle compact
- bonne robustesse

Cas idéal : données parfaitement séparables

On impose :

$$y_i(w^T x_i + b) \geq 1 \quad \forall i$$

Objectif mathématique :

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

Interprétation :

- minimiser $\|w\|$ revient à maximiser la marge

Cas réel : données non séparables

Dans la pratique :

- les classes se chevauchent
- présence de bruit et d'outliers

Solution : SVM à marge souple (soft margin)

On autorise certaines erreurs avec des variables de relâchement ξ_i :

$$y_i(w^T x_i + b) \geq 1 - \xi_i$$

Fonction objectif :

$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$

Le rôle du paramètre C

Paramètre C : hyperparamètre clé

- contrôle le compromis marge / erreurs

C grand :

- pénalise fortement les erreurs
- frontière plus stricte
- risque d'overfitting

C petit :

- accepte plus d'erreurs
- marge plus large
- meilleure généralisation

SVM non linéaire : idée du noyau

Problème :

- certaines données ne sont pas séparables linéairement

Idée du noyau (kernel trick) :

- projeter les données dans un espace de dimension plus élevée
- où la séparation devient possible

Avantage :

- pas besoin de calculer explicitement la projection

Noyaux (kernels) les plus courants

- **Linéaire** :

$$K(x, z) = x^T z$$

- **Polynomial** :

$$K(x, z) = (x^T z + c)^d$$

- **RBF (Gaussien)** :

$$K(x, z) = \exp(-\gamma \|x - z\|^2)$$

RBF :

- le plus utilisé
- très flexible

Le paramètre γ (noyau RBF)

Rôle de γ :

- contrôle l'influence d'un point d'entraînement

γ grand :

- influence très locale
- frontière très complexe
- risque d'overfitting

γ petit :

- influence plus globale
- frontière plus lisse
- risque d'underfitting

Lien avec biais / variance

- C et γ contrôlent la complexité du modèle
- mauvais réglage $\Rightarrow$ sur- ou sous-apprentissage

Bon réglage :

- validation croisée
- grid search ou random search

Avantages et limites de SVM

Avantages :

- très bonne généralisation
- efficace en haute dimension
- robuste au bruit modéré

Limites :

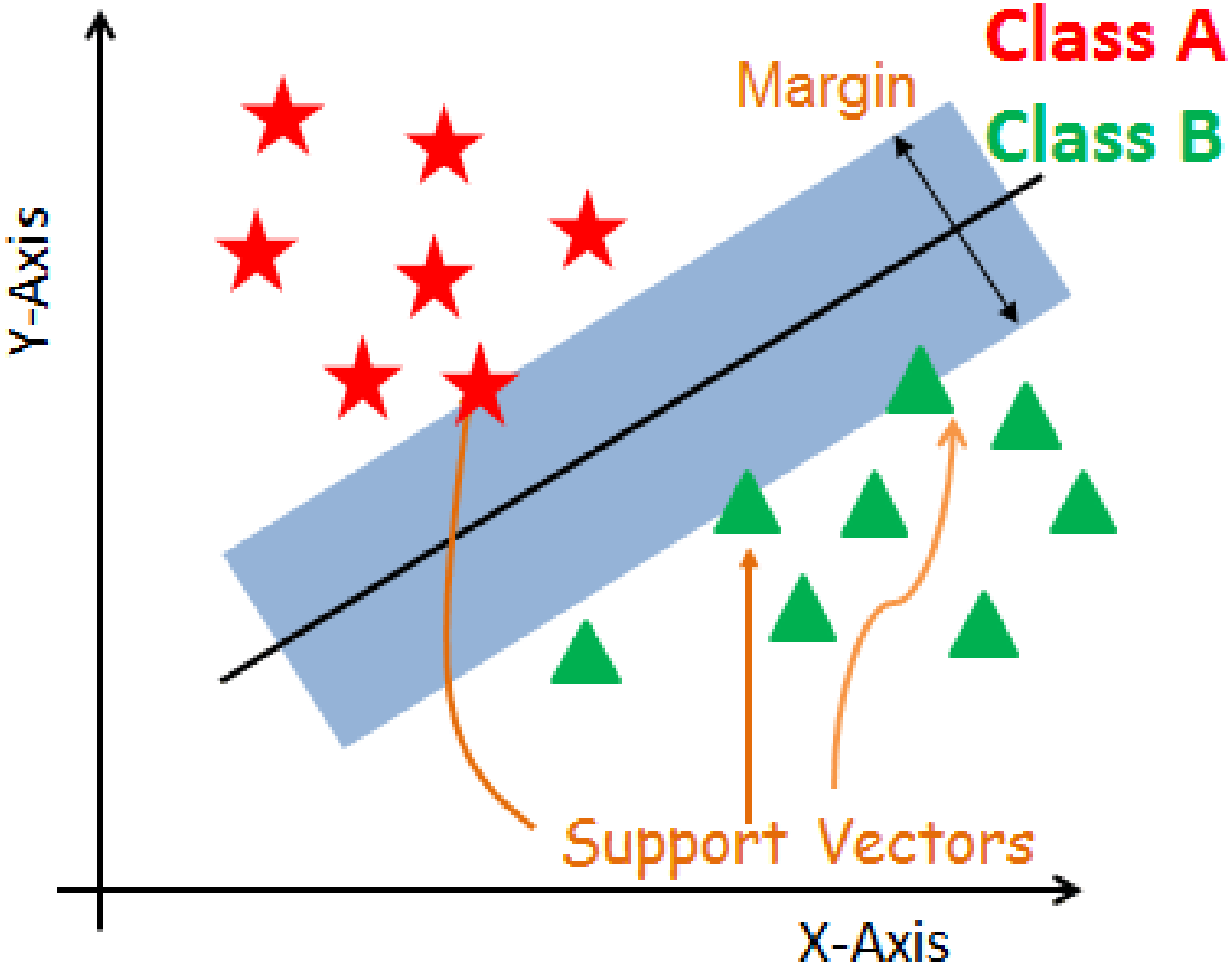
- choix des hyperparamètres délicat
- coût de calcul élevé pour grands jeux de données
- moins interprétable que les arbres

Résumé : Support Vector Machine

- Modèle supervisé basé sur la maximisation de la marge
- Dépend uniquement des vecteurs de support
- Utilise des noyaux pour gérer la non-linéarité
- Hyperparamètres clés : C, noyau, γ

Message clé : SVM privilégie une séparation robuste plutôt qu'une séparation parfaite.

SVM



Modèle d'apprentissage non supervisé

k-means (Clustering)

Regrouper automatiquement des données similaires

k-means : définition

Type : Apprentissage non supervisé

Objectif :

- Regrouper les données en k groupes (clusters)
- Les points d'un même groupe sont similaires
- Les groupes sont différents entre eux

Important :

- Pas de labels
- Pas de vérité terrain

Intuition de k-means

- Chaque cluster est représenté par un centroïde
- Un centroïde est la moyenne des points du cluster
- Chaque point appartient au cluster dont le centroïde est le plus proche

Distance la plus utilisée :

$$d(x, z) = \sqrt{\sum_{j=1}^{p} (x_j - z_j)^2}$$

Algorithme k-means (étapes)

1. Choisir le nombre de clusters k
2. Initialiser k centroïdes (souvent au hasard)
3. Assigner chaque point au centroïde le plus proche
4. Recalculer les centroïdes
5. Répéter jusqu'à convergence

Convergence :

- les affectations ne changent plus
- ou les centroïdes bougent très peu

Fonction objectif de k-means

k-means minimise la somme des distances intra-clusters :

$$J = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

- C_k : cluster k
- μ_k : centroïde du cluster k

Intuition :

Les points doivent être proches de leur centre.

Choix du nombre de clusters k

Problème : k doit être choisi à l'avance.

Méthodes courantes :

- Méthode du coude (Elbow method)
- Silhouette score
- Connaissance métier

Effet de k :

- k trop petit $\rightarrow$ groupes trop larges
- k trop grand $\rightarrow$ groupes artificiels

Méthode du coude (Elbow Method)

Contexte : Clustering k-means

Problème :

- k-means nécessite de choisir le nombre de clusters k
- ce choix n'est pas automatique

Idée de la méthode du coude :

- tester plusieurs valeurs de k
- mesurer la qualité du clustering pour chaque k

Mesure utilisée : inertie (ou SSE)

$$\text{Inertie} = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

Principe :

- plus k augmente, plus l'inertie diminue
- on trace la courbe inertie vs k
- on choisit k au niveau du coude

Importance de la normalisation

k-means est très sensible à l'échelle des variables.

Sans normalisation :

- les variables à grande échelle dominent

Solution :

- standardisation ou normalisation des données

Avantages et limites

Avantages :

- Simple et rapide
- Facile à implémenter
- Fonctionne bien sur des clusters compacts

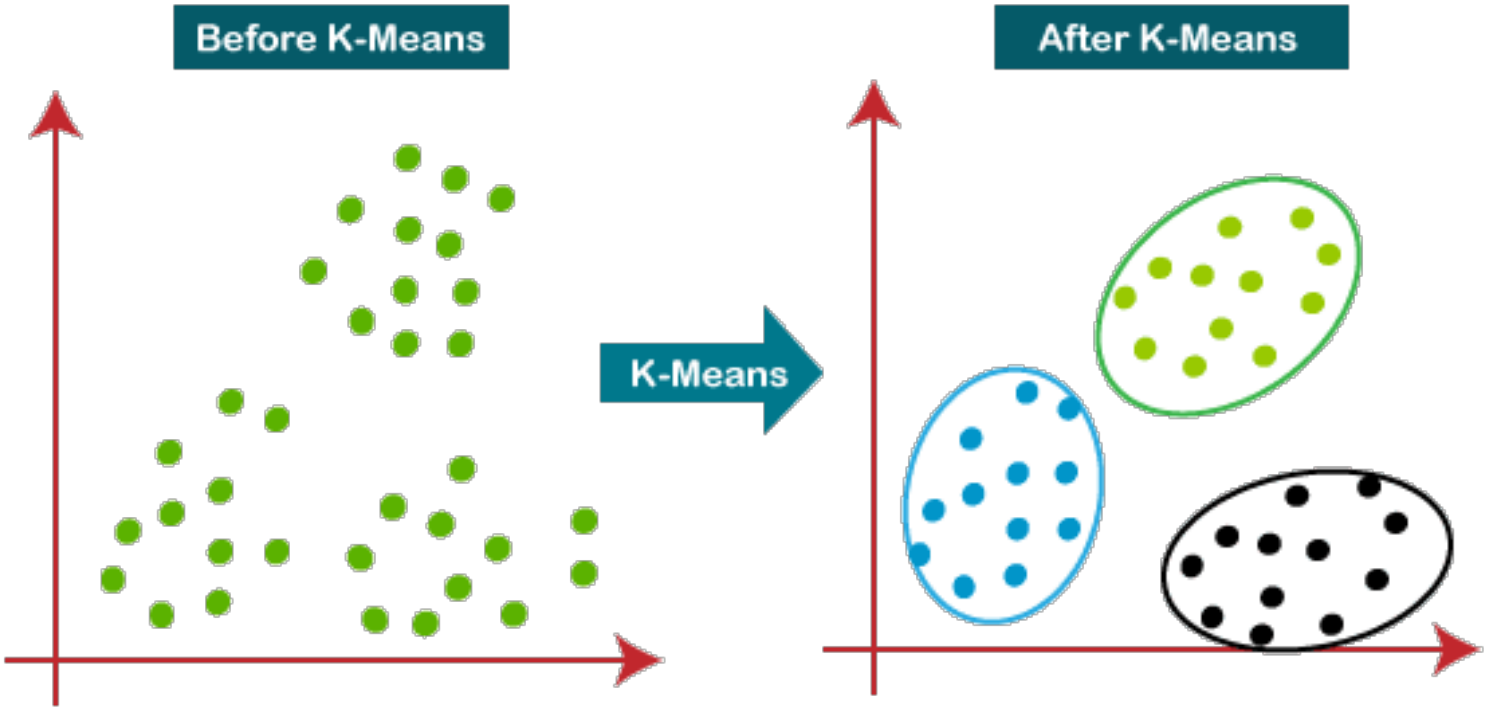
Limites :

- Choix de k délicat
- Sensible à l'initialisation
- Mauvais pour des clusters non sphériques

Résumé : k-means

- Algorithme non supervisé de clustering
- Regroupe les données sans labels
- Basé sur la distance aux centroïdes
- Sensible à l'échelle et au choix de k

K-means



Algorithme non supervisé

Clustering hiérarchique

Regrouper les données de manière progressive et interprétable

Clustering hiérarchique : définition

Type : Apprentissage non supervisé

Objectif :

- Regrouper automatiquement des données similaires
- Construire une hiérarchie de groupes
- Explorer la structure des données sans labels

Résultat principal :

- un dendrogramme (arbre de regroupement)

Idée intuitive

- Chaque observation est au départ un groupe
- On fusionne progressivement les groupes les plus proches
- Le processus continue jusqu'à former un seul groupe

Analogie pédagogique :

Former des équipes en regroupant progressivement les étudiants les plus semblables.

Deux grandes approches

Clustering hiérarchique agglomératif (le plus utilisé) :

- chaque point commence seul
- on fusionne les groupes proches

Clustering hiérarchique divisif :

- tous les points commencent ensemble
- on divise progressivement

Dans la pratique : on utilise presque toujours la version agglomérative.

Comment mesurer la proximité entre groupes ?

Le clustering hiérarchique repose sur :

- une distance entre les points
- une méthode de liaison (linkage)

Distances courantes :

- Euclidienne
- Manhattan

Méthodes de liaison (linkage) :

- **Single** : distance minimale
- **Complete** : distance maximale
- **Average** : distance moyenne
- **Ward** : minimise la variance intra-groupe

Le dendrogramme

Dendrogramme :

- arbre qui montre les fusions successives
- la hauteur représente la distance de fusion

Comment choisir le nombre de clusters ?

- couper le dendrogramme à une certaine hauteur
- plus la coupe est basse $\rightarrow$ plus de clusters

Avantage clé :

Pas besoin de fixer le nombre de clusters à l'avance.

Avantages et limites

Avantages :

- Très interprétable
- Pas besoin de choisir k au départ
- Bon pour l'exploration des données

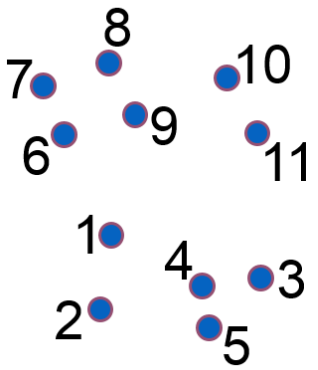
Limites :

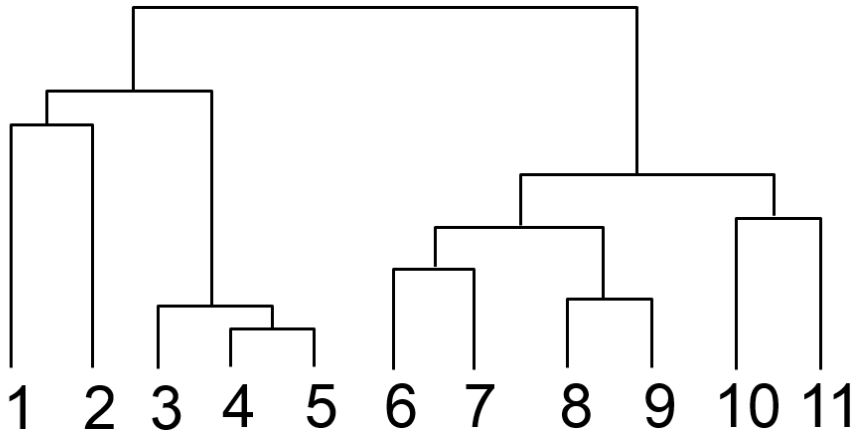
- Coût de calcul élevé pour grands jeux de données
- Sensible au bruit et aux outliers

Résumé : Clustering hiérarchique

- Algorithme non supervisé de regroupement progressif
- Basé sur distances et méthodes de liaison
- Produit un dendrogramme interprétable
- Idéal pour l'exploration de données

Clustering Hierarchique





Algorithme non supervisé

Analyse en Composantes Principales (ACP)

Réduction de dimension basée sur la variance et la covariance

ACP : contexte et objectif

Type : Apprentissage non supervisé

Problème :

- Les jeux de données contiennent souvent beaucoup de variables
- Certaines variables sont redondantes ou corrélées

Objectif de l'ACP :

- Réduire le nombre de variables
- Conserver un maximum d'information
- Simplifier l'analyse et la visualisation

Idée intuitive de l'ACP

- Les données forment un nuage de points
- Certaines directions expliquent mieux la dispersion des points

Principe :

Trouver les directions où les données varient le plus.

Résultat :

- nouvelles variables appelées composantes principales
- moins nombreuses mais plus informatives

Étape 1 : centrage et normalisation des données

Soit une matrice de données :

$$X \in \mathbb{R}^{n \times p}$$

- n : nombre d'observations
- p : nombre de variables

Centrage :

$$x_{ij}^{(c)} = x_{ij} - \bar{x}_j$$

Pourquoi ?

- l'ACP analyse la variance autour de la moyenne

Normalisation (souvent nécessaire) :

- rend les variables comparables
- évite qu'une variable à grande échelle domine

Étape 2 : qu'est-ce que la covariance ?

Covariance entre deux variables X et Y :

$$\text{Cov}(X, Y) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$$

Interprétation :

- $\text{Cov}(X, Y) > 0$: les variables augmentent ensemble
- $\text{Cov}(X, Y) < 0$: l'une augmente, l'autre diminue
- $\text{Cov}(X, Y) \approx 0$: pas de lien linéaire

Cas particulier :

$$\text{Cov}(X, X) = \text{Var}(X)$$

Étape 3 : matrice de covariance

Pour p variables, on construit la matrice de covariance :

$$\Sigma = \begin{pmatrix} \mathrm{Var}(X_1) & \mathrm{Cov}(X_1, X_2) & \cdots & \mathrm{Cov}(X_1, X_p) \\ \mathrm{Cov}(X_2, X_1) & \mathrm{Var}(X_2) & \cdots & \mathrm{Cov}(X_2, X_p) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}(X_p, X_1) & \mathrm{Cov}(X_p, X_2) & \cdots & \mathrm{Var}(X_p) \end{pmatrix}$$

À retenir :

- matrice symétrique
- diagonale = variances
- hors diagonale = relations entre variables

Pourquoi la matrice de covariance est centrale en ACP

- L'ACP cherche les directions de variance maximale
- Ces directions dépendent directement de la covariance

Idée clé :

Les composantes principales sont les directions qui expliquent le mieux la structure de la matrice de covariance.

Conséquence :

- si les variables sont corrélées $\rightarrow$ ACP très utile
- si elles sont indépendantes $\rightarrow$ gain limité

Étape 4 : valeurs propres et vecteurs propres (intuition)

On calcule :

- les vecteurs propres de la matrice de covariance
- les valeurs propres associées

Interprétation :

- vecteurs propres $\rightarrow$ directions principales (axes)
- valeurs propres $\rightarrow$ quantité de variance expliquée

Ordre :

- PC1 : variance maximale
- PC2 : variance maximale restante
- etc.

Étape 5 : projection des données

Une fois les directions trouvées :

- on projette les données sur les premières composantes

$$Z = XW$$

- W : matrice des vecteurs propres sélectionnés
- Z : nouvelles variables (composantes principales)

Résultat :

- moins de dimensions
- moins de bruit

Variance expliquée

Chaque composante explique une partie de la variance totale.

Ratio de variance expliquée :

$$\frac{\lambda_k}{\sum_j \lambda_j}$$

Choix du nombre de composantes :

- conserver 80–95% de la variance
- utiliser un graphique en coude

Applications typiques de l'ACP

- Visualisation de données en 2D / 3D
- Réduction de dimension avant clustering
- Prétraitement avant classification
- Réduction du bruit

Limites de l'ACP

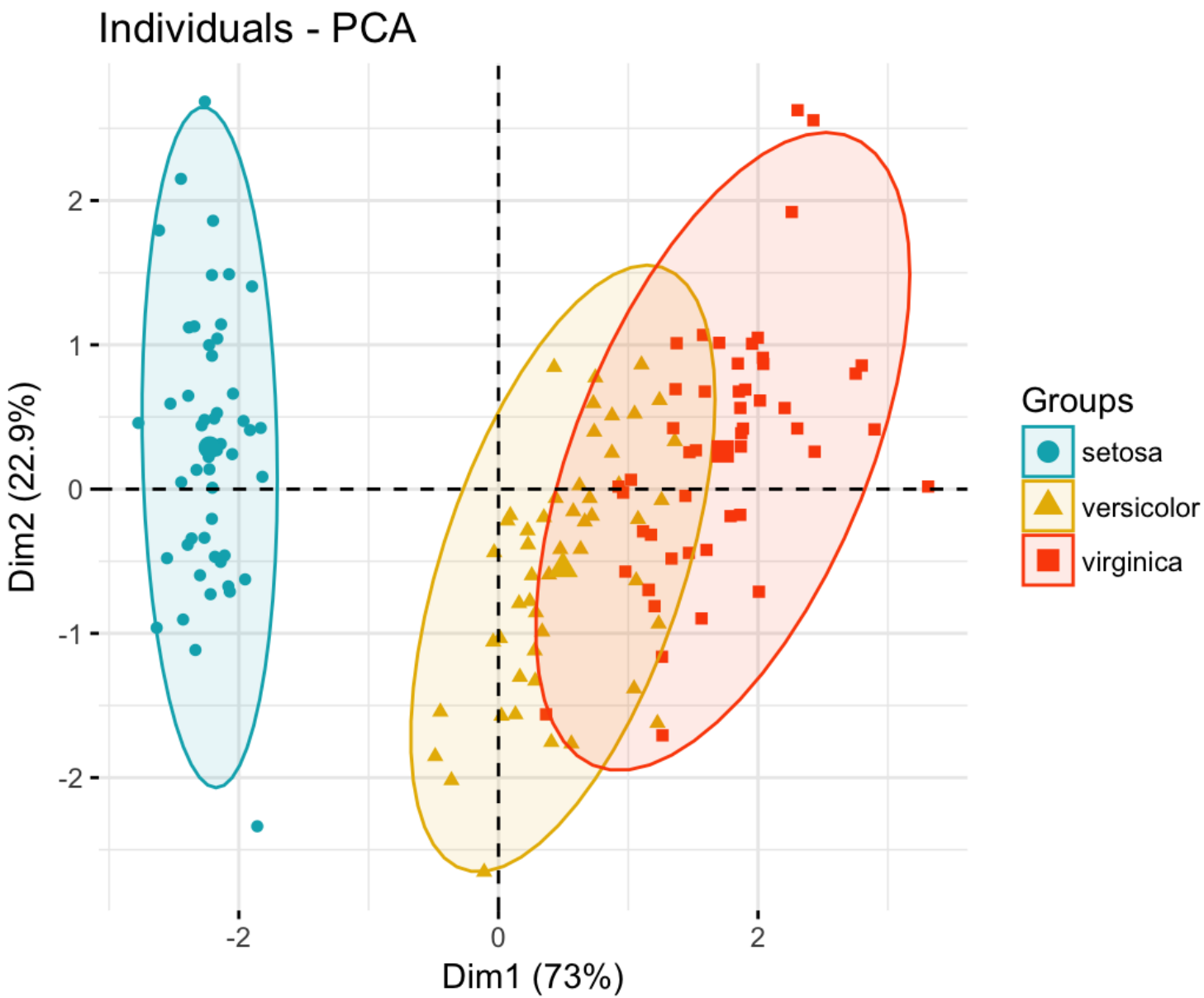
- Méthode linéaire
- Perte d'interprétation des variables originales
- Sensible à l'échelle des variables

Résumé : ACP

- Algorithme non supervisé de réduction de dimension
- Basé sur la matrice de covariance
- Recherche les directions de variance maximale
- Étape clé avant de nombreux algorithmes

Message clé : l'ACP résume l'information sans utiliser de labels.

ACP


Individuals - PCA
Dim2 (22.9%)
Dim1 (73%)
2
1
0
-1
-2
-2
0
2
Groups
setosa
versicolor
virginica

Travaux Pratiques

Travaux Pratiques

- Voir Notebook Cours4-TP1
- Voir Notebook Cours4-TP2

Introduction à scikit-learn

- **scikit-learn** est une bibliothèque Python pour :
  - la **préparation** des données ;
  - l'**entraînement** de modèles de machine learning ;
  - l'**évaluation** et la **sélection** de modèles.

- Principales familles de modèles :
  - Régression, classification
  - Clustering, réduction de dimension
  - Sélection de variables, pipelines, etc.

- **API unifiée** basée sur les méthodes `.fit()`, `.predict()`, `.transform()`.

Cycle typique avec scikit-learn

1. Charger les données (`pandas`, fichiers CSV, etc.).
2. Séparer en jeux **train** et **test**.
3. Préparer les données :
   - gestion des valeurs manquantes,
   - encodage des variables catégorielles,
   - normalisation / standardisation.
4. Choisir un modèle (régression, classification, clustering. . .).
5. Entraîner le modèle : `model.fit(X_train, y_train)`.
6. Prédire : `model.predict(X_test)`.
7. Évaluer les performances avec des métriques adaptées.

Scikit-learn : syntaxe de base

- Exemple minimal pour un problème supervisé :

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# X : features, y : cible
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 1. Définir le modèle
model = LinearRegression()

# 2. Entraîner le modèle
model.fit(X_train, y_train)

# 3. Faire des prédictions
y_pred = model.predict(X_test)
```

Exemple : régression avec scikit-learn

- Prédiction d'une variable numérique (prix, score, etc.) :

```
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import Ridge

# Définir un modèle de régression Ridge
model = Ridge(alpha=1.0)

# Entraînement
model.fit(X_train, y_train)

# Prédictions sur le test
y_pred = model.predict(X_test)

# Evaluation
mse = mean_squared_error(y_test, y_pred)
r2  = r2_score(y_test, y_pred)

print("MSE :", mse)
```

Exemple : classification avec scikit-learn

- Prédiction d'une classe (spam / non-spam, malade / sain, etc.) :

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score

# X, y : donnees et labels
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

model = LogisticRegression(max_iter=1000)

model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Accuracy :", accuracy_score(y_test, y_pred))
print("F1-score :", f1_score(y_test, y_pred, average="bi
```

Prétraitement et Pipelines

- **Pipeline** : enchaîner prétraitements et modèle, pour éviter les fuites de données et simplifier le code.

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("svc", SVC(kernel="rbf", C=1.0))
])

pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
```

- Avantages :
  - même transformation appliquée à train et test ;
  - code plus clair ;
  - compatible avec validation croisée et GridSearch.

Validation croisée et recherche d'hyperparamètres

- scikit-learn intègre :
  - la **validation croisée** : `cross_val_score`
  - la **recherche d'hyperparamètres** : `GridSearchCV`, `RandomizedSearchCV`.

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

param_grid = {
    "n_estimators": [50, 100, 200],
    "max_depth": [None, 5, 10]
}

rf = RandomForestClassifier(random_state=42)

grid = GridSearchCV(
    rf, param_grid,
    cv=5, scoring="f1", n_jobs=-1
)
```

Résumé du Chapitre 4 (Partie 1)

- Concepts fondamentaux : données, features, labels, tâches supervisées et non supervisées.
- Modèles d'apprentissage : classification et régression (principaux algorithmes).
- Fonctionnement des modèles : paramètres, hyperparamètres, frontières de décision, fonctions d'activation.
- Évaluation : métriques pour classification et régression, ROC/AUC, matrice de confusion.
- Généralisation : biais, variance, sur-apprentissage et sous-apprentissage.
- Régularisation : L1 (Lasso), L2 (Ridge), Elastic Net, Dropout, Early stopping.

Auto-évaluation

- Je comprends la différence entre classification et régression.
- Je sais expliquer ce qu'est une fonction de perte et une fonction d'activation.
- Je peux interpréter une frontière de décision et un ajustement de régression.
- Je sais distinguer paramètres et hyperparamètres d'un modèle.
- Je comprends ce qu'est la généralisation et comment la mesurer.
- Je peux identifier des situations de sur-apprentissage et sous-apprentissage.
- Je comprends le rôle des techniques de régularisation (L1, L2, Dropout, etc.).