Cours Machine Learning

Chapitre 04

Classification Supervisée

Objectifs d'apprentissage :

— Maîtriser les algorithmes de classification (KNN, arbres, SVM)
— Comprendre les méthodes d'ensemble (bagging, boosting)
— Implémenter et optimiser des classifieurs
— Évaluer et comparer différents modèles

Prérequis : Chapitres 00, 01, 02, 03

Durée estimée : 6-8 heures

Notebooks : `04_*.ipynb`

Cours ML - Sandbox-ML
Version 1.0 - 2026

Table des matières

1 Introduction à la Classification **2**
1.1 Frontière de Décision . . . 2

2 K-Nearest Neighbors (KNN) **3**
2.1 Principe . . . 3
2.2 Hyperparamètres . . . 3
2.3 Avantages et Limites . . . 3

3 Arbres de Décision **4**
3.1 Principe . . . 4
3.2 Construction de l'Arbre (CART) . . . 4
3.3 Mesures d'Impureté . . . 5
3.4 Hyperparamètres et Régularisation . . . 5
3.5 Avantages et Limites . . . 5

4 Random Forest **6**
4.1 Principe : Ensemble Learning . . . 6
4.2 Bagging (Bootstrap Aggregating) . . . 6
4.3 Random Forest = Bagging + Randomisation . . . 7
4.4 Hyperparamètres . . . 7
4.5 Out-of-Bag (OOB) Error . . . 7
4.6 Feature Importance . . . 7
4.7 Avantages et Limites . . . 8

5 Gradient Boosting **8**
5.1 Principe : Boosting . . . 8
5.2 Gradient Boosting (GBDT) . . . 8
5.3 Hyperparamètres . . . 8
5.4 XGBoost, LightGBM, CatBoost . . . 9

6 Support Vector Machines (SVM) **9**
6.1 Principe : Maximum Margin . . . 9
6.2 Formulation Mathématique . . . 10
6.3 Kernel Trick . . . 10
6.4 Hyperparamètres . . . 11
6.5 Avantages et Limites . . . 11

7 Comparaison des Algorithmes **12**

8 Résumé **12**
8.1 Points Clés . . . 12

9 Exercices **12**

10 Pour Aller Plus Loin **12**

1 Introduction à la Classification

Définition

Classification La classification est une tâche d'apprentissage supervisé où l'objectif est de prédire une variable cible **catégorielle** $y \in \{1, 2, \ldots, K\}$ à partir de features $\mathbf{x} \in \mathbb{R}^d$.

Types de classification :

- **Binaire :** $K = 2$ (ex : spam/non-spam, malade/sain)
- **Multi-classe :** $K > 2$ (ex : classification de chiffres 0-9, types d'animaux)
- **Multi-label :** Une instance peut appartenir à plusieurs classes simultanément

Exemple

Applications

- **Médical :** Diagnostic de maladies (sain, malade A, malade B)
- **Finance :** Détection de fraude (frauduleux/légitime), scoring crédit (bon/mauvais)
- **Vision :** Classification d'images (chat, chien, oiseau, etc.)
- **NLP :** Analyse de sentiment (positif/négatif/neutre), classification de documents

1.1 Frontière de Décision

Un classifieur apprend une **frontière de décision** qui sépare les différentes classes dans l'espace des features.

Exemples :

- **Linéaire :** Droite (2D), hyperplan (d-dimensions) - régression logistique, SVM linéaire
- **Non-linéaire :** Courbes, surfaces complexes - arbres, SVM kernel, réseaux de neurones

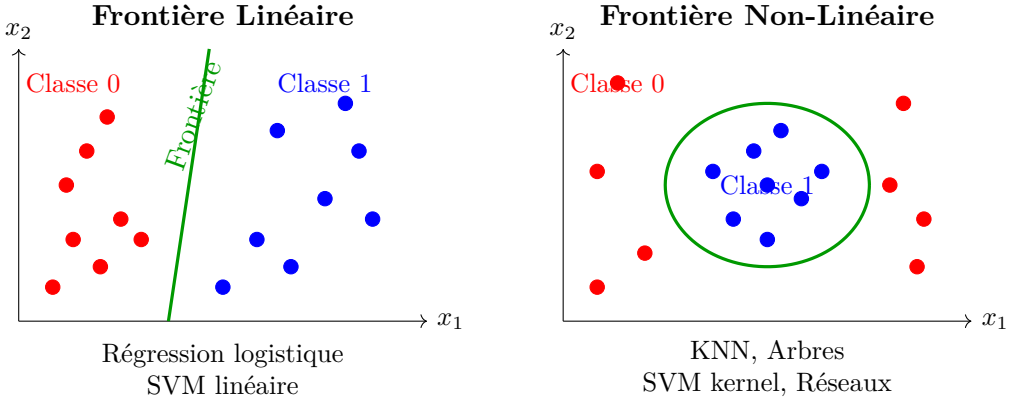

FIGURE 1 – Frontières de décision : linéaire (gauche) séparant par une droite/hyperplan, et non-linéaire (droite) capturant des patterns complexes. Le choix dépend de la distribution des données.

2 K-Nearest Neighbors (KNN)

2.1 Principe

Définition

K-Nearest Neighbors KNN est un algorithme **basé sur l'instance** : pour prédire la classe d'un nouveau point $\mathbf{x}$, on cherche les k voisins les plus proches dans le train set et on vote à la majorité.

Algorithm 1 KNN Classification

Require: Dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, point de test $\mathbf{x}_{\text{test}}$, k

Ensure: Classe prédite $\hat{y}$

1: Calculer distances : $d_i = \|\mathbf{x}_{\text{test}} - \mathbf{x}_i\|$ pour tout i

2: Trouver les k plus petites distances (k voisins)

3: $\hat{y}$ = classe majoritaire parmi ces k voisins

4: **return** $\hat{y}$

Distance utilisée : Typiquement euclidienne L^2, mais peut être Manhattan L^1, Minkowski, etc.

2.2 Hyperparamètres

- k **(nombre de voisins)** :
  - k petit : Modèle complexe, sensible au bruit, overfitting
  - k grand : Modèle simple, lisse, underfitting
  - Règle empirique : $k = \sqrt{n}$ ou validation croisée
- **Métrique de distance** : Euclidienne, Manhattan, etc.
- **Pondération** : Uniforme ou par distance (voisins proches pèsent plus)

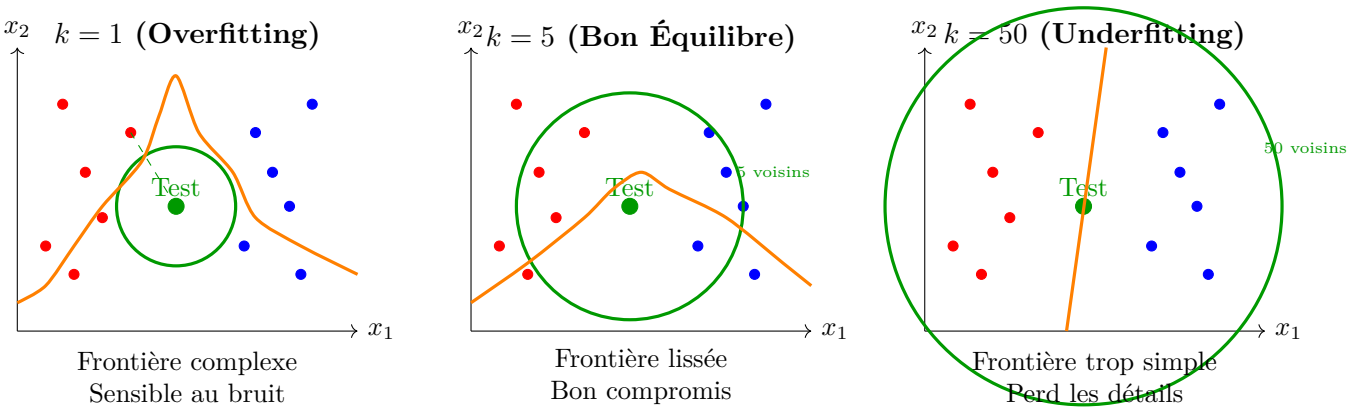

FIGURE 2 – Impact du paramètre k dans KNN : $k = 1$ crée une frontière complexe qui s'adapte au bruit (overfitting), $k = 5$ donne un bon équilibre, $k = 50$ sur-lisse et perd les détails (underfitting).

2.3 Avantages et Limites

Avantages :

— Simple à comprendre et implémenter
— Pas de phase d'entraînement (lazy learning)
— Fonctionne bien pour frontières complexes
— Naturellement multi-classe

Limites :

— Coût de prédiction élevé : $O(nd)$ par prédiction
— Sensible à l'échelle des features (nécessite normalisation)
— Curse of dimensionality : performance dégradée en haute dimension
— Nécessite beaucoup de mémoire (stocke tout le train set)

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler

# Normalisation (important pour KNN!)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# KNN
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)
y_pred = knn.predict(X_test_scaled)
```

Listing 1 – KNN avec scikit-learn

3 Arbres de Décision

3.1 Principe

Définition

Arbre de décision Un arbre de décision partitionne récursivement l'espace des features en régions homogènes via des tests binaires sur les features.

Structure :

— **Nœud interne :** Test sur une feature ($x_j \leq$ seuil)
— **Branche :** Résultat du test (gauche : vrai, droite : faux)
— **Feuille :** Prédiction finale (classe majoritaire)

3.2 Construction de l'Arbre (CART)

L'algorithme CART (Classification and Regression Trees) construit l'arbre de manière gloutonne (greedy) :

Algorithm 2 Construction Arbre (CART)

Require: Dataset $D = \{(\mathbf{x}_i, y_i)\}$

Si critère d'arrêt atteint : créer feuille avec classe majoritaire
Sinon :
    Trouver meilleure feature j et seuil θ qui minimise impureté
    Séparer D en D_{left} et D_{right}
    Construire récursivement sous-arbres gauche et droit

3.3 Mesures d'Impureté

Pour choisir le meilleur split, on minimise l'impureté.

1. Entropie (Shannon) :

$$H(D) = -\sum_{k=1}^{K} p_k \log_2(p_k) \tag{1}$$

où p_k est la proportion de la classe k dans D.

2. Indice de Gini :

$$\text{Gini}(D) = 1 - \sum_{k=1}^{K} p_k^2 \tag{2}$$

3. Misclassification Error :

$$\text{Error}(D) = 1 - \max_k p_k \tag{3}$$

Gain d'information :

$$\text{IG} = H(D) - \frac{|D_{\text{left}}|}{|D|} H(D_{\text{left}}) - \frac{|D_{\text{right}}|}{|D|} H(D_{\text{right}}) \tag{4}$$

On choisit le split qui maximise le gain d'information (ou minimise l'impureté).

3.4 Hyperparamètres et Régularisation

Critères d'arrêt :

- `max_depth` : Profondeur maximale de l'arbre
- `min_samples_split` : Nombre min d'échantillons pour split
- `min_samples_leaf` : Nombre min d'échantillons par feuille
- `max_features` : Nombre max de features considérées pour split

Pruning (élagage) :

- **Pre-pruning :** Arrêter tôt la construction (via critères ci-dessus)
- **Post-pruning :** Construire arbre complet puis élaguer (cost-complexity pruning)

3.5 Avantages et Limites

Avantages :

- Très interprétables (visualisation facile)
- Gèrent features numériques et catégorielles
- Pas besoin de normalisation

— Capturent des interactions non-linéaires
— Rapides en prédiction

Limites :

— Instables : petits changements dans les données ⇒ arbre très différent
— Tendance à l'overfitting
— Frontières de décision orthogonales (alignées sur axes)
— Biais vers features avec beaucoup de valeurs

```python
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(
    max_depth=5,
    min_samples_split=20,
    criterion='gini'
)
tree.fit(X_train, y_train)
y_pred = tree.predict(X_test)

# Visualisation
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
plt.figure(figsize=(20, 10))
plot_tree(tree, filled=True, feature_names=feature_names)
plt.show()
```

Listing 2 – Arbre de décision

4 Random Forest

4.1 Principe : Ensemble Learning

Définition

Ensemble Learning Combiner plusieurs modèles faibles pour créer un modèle fort. L'idée : agréger les prédictions de multiples arbres pour réduire variance et overfitting.

4.2 Bagging (Bootstrap Aggregating)

Algorithme :

1. Pour $b = 1, \ldots, B$:
   — Créer un échantillon bootstrap (tirage avec remise) de taille n
   — Entraîner un arbre de décision T_b sur cet échantillon
2. Prédiction finale : vote majoritaire des B arbres

Pourquoi ça marche ?

— Arbres individuels : haute variance (overfitting)
— Moyenner B arbres : réduit la variance
— Les arbres sont **décorrélés** grâce au bootstrap

4.3 Random Forest = Bagging + Randomisation

Définition

Random Forest Extension du bagging avec randomisation supplémentaire :

- **Bootstrap des instances** (comme bagging)
- **Sélection aléatoire de features** : À chaque split, on considère seulement m features aléatoires (typiquement $m = \sqrt{d}$)

Bénéfice : Décorrèle encore plus les arbres ⇒ variance réduite.

4.4 Hyperparamètres

- `n_estimators` : Nombre d'arbres B (plus = mieux, mais diminishing returns)
- `max_features` : Nombre de features considérées par split
- `max_depth`, `min_samples_split`, etc. : Paramètres des arbres
- `bootstrap` : Utiliser bootstrap ou non

4.5 Out-of-Bag (OOB) Error

Chaque arbre est entraîné sur 63% des données (bootstrap). Les 37% restants (out-of-bag) servent à estimer l'erreur de généralisation sans validation set séparé.

4.6 Feature Importance

Random Forest peut calculer l'importance de chaque feature :

- **Mean Decrease Impurity (MDI)** : Moyenne de la réduction d'impureté apportée par chaque feature
- **Mean Decrease Accuracy (MDA)** : Dégradation de l'accuracy quand on permute la feature

```python
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(
    n_estimators=100,
    max_depth=10,
    max_features='sqrt',
    random_state=42
)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

# Feature importance
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]
print("Top features:")
for i in range(5):
    print(f"{i+1}. {feature_names[indices[i]]}: {importances[indices[i]]:.4f}")
```

Listing 3 – Random Forest

4.7 Avantages et Limites

Avantages :

- Performances excellentes (souvent meilleures que arbre seul)
- Réduction overfitting
- Robuste au bruit et outliers
- Feature importance automatique
- Parallélisable (arbres indépendants)

Limites :

- Moins interprétable qu'un arbre seul
- Plus lent en prédiction (doit consulter B arbres)
- Nécessite plus de mémoire

5 Gradient Boosting

5.1 Principe : Boosting

Définition

Boosting Entraîner des modèles séquentiellement, où chaque nouveau modèle se concentre sur les erreurs des modèles précédents.

Différence avec Bagging :

- **Bagging :** Modèles indépendants en parallèle
- **Boosting :** Modèles séquentiels, chaque modèle corrige les erreurs du précédent

5.2 Gradient Boosting (GBDT)

Idée : Construire un ensemble additif de modèles (arbres faibles) en minimisant une fonction de perte via descente de gradient.

Algorithm 3 Gradient Boosting

Require: Dataset $\{(\mathbf{x}_i, y_i)\}$, fonction de perte L, learning rate η, M iterations

1: Initialiser $F_0(\mathbf{x}) = \text{argmin}_c \sum_{i=1}^{n} L(y_i, c)$
2: **for** $m = 1$ to M **do**
3: Calculer pseudo-résidus : $r_{im} = -\frac{\partial L(y_i, F_{m-1}(\mathbf{x}_i))}{\partial F_{m-1}(\mathbf{x}_i)}$
4: Entraîner arbre h_m sur $\{(\mathbf{x}_i, r_{im})\}$
5: Mettre à jour : $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta h_m(\mathbf{x})$
6: **end for**
7: **return** $F_M(\mathbf{x})$

Intuition : Chaque nouvel arbre h_m prédit les résidus (erreurs) du modèle actuel F_{m-1}, on l'ajoute avec un petit poids η.

5.3 Hyperparamètres

- `n_estimators` : Nombre d'arbres M

— `learning_rate` η : Poids de chaque arbre (trade-off avec M)
— `max_depth` : Profondeur des arbres (typiquement faible : 3-6)
— `subsample` : Fraction de données par arbre (stochastic gradient boosting)
— `min_samples_split`, `min_samples_leaf` : Régularisation

5.4 XGBoost, LightGBM, CatBoost

Implémentations optimisées et étendues de Gradient Boosting :

XGBoost (Extreme Gradient Boosting) :
— Régularisation L^1 et L^2
— Gestion des valeurs manquantes
— Parallélisation et optimisations
— Très utilisé en compétitions Kaggle

LightGBM :
— Gradient-based One-Side Sampling (GOSS)
— Exclusive Feature Bundling (EFB)
— Très rapide et efficace en mémoire
— Excellentes performances sur gros datasets

CatBoost :
— Gestion native des features catégorielles
— Ordered boosting (évite target leakage)
— Robuste aux hyperparamètres

```python
from xgboost import XGBClassifier

xgb = XGBClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=5,
    subsample=0.8,
    random_state=42
)
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
```

Listing 4 – Gradient Boosting avec XGBoost

6 Support Vector Machines (SVM)

6.1 Principe : Maximum Margin

Définition

SVM SVM cherche l'hyperplan qui sépare les classes avec la **marge maximale**. La marge est la distance entre l'hyperplan et les points les plus proches (support vectors).

Cas linéairement séparable :

L'hyperplan est défini par :

$$\mathbf{w}^T\mathbf{x} + b = 0 \tag{5}$$

Prédiction : $\hat{y} = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$

Objectif : Maximiser la marge $\frac{2}{\|\mathbf{w}\|} \Leftrightarrow$ Minimiser $\|\mathbf{w}\|^2$ sous contraintes.

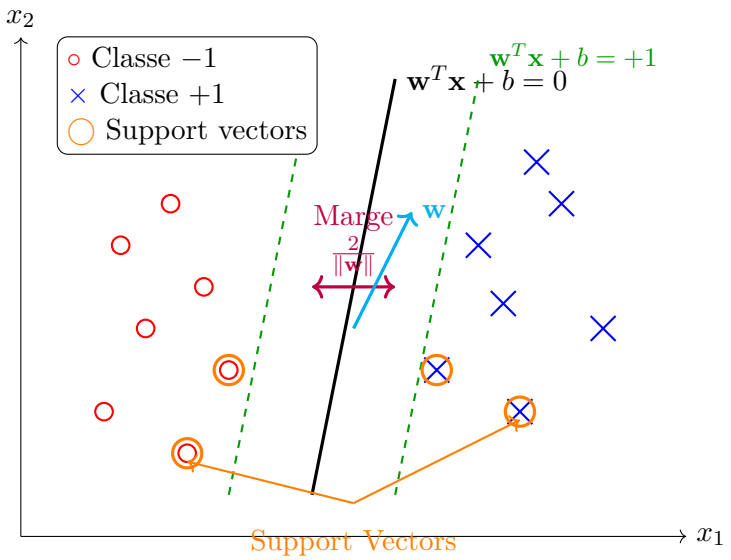

FIGURE 3 – SVM avec marge maximale : l'hyperplan noir sépare les classes, les marges (vertes) définissent la zone, et les support vectors (orange) sont les points les plus proches qui déterminent l'hyperplan. La marge est maximisée.

6.2 Formulation Mathématique

Hard-margin SVM (séparable) :

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \ \forall i \tag{6}$$

Soft-margin SVM (non-séparable) : Introduire des variables de slack $\xi_i \geq 0$ pour permettre des erreurs :

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \ \xi_i \geq 0 \tag{7}$$

Hyperparamètre C :

- C grand : Peu d'erreurs tolérées (risque overfitting)
- C petit : Plus d'erreurs tolérées (underfitting)

6.3 Kernel Trick

Pour gérer des données non-linéairement séparables, on projette les données dans un espace de dimension supérieure via une fonction $\phi(\mathbf{x})$.

Problème : Calcul coûteux de $\phi(\mathbf{x})$ en haute dimension.

Solution : Kernel Trick On n'a besoin que des produits scalaires $\phi(\mathbf{x})^T\phi(\mathbf{x}')$, qu'on calcule efficacement via un **kernel** $K(\mathbf{x}, \mathbf{x}')$.

Kernels courants :

- **Linéaire :** $K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$
- **Polynomial :** $K(\mathbf{x}, \mathbf{x}') = (\gamma\mathbf{x}^T\mathbf{x}' + r)^d$
- **RBF (Gaussian) :** $K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$
- **Sigmoid :** $K(\mathbf{x}, \mathbf{x}') = \tanh(\gamma\mathbf{x}^T\mathbf{x}' + r)$

Le kernel RBF est le plus populaire (frontières très flexibles).

6.4 Hyperparamètres

- `C` : Régularisation (trade-off marge vs erreurs)
- `kernel` : Type de kernel
- `gamma` (pour RBF) : Inverse du rayon d'influence
  - γ petit : Influence large, modèle simple
  - γ grand : Influence locale, modèle complexe (overfitting)

6.5 Avantages et Limites

Avantages :

- Efficace en haute dimension
- Robuste à l'overfitting (régularisation C)
- Versatile (différents kernels)
- Bonne performance théorique

Limites :

- Coût d'entraînement : $O(n^2)$ à $O(n^3)$ (mauvais pour gros datasets)
- Sensible au choix de kernel et hyperparamètres
- Nécessite normalisation des features
- Difficile à interpréter

```python
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

# Normalisation importante pour SVM
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# SVM avec kernel RBF
svm = SVC(kernel='rbf', C=1.0, gamma='scale')
svm.fit(X_train_scaled, y_train)
y_pred = svm.predict(X_test_scaled)
```

Listing 5 – SVM

Table 1 – Comparaison des algorithmes de classification

Algorithme	Interprétabilité	Vitesse	Performance	Scalabilité
KNN	Moyenne	Lente (test)	Bonne	Mauvaise
Arbre de décision	Excellente	Rapide	Moyenne	Bonne
Random Forest	Faible	Moyenne	Excellente	Bonne
Gradient Boosting	Faible	Lente (train)	Excellente	Bonne
SVM	Faible	Lente (train)	Bonne	Mauvaise

7 Comparaison des Algorithmes

(i) Astuce

Recommandations pratiques :

- **Baseline rapide :** KNN, arbre de décision
- **Haute performance :** Random Forest, Gradient Boosting (XGBoost/LightGBM)
- **Interprétabilité :** Arbre de décision peu profond
- **Haute dimension :** SVM (kernel RBF)
- **Gros datasets :** LightGBM, régression logistique

8 Résumé

8.1 Points Clés

- **KNN :** Simple, basé instance, nécessite normalisation
- **Arbres :** Interprétables, overfitting facile, frontières orthogonales
- **Random Forest :** Bagging + randomisation, excellente performance, réduit overfitting
- **Gradient Boosting :** Boosting séquentiel, état de l'art (XGBoost, LightGBM)
- **SVM :** Maximum margin, kernel trick, efficace haute dimension
- **Ensemble learning :** Combiner modèles améliore performance

9 Exercices

Voir notebooks `04_exercices.ipynb`

10 Pour Aller Plus Loin

Chapitre suivant : **Chapitre 05 - Apprentissage Non-Supervisé**

Références

1. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning.* Springer.
2. James, G. et al. (2021). *An Introduction to Statistical Learning* (2e éd.). Springer.
3. Géron, A. (2022). *Hands-On Machine Learning* (3e éd.). O'Reilly.