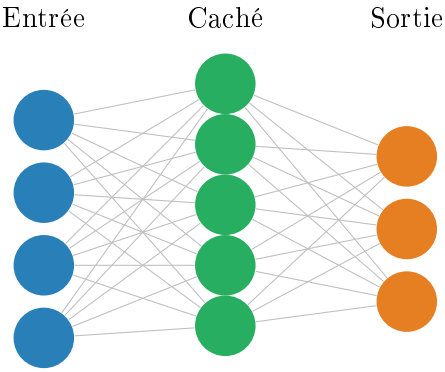
Guide Complet d'Apprentissage en Intelligence Artificielle

Volume 1

Fondations Mathématiques et Machine Learning



De Débutant à Expert

Modou DIAGNE

Élève Ingénieur IA/Data

www.linkedin.com/in/modou-diagne-94b451247

2026

Table des matières

Table des figures

Liste des tableaux

List of Algorithms

Préface

Bienvenue dans ce guide complet d'apprentissage en Intelligence Artificielle. Ce premier volume est consacré aux **fondations mathématiques** essentielles et au **Machine Learning classique**.

Objectifs de ce Guide

Ce guide a été conçu pour vous accompagner de manière progressive, depuis les bases mathématiques jusqu'à l'implémentation de modèles de Machine Learning sophistiqués. Chaque concept est présenté avec :

— Une **explication intuitive** pour comprendre le "pourquoi"
— Une **formalisation mathématique** rigoureuse
— Des **exemples concrets** et visualisations
— Du **code Python** implémentant les concepts
— Des **exercices pratiques** pour consolider vos acquis

Prérequis

Pour tirer le meilleur parti de ce guide, vous devriez avoir :

— Des notions de base en mathématiques (niveau lycée)
— Une familiarité avec la programmation (idéalement Python)
— De la curiosité et de la persévérance !

Structure du Guide

Ce volume est organisé en **quatre parties** :

1. **Partie I : Introduction et Prérequis** – Présentation de l'IA et maîtrise de Python pour la Data Science
2. **Partie II : Fondations Mathématiques** – Les outils mathématiques indispensables

3. **Partie III : Apprentissage Supervisé** – Régression, Classification, et algorithmes classiques
4. **Partie IV : Apprentissage Non Supervisé et Évaluation** – Clustering, réduction dimensionnelle, et métriques

Comment Utiliser ce Guide

Astuce

Nous vous recommandons de suivre ce guide de manière **séquentielle**, car chaque chapitre s'appuie sur les précédents. N'hésitez pas à revenir sur les concepts mathématiques lorsque vous en avez besoin.

Bonne lecture et bon apprentissage !

Introduction : Parcours vers l'IA

Qu'est-ce que l'Intelligence Artificielle ?

L'**Intelligence Artificielle** (IA) est un domaine de l'informatique qui vise à créer des systèmes capables d'effectuer des tâches qui nécessitent normalement l'intelligence humaine.

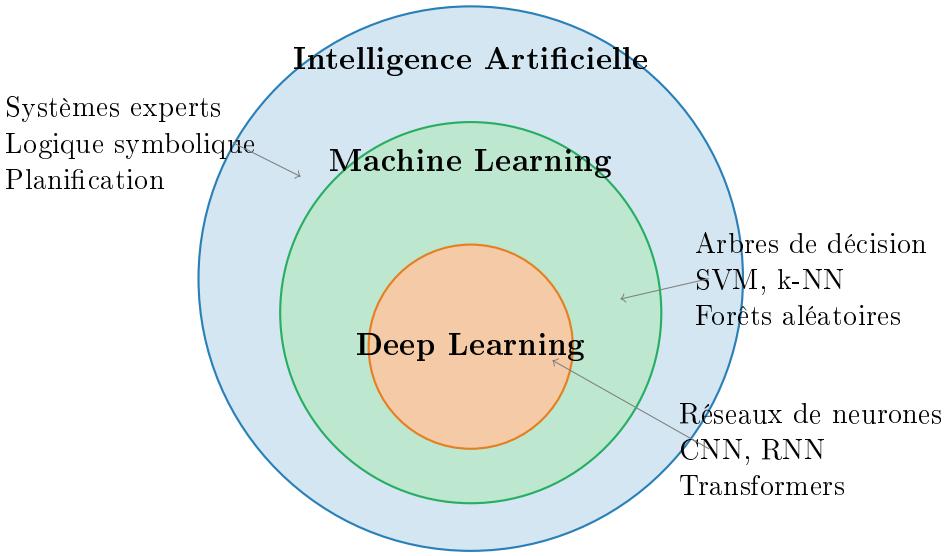


FIGURE 1 – Hiérarchie des domaines de l'IA

Plan d'Apprentissage

Voici le parcours recommandé pour maîtriser l'IA :

Pourquoi les Mathématiques sont Essentielles

Les mathématiques constituent le **langage** de l'IA. Comprendre les fondements mathématiques vous permettra de :

1. **Comprendre** les algorithmes en profondeur, pas juste les utiliser

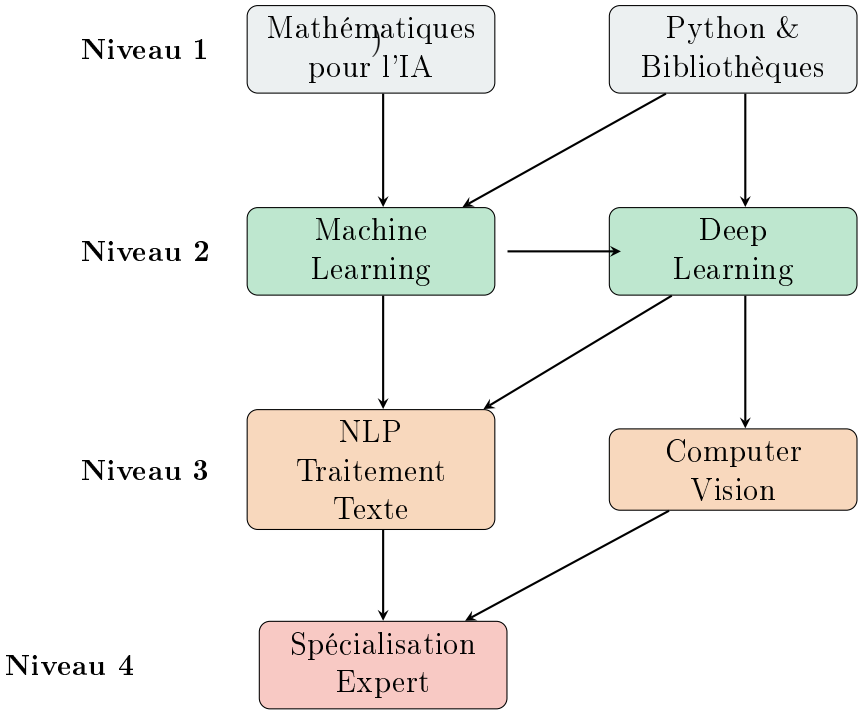


FIGURE 2 – Plan d'apprentissage en IA

2. **Diagnostiquer** les problèmes et optimiser les performances
3. **Innover** en adaptant ou créant de nouveaux algorithmes
4. **Communiquer** avec la communauté scientifique

Important

Ne sautez pas les chapitres mathématiques ! Même si vous êtes impatient d'arriver au Machine Learning, ces fondations sont **indispensables** pour une compréhension profonde.

Première partie

Introduction à l'IA et Prérequis

Chapitre 1

Introduction à l'Intelligence Artificielle

Objectifs du chapitre

Ce chapitre introductif présente une vue d'ensemble de l'Intelligence Artificielle (IA), son histoire, ses différentes branches et ses applications. Nous verrons comment l'IA, le Machine Learning et le Deep Learning s'articulent, et nous poserons les bases pour comprendre la place du Machine Learning dans cet écosystème plus large.

1.1 Qu'est-ce que l'Intelligence Artificielle ?

Définition 1.1 (Intelligence Artificielle)**.** L'**Intelligence Artificielle** (IA) est le domaine de l'informatique qui cherche à créer des systèmes capables de réaliser des tâches qui, normalement, nécessitent l'intelligence humaine : perception, raisonnement, apprentissage, prise de décision, résolution de problèmes, et compréhension du langage naturel.

1.1.1 Les Différentes Définitions de l'IA

Historiquement, plusieurs définitions ont été proposées :

1. **Approche comportementale (Test de Turing, 1950)** : Une machine est intelligente si son comportement est indiscernable de celui d'un humain.
2. **Approche cognitive** : L'IA cherche à reproduire les processus cognitifs humains.
3. **Approche pragmatique** : L'IA crée des systèmes qui accomplissent des tâches utiles de manière efficace.
4. **Approche rationnelle** : L'IA crée des agents qui maximisent l'atteinte de leurs objectifs étant donné leurs connaissances.

Intuition

L'IA n'est pas une technologie unique, mais plutôt un **ensemble de techniques** et d'**approches** visant à créer des systèmes "intelligents". C'est un domaine parapluie qui englobe de nombreuses sous-disciplines.

1.2 Histoire de l'Intelligence Artificielle

1.2.1 Les Pionniers (1940-1960)

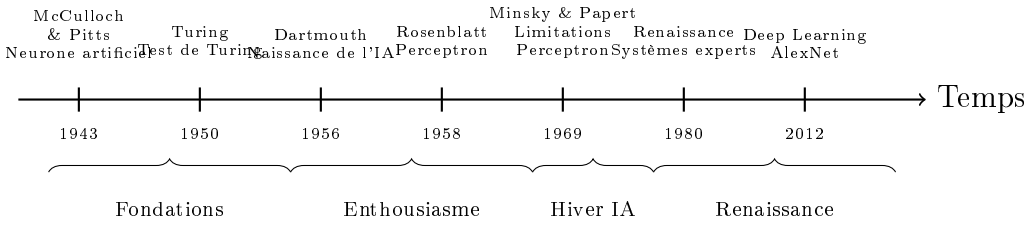


FIGURE 1.1 – Chronologie simplifiée de l'Intelligence Artificielle

1943 : Le Neurone Artificiel

Warren McCulloch et Walter Pitts proposent le premier modèle mathématique d'un neurone artificiel :

$$y = \begin{cases} 1 & \text{si } \sum_{i=1}^{n} w_i x_i \geq \theta \\ 0 & \text{sinon} \end{cases} \tag{1.1}$$

où x_i sont les entrées, w_i les poids synaptiques, et θ le seuil d'activation.

1950 : Le Test de Turing

Alan Turing propose le célèbre **Test de Turing** : une machine est intelligente si un interrogateur humain ne peut pas distinguer ses réponses de celles d'un humain.

1956 : La Conférence de Dartmouth

John McCarthy organise la conférence de Dartmouth où le terme "Intelligence Artificielle" est officiellement adopté. Les participants incluent Marvin Minsky, Claude Shannon, et Herbert Simon.

Astuce

La conférence de Dartmouth est considérée comme l'**acte de naissance** de l'IA en tant que discipline académique.

1.2.2 Les Hivers et les Renaissances de l'IA

L'histoire de l'IA est marquée par des cycles d'enthousiasme et de déception :

Période	**Caractéristique**	**Technologies clés**
1956-1974	Enthousiasme initial	Perceptron, LISP
1974-1980	Premier hiver IA	Critiques du Perceptron
1980-1987	Renaissance	Systèmes experts
1987-1993	Second hiver IA	Limites des systèmes experts
1993-2011	Renouveau discret	SVM, Forêts aléatoires
2012-présent	Ère du Deep Learning	CNN, RNN, Transformers

TABLE 1.1 – Les cycles de l'Intelligence Artificielle

1.3 Les Branches de l'Intelligence Artificielle

1.3.1 IA Symbolique vs IA Connexionniste

Il existe deux grandes approches en IA :

IA Symbolique	**IA Connexionniste**
Représentation explicite des connaissances	Apprentissage à partir de données
Règles logiques (si-alors)	Réseaux de neurones
Raisonnement déductif	Reconnaissance de patterns
Interprétable	Boîte noire
Nécessite l'expertise humaine	Nécessite beaucoup de données
Exemples : Systèmes experts, PROLOG	Exemples : Deep Learning, ML

TABLE 1.2 – Comparaison IA symbolique vs connexionniste

Attention

Aujourd'hui, la tendance est à l'**IA hybride** qui combine les avantages des deux approches : la puissance d'apprentissage du Deep Learning avec la capacité de raisonnement explicite de l'IA symbolique.

1.4 La Hiérarchie : IA, Machine Learning, Deep Learning

1.4.1 Intelligence Artificielle (IA)

Définition 1.2 (IA - Définition Large)**.** L'IA englobe **toute technique** permettant à une machine de simuler un comportement intelligent, qu'il soit basé sur des règles pré-

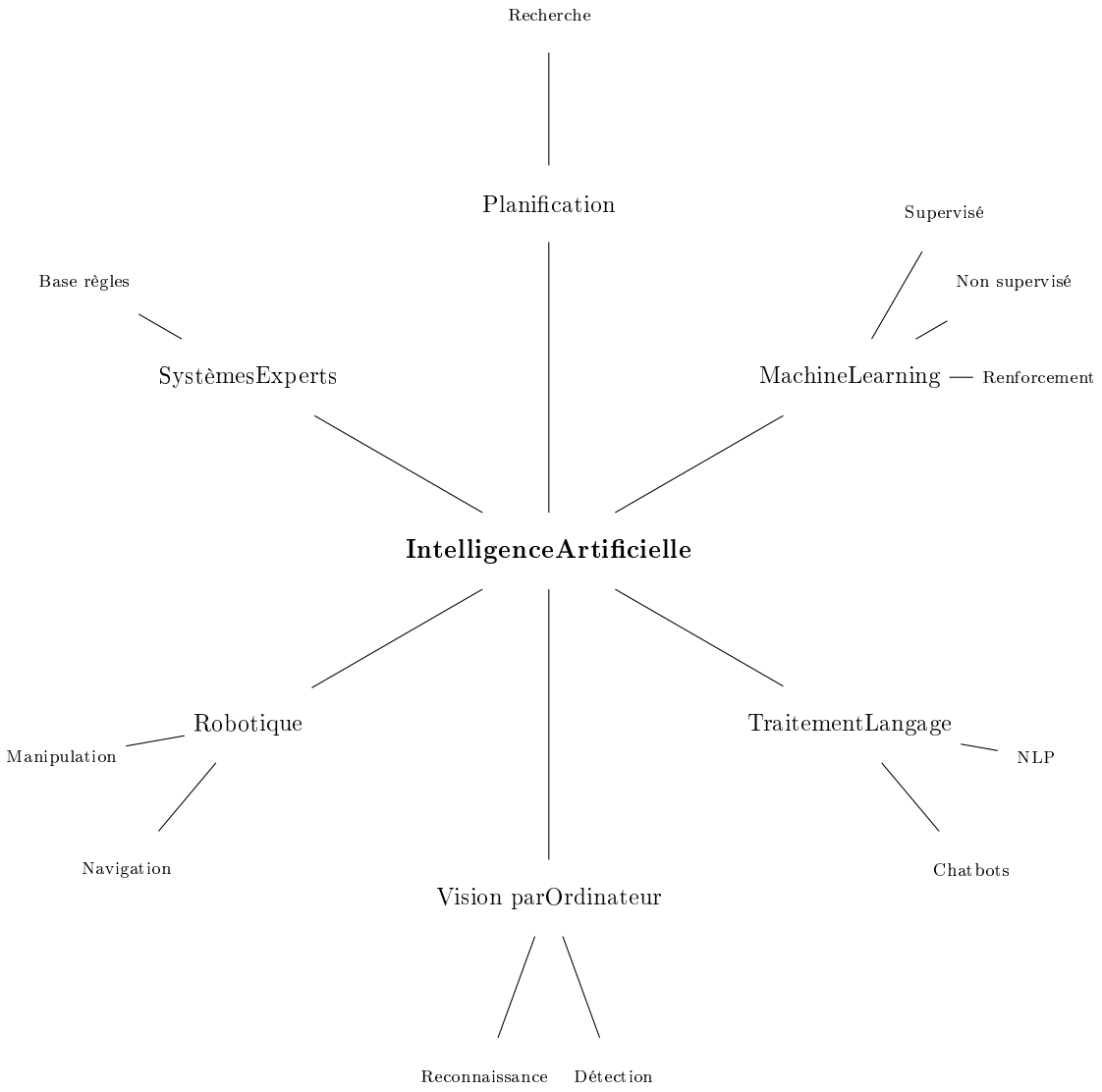


FIGURE 1.2 – Les principales branches de l'Intelligence Artificielle

programmées ou sur l'apprentissage à partir de données.

Exemples de techniques IA non-ML :

- **Systèmes experts** : Règles if-then codées par des experts humains
- **Recherche** : Algorithmes A*, minimax pour les jeux
- **Planification** : STRIPS, planificateurs automatiques
- **Logique floue** : Raisonnement avec incertitude

1.4.2 Machine Learning (ML)

Définition 1.3 (Machine Learning)**.** Le **Machine Learning** est un sous-domaine de l'IA où les systèmes **apprennent automatiquement** à partir de données, sans être explicitement programmés pour chaque tâche.

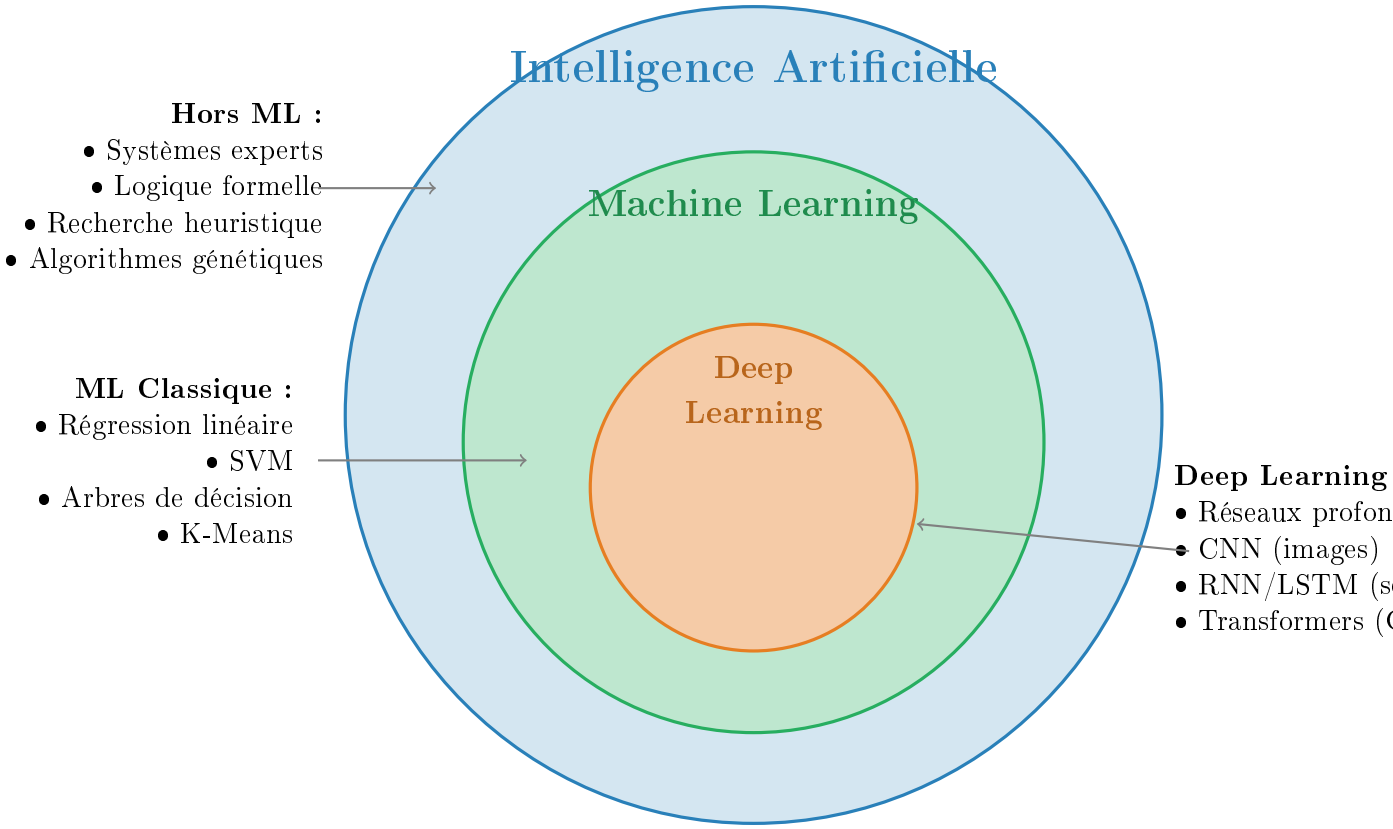


FIGURE 1.3 – Relation d'inclusion : IA ⊃ ML ⊃ DL

$$\text{Programme traditionnel :} \quad \text{Règles} + \text{Données} \rightarrow \text{Réponses} \tag{1.2}$$

$$\text{Machine Learning :} \quad \text{Données} + \text{Réponses} \rightarrow \text{Règles (Modèle)} \tag{1.3}$$

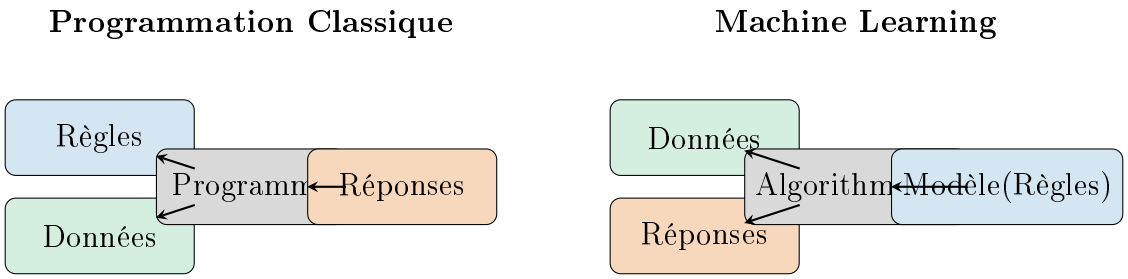


FIGURE 1.4 – Différence entre programmation classique et Machine Learning

1.4.3 Deep Learning (DL)

Définition 1.4 (Deep Learning)**.** Le **Deep Learning** est un sous-ensemble du Machine Learning utilisant des **réseaux de neurones artificiels** avec plusieurs couches cachées (d'où "deep" = profond) pour apprendre des représentations hiérarchiques des données.

Caractéristiques du Deep Learning :

— Apprentissage automatique de caractéristiques (features)

— Nécessite de grandes quantités de données

— Nécessite une puissance de calcul importante (GPU)

— Excellents résultats en vision, NLP, audio

1.5 Les Types d'Apprentissage en Machine Learning

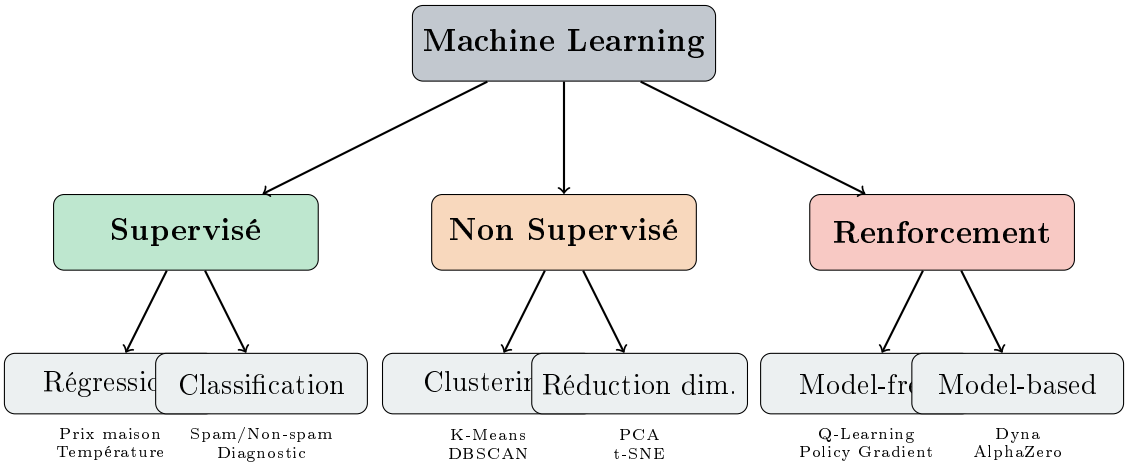


FIGURE 1.5 – Taxonomie des types d'apprentissage en Machine Learning

1.5.1 Apprentissage Supervisé

Définition 1.5 (Apprentissage Supervisé)**.** L'algorithme apprend à partir de données **étiquetées** : chaque exemple d'entraînement est une paire (entrée, sortie attendue). L'objectif est d'apprendre une fonction $f : X \to Y$ qui prédit la sortie pour de nouvelles entrées.

$$\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\} \tag{1.4}$$

— **Régression** : $y \in \mathbb{R}$ (variable continue)

— **Classification** : $y \in \{c_1, c_2, \ldots, c_K\}$ (classes discrètes)

Exemple 1.1.
— **Régression** : Prédire le prix d'une maison à partir de ses caractéristiques

— **Classification** : Détecter si un email est spam ou non

1.5.2 Apprentissage Non Supervisé

Définition 1.6 (Apprentissage Non Supervisé)**.** L'algorithme apprend à partir de données **sans étiquettes**. L'objectif est de découvrir des structures cachées ou des patterns dans les données.

$$\mathcal{D} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\} \tag{1.5}$$

— **Clustering** : Regrouper les données similaires

— **Réduction de dimensionnalité** : Compresser les données en gardant l'essentiel

— **Détection d'anomalies** : Identifier les données atypiques

1.5.3 Apprentissage par Renforcement

Définition 1.7 (Apprentissage par Renforcement)**.** Un **agent** apprend à prendre des **actions** dans un **environnement** pour maximiser une **récompense cumulative**. Il n'y a pas de supervision directe, mais des récompenses/pénalités.

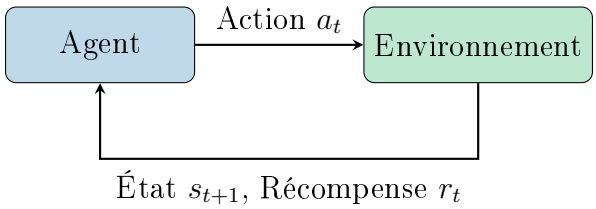


FIGURE 1.6 – Cycle d'interaction en apprentissage par renforcement

1.6 Applications de l'Intelligence Artificielle

L'IA est aujourd'hui omniprésente. Voici quelques domaines d'application majeurs :

1.6.1 Vision par Ordinateur

— **Reconnaissance d'images** : Classification, détection d'objets

— **Reconnaissance faciale** : Authentification, surveillance

— **Véhicules autonomes** : Perception de l'environnement

— **Imagerie médicale** : Détection de tumeurs, analyse de radiographies

1.6.2 Traitement du Langage Naturel (NLP)

— **Traduction automatique** : Google Translate, DeepL

— **Chatbots et assistants** : ChatGPT, Siri, Alexa

— **Analyse de sentiments** : Opinion mining sur réseaux sociaux
— **Résumé automatique** : Condensation de documents

1.6.3 Domaines Spécialisés

Domaine	Applications IA
Santé	Diagnostic, découverte de médicaments, chirurgie robotique
Finance	Trading algorithmique, détection de fraude, scoring crédit
Commerce	Recommandations, pricing dynamique, chatbots support
Industrie	Maintenance prédictive, contrôle qualité, optimisation
Transport	Véhicules autonomes, optimisation de routes, logistique
Éducation	Tuteurs intelligents, correction automatique, personnalisation

TABLE 1.3 – Applications de l'IA par domaine

1.7 Les Défis et Enjeux de l'IA

1.7.1 Défis Techniques

— **Interprétabilité** : Comprendre les décisions des modèles (explainable AI)
— **Robustesse** : Résister aux attaques adverses et aux cas limites
— **Généralisation** : Bien performer sur des données jamais vues
— **Efficacité** : Réduire les besoins en données et en calcul

1.7.2 Enjeux Éthiques

Attention

L'IA soulève des questions éthiques importantes :

— **Biais algorithmiques** : Les modèles peuvent perpétuer ou amplifier les biais présents dans les données
— **Vie privée** : Collecte massive de données personnelles
— **Emploi** : Automatisation et transformation du marché du travail
— **Autonomie des armes** : Utilisation militaire de l'IA
— **Désinformation** : Deepfakes, génération de faux contenus

1.8 Pourquoi le Machine Learning est au Cœur de l'IA Moderne

Le Machine Learning est devenu le paradigme dominant de l'IA pour plusieurs raisons :

1. **Disponibilité des données** : Explosion des données (Big Data)
2. **Puissance de calcul** : GPU, TPU, cloud computing
3. **Algorithmes avancés** : Deep Learning, techniques d'optimisation
4. **Outils accessibles** : Frameworks open source (TensorFlow, PyTorch, scikit-learn)
5. **Résultats impressionnants** : Performances surhumaines dans certaines tâches

Important

Ce guide se concentre sur le **Machine Learning** car c'est :

— La branche la plus **pratique et applicable** de l'IA aujourd'hui

— La **fondation** nécessaire pour comprendre le Deep Learning

— Le **cœur** de la plupart des systèmes IA modernes

1.9 Prérequis pour ce Guide

Pour suivre ce guide efficacement, voici les prérequis recommandés :

Domaine	**Niveau requis**	**Couvert dans ce guide ?**
Mathématiques de base	Lycée	Rappels fournis
Algèbre linéaire	Notions	Chapitre détaillé
Probabilités/Statistiques	Notions	Chapitre détaillé
Calcul différentiel	Notions	Chapitre détaillé
Programmation Python	Bases	Chapitre dédié

TABLE 1.4 – Prérequis recommandés

Synthèse du Chapitre

Synthèse

Dans ce chapitre introductif, nous avons couvert :

— **Définition de l'IA** : Domaine visant à créer des systèmes "intelligents"

— **Histoire** : Des pionniers des années 1940 aux Transformers modernes

— **Hiérarchie IA $\supset$ ML $\supset$ DL** :

  — IA : Tout système intelligent

  — ML : Apprentissage à partir de données

  — DL : Réseaux de neurones profonds

— **Types d'apprentissage** :

  — Supervisé (avec étiquettes)

  - Non supervisé (sans étiquettes)
  - Par renforcement (récompenses)
- **Applications** : Vision, NLP, santé, finance, etc.
- **Enjeux** : Interprétabilité, biais, éthique

Prochaine étape : Avant de plonger dans les algorithmes de ML, nous devons maîtriser les outils de base - Python et ses bibliothèques de Data Science.

Exercices

Exercice 1.1 (QCM - Compréhension).

1. Laquelle de ces affirmations est correcte ?
   a) Le Deep Learning est plus général que le Machine Learning
   b) Le Machine Learning est un sous-ensemble de l'Intelligence Artificielle
   c) L'Intelligence Artificielle est un sous-ensemble du Machine Learning
   d) Le Machine Learning et le Deep Learning sont des domaines séparés
2. Un système qui apprend à jouer aux échecs en jouant contre lui-même utilise :
   a) L'apprentissage supervisé
   b) L'apprentissage non supervisé
   c) L'apprentissage par renforcement
   d) Un système expert
3. La détection de groupes de clients similaires dans une base de données est un problème de :
   a) Régression
   b) Classification
   c) Clustering
   d) Renforcement

Exercice 1.2 (Réflexion).

1. Identifiez 3 applications d'IA que vous utilisez quotidiennement. Pour chacune, déterminez s'il s'agit probablement de Machine Learning, et si oui, quel type d'apprentissage.
2. Pourquoi pensez-vous que le Deep Learning a connu un tel succès après 2012 alors que les réseaux de neurones existaient depuis les années 1950 ?
3. Quels pourraient être les biais dans un système de recrutement basé sur l'IA entraîné sur les décisions de recrutement passées ?

Chapitre 2

Python pour la Data Science

Objectifs du chapitre

Ce chapitre présente les outils Python essentiels pour la Data Science et le Machine Learning. Nous verrons l'environnement Python, les bibliothèques fondamentales (NumPy, Pandas, Matplotlib, Scikit-learn), et les bonnes pratiques pour structurer vos projets d'IA.

2.1 Pourquoi Python pour l'IA ?

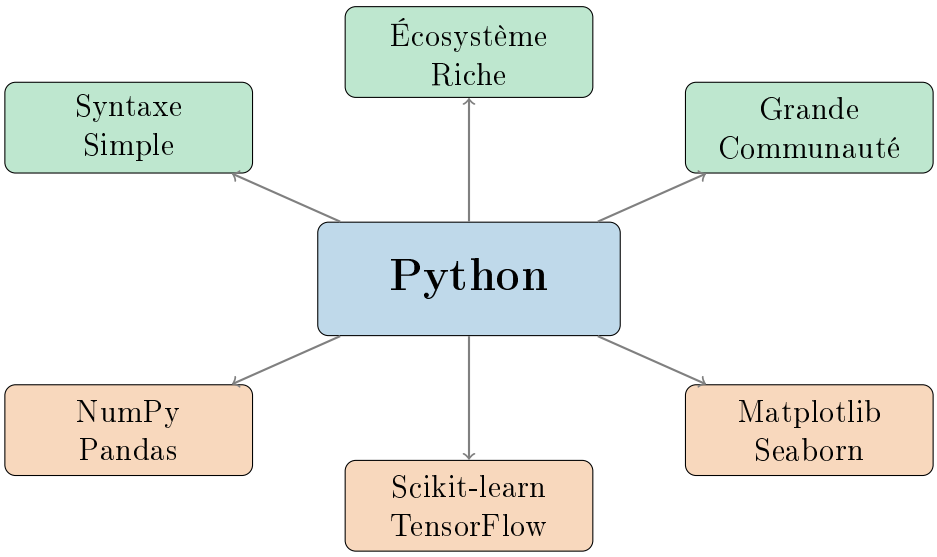


FIGURE 2.1 – Python : le langage de référence pour la Data Science

Python s'est imposé comme le langage de référence en Data Science et IA pour plusieurs raisons :

1. **Syntaxe claire et lisible** : Code proche du pseudo-code mathématique

2. **Écosystème scientifique complet** : NumPy, Pandas, Scikit-learn, TensorFlow, PyTorch
3. **Prototypage rapide** : Idéal pour l'expérimentation
4. **Communauté massive** : Documentation, tutoriels, forums
5. **Intégration facile** : Avec d'autres langages (C, Fortran) pour les performances

Astuce

Python n'est pas le langage le plus rapide, mais ses bibliothèques scientifiques (NumPy, TensorFlow) sont optimisées en C/C++ et Fortran. Vous bénéficiez ainsi de la **simplicité de Python** avec les **performances du C**.

2.2 Configuration de l'Environnement

2.2.1 Installation de Python et Anaconda

Définition 2.1 (Anaconda)**. Anaconda** est une distribution Python qui inclut automatiquement la plupart des bibliothèques scientifiques et un gestionnaire d'environnements (conda).

```
# Creer un environnement dedie a l'IA
conda create -n ia_env python=3.11

# Activer l'environnement
conda activate ia_env

# Installer les bibliotheques essentielles
conda install numpy pandas matplotlib scikit-learn jupyter

# Pour le Deep Learning
conda install pytorch torchvision -c pytorch
# ou
pip install tensorflow
```

Listing 2.1 – Installation via conda

2.2.2 Jupyter Notebook : L'Outil Interactif

Définition 2.2 (Jupyter Notebook)**. Jupyter Notebook** est un environnement interactif qui permet de combiner code, texte (Markdown), équations LaTeX et visualisations dans un même document.

```
# Lancer Jupyter Notebook
jupyter notebook
```

```
# Ou Jupyter Lab (interface plus moderne)
jupyter lab
```

Listing 2.2 – Lancement de Jupyter

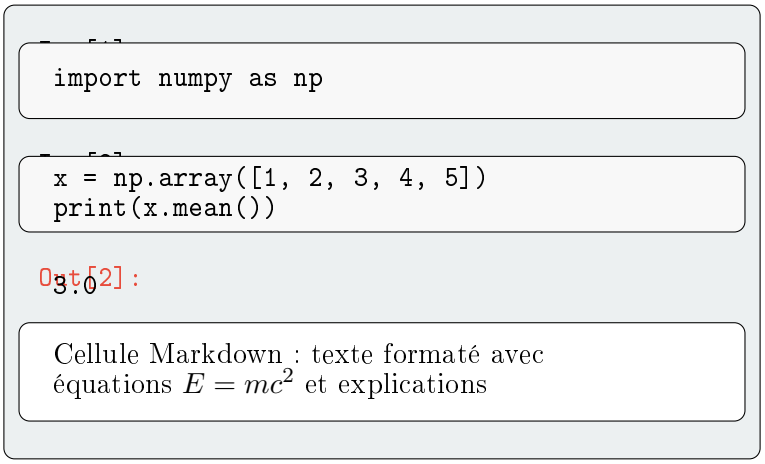


FIGURE 2.2 – Structure d'un Jupyter Notebook

Important

Jupyter Notebook est l'outil **idéal pour l'apprentissage** car il permet d'expérimenter de manière interactive et de documenter votre réflexion. Tous les exemples de ce guide peuvent être reproduits dans un notebook.

2.3 NumPy : Le Calcul Numérique

Définition 2.3 (NumPy)**. NumPy** (Numerical Python) est la bibliothèque fondamentale pour le calcul scientifique en Python. Elle fournit le type `ndarray` (tableau N-dimensionnel) et des opérations vectorisées efficaces.

2.3.1 Création de Tableaux (Arrays)

```
import numpy as np

# A partir d'une liste Python
a = np.array([1, 2, 3, 4, 5])
print(f"Array 1D: {a}")
print(f"Shape: {a.shape}")   # (5,)
print(f"Type: {a.dtype}")    # int64

# Matrice 2D
M = np.array([[1, 2, 3],
```

```
                [4, 5, 6],
                [7, 8, 9]])
print(f"Matrice 2D shape: {M.shape}")  # (3, 3)

# Fonctions de creation
zeros = np.zeros((3, 4))       # Matrice 3x4 de zeros
ones = np.ones((2, 3))         # Matrice 2x3 de uns
eye = np.eye(4)                # Matrice identite 4x4
rand = np.random.rand(3, 3)    # Matrice aleatoire uniforme [0,1]
randn = np.random.randn(3, 3) # Matrice aleatoire normale N(0,1)

# Sequences
seq = np.arange(0, 10, 2)      # [0, 2, 4, 6, 8]
lin = np.linspace(0, 1, 5)     # [0, 0.25, 0.5, 0.75, 1]
```

Listing 2.3 – Création d'arrays NumPy

2.3.2 Indexation et Slicing

```
import numpy as np

M = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12]])

# Indexation simple
print(M[0, 0])     # 1 (element ligne 0, colonne 0)
print(M[1, 2])     # 7 (element ligne 1, colonne 2)

# Slicing [debut:fin:pas]
print(M[0, :])     # [1, 2, 3, 4] (premiere ligne)
print(M[:, 0])     # [1, 5, 9] (premiere colonne)
print(M[0:2, 1:3]) # [[2, 3], [6, 7]] (sous-matrice)

# Indexation booleenne (tres puissant !)
mask = M > 5
print(mask)
# [[False False False False]
#  [False  True  True  True]
#  [ True  True  True  True]]
print(M[mask])     # [6, 7, 8, 9, 10, 11, 12]

# Fancy indexing
indices = [0, 2]
print(M[indices, :])  # Lignes 0 et 2
```

Listing 2.4 – Indexation des arrays

2.3.3 Opérations Vectorisées

Intuition

Les opérations **vectorisées** de NumPy sont exécutées en C optimisé, ce qui les rend **10 à 100 fois plus rapides** que les boucles Python équivalentes.

```python
import numpy as np

a = np.array([1, 2, 3, 4])
b = np.array([10, 20, 30, 40])

# Operations element par element
print(a + b)       # [11, 22, 33, 44]
print(a * b)       # [10, 40, 90, 160]
print(a ** 2)      # [1, 4, 9, 16]
print(np.sqrt(a)) # [1., 1.41, 1.73, 2.]
print(np.exp(a))   # [2.72, 7.39, 20.09, 54.60]
print(np.log(a))   # [0., 0.69, 1.10, 1.39]

# Broadcasting : operations entre arrays de shapes differentes
M = np.array([[1, 2, 3],
              [4, 5, 6]])
v = np.array([10, 20, 30])

print(M + v)  # Ajoute v a chaque ligne de M
# [[11, 22, 33],
#  [14, 25, 36]]
```

Listing 2.5 – Opérations vectorisées

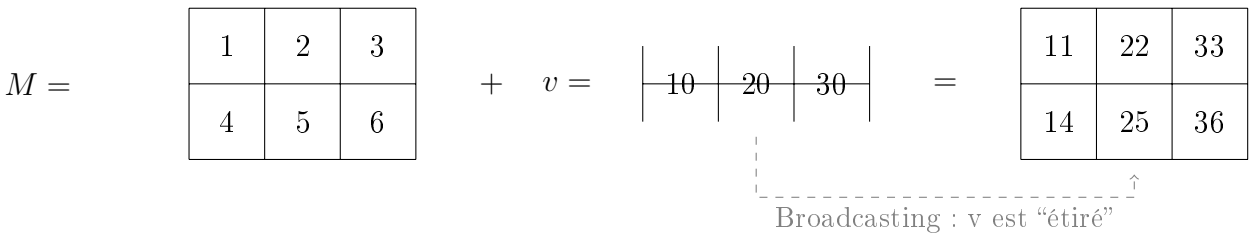


FIGURE 2.3 – Broadcasting NumPy : ajout d'un vecteur à une matrice

2.3.4 Algèbre Linéaire avec NumPy

```python
import numpy as np

A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
v = np.array([1, 2])

# Produit matriciel (3 syntaxes equivalentes)
C = np.dot(A, B)
C = A @ B            # Syntaxe recommandee (Python 3.5+)
C = np.matmul(A, B)

# Produit matrice-vecteur
y = A @ v

# Transposee
print(A.T)

# Determinant et inverse
det = np.linalg.det(A)
inv = np.linalg.inv(A)

# Valeurs propres et vecteurs propres
eigenvalues, eigenvectors = np.linalg.eig(A)

# Resolution de systeme lineaire Ax = b
b = np.array([1, 2])
x = np.linalg.solve(A, b)

# Decomposition SVD
U, S, Vt = np.linalg.svd(A)

# Norme
norm_2 = np.linalg.norm(v)         # Norme L2
norm_1 = np.linalg.norm(v, ord=1) # Norme L1
```

Listing 2.6 – Opérations d'algèbre linéaire

Attention

Le produit `A * B` fait une multiplication **élément par élément**, pas un produit matriciel ! Utilisez `A @ B` ou `np.dot(A, B)` pour le produit matriciel.

2.4 Pandas : Manipulation de Données

Définition 2.4 (Pandas)**. Pandas** est la bibliothèque de référence pour la manipulation et l'analyse de données tabulaires. Elle fournit deux structures principales : `Series` (1D) et `DataFrame` (2D).

2.4.1 Structures de Données

```python
import pandas as pd
import numpy as np

# Series : tableau 1D avec index
s = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
print(s['b'])  # 20

# DataFrame : tableau 2D avec index et colonnes
data = {
    'Nom': ['Alice', 'Bob', 'Charlie', 'Diana'],
    'Age': [25, 30, 35, 28],
    'Ville': ['Paris', 'Lyon', 'Marseille', 'Paris'],
    'Salaire': [45000, 55000, 48000, 52000]
}
df = pd.DataFrame(data)
print(df)
#       Nom  Age       Ville  Salaire
# 0   Alice   25       Paris    45000
# 1     Bob   30        Lyon    55000
# 2 Charlie   35   Marseille    48000
# 3   Diana   28       Paris    52000
```

Listing 2.7 – Series et DataFrame Pandas

2.4.2 Chargement et Exploration des Données

```python
import pandas as pd

# Chargement depuis fichiers
df = pd.read_csv('donnees.csv')
df = pd.read_excel('donnees.xlsx')
df = pd.read_json('donnees.json')

# Exploration rapide
print(df.head())         # 5 premieres lignes
print(df.tail())         # 5 dernieres lignes
print(df.shape)          # (nb_lignes, nb_colonnes)
```

```
print(df.columns)          # Noms des colonnes
print(df.dtypes)           # Types de donnees
print(df.info())           # Resume du DataFrame
print(df.describe())       # Statistiques descriptives

# Verification des valeurs manquantes
print(df.isnull().sum())
```

Listing 2.8 – Chargement et exploration

2.4.3 Sélection et Filtrage

```
# Selection de colonnes
print(df['Nom'])                # Une colonne (Series)
print(df[['Nom', 'Age']])       # Plusieurs colonnes (DataFrame)

# Selection par position (iloc)
print(df.iloc[0])               # Premiere ligne
print(df.iloc[0:3])             # Lignes 0, 1, 2
print(df.iloc[0, 1])            # Element ligne 0, colonne 1

# Selection par label (loc)
print(df.loc[0, 'Nom'])         # Element avec index 0, colonne 'Nom'
print(df.loc[0:2, ['Nom', 'Age']])

# Filtrage conditionnel
jeunes = df[df['Age'] < 30]
parisiens = df[df['Ville'] == 'Paris']
riches_jeunes = df[(df['Salaire'] > 50000) & (df['Age'] < 35)]

# Filtrage avec isin
grandes_villes = df[df['Ville'].isin(['Paris', 'Lyon'])]
```

Listing 2.9 – Sélection et filtrage

2.4.4 Transformation et Agrégation

```
# Nouvelles colonnes
df['Salaire_Mensuel'] = df['Salaire'] / 12
df['Senior'] = df['Age'] > 30

# Application de fonctions
df['Nom_Majuscule'] = df['Nom'].apply(lambda x: x.upper())
df['Age_Decade'] = df['Age'].apply(lambda x: x // 10 * 10)

```

```python
# Agregation simple
print(df['Salaire'].mean())    # Moyenne
print(df['Salaire'].median()) # Mediane
print(df['Salaire'].std())     # Ecart-type
print(df['Age'].max())         # Maximum

# GroupBy : agregation par groupe
par_ville = df.groupby('Ville')['Salaire'].mean()
print(par_ville)
# Ville
# Lyon          55000.0
# Marseille     48000.0
# Paris         48500.0

# Agregations multiples
stats = df.groupby('Ville').agg({
    'Salaire': ['mean', 'min', 'max'],
    'Age': 'mean'
})
```

Listing 2.10 – Transformations et agrégations

2.4.5 Gestion des Valeurs Manquantes

```python
import pandas as pd
import numpy as np

# Detecter les valeurs manquantes
print(df.isnull())
print(df.isnull().sum())

# Supprimer les lignes avec valeurs manquantes
df_clean = df.dropna()
df_clean = df.dropna(subset=['Salaire'])  # Seulement pour
    certaines colonnes

# Remplir les valeurs manquantes
df['Age'].fillna(df['Age'].mean(), inplace=True)  # Par la moyenne
df['Ville'].fillna('Inconnu', inplace=True)       # Par une valeur
    fixe
df['Salaire'].fillna(method='ffill', inplace=True) # Forward fill
```

Listing 2.11 – Gestion des valeurs manquantes

Synthèse

Workflow Pandas typique :

1. **Charger** les données (`read_csv`)
2. **Explorer** (`head`, `describe`, `info`)
3. **Nettoyer** (valeurs manquantes, types)
4. **Transformer** (nouvelles colonnes, filtrage)
5. **Analyser** (groupby, agrégations)
6. **Exporter** (`to_csv`)

2.5 Matplotlib et Seaborn : Visualisation

Définition 2.5 (Matplotlib)**. Matplotlib** est la bibliothèque de visualisation de base en Python. **Seaborn** est construite sur Matplotlib et offre une interface plus simple pour les visualisations statistiques.

2.5.1 Graphiques de Base avec Matplotlib

```
import matplotlib.pyplot as plt
import numpy as np

# Configuration pour les notebooks Jupyter
# %matplotlib inline  # A decommenter dans Jupyter

# Courbe simple
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)

plt.figure(figsize=(10, 6))
plt.plot(x, y, label='sin(x)', color='blue', linewidth=2)
plt.plot(x, np.cos(x), label='cos(x)', color='red', linestyle='--')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Fonctions trigonometriques')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

Listing 2.12 – Graphiques Matplotlib de base

```
import matplotlib.pyplot as plt
import numpy as np

```

```
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# 1. Nuage de points (scatter)
x = np.random.randn(100)
y = 2*x + np.random.randn(100)*0.5
axes[0, 0].scatter(x, y, alpha=0.7)
axes[0, 0].set_title('Nuage de points')

# 2. Histogramme
data = np.random.randn(1000)
axes[0, 1].hist(data, bins=30, edgecolor='black', alpha=0.7)
axes[0, 1].set_title('Histogramme')

# 3. Diagramme en barres
categories = ['A', 'B', 'C', 'D']
valeurs = [23, 45, 56, 78]
axes[1, 0].bar(categories, valeurs, color=['blue', 'green', 'red',
    'orange'])
axes[1, 0].set_title('Diagramme en barres')

# 4. Boite a moustaches (boxplot)
data = [np.random.randn(100) + i for i in range(4)]
axes[1, 1].boxplot(data, labels=['G1', 'G2', 'G3', 'G4'])
axes[1, 1].set_title('Boxplot')

plt.tight_layout()
plt.show()
```

Listing 2.13 – Types de graphiques courants

2.5.2 Seaborn pour la Visualisation Statistique

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# Charger un dataset exemple
df = sns.load_dataset('iris')

# Pairplot : relations entre toutes les variables
sns.pairplot(df, hue='species')
plt.show()

# Heatmap de correlation
plt.figure(figsize=(8, 6))
```

```
correlation = df.drop('species', axis=1).corr()
sns.heatmap(correlation, annot=True, cmap='coolwarm', center=0)
plt.title('Matrice de correlation')
plt.show()

# Distribution
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.histplot(data=df, x='sepal_length', hue='species', kde=True,
    ax=axes[0])
axes[0].set_title('Distribution de sepal_length')

sns.boxplot(data=df, x='species', y='sepal_length', ax=axes[1])
axes[1].set_title('Boxplot par espece')

plt.tight_layout()
plt.show()
```

Listing 2.14 – Visualisations Seaborn

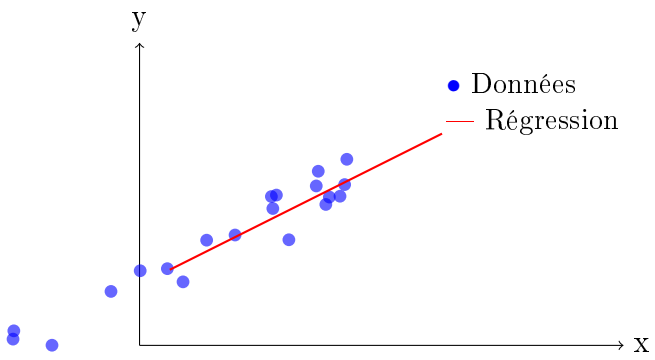


FIGURE 2.4 – Visualisation typique : nuage de points avec régression

2.6 Scikit-learn : Apprentissage Automatique

Définition 2.6 (Scikit-learn)**. Scikit-learn** (sklearn) est la bibliothèque de référence pour le Machine Learning classique en Python. Elle offre une API cohérente pour tous les algorithmes : classification, régression, clustering, réduction dimensionnelle.

2.6.1 L'API Unifiée de Scikit-learn

Tous les modèles Scikit-learn suivent la même interface :

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

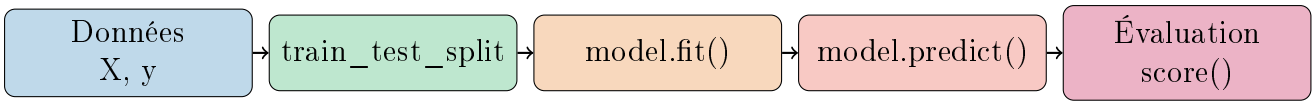


FIGURE 2.5 – Pipeline typique Scikit-learn

```python
from sklearn.metrics import mean_squared_error, r2_score

# 1. Preparer les donnees
X = ...  # Features (matrice)
y = ...  # Target (vecteur)

# 2. Separer train/test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 3. Creer et entrainer le modele
model = LinearRegression()
model.fit(X_train, y_train)

# 4. Faire des predictions
y_pred = model.predict(X_test)

# 5. Evaluer
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"MSE: {mse:.4f}, R2: {r2:.4f}")
```

Listing 2.15 – API unifiée Scikit-learn

2.6.2 Exemple Complet : Classification Iris

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt

# 1. Charger les donnees
iris = load_iris()
X, y = iris.data, iris.target
print(f"Shape X: {X.shape}, Shape y: {y.shape}")

```

```python
# 2. Separer train/test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# 3. Normaliser les features (important pour k-NN)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 4. Entrainer un classifieur k-NN
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)

# 5. Predire et evaluer
y_pred = knn.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2%}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred,
   target_names=iris.target_names))
```

Listing 2.16 – Classification sur le dataset Iris

2.6.3 Les Algorithmes Principaux

Type	Algorithme	Classe Scikit-learn
Régression	Régression Linéaire	`LinearRegression`
	Ridge / Lasso	`Ridge`, `Lasso`
	SVR	`SVR`
	Arbre de décision	`DecisionTreeRegressor`
	Random Forest	`RandomForestRegressor`
Classification	Régression Logistique	`LogisticRegression`
	k-NN	`KNeighborsClassifier`
	SVM	`SVC`
	Arbre de décision	`DecisionTreeClassifier`
	Random Forest	`RandomForestClassifier`
Clustering	K-Means	`KMeans`
	DBSCAN	`DBSCAN`
	Hiérarchique	`AgglomerativeClustering`
Réduction dim.	PCA	`PCA`
	t-SNE	`TSNE`

TABLE 2.1 – Principaux algorithmes Scikit-learn

2.7 Bonnes Pratiques en Data Science

2.7.1 Structure d'un Projet Data Science

```
mon_projet/
|-- data/
|   |-- raw/                # Donnees brutes (non modifiees)
|   |-- processed/          # Donnees nettoyees/transformees
|   `-- external/           # Donnees externes
|-- notebooks/
|   |-- 01_exploration.ipynb
|   |-- 02_preprocessing.ipynb
|   `-- 03_modeling.ipynb
|-- src/
|   |-- __init__.py
|   |-- data/               # Scripts de chargement/nettoyage
|   |-- features/           # Feature engineering
|   |-- models/             # Code des modeles
|   `-- visualization/      # Code de visualisation
|-- models/                 # Modeles entraines sauvegardes
|-- reports/
|   |-- figures/            # Graphiques pour rapports
|   `-- rapport_final.pdf
|-- requirements.txt        # Dependances Python
|-- README.md
`-- config.yaml             # Configuration du projet
```

Listing 2.17 – Structure de projet recommandée

2.7.2 Gestion de la Reproductibilité

```python
import numpy as np
import random
import os

# Fixer les seeds pour reproductibilite
SEED = 42

def set_seed(seed=SEED):
    """Fixe toutes les sources d'aleatoire."""
    np.random.seed(seed)
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    # Pour PyTorch
    # torch.manual_seed(seed)
```

```
    # torch.cuda.manual_seed_all(seed)
    # Pour TensorFlow
    # tf.random.set_seed(seed)

set_seed(42)

# Dans train_test_split, toujours specifier random_state
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=SEED
)
```

Listing 2.18 – Assurer la reproductibilité

Important

Les 3 règles d'or de la reproductibilité :

1. **Fixer les seeds** : `random_state=42` partout
2. **Documenter les dépendances** : `pip freeze > requirements.txt`
3. **Versionner le code** : Git + commits réguliers

2.7.3 Checklist Avant de Commencer un Projet ML

- ☐ **Comprendre le problème métier** : Quel est l'objectif ? Comment sera utilisé le modèle ?
- ☐ **Explorer les données** : EDA complète, visualisations, statistiques
- ☐ **Vérifier la qualité** : Valeurs manquantes, outliers, cohérence
- ☐ **Définir la métrique** : Quelle métrique optimiser ?
- ☐ **Établir une baseline** : Modèle simple de référence
- ☐ **Séparer les données** : Train/Validation/Test AVANT toute transformation
- ☐ **Itérer** : Feature engineering, sélection de modèle, hyperparamètres
- ☐ **Documenter** : Code commenté, notebooks clairs, README

2.8 Exercices Pratiques

Exercice 2.1 (NumPy - Opérations matricielles)**.** Soit la matrice $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$.

1. Créez cette matrice avec NumPy
2. Calculez la somme de chaque ligne et de chaque colonne
3. Normalisez chaque colonne (soustraire la moyenne, diviser par l'écart-type)

4. Calculez la trace (somme des éléments diagonaux)
5. Trouvez les valeurs propres de A

Exercice 2.2 (Pandas - Analyse de données)**.** Chargez le dataset "titanic" de Seaborn et :

1. Explorez les données : dimensions, types, valeurs manquantes
2. Calculez le taux de survie global et par classe de passager
3. Créez une nouvelle colonne "FamilySize" = SibSp + Parch + 1
4. Visualisez la distribution des âges par classe avec Seaborn
5. Identifiez les corrélations entre les variables numériques

Exercice 2.3 (Scikit-learn - Premier modèle)**.** Sur le dataset Iris :

1. Séparez les données en train (70%) et test (30%)
2. Entraînez 3 modèles : Régression Logistique, k-NN (k=5), Arbre de décision
3. Comparez leurs performances (accuracy) sur le set de test
4. Visualisez les frontières de décision pour les 2 premières features

Résumé du Chapitre

Synthèse

Ce qu'il faut retenir :

- **NumPy** : Calcul numérique efficace avec les arrays N-dimensionnels
  - Création : `np.array`, `np.zeros`, `np.random.randn`
  - Opérations : vectorisées, broadcasting, algèbre linéaire
- **Pandas** : Manipulation de données tabulaires
  - Structures : `Series` (1D) et `DataFrame` (2D)
  - Opérations : sélection, filtrage, groupby, valeurs manquantes
- **Matplotlib/Seaborn** : Visualisation de données
  - Matplotlib : graphiques de base, contrôle fin
  - Seaborn : visualisations statistiques élégantes
- **Scikit-learn** : Machine Learning
  - API unifiée : `fit()`, `predict()`, `score()`
  - Pipeline : split → preprocess → train → evaluate

Astuce

Pour aller plus loin :

— Documentation NumPy : https://numpy.org/doc/

— Documentation Pandas : https://pandas.pydata.org/docs/

— Documentation Scikit-learn : https://scikit-learn.org/

— Kaggle : Compétitions et datasets pour pratiquer

Deuxième partie

Fondations Mathématiques pour l'IA

Chapitre 3

Algèbre Linéaire pour l'Intelligence Artificielle

Objectifs du chapitre

L'algèbre linéaire est le **pilier fondamental** de l'Intelligence Artificielle. Ce chapitre vous guide à travers les concepts essentiels : vecteurs, matrices, transformations linéaires, et décompositions. Chaque notion est illustrée par son application directe en Machine Learning.

3.1 Introduction : Pourquoi l'Algèbre Linéaire ?

L'algèbre linéaire est omniprésente en IA car :

1. **Les données sont des matrices** : Un dataset de n exemples avec p caractéristiques est une matrice $n \times p$
2. **Les images sont des tenseurs** : Une image couleur est un tenseur 3D (hauteur $\times$ largeur $\times$ canaux)
3. **Les modèles sont des opérations matricielles** : Réseaux de neurones, PCA, régression...
4. **L'optimisation utilise le calcul vectoriel** : Gradient, Hessien, descente de gradient

Intuition

Imaginez que vous avez un tableau Excel avec des milliers de lignes (clients) et des dizaines de colonnes (caractéristiques). L'algèbre linéaire vous donne les outils pour manipuler, transformer et analyser ce tableau de manière **efficace et élégante**.

3.2 Scalaires, Vecteurs et Matrices

3.2.1 Scalaires

Définition 3.1 (Scalaire)**.** Un **scalaire** est un nombre unique, appartenant généralement à $\mathbb{R}$ (nombres réels) ou $\mathbb{C}$ (nombres complexes).

Notation : lettres minuscules italiques : a, b, x, α, λ

Exemple 3.1. En Machine Learning, les scalaires représentent :

— Le **taux d'apprentissage** : $\eta = 0.01$

— Une **prédiction unique** : $\hat{y} = 42.5$

— Un **coefficient** de régression : $\beta_1 = 2.3$

3.2.2 Vecteurs

Définition 3.2 (Vecteur)**.** Un **vecteur** est une liste ordonnée de nombres. Un vecteur de dimension n appartient à $\mathbb{R}^n$.

Notation : lettres minuscules en gras : $\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{\theta}$

Un vecteur colonne de dimension n :

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n \tag{3.1}$$

Un vecteur ligne (transposé) :

$$\boldsymbol{x}^\top = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix} \in \mathbb{R}^{1 \times n} \tag{3.2}$$

Exemple 3.2 (Représentation d'une donnée)**.** Un client d'une banque peut être représenté par un vecteur de caractéristiques :

$$\boldsymbol{x} = \begin{pmatrix} 35 \\ 50000 \\ 2 \\ 15 \end{pmatrix} = \begin{pmatrix} \text{Âge} \\ \text{Revenu annuel ()} \\ \text{Nombre d'enfants} \\ \text{Ancienneté (années)} \end{pmatrix}$$

Opérations sur les Vecteurs

Addition et Soustraction Pour $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$:

$$\boldsymbol{x} + \boldsymbol{y} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{pmatrix} \tag{3.3}$$

Multiplication par un Scalaire Pour $\alpha \in \mathbb{R}$ et $\boldsymbol{x} \in \mathbb{R}^n$:

$$\alpha \boldsymbol{x} = \begin{pmatrix} \alpha x_1 \\ \alpha x_2 \\ \vdots \\ \alpha x_n \end{pmatrix} \tag{3.4}$$

Produit Scalaire (Dot Product)

Définition 3.3 (Produit Scalaire)**.** Le **produit scalaire** de deux vecteurs $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ est :

$$\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x}^\top \boldsymbol{y} = \sum_{i=1}^{n} x_i y_i = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n \tag{3.5}$$

Important

Le produit scalaire est **fondamental** en ML :

- **Prédiction linéaire** : $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x} + b$
- **Similarité cosinus** : mesure de ressemblance entre documents/mots
- **Noyaux (kernels)** : base des SVM

Norme d'un Vecteur

Définition 3.4 (Norme L^p)**.** La **norme** L^p d'un vecteur $\boldsymbol{x}$ est :

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} \tag{3.6}$$

Les normes les plus utilisées en ML :

- **Norme** L^1 (Manhattan) : $\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i|$
  - Utilisée pour la **régularisation Lasso** (encourage la parcimonie)
- **Norme** L^2 (Euclidienne) : $\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$

  - Utilisée pour la **régularisation Ridge**
  - Mesure la **distance géométrique**
- **Norme** L^∞ (Max) : $\|\boldsymbol{x}\|_\infty = \max_i |x_i|$

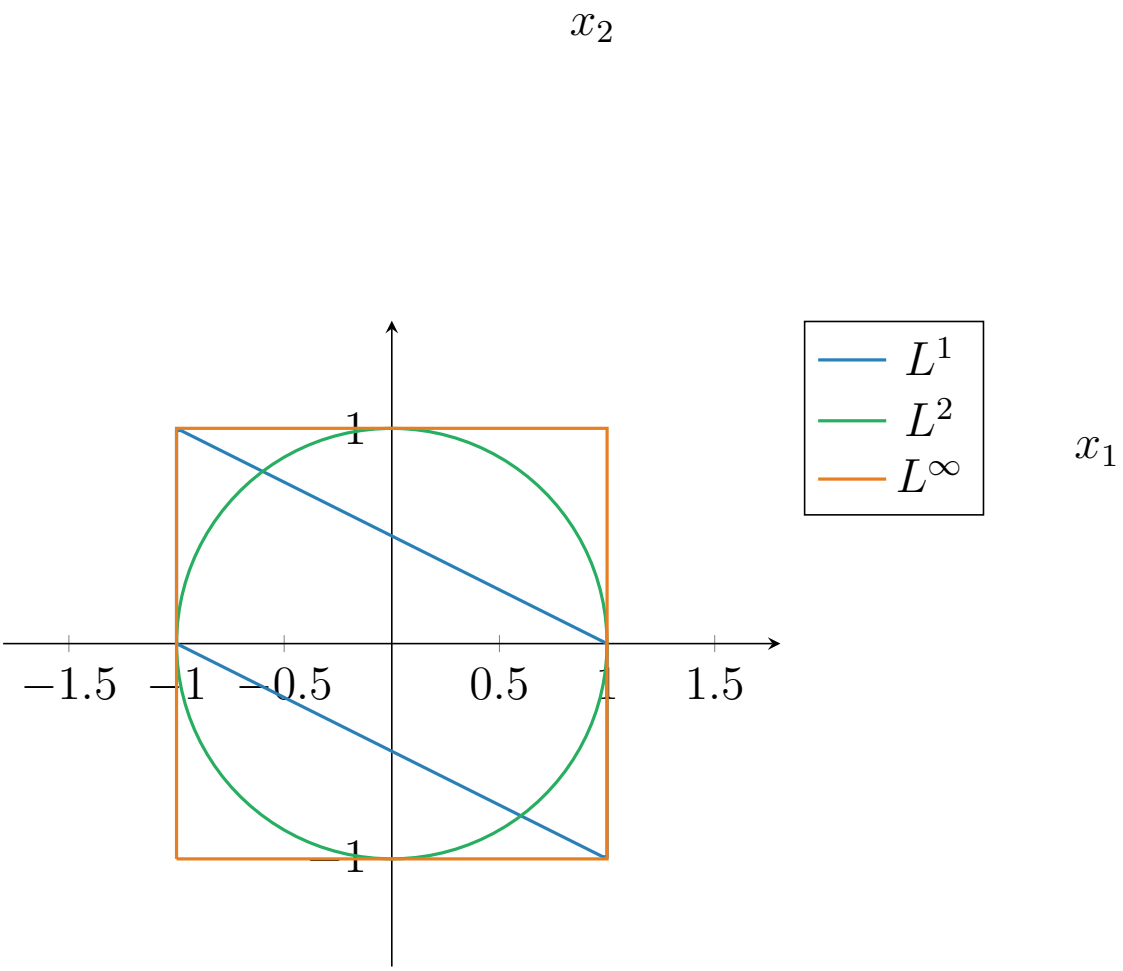


FIGURE 3.1 – Boules unitaires pour différentes normes : $\|\boldsymbol{x}\|_p = 1$

3.2.3 Matrices

Définition 3.5 (Matrice)**.** Une **matrice** est un tableau rectangulaire de nombres. Une matrice de taille $m \times n$ a m lignes et n colonnes.

Notation : lettres majuscules en gras : $\mathbf{A}, \mathbf{X}, \mathbf{W}$

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \in \mathbb{R}^{m \times n} \tag{3.7}$$

Exemple 3.3 (Matrice de Données)**.** En ML, un dataset est souvent représenté par une **matrice de design X** :

$$\mathbf{X} = \begin{pmatrix} \leftarrow & \boldsymbol{x}^{(1)\top} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(2)\top} & \rightarrow \\ & \vdots & \\ \leftarrow & \boldsymbol{x}^{(n)\top} & \rightarrow \end{pmatrix}_{n \times p}$$

où n = nombre d'exemples et p = nombre de features.

Matrices Spéciales

- **Matrice carrée** : $m = n$
- **Matrice identité $\mathbf{I}_n$** : diagonale de 1, reste 0

$$\mathbf{I}_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.8}$$

- **Matrice diagonale** : seule la diagonale est non nulle
- **Matrice symétrique** : $\mathbf{A} = \mathbf{A}^\top$
- **Matrice orthogonale** : $\mathbf{Q}\mathbf{Q}^\top = \mathbf{Q}^\top\mathbf{Q} = \mathbf{I}$

Opérations Matricielles

Transposition

$$\text{Si } \mathbf{A} \in \mathbb{R}^{m \times n}, \text{ alors } \mathbf{A}^\top \in \mathbb{R}^{n \times m} \text{ avec } (\mathbf{A}^\top)_{ij} = a_{ji} \tag{3.9}$$

Propriétés :

$$(\mathbf{A}^\top)^\top = \mathbf{A} \tag{3.10}$$

$$(\mathbf{A} + \mathbf{B})^\top = \mathbf{A}^\top + \mathbf{B}^\top \tag{3.11}$$

$$(\mathbf{AB})^\top = \mathbf{B}^\top\mathbf{A}^\top \quad \text{(attention à l'ordre !)} \tag{3.12}$$

Multiplication Matricielle

Définition 3.6 (Produit Matriciel)**.** Pour $\mathbf{A} \in \mathbb{R}^{m \times n}$ et $\mathbf{B} \in \mathbb{R}^{n \times p}$, le produit $\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times p}$ est défini par :

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \tag{3.13}$$

Attention

La multiplication matricielle n'est **pas commutative** : $\mathbf{AB} \neq \mathbf{BA}$ en général !

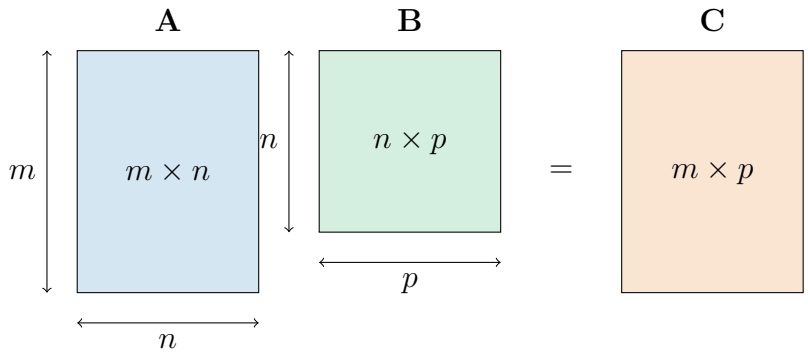


FIGURE 3.2 – Dimensions pour la multiplication matricielle

Exemple 3.4 (Prédiction avec Régression Linéaire)**.** Pour un dataset $\mathbf{X} \in \mathbb{R}^{n \times p}$ et des poids $\boldsymbol{w} \in \mathbb{R}^p$, les prédictions pour tous les exemples sont :

$$\hat{\boldsymbol{y}} = \mathbf{X}\boldsymbol{w} \in \mathbb{R}^n \tag{3.14}$$

C'est **une seule opération** matricielle au lieu de n boucles !

3.3 Systèmes d'Équations Linéaires

Définition 3.7 (Système Linéaire)**.** Un système de m équations linéaires à n inconnues :

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b} \tag{3.15}$$

où $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{b} \in \mathbb{R}^m$.

3.3.1 Inverse d'une Matrice

Définition 3.8 (Matrice Inverse)**.** Pour une matrice carrée $\mathbf{A}$, son inverse $\mathbf{A}^{-1}$ (si elle existe) satisfait :

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I} \tag{3.16}$$

Si $\mathbf{A}$ est inversible, la solution du système $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ est :

$$\boldsymbol{x} = \mathbf{A}^{-1}\boldsymbol{b} \tag{3.17}$$

Attention

En pratique, on **n'inverse jamais** explicitement une matrice ! C'est :

- Numériquement **instable**
- Computationnellement **coûteux** : $O(n^3)$

On utilise plutôt des **décompositions** (LU, QR, Cholesky).

3.3.2 Déterminant

Définition 3.9 (Déterminant)**.** Le **déterminant** d'une matrice carrée $\mathbf{A}$ est un scalaire noté $\det(\mathbf{A})$ ou $|\mathbf{A}|$.

Pour une matrice 2×2 :

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc \tag{3.18}$$

Théorème 3.1. *Une matrice* $\mathbf{A}$ *est inversible si et seulement si* $\det(\mathbf{A}) \neq 0$.

Intuition

Géométriquement, le déterminant représente le **facteur de changement de volume** lors d'une transformation linéaire. Un déterminant nul signifie que la transformation "écrase" l'espace.

3.4 Espaces Vectoriels et Sous-espaces

Définition 3.10 (Espace Vectoriel)**.** Un **espace vectoriel** V sur $\mathbb{R}$ est un ensemble muni de l'addition et de la multiplication par un scalaire, satisfaisant les axiomes standards (associativité, commutativité, distributivité, etc.).

3.4.1 Indépendance Linéaire

Définition 3.11 (Indépendance Linéaire)**.** Des vecteurs $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ sont **linéairement indépendants** si :

$$\alpha_1 \boldsymbol{v}_1 + \alpha_2 \boldsymbol{v}_2 + \cdots + \alpha_k \boldsymbol{v}_k = \mathbf{0} \implies \alpha_1 = \alpha_2 = \cdots = \alpha_k = 0 \tag{3.19}$$

Exemple 3.5. Dans $\mathbb{R}^2$, les vecteurs $\boldsymbol{e}_1 = (1, 0)$ et $\boldsymbol{e}_2 = (0, 1)$ sont indépendants.

Les vecteurs $\boldsymbol{u} = (1, 2)$ et $\boldsymbol{v} = (2, 4) = 2\boldsymbol{u}$ sont **dépendants**.

3.4.2 Base et Dimension

Définition 3.12 (Base)**.** Une **base** d'un espace vectoriel V est un ensemble de vecteurs linéairement indépendants qui engendrent V.

Définition 3.13 (Dimension)**.** La **dimension** d'un espace vectoriel est le nombre de vecteurs dans une base.

3.4.3 Rang d'une Matrice

Définition 3.14 (Rang)**.** Le **rang** d'une matrice $\mathbf{A}$, noté $\text{rang}(\mathbf{A})$ ou $\text{rk}(\mathbf{A})$, est :

— Le nombre de lignes linéairement indépendantes

— = Le nombre de colonnes linéairement indépendantes

— = La dimension de l'image de $\mathbf{A}$

Théorème 3.2 (Rang et Inversibilité). *Une matrice $\mathbf{A} \in \mathbb{R}^{n \times n}$ est inversible si et seulement si rang$(\mathbf{A}) = n$ (rang plein).*

Synthèse

Pour une matrice $\mathbf{A} \in \mathbb{R}^{m \times n}$:

— $\text{rang}(\mathbf{A}) \leq \min(m, n)$

— Si $\text{rang}(\mathbf{A}) = \min(m, n)$, on dit que $\mathbf{A}$ est de **rang plein**

— En ML, une matrice de données de rang déficient peut causer des problèmes (multicolinéarité)

3.5 Transformations Linéaires

Définition 3.15 (Transformation Linéaire). Une fonction $T : \mathbb{R}^n \to \mathbb{R}^m$ est une **transformation linéaire** si :

$$T(\boldsymbol{u} + \boldsymbol{v}) = T(\boldsymbol{u}) + T(\boldsymbol{v}) \qquad \text{(additivité)} \tag{3.20}$$

$$T(\alpha \boldsymbol{u}) = \alpha T(\boldsymbol{u}) \qquad \text{(homogénéité)} \tag{3.21}$$

Théorème 3.3. *Toute transformation linéaire $T : \mathbb{R}^n \to \mathbb{R}^m$ peut être représentée par une matrice $\mathbf{A} \in \mathbb{R}^{m \times n}$ telle que :*

$$T(\boldsymbol{x}) = \mathbf{A}\boldsymbol{x} \tag{3.22}$$

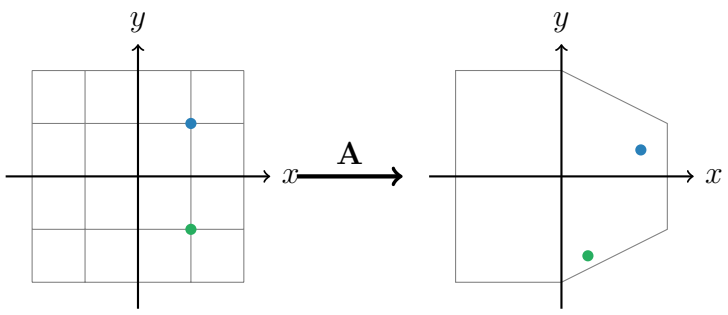


FIGURE 3.3 – Une transformation linéaire déforme l'espace de manière uniforme

Exemple 3.6 (Transformations Courantes en ML). — **Rotation** : $\mathbf{R}_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$

— **Mise à l'échelle** : $\mathbf{S} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}$

— **Projection** : Réduction dimensionnelle (PCA)

3.6 Valeurs Propres et Vecteurs Propres

Définition 3.16 (Valeurs et Vecteurs Propres)**.** Pour une matrice carrée $\mathbf{A}$, un scalaire λ est une **valeur propre** et un vecteur non nul $\boldsymbol{v}$ est un **vecteur propre** associé si :

$$\boxed{\mathbf{A}\boldsymbol{v} = \lambda\boldsymbol{v}} \tag{3.23}$$

Intuition

Les vecteurs propres sont les directions qui ne sont **pas déviées** par la transformation, seulement **étirées ou compressées** par un facteur λ.

3.6.1 Calcul des Valeurs Propres

Les valeurs propres sont les solutions de l'**équation caractéristique** :

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0 \tag{3.24}$$

Exemple 3.7. Pour $\mathbf{A} = \begin{pmatrix} 4 & 2 \\ 1 & 3 \end{pmatrix}$:

$$\begin{aligned} \det\begin{pmatrix} 4-\lambda & 2 \\ 1 & 3-\lambda \end{pmatrix} &= 0 \\ (4-\lambda)(3-\lambda) - 2 &= 0 \\ \lambda^2 - 7\lambda + 10 &= 0 \\ (\lambda - 5)(\lambda - 2) &= 0 \end{aligned}$$

Donc $\lambda_1 = 5$ et $\lambda_2 = 2$.

3.6.2 Diagonalisation

Définition 3.17 (Diagonalisation)**.** Une matrice $\mathbf{A}$ est **diagonalisable** s'il existe une matrice inversible $\mathbf{P}$ et une matrice diagonale $\mathbf{D}$ telles que :

$$\mathbf{A} = \mathbf{P}\mathbf{D}\mathbf{P}^{-1} \tag{3.25}$$

où les colonnes de $\mathbf{P}$ sont les vecteurs propres et $\mathbf{D}$ contient les valeurs propres.

Théorème 3.4 (Diagonalisation des Matrices Symétriques). *Toute matrice symétrique réelle* $\mathbf{A} = \mathbf{A}^\top$ *:*

1. *A des valeurs propres* ***réelles***
2. *Est* ***orthogonalement diagonalisable*** *:* $\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top$
3. *A des vecteurs propres* ***orthogonaux***

Important

Les matrices symétriques apparaissent partout en ML :

— Matrice de **covariance** : $\boldsymbol{\Sigma} = \frac{1}{n}\mathbf{X}^\top\mathbf{X}$

— Matrice **Hessienne** : dérivées secondes

— Matrices de **similarité/noyau**

3.6.3 Application : Analyse en Composantes Principales (PCA)

La PCA utilise les valeurs/vecteurs propres pour réduire la dimensionnalité :

1. Calculer la matrice de covariance $\boldsymbol{\Sigma}$
2. Trouver ses valeurs/vecteurs propres
3. Trier par valeur propre décroissante
4. Projeter sur les k premiers vecteurs propres

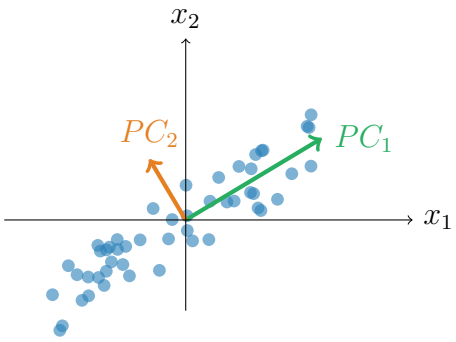


FIGURE 3.4 – PCA : les composantes principales sont les vecteurs propres de la covariance

3.7 Décomposition en Valeurs Singulières (SVD)

Définition 3.18 (SVD - Singular Value Decomposition). Toute matrice $\mathbf{A} \in \mathbb{R}^{m \times n}$ peut être décomposée en :

$$\boxed{\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top} \tag{3.26}$$

où :

— $\mathbf{U} \in \mathbb{R}^{m \times m}$: matrice orthogonale (vecteurs singuliers gauches)

— $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$: matrice diagonale (valeurs singulières $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$)

— $\mathbf{V} \in \mathbb{R}^{n \times n}$: matrice orthogonale (vecteurs singuliers droits)

Théorème 3.5. *La SVD existe pour* ***toute*** *matrice, contrairement à la diagonalisation par valeurs propres.*

3.7.1 Relation avec les Valeurs Propres

— Les valeurs singulières σ_i sont les racines carrées des valeurs propres de $\mathbf{A}^\top \mathbf{A}$

— Les colonnes de $\mathbf{V}$ sont les vecteurs propres de $\mathbf{A}^\top \mathbf{A}$

— Les colonnes de $\mathbf{U}$ sont les vecteurs propres de $\mathbf{A}\mathbf{A}^\top$

3.7.2 Applications en ML

1. **Réduction dimensionnelle** : SVD tronquée
2. **Systèmes de recommandation** : factorisation matricielle
3. **Compression d'images**
4. **Pseudo-inverse** : $\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^\top$
5. **Régression aux moindres carrés**

Exemple 3.8 (Compression d'Image)**.** Une image en niveaux de gris est une matrice. En gardant seulement les k plus grandes valeurs singulières :

$$\mathbf{A}_k = \sum_{i=1}^{k} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^\top \tag{3.27}$$

On obtient une approximation de rang k avec compression $\frac{k(m+n+1)}{mn}$.

3.8 Exercices

Exercice 3.1 (Opérations Vectorielles)**.** Soient $\boldsymbol{u} = (1, 2, 3)$ et $\boldsymbol{v} = (4, 5, 6)$. Calculer :

1. $\boldsymbol{u} + \boldsymbol{v}$
2. $3\boldsymbol{u} - 2\boldsymbol{v}$
3. $\boldsymbol{u} \cdot \boldsymbol{v}$
4. $\|\boldsymbol{u}\|_2$ et $\|\boldsymbol{v}\|_1$
5. L'angle θ entre $\boldsymbol{u}$ et $\boldsymbol{v}$

Exercice 3.2 (Multiplication Matricielle)**.** Soient $\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ et $\mathbf{B} = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$.

1. Calculer $\mathbf{AB}$ et $\mathbf{BA}$. Sont-ils égaux ?

2. Calculer $(\mathbf{AB})^\top$ et vérifier que c'est $\mathbf{B}^\top\mathbf{A}^\top$
3. Calculer $\det(\mathbf{A})$ et $\det(\mathbf{B})$

Exercice 3.3 (Valeurs Propres)**.** Pour $\mathbf{A} = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$:

1. Trouver les valeurs propres
2. Trouver les vecteurs propres correspondants
3. Vérifier que les vecteurs propres sont orthogonaux
4. Écrire la diagonalisation $\mathbf{A} = \mathbf{PDP}^{-1}$

Exercice 3.4 (Application ML)**.** Un dataset a 3 exemples avec 2 features :

$$\mathbf{X} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

1. Centrer les données (soustraire la moyenne par colonne)
2. Calculer la matrice de covariance $\mathbf{\Sigma} = \frac{1}{n-1}\mathbf{X}^\top\mathbf{X}$
3. Trouver les composantes principales (vecteurs propres de $\mathbf{\Sigma}$)

3.9 Implémentation Python

```python
import numpy as np

# --- Cr ation de vecteurs et matrices ---
v = np.array([1, 2, 3])
u = np.array([4, 5, 6])
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

# --- Operations vectorielles ---
print("Addition:", v + u)
print("Produit scalaire:", np.dot(v, u))  # ou v @ u
print("Norme L2:", np.linalg.norm(v))
print("Norme L1:", np.linalg.norm(v, ord=1))

# --- Operations matricielles ---
print("Produit matriciel:", A @ B)
print("Transpose:", A.T)
print("Determinant:", np.linalg.det(A))
print("Inverse:", np.linalg.inv(A))

# --- Valeurs et vecteurs propres ---
```

```python
eigenvalues, eigenvectors = np.linalg.eig(A)
print("Valeurs propres:", eigenvalues)
print("Vecteurs propres:\n", eigenvectors)

# --- SVD ---
U, S, Vt = np.linalg.svd(A)
print("Valeurs singulieres:", S)

# --- Resolution de systemes ---
# Ax = b
b = np.array([5, 11])
x = np.linalg.solve(A, b)
print("Solution:", x)
```

Listing 3.1 – Algèbre linéaire avec NumPy

Synthèse

Points clés du chapitre :

- Les **vecteurs** représentent des données, les **matrices** des datasets ou transformations
- Le **produit scalaire** est au cœur des prédictions linéaires
- Les **normes** sont utilisées pour la régularisation et les distances
- Les **valeurs/vecteurs propres** révèlent la structure des données
- La **SVD** est l'outil universel pour la décomposition matricielle

Chapitre 4

Calcul Différentiel et Optimisation

Objectifs du chapitre

Le calcul différentiel est le moteur de l'apprentissage automatique. Ce chapitre couvre les dérivées, le gradient, et les techniques d'optimisation essentielles pour entraîner des modèles de ML.

4.1 Pourquoi le Calcul Différentiel en ML ?

L'apprentissage automatique consiste souvent à **minimiser une fonction de coût**. Le calcul différentiel nous donne les outils pour :

1. **Mesurer** comment les paramètres affectent l'erreur (gradient)
2. **Trouver** la direction de descente optimale
3. **Mettre à jour** les paramètres pour réduire l'erreur

Intuition

Imaginez être au sommet d'une montagne dans le brouillard. Vous voulez descendre mais ne voyez rien. La stratégie : tâtez le sol autour de vous et avancez vers la pente descendante la plus raide. C'est exactement ce que fait la **descente de gradient** !

4.2 Dérivées : Les Fondamentaux

4.2.1 Dérivée d'une Fonction à Une Variable

Définition 4.1 (Dérivée)**.** La **dérivée** d'une fonction $f : \mathbb{R} \to \mathbb{R}$ au point x est :

$$f'(x) = \frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{4.1}$$

Elle représente le **taux de variation instantané** de f.

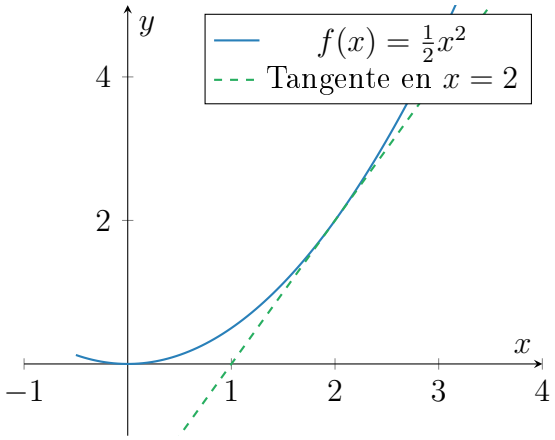


FIGURE 4.1 – La dérivée est la pente de la tangente

4.2.2 Règles de Dérivation

TABLE 4.1 – Règles de dérivation essentielles

Règle	**Fonction**	**Dérivée**
Constante	$f(x) = c$	$f'(x) = 0$
Puissance	$f(x) = x^n$	$f'(x) = nx^{n-1}$
Exponentielle	$f(x) = e^x$	$f'(x) = e^x$
Logarithme	$f(x) = \ln(x)$	$f'(x) = \frac{1}{x}$
Somme	$(f + g)'$	$f' + g'$
Produit	$(fg)'$	$f'g + fg'$
Quotient	$\left(\frac{f}{g}\right)'$	$\frac{f'g - fg'}{g^2}$
Chaîne	$(f \circ g)'$	$f'(g(x)) \cdot g'(x)$

Important

La **règle de la chaîne** est fondamentale pour la **rétropropagation** dans les réseaux de neurones !

4.2.3 Dérivées des Fonctions d'Activation

En ML, on utilise souvent ces fonctions d'activation :

$$\text{Sigmoïde} : \sigma(x) = \frac{1}{1+e^{-x}} \qquad \sigma'(x) = \sigma(x)(1-\sigma(x)) \tag{4.2}$$

$$\text{Tanh} : \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \qquad \tanh'(x) = 1-\tanh^2(x) \tag{4.3}$$

$$\text{ReLU} : \text{ReLU}(x) = \max(0,x) \qquad \text{ReLU}'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{4.4}$$

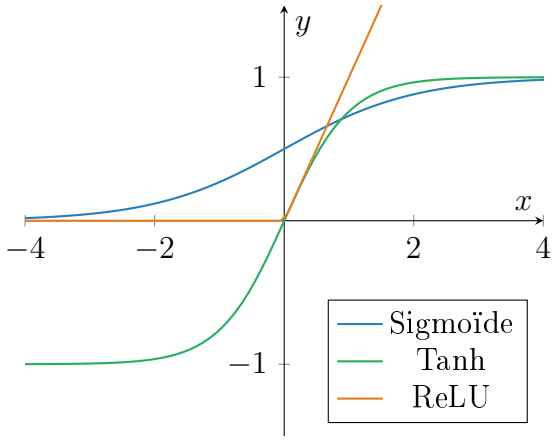


FIGURE 4.2 – Fonctions d'activation courantes

4.3 Gradient : Dérivées Multivariées

4.3.1 Dérivées Partielles

Définition 4.2 (Dérivée Partielle)**.** Pour $f : \mathbb{R}^n \to \mathbb{R}$, la **dérivée partielle** par rapport à x_i est :

$$\frac{\partial f}{\partial x_i} = \lim_{h \to 0} \frac{f(x_1, \ldots, x_i + h, \ldots, x_n) - f(x_1, \ldots, x_n)}{h} \tag{4.5}$$

On dérive par rapport à x_i en considérant les autres variables comme constantes.

Exemple 4.1. Pour $f(x, y) = x^2 + 3xy + y^2$:

$$\frac{\partial f}{\partial x} = 2x + 3y$$
$$\frac{\partial f}{\partial y} = 3x + 2y$$

4.3.2 Le Gradient

Définition 4.3 (Gradient)**.** Le **gradient** d'une fonction $f : \mathbb{R}^n \to \mathbb{R}$ est le vecteur des dérivées partielles :

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \tag{4.6}$$

Théorème 4.1 (Direction de Plus Grande Pente)**.** *Le gradient $\nabla f(\boldsymbol{x})$ pointe dans la direction de **plus grande croissance** de f au point $\boldsymbol{x}$.*

*Donc $-\nabla f(\boldsymbol{x})$ pointe vers la **plus grande décroissance**.*

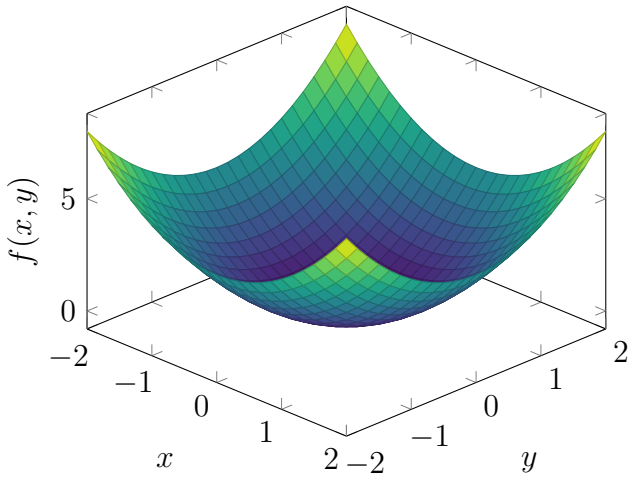


FIGURE 4.3 – Surface $f(x, y) = x^2 + y^2$ avec courbes de niveau

4.3.3 Jacobienne et Hessienne

Définition 4.4 (Matrice Jacobienne)**.** Pour $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$, la **Jacobienne** est la matrice des dérivées partielles :

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} \in \mathbb{R}^{m \times n} \tag{4.7}$$

Définition 4.5 (Matrice Hessienne)**.** Pour $f : \mathbb{R}^n \to \mathbb{R}$, la **Hessienne** est la matrice des dérivées secondes :

$$\mathbf{H} = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \tag{4.8}$$

Astuce

La Hessienne donne des informations sur la **courbure** :

- **H** définie positive ⇒ minimum local
- **H** définie négative ⇒ maximum local
- **H** indéfinie ⇒ point selle

4.4 La Descente de Gradient

4.4.1 Algorithme de Base

L'objectif : minimiser une fonction de coût $J(\boldsymbol{\theta})$ par rapport aux paramètres $\boldsymbol{\theta}$.

Algorithm 1 Descente de Gradient

Require: Fonction $J(\boldsymbol{\theta})$, taux d'apprentissage η, critère d'arrêt

1: Initialiser $\boldsymbol{\theta}$ aléatoirement
2: **while** non convergé **do**
3: Calculer le gradient $\nabla J(\boldsymbol{\theta})$
4: Mettre à jour : $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla J(\boldsymbol{\theta})$
5: **end while**
6: **return** $\boldsymbol{\theta}$

$$\boxed{\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla J(\boldsymbol{\theta}^{(t)})} \tag{4.9}$$

4.4.2 Le Taux d'Apprentissage

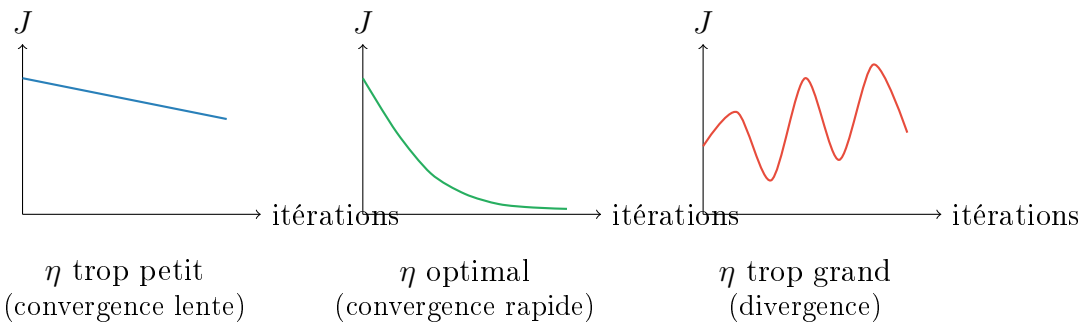


FIGURE 4.4 – Impact du taux d'apprentissage sur la convergence

4.4.3 Variantes de la Descente de Gradient

Batch Gradient Descent

Utilise **tout le dataset** pour chaque mise à jour :

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla J_i(\boldsymbol{\theta}) \tag{4.10}$$

Avantages : Stable, converge vers le minimum **Inconvénients** : Lent pour grands datasets, mémoire

Stochastic Gradient Descent (SGD)

Utilise **un seul exemple** par mise à jour :

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla J_i(\boldsymbol{\theta}) \tag{4.11}$$

Avantages : Rapide, peut échapper aux minima locaux **Inconvénients** : Bruyant, peut osciller

Mini-batch Gradient Descent

Utilise un **sous-ensemble** (batch) de B exemples :

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{1}{B} \sum_{i \in \text{batch}} \nabla J_i(\boldsymbol{\theta}) \tag{4.12}$$

Astuce

Le **mini-batch** (typiquement $B = 32, 64, 128$) est le meilleur compromis :

- Profite du parallélisme GPU
- Assez stable pour converger
- Assez bruyant pour explorer

4.4.4 Optimiseurs Avancés

Momentum

Ajoute une "inertie" pour accélérer la convergence :

$$\boldsymbol{v}^{(t)} = \gamma \boldsymbol{v}^{(t-1)} + \eta \nabla J(\boldsymbol{\theta}^{(t)}) \tag{4.13}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \boldsymbol{v}^{(t)} \tag{4.14}$$

Adam (Adaptive Moment Estimation)

Combine momentum et adaptation du learning rate :

$$\boldsymbol{m}^{(t)} = \beta_1 \boldsymbol{m}^{(t-1)} + (1-\beta_1)\nabla J \tag{4.15}$$

$$\boldsymbol{v}^{(t)} = \beta_2 \boldsymbol{v}^{(t-1)} + (1-\beta_2)(\nabla J)^2 \tag{4.16}$$

$$\hat{\boldsymbol{m}} = \frac{\boldsymbol{m}^{(t)}}{1-\beta_1^t}, \quad \hat{\boldsymbol{v}} = \frac{\boldsymbol{v}^{(t)}}{1-\beta_2^t} \tag{4.17}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta}{\sqrt{\hat{\boldsymbol{v}}} + \epsilon}\hat{\boldsymbol{m}} \tag{4.18}$$

Important

Adam est l'optimiseur par défaut recommandé pour la plupart des problèmes. Valeurs typiques : $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$.

4.5 Conditions d'Optimalité

4.5.1 Points Critiques

Définition 4.6 (Point Critique)**.** Un point $\boldsymbol{x}^*$ est un **point critique** de f si $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$.

4.5.2 Conditions du Premier Ordre

Théorème 4.2 (Condition Nécessaire)**.** *Si $\boldsymbol{x}^*$ est un minimum local de f, alors $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$.*

4.5.3 Conditions du Second Ordre

Théorème 4.3 (Condition Suffisante)**.** *Si $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$ et $\mathbf{H}(\boldsymbol{x}^*)$ est **définie positive**, alors $\boldsymbol{x}^*$ est un **minimum local strict**.*

Définition 4.7 (Matrice Définie Positive)**.** Une matrice symétrique $\mathbf{A}$ est **définie positive** si :

$$\boldsymbol{x}^\top \mathbf{A} \boldsymbol{x} > 0 \quad \forall \boldsymbol{x} \neq \mathbf{0} \tag{4.19}$$

Équivalent : toutes ses valeurs propres sont > 0.

4.5.4 Convexité

Définition 4.8 (Fonction Convexe)**.** f est **convexe** si pour tout $\boldsymbol{x}, \boldsymbol{y}$ et $\lambda \in [0,1]$:

$$f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}) \tag{4.20}$$

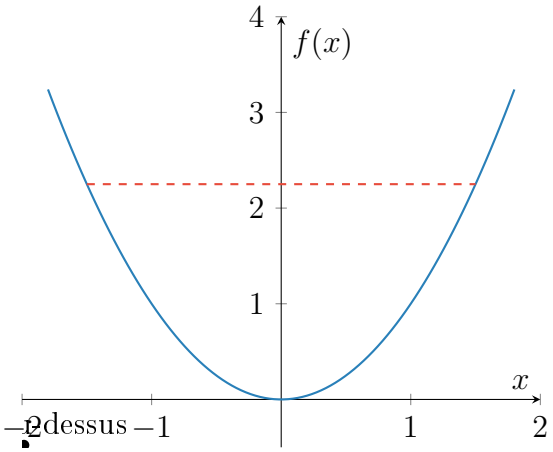


FIGURE 4.5 – Une fonction convexe : la corde est toujours au-dessus de la courbe

Théorème 4.4 (Minimum Global pour Fonctions Convexes)**.** *Pour une fonction* ***convexe****, tout minimum local est un* ***minimum global****.*

Synthèse

C'est pourquoi on préfère les problèmes convexes en ML :

- **Régression linéaire** : fonction de coût MSE est convexe
- **Régression logistique** : cross-entropy est convexe
- **SVM** : problème dual est convexe

4.6 Exercices

Exercice 4.1 (Dérivées)**.** Calculer les dérivées de :

1. $f(x) = 3x^4 - 2x^2 + x - 5$
2. $g(x) = e^{-x^2}$
3. $h(x) = \ln(1 + e^x)$ (softplus)
4. $\sigma(x) = \frac{1}{1+e^{-x}}$ (sigmoïde)

Exercice 4.2 (Gradient)**.** Pour $f(x, y, z) = x^2y + yz^2 + \ln(xyz)$:

1. Calculer ∇f
2. Évaluer ∇f au point $(1, 2, 1)$
3. Dans quelle direction f croît le plus rapidement en ce point ?

Exercice 4.3 (Descente de Gradient)**.** Soit $f(x) = x^2 - 4x + 5$.

1. Trouver le minimum analytiquement
2. Implémenter la descente de gradient avec $\eta = 0.1$, $x_0 = 0$
3. Tracer la convergence vers le minimum

Exercice 4.4 (Application ML). Pour la régression linéaire avec MSE :

$$J(\boldsymbol{w}) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2$$

1. Calculer $\nabla_{\boldsymbol{w}} J$
2. Montrer que la solution analytique est $\boldsymbol{w}^* = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\boldsymbol{y}$
3. Implémenter la descente de gradient pour trouver $\boldsymbol{w}$

4.7 Implémentation Python

```python
import numpy as np
import matplotlib.pyplot as plt

def gradient_descent(f, grad_f, x0, lr=0.01, n_iter=100):
    """
    Descente de gradient generique

    Args:
        f: fonction a minimiser
        grad_f: gradient de f
        x0: point initial
        lr: taux d'apprentissage
        n_iter: nombre d'iterations

    Returns:
        historique des x et f(x)
    """
    x = x0
    history = {'x': [x], 'f': [f(x)]}

    for _ in range(n_iter):
        grad = grad_f(x)
        x = x - lr * grad
        history['x'].append(x)
        history['f'].append(f(x))

    return x, history

# Exemple : minimiser f(x) = x^2 - 4x + 5
f = lambda x: x**2 - 4*x + 5
grad_f = lambda x: 2*x - 4

```

```
x_opt, history = gradient_descent(f, grad_f, x0=0, lr=0.1,
    n_iter=50)
print(f"Minimum trouve: x = {x_opt:.4f}, f(x) = {f(x_opt):.4f}")

# Visualisation
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Trajectoire sur la fonction
x_range = np.linspace(-1, 5, 100)
axes[0].plot(x_range, f(x_range), 'b-', label='f(x)')
axes[0].scatter(history['x'], history['f'],
    c=range(len(history['x'])),
                cmap='Reds', s=50, zorder=5)
axes[0].set_xlabel('x')
axes[0].set_ylabel('f(x)')
axes[0].set_title('Trajectoire de la descente')
axes[0].legend()

# Convergence
axes[1].plot(history['f'], 'g-o')
axes[1].set_xlabel('Iteration')
axes[1].set_ylabel('f(x)')
axes[1].set_title('Convergence')

plt.tight_layout()
plt.show()
```

Listing 4.1 – Descente de Gradient en Python

```
import numpy as np

class LinearRegressionGD:
    """Regression lineaire avec descente de gradient"""

    def __init__(self, lr=0.01, n_iter=1000):
        self.lr = lr
        self.n_iter = n_iter
        self.weights = None
        self.bias = None
        self.loss_history = []

    def fit(self, X, y):
        n_samples, n_features = X.shape

        # Initialisation
        self.weights = np.zeros(n_features)
```

```
        self.bias = 0

        for _ in range(self.n_iter):
            # Predictions
            y_pred = X @ self.weights + self.bias

            # Gradients
            dw = (1/n_samples) * X.T @ (y_pred - y)
            db = (1/n_samples) * np.sum(y_pred - y)

            # Mise a jour
            self.weights -= self.lr * dw
            self.bias -= self.lr * db

            # Loss (MSE)
            loss = np.mean((y_pred - y)**2)
            self.loss_history.append(loss)

        return self

    def predict(self, X):
        return X @ self.weights + self.bias

# Exemple d'utilisation
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X.flatten() + np.random.randn(100)

model = LinearRegressionGD(lr=0.1, n_iter=100)
model.fit(X, y)

print(f"Poids: {model.weights[0]:.4f}")
print(f"Biais: {model.bias:.4f}")
```

Listing 4.2 – Gradient pour la Régression Linéaire

Synthèse

Points clés du chapitre :

- Le **gradient** indique la direction de plus grande croissance
- La **descente de gradient** minimise en allant dans la direction opposée
- Le **taux d'apprentissage** contrôle la vitesse de convergence
- **Adam** est l'optimiseur recommandé par défaut
- Les fonctions **convexes** garantissent un minimum global unique

Chapitre 5

Probabilités et Statistiques pour le Machine Learning

Objectifs du chapitre

Les probabilités et statistiques sont au cœur du Machine Learning. Ce chapitre couvre les distributions de probabilité, l'estimation, les tests statistiques, et l'inférence bayésienne - les outils essentiels pour modéliser l'incertitude.

5.1 Fondements des Probabilités

5.1.1 Espace de Probabilité

Définition 5.1 (Espace de Probabilité)**.** Un **espace de probabilité** est un triplet $(\Omega, \mathcal{F}, P)$ où :

- Ω : **espace échantillon** (ensemble de tous les résultats possibles)
- $\mathcal{F}$: **tribu** (ensemble des événements mesurables)
- P : **mesure de probabilité** ($P : \mathcal{F} \to [0, 1]$)

5.1.2 Axiomes de Kolmogorov

1. **Non-négativité** : $P(A) \geq 0$ pour tout événement A
2. **Normalisation** : $P(\Omega) = 1$
3. **Additivité** : Si $A \cap B = \emptyset$, alors $P(A \cup B) = P(A) + P(B)$

5.1.3 Probabilité Conditionnelle

Définition 5.2 (Probabilité Conditionnelle)**.** La probabilité de A sachant B est :

$$P(A|B) = \frac{P(A \cap B)}{P(B)}, \quad P(B) > 0 \tag{5.1}$$

Théorème 5.1 (Formule de Bayes)**.**

$$\boxed{P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}} \tag{5.2}$$

Où :

- $P(A|B)$ *:* ***posterior*** *(probabilité a posteriori)*
- $P(B|A)$ *:* ***vraisemblance*** *(likelihood)*
- $P(A)$ *:* ***prior*** *(probabilité a priori)*
- $P(B)$ *:* ***évidence*** *(normalisation)*

Important

La formule de Bayes est le fondement de :

- **Classification Naïve Bayes**
- **Inférence bayésienne**
- **Filtrage spam**
- **Diagnostic médical**

Exemple 5.1 (Test Médical)**.** Un test a une sensibilité de 99% (vrai positif) et une spécificité de 95% (vrai négatif). La prévalence de la maladie est de 1%.

Si le test est positif, quelle est la probabilité d'être malade ?

$$\begin{aligned} P(\text{Malade}|\text{Test}+) &= \frac{P(\text{Test}+|\text{Malade}) \cdot P(\text{Malade})}{P(\text{Test}+)} \\ &= \frac{0.99 \times 0.01}{0.99 \times 0.01 + 0.05 \times 0.99} \\ &= \frac{0.0099}{0.0099 + 0.0495} = \frac{0.0099}{0.0594} \\ &\approx 16.7\% \end{aligned}$$

Malgré un test "très fiable", seulement 16.7% de chance d'être malade !

5.1.4 Indépendance

Définition 5.3 (Indépendance)**.** Deux événements A et B sont **indépendants** si :

$$P(A \cap B) = P(A) \cdot P(B) \tag{5.3}$$

Équivalent : $P(A|B) = P(A)$

5.2 Variables Aléatoires

Définition 5.4 (Variable Aléatoire)**.** Une **variable aléatoire** X est une fonction $X : \Omega \to \mathbb{R}$ qui associe un nombre réel à chaque résultat de l'expérience aléatoire.

5.2.1 Variables Discrètes

Définition 5.5 (PMF - Probability Mass Function)**.** Pour une variable discrète X, la **fonction de masse** est :

$$p_X(x) = P(X = x) \tag{5.4}$$

avec $\sum_x p_X(x) = 1$

Distributions Discrètes Importantes

Bernoulli

$$X \sim \text{Bernoulli}(p) : P(X = 1) = p, \quad P(X = 0) = 1 - p \tag{5.5}$$

Usage : Succès/échec, classification binaire

Binomiale

$$X \sim \text{Binomiale}(n, p) : P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k} \tag{5.6}$$

Usage : Nombre de succès sur n essais

Poisson

$$X \sim \text{Poisson}(\lambda) : P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{5.7}$$

Usage : Comptage d'événements rares

Catégorielle (Multinomiale)

$$X \sim \text{Cat}(\boldsymbol{p}) : P(X = k) = p_k, \quad \sum_k p_k = 1 \tag{5.8}$$

Usage : Classification multi-classes

5.2.2 Variables Continues

Définition 5.6 (PDF - Probability Density Function)**.** Pour une variable continue X, la **densité de probabilité** $f_X(x)$ satisfait :

$$P(a \leq X \leq b) = \int_a^b f_X(x)dx \tag{5.9}$$

avec $\int_{-\infty}^{+\infty} f_X(x)dx = 1$

Attention

Pour une variable continue, $P(X = x) = 0$ pour tout x particulier ! On parle de probabilité sur des **intervalles**.

Distributions Continues Importantes

Uniforme

$$X \sim \mathcal{U}(a, b) : f(x) = \frac{1}{b-a} \quad \text{pour } x \in [a, b] \tag{5.10}$$

Gaussienne (Normale)

$$X \sim \mathcal{N}(\mu, \sigma^2) : f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{5.11}$$

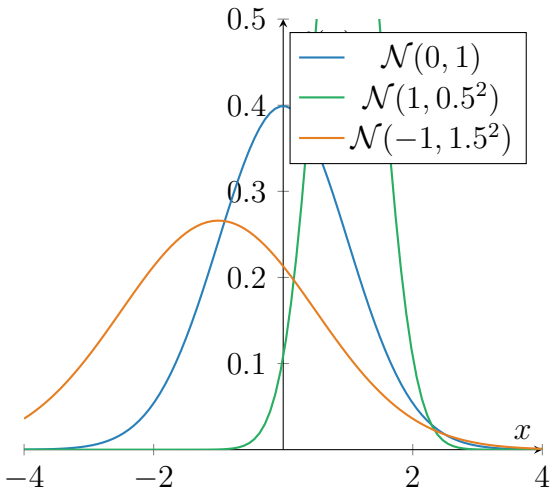


FIGURE 5.1 – Différentes distributions gaussiennes

Important

La distribution gaussienne est omniprésente en ML :

- **Théorème Central Limite** : somme de variables → gaussienne
- **Bruit** : souvent modélisé comme gaussien
- **Régression linéaire** : hypothèse de résidus gaussiens
- **Gaussian Process**

Exponentielle

$$X \sim \text{Exp}(\lambda) : f(x) = \lambda e^{-\lambda x} \quad \text{pour } x \geq 0 \tag{5.12}$$

Usage : Temps entre événements

Gamma

$$X \sim \text{Gamma}(\alpha, \beta) : f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} \tag{5.13}$$

Beta

$$X \sim \text{Beta}(\alpha, \beta) : f(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)} \quad \text{pour } x \in [0, 1] \tag{5.14}$$

Usage : Prior pour des probabilités (Bayésien)

5.3 Espérance et Moments

5.3.1 Espérance

Définition 5.7 (Espérance)**.** L'**espérance** (moyenne) d'une variable aléatoire :

$$\mathbb{E}[X] = \begin{cases} \sum_x x \cdot p_X(x) & \text{(discret)} \\ \int x \cdot f_X(x) dx & \text{(continu)} \end{cases} \tag{5.15}$$

Propriétés :

$$\mathbb{E}[aX + b] = a\mathbb{E}[X] + b \tag{5.16}$$
$$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y] \quad \text{(toujours vrai)} \tag{5.17}$$
$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y] \quad \text{(si indépendants)} \tag{5.18}$$

5.3.2 Variance et Écart-type

Définition 5.8 (Variance)**.** La **variance** mesure la dispersion autour de la moyenne :

$$\text{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2 \tag{5.19}$$

Définition 5.9 (Écart-type)**.** L'**écart-type** est $\sigma = \sqrt{\text{Var}(X)}$

Propriétés :

$$\text{Var}(aX + b) = a^2 \text{Var}(X) \tag{5.20}$$
$$\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) + 2\text{Cov}(X, Y) \tag{5.21}$$

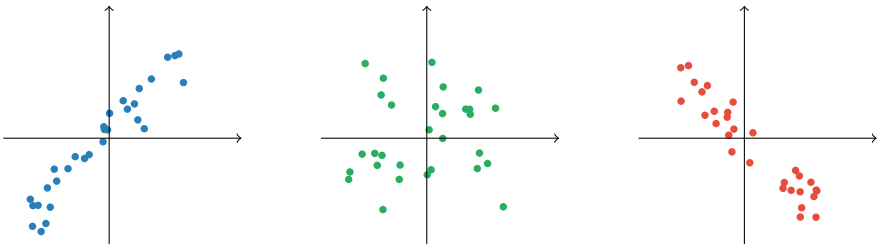
5.3.3 Covariance et Corrélation

Définition 5.10 (Covariance)**.**

$$\text{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] \tag{5.22}$$

Définition 5.11 (Corrélation)**.**

$$\rho(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\sqrt{\mathrm{Var}(X)\mathrm{Var}(Y)}} \in [-1,1] \tag{5.23}$$



$\rho \approx +0.9$ $\rho \approx 0$ $\rho \approx -0.9$

FIGURE 5.2 – Différents niveaux de corrélation

5.3.4 Tableau Récapitulatif des Distributions

TABLE 5.1 – Distributions courantes et leurs moments

Distribution	**Paramètres**	$\mathbb{E}[X]$	$\mathbf{Var}(X)$
Bernoulli	p	p	$p(1-p)$
Binomiale	n, p	np	$np(1-p)$
Poisson	λ	λ	λ
Uniforme	a, b	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
Exponentielle	λ	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$
Normale	μ, σ^2	μ	σ^2

5.4 Distribution Gaussienne Multivariée

Définition 5.12 (Gaussienne Multivariée)**.** Un vecteur aléatoire $\boldsymbol{X} \in \mathbb{R}^d$ suit une loi **gaussienne multivariée** $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ si :

$$f(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \tag{5.24}$$

où :

— $\boldsymbol{\mu} \in \mathbb{R}^d$: vecteur moyenne

— $\boldsymbol{\Sigma} \in \mathbb{R}^{d\times d}$: matrice de covariance (symétrique définie positive)

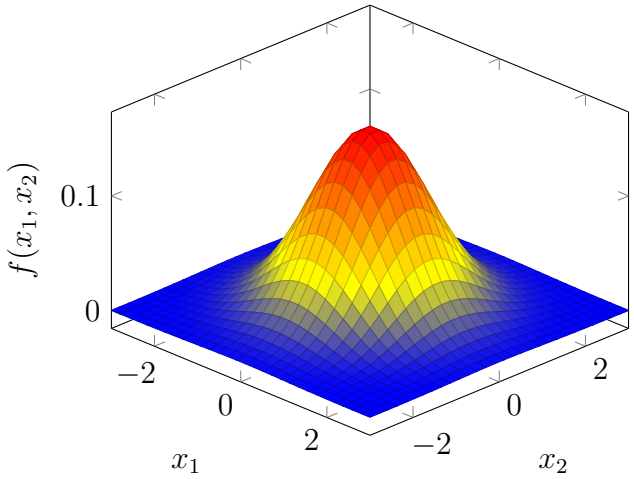


FIGURE 5.3 – Gaussienne bivariée standard $\mathcal{N}(\mathbf{0}, \mathbf{I})$

Astuce

La forme $(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})$ est la **distance de Mahalanobis** au carré. Elle généralise la distance euclidienne en tenant compte de la corrélation.

5.5 Estimation

5.5.1 Maximum de Vraisemblance (MLE)

Définition 5.13 (Vraisemblance)**.** Pour un modèle paramétrique $p(x|\theta)$ et des données $\mathcal{D} = \{x_1, \ldots, x_n\}$, la **vraisemblance** est :

$$\mathcal{L}(\theta|\mathcal{D}) = \prod_{i=1}^{n} p(x_i|\theta) \tag{5.25}$$

Définition 5.14 (Estimateur du Maximum de Vraisemblance)**.** L'estimateur MLE est :

$$\hat{\theta}_{MLE} = \arg\max_{\theta} \mathcal{L}(\theta|\mathcal{D}) = \arg\max_{\theta} \sum_{i=1}^{n} \log p(x_i|\theta) \tag{5.26}$$

Exemple 5.2 (MLE pour la Gaussienne)**.** Pour $X_1, \ldots, X_n \sim \mathcal{N}(\mu, \sigma^2)$:

$$\hat{\mu}_{MLE} = \frac{1}{n} \sum_{i=1}^{n} x_i = \bar{x} \tag{5.27}$$

$$\hat{\sigma}^2_{MLE} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2 \tag{5.28}$$

Attention

L'estimateur MLE de la variance est **biaisé**. L'estimateur non biaisé utilise $\frac{1}{n-1}$ au lieu de $\frac{1}{n}$.

5.5.2 Maximum A Posteriori (MAP)

Définition 5.15 (Estimateur MAP)**.** En incorporant un **prior** $p(\theta)$:

$$\hat{\theta}_{MAP} = \arg\max_{\theta} p(\theta|\mathcal{D}) = \arg\max_{\theta} \left[\log p(\mathcal{D}|\theta) + \log p(\theta)\right] \tag{5.29}$$

Synthèse

- **MLE** : maximise la vraisemblance seule
- **MAP** : inclut une connaissance a priori
- Si prior uniforme : MAP = MLE
- Prior gaussien sur $\theta \Rightarrow$ régularisation L_2 (Ridge)

5.6 Inférence Bayésienne

Définition 5.16 (Inférence Bayésienne)**.** Au lieu d'estimer un $\hat{\theta}$ unique, on calcule la **distribution posterior** :

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta) \cdot p(\theta)}{p(\mathcal{D})} \tag{5.30}$$

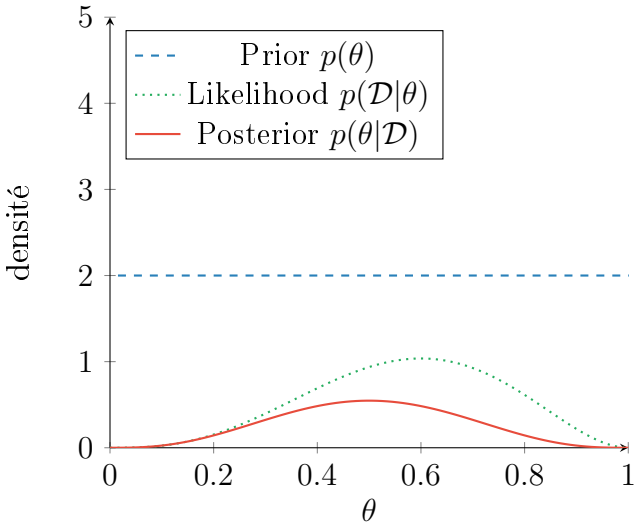


FIGURE 5.4 – Mise à jour bayésienne : Prior $\times$ Likelihood $\propto$ Posterior

5.6.1 Prédiction Bayésienne

Pour faire une prédiction sur un nouveau point x^* :

$$p(y^*|x^*, \mathcal{D}) = \int p(y^*|x^*, \theta)p(\theta|\mathcal{D})d\theta \tag{5.31}$$

On **marginalise** sur tous les paramètres possibles, pondérés par leur posterior.

Astuce

L'approche bayésienne fournit naturellement une **mesure d'incertitude** sur les prédictions, contrairement aux approches point-estimate (MLE, MAP).

5.7 Théorie de l'Information

5.7.1 Entropie

Définition 5.17 (Entropie)**.** L'**entropie** mesure l'incertitude d'une distribution :

$$H(X) = -\sum_x p(x) \log p(x) = -\mathbb{E}[\log p(X)] \tag{5.32}$$

— Entropie maximale : distribution uniforme

— Entropie minimale ($= 0$) : distribution déterministe

Exemple 5.3. Pour une pièce de monnaie avec $P(\text{pile}) = p$:

$$H = -p \log p - (1-p) \log(1-p)$$

Maximum à $p = 0.5$ (équiprobable $\Rightarrow$ plus d'incertitude).

5.7.2 Entropie Croisée et KL-Divergence

Définition 5.18 (Entropie Croisée)**.** Entre la vraie distribution p et la distribution prédite q :

$$H(p, q) = -\sum_x p(x) \log q(x) \tag{5.33}$$

Définition 5.19 (Divergence KL)**.** La **divergence de Kullback-Leibler** mesure la "distance" entre distributions :

$$D_{KL}(p\|q) = \sum_x p(x) \log \frac{p(x)}{q(x)} = H(p, q) - H(p) \tag{5.34}$$

Important

En classification, minimiser l'**entropie croisée** est équivalent à maximiser la vraisemblance ! C'est pourquoi c'est la loss standard.

Attention

$D_{KL}(p\|q) \neq D_{KL}(q\|p)$! La KL-divergence n'est pas symétrique.

5.8 Exercices

Exercice 5.1 (Probabilités de Base)**.** On lance deux dés équilibrés.

1. Quelle est la probabilité que la somme soit 7 ?
2. Sachant que la somme est impaire, quelle est la probabilité qu'elle soit ≥ 9 ?

Exercice 5.2 (Bayes)**.** Une usine a deux machines. Machine A produit 60% des pièces avec 2% de défauts. Machine B produit 40% avec 5% de défauts.

1. Quelle est la probabilité qu'une pièce soit défectueuse ?
2. Si une pièce est défectueuse, quelle est la probabilité qu'elle vienne de B ?

Exercice 5.3 (Distributions)**.** Le nombre d'emails spam reçus par heure suit une loi de Poisson avec $\lambda = 4$.

1. Probabilité de recevoir exactement 3 spams en une heure ?
2. Probabilité d'en recevoir au moins 1 ?
3. Espérance et variance du nombre de spams en 8 heures ?

Exercice 5.4 (MLE)**.** Soit $X_1, \ldots, X_n$ un échantillon de loi exponentielle Exp(λ).

1. Écrire la log-vraisemblance
2. Trouver $\hat{\lambda}_{MLE}$
3. Vérifier que c'est un maximum (condition du second ordre)

5.9 Implémentation Python

```python
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt

# --- Distributions ---

# Bernoulli
X_bern = stats.bernoulli(p=0.7)
print(f"Bernoulli: E[X]={X_bern.mean()}, Var={X_bern.var()}")
```

```python

# Binomiale
X_binom = stats.binom(n=10, p=0.3)
print(f"Binomiale: P(X=3)={X_binom.pmf(3):.4f}")

# Poisson
X_pois = stats.poisson(mu=4)
print(f"Poisson: P(X>=3)={1 - X_pois.cdf(2):.4f}")

# Gaussienne
X_norm = stats.norm(loc=0, scale=1)
print(f"Normale: P(-1<X<1)={X_norm.cdf(1) - X_norm.cdf(-1):.4f}")

# --- Estimation MLE ---
# Exemple: donnees gaussiennes
np.random.seed(42)
data = np.random.normal(loc=5, scale=2, size=100)

mu_mle = np.mean(data)
sigma_mle = np.std(data, ddof=0)  # MLE (biaise)
sigma_unbiased = np.std(data, ddof=1)  # non biaise

print(f"MLE: mu={mu_mle:.3f}, sigma={sigma_mle:.3f}")
print(f"Unbiased sigma={sigma_unbiased:.3f}")

# --- Gaussienne Multivariee ---
mean = [0, 0]
cov = [[1, 0.8], [0.8, 1]]
X_multi = stats.multivariate_normal(mean=mean, cov=cov)

# Generer des echantillons
samples = X_multi.rvs(size=1000)

plt.figure(figsize=(8, 6))
plt.scatter(samples[:, 0], samples[:, 1], alpha=0.5)
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.title('Echantillons d\'une Gaussienne bivariee')
plt.axis('equal')
plt.show()

# --- Entropie ---
def entropy(p):
    """Calcul de l'entropie"""
    p = np.array(p)
```

```python
    p = p[p > 0]  # eviter log(0)
    return -np.sum(p * np.log2(p))

def cross_entropy(p_true, q_pred):
    """Entropie croisee"""
    return -np.sum(p_true * np.log2(q_pred))

def kl_divergence(p, q):
    """Divergence KL"""
    return np.sum(p * np.log2(p / q))

# Exemple
p = np.array([0.5, 0.5])  # distribution uniforme
q = np.array([0.9, 0.1])  # distribution biaisee

print(f"Entropie de p: {entropy(p):.4f} bits")
print(f"Entropie de q: {entropy(q):.4f} bits")
print(f"Cross-entropy H(p,q): {cross_entropy(p, q):.4f}")
print(f"KL(p||q): {kl_divergence(p, q):.4f}")
```

Listing 5.1 – Probabilités et Statistiques avec Python

Synthèse

Points clés du chapitre :

- Le **théorème de Bayes** permet de mettre à jour nos croyances
- Les **distributions** modélisent différents types de données
- L'**espérance** et la **variance** caractérisent une distribution
- Le **MLE** maximise la vraisemblance, le **MAP** inclut un prior
- L'**entropie croisée** est la loss standard en classification

Chapitre 6

Optimisation pour le Machine Learning

Objectifs du chapitre

L'optimisation est au cœur de l'apprentissage automatique. Ce chapitre approfondit les techniques d'optimisation numérique, la régularisation, et les méthodes avancées utilisées pour entraîner les modèles de ML.

6.1 Le Problème d'Optimisation en ML

6.1.1 Formulation Générale

En Machine Learning, on cherche généralement à résoudre :

$$\boxed{\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)) + \lambda \Omega(\boldsymbol{\theta})} \tag{6.1}$$

Où :

— $\mathcal{L}$: fonction de perte (loss)

— $f_{\boldsymbol{\theta}}$: modèle paramétré

— $\Omega(\boldsymbol{\theta})$: terme de régularisation

— λ : hyperparamètre de régularisation

TABLE 6.1 – Fonctions de perte selon le type de problème

Problème	**Loss**	**Formule**
Régression	MSE	$\mathcal{L} = (y - \hat{y})^2$
Régression	MAE	$\mathcal{L} = \lvert y - \hat{y} \rvert$
Régression	Huber	$\mathcal{L} = \begin{cases} \frac{1}{2}(y-\hat{y})^2 & \lvert y-\hat{y}\rvert \leq \delta \\ \delta(\lvert y-\hat{y}\rvert - \frac{\delta}{2}) & \text{sinon} \end{cases}$
Classification binaire	Log Loss	$\mathcal{L} = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$
Multi-classes	Cross-Entropy	$\mathcal{L} = -\sum_k y_k \log(\hat{y}_k)$

6.1.2 Fonctions de Perte Courantes

6.2 Optimisation Convexe

6.2.1 Rappel : Convexité

Définition 6.1 (Fonction Convexe)**.** $f : \mathbb{R}^n \to \mathbb{R}$ est **convexe** si pour tout $\boldsymbol{x}, \boldsymbol{y}$ et $\lambda \in [0,1]$:

$$f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}) \tag{6.2}$$

Théorème 6.1 (Caractérisation par la Hessienne)**.** *Si f est deux fois différentiable, alors f est convexe si et seulement si sa Hessienne $\mathbf{H}$ est **semi-définie positive** partout :*

$$\mathbf{H}(\boldsymbol{x}) \succeq 0 \quad \forall \boldsymbol{x} \tag{6.3}$$

6.2.2 Pourquoi la Convexité est Importante

Théorème 6.2 (Minimum Global)**.** *Pour une fonction **convexe stricte**, tout minimum local est un **minimum global unique**.*

Synthèse

Problèmes convexes en ML :

- Régression linéaire (MSE)
- Régression logistique (Log Loss)
- SVM (problème dual)
- Régression Ridge/Lasso

Problèmes **non convexes** :

- Réseaux de neurones
- Clustering (k-means)
- Matrix factorization

6.3 Méthodes du Premier Ordre

6.3.1 Descente de Gradient (Rappel)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla J(\boldsymbol{\theta}^{(t)}) \tag{6.4}$$

6.3.2 Momentum

Le momentum ajoute une "inertie" qui aide à :

— Accélérer la convergence dans les directions de faible courbure

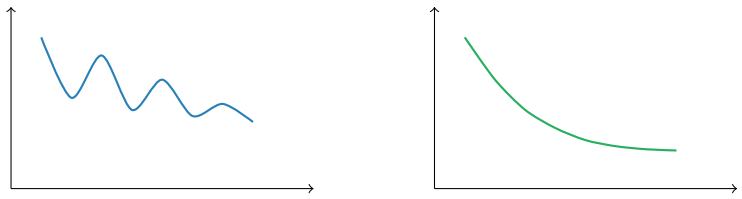
— Réduire les oscillations

— Traverser les plateaux

Algorithm 2 SGD avec Momentum

Require: η (learning rate), γ (momentum, typiquement 0.9)

1: $\boldsymbol{v} \leftarrow \mathbf{0}$
2: **for** $t = 1, 2, \ldots$ **do**
3: $\quad \boldsymbol{g} \leftarrow \nabla J(\boldsymbol{\theta}^{(t)})$
4: $\quad \boldsymbol{v} \leftarrow \gamma \boldsymbol{v} + \eta \boldsymbol{g}$
5: $\quad \boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \boldsymbol{v}$
6: **end for**



Sans Momentum Avec Momentum

FIGURE 6.1 – Le momentum réduit les oscillations et accélère la convergence

6.3.3 Nesterov Accelerated Gradient (NAG)

Une amélioration du momentum qui "regarde en avant" :

$$\boldsymbol{v}^{(t+1)} = \gamma \boldsymbol{v}^{(t)} + \eta \nabla J(\boldsymbol{\theta}^{(t)} - \gamma \boldsymbol{v}^{(t)}) \tag{6.5}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \boldsymbol{v}^{(t+1)} \tag{6.6}$$

Intuition

NAG calcule le gradient à la position "anticipée" au lieu de la position actuelle, permettant une correction plus intelligente.

6.4 Méthodes Adaptatives

6.4.1 AdaGrad

AdaGrad adapte le learning rate pour chaque paramètre :

$$\boldsymbol{s}^{(t)} = \boldsymbol{s}^{(t-1)} + \boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)} \tag{6.7}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta}{\sqrt{\boldsymbol{s}^{(t)} + \epsilon}} \odot \boldsymbol{g}^{(t)} \tag{6.8}$$

Avantage : Adapte automatiquement le LR par paramètre **Inconvénient** : LR décroît trop vite (accumulation)

6.4.2 RMSprop

RMSprop utilise une moyenne mobile exponentielle au lieu d'accumuler :

$$\boldsymbol{s}^{(t)} = \beta \boldsymbol{s}^{(t-1)} + (1-\beta)\boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)} \tag{6.9}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta}{\sqrt{\boldsymbol{s}^{(t)} + \epsilon}} \odot \boldsymbol{g}^{(t)} \tag{6.10}$$

Typiquement $\beta = 0.9$.

6.4.3 Adam (Adaptive Moment Estimation)

Algorithm 3 Adam

Require: $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$

$\boldsymbol{m} \leftarrow \mathbf{0}$, $\boldsymbol{v} \leftarrow \mathbf{0}$, $t \leftarrow 0$
while non convergé **do**
  $t \leftarrow t + 1$
  $\boldsymbol{g} \leftarrow \nabla J(\boldsymbol{\theta})$
  $\boldsymbol{m} \leftarrow \beta_1 \boldsymbol{m} + (1 - \beta_1)\boldsymbol{g}$ ▷ 1er moment (moyenne)
  $\boldsymbol{v} \leftarrow \beta_2 \boldsymbol{v} + (1 - \beta_2)\boldsymbol{g}^2$ ▷ 2ème moment (variance)
  $\hat{\boldsymbol{m}} \leftarrow \boldsymbol{m}/(1 - \beta_1^t)$ ▷ Correction de biais
  $\hat{\boldsymbol{v}} \leftarrow \boldsymbol{v}/(1 - \beta_2^t)$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \hat{\boldsymbol{m}}/(\sqrt{\hat{\boldsymbol{v}}} + \epsilon)$
end while

Important

Adam est généralement le meilleur choix par défaut :

— Combine les avantages du momentum et de RMSprop

— Robuste aux hyperparamètres

— Converge rapidement

6.4.4 Comparaison des Optimiseurs

TABLE 6.2 – Comparaison des optimiseurs

Optimiseur	LR Adaptatif	Momentum	Usage Recommandé
SGD	Non	Non	Problèmes simples
SGD + Momentum	Non	Oui	Vision par ordinateur
AdaGrad	Oui	Non	NLP (sparse)
RMSprop	Oui	Non	RNN
Adam	Oui	Oui	Usage général
AdamW	Oui	Oui	Avec weight decay

6.5 Régularisation

La régularisation prévient le **surapprentissage** (overfitting) en ajoutant une pénalité sur les paramètres.

6.5.1 Régularisation L^2 (Ridge)

$$J_{Ridge}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_2^2 = J(\boldsymbol{\theta}) + \lambda \sum_j \theta_j^2 \tag{6.11}$$

— Encourage des **petits** paramètres
— Ne met pas de paramètres exactement à zéro
— Équivalent à un prior **gaussien** sur $\boldsymbol{\theta}$

6.5.2 Régularisation L^1 (Lasso)

$$J_{Lasso}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1 = J(\boldsymbol{\theta}) + \lambda \sum_j |\theta_j| \tag{6.12}$$

— Encourage la **sparsité** (paramètres exactement à zéro)
— Utile pour la **sélection de features**
— Équivalent à un prior **Laplace** sur $\boldsymbol{\theta}$

6.5.3 Elastic Net

$$J_{EN}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda_1 \|\boldsymbol{\theta}\|_1 + \lambda_2 \|\boldsymbol{\theta}\|_2^2 \tag{6.13}$$

Combine les avantages de L1 et L2.

6.6 Learning Rate Scheduling

Le learning rate peut évoluer pendant l'entraînement.

6.6.1 Schedules Courants

Step Decay

$$\eta_t = \eta_0 \cdot \gamma^{\lfloor t/s \rfloor} \tag{6.14}$$

Réduire par facteur γ tous les s epochs.

Exponential Decay

$$\eta_t = \eta_0 \cdot e^{-kt} \tag{6.15}$$

Cosine Annealing

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min}) \left(1 + \cos\left(\frac{t\pi}{T}\right)\right) \tag{6.16}$$

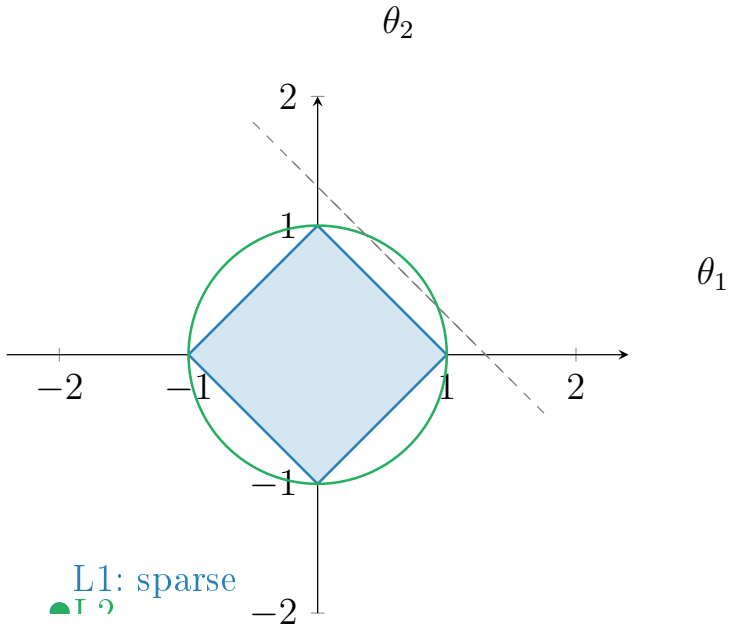


FIGURE 6.2 – L1 vs L2 : L1 favorise les solutions sur les axes (sparse)

Warmup Commencer avec un petit LR, puis augmenter linéairement pendant les premiers epochs :

$$\eta_t = \eta_{max} \cdot \frac{t}{T_{warmup}} \quad \text{pour } t < T_{warmup} \tag{6.17}$$

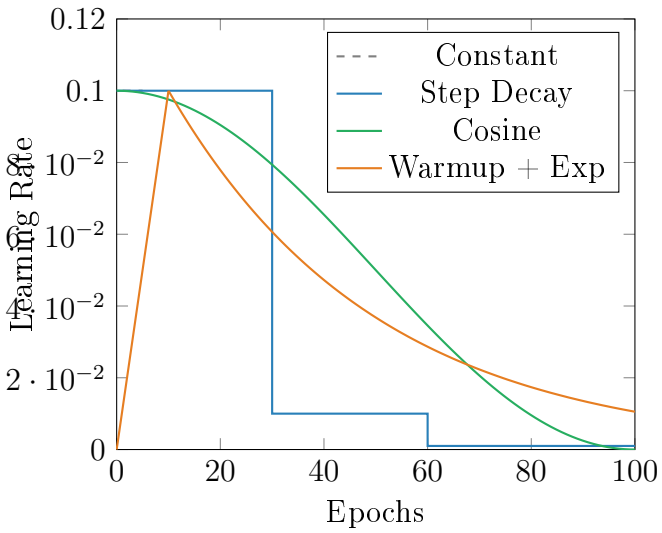


FIGURE 6.3 – Différentes stratégies de learning rate scheduling

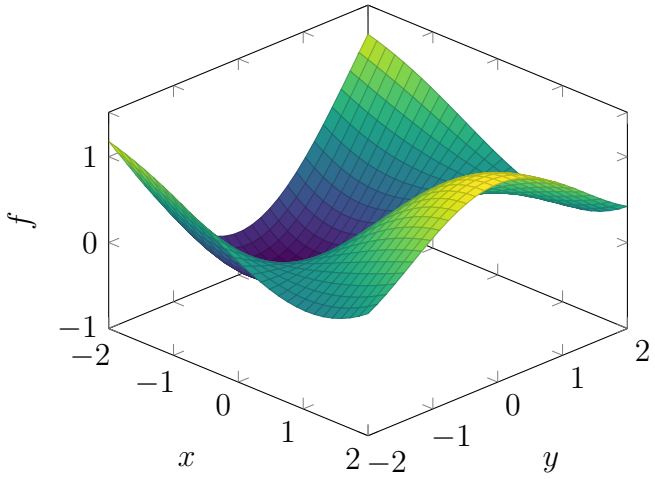


FIGURE 6.4 – Surface non convexe avec plusieurs minima locaux

6.7 Problèmes d'Optimisation Non Convexe

6.7.1 Minima Locaux et Points Selle

Définition 6.2 (Point Selle)**.** Un point $\boldsymbol{x}^*$ est un **point selle** si $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$ mais $\boldsymbol{x}^*$ n'est ni un maximum ni un minimum.

Astuce

En haute dimension, les **points selle** sont plus problématiques que les minima locaux. Le SGD (grâce à son bruit) aide souvent à les éviter.

6.7.2 Techniques pour Échapper aux Mauvais Minima

1. **Initialisation intelligente** : Xavier, He
2. **Bruit stochastique** : Mini-batch SGD
3. **Momentum** : Aide à traverser les plateaux
4. **Learning rate cyclique** : Exploration périodique
5. **Ensembles** : Plusieurs initialisations

6.8 Optimisation sous Contraintes

6.8.1 Méthode de Lagrange

Pour minimiser $f(\boldsymbol{x})$ sous contraintes $g_i(\boldsymbol{x}) = 0$:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \sum_i \lambda_i g_i(\boldsymbol{x}) \tag{6.18}$$

Conditions nécessaires (KKT) :

$$\nabla_{\boldsymbol{x}} \mathcal{L} = \mathbf{0} \tag{6.19}$$

$$g_i(\boldsymbol{x}) = 0 \quad \forall i \tag{6.20}$$

6.8.2 Application : SVM

Le problème SVM utilise l'optimisation sous contraintes :

$$\min_{\boldsymbol{w},b} \frac{1}{2} \|\boldsymbol{w}\|^2 \quad \text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 \tag{6.21}$$

Le dual (résolu en pratique) fait intervenir les multiplicateurs de Lagrange.

6.9 Exercices

Exercice 6.1 (Convexité)**.** Montrer que les fonctions suivantes sont convexes :

1. $f(x) = x^2 + 3x + 5$
2. $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2$
3. $f(\boldsymbol{x}) = \max(x_1, x_2, \ldots, x_n)$

Exercice 6.2 (Optimiseurs)**.** Implémentez SGD avec momentum et Adam from scratch, puis comparez-les sur la minimisation de la fonction de Rosenbrock :

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

Exercice 6.3 (Régularisation)**.** Pour un problème de régression avec 100 features et seulement 50 exemples :

1. Quel type de régularisation recommandez-vous ? Pourquoi ?
2. Implémentez Ridge et Lasso et comparez les coefficients

6.10 Implémentation Python

```python
import numpy as np

class SGD:
    def __init__(self, lr=0.01):
        self.lr = lr

    def update(self, params, grads):
        for p, g in zip(params, grads):
            p -= self.lr * g
```

```
        return params

class SGDMomentum:
    def __init__(self, lr=0.01, momentum=0.9):
        self.lr = lr
        self.momentum = momentum
        self.v = None

    def update(self, params, grads):
        if self.v is None:
            self.v = [np.zeros_like(p) for p in params]

        for i, (p, g) in enumerate(zip(params, grads)):
            self.v[i] = self.momentum * self.v[i] + self.lr * g
            p -= self.v[i]
        return params

class Adam:
    def __init__(self, lr=0.001, beta1=0.9, beta2=0.999,
       epsilon=1e-8):
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.epsilon = epsilon
        self.m = None
        self.v = None
        self.t = 0

    def update(self, params, grads):
        if self.m is None:
            self.m = [np.zeros_like(p) for p in params]
            self.v = [np.zeros_like(p) for p in params]

        self.t += 1

        for i, (p, g) in enumerate(zip(params, grads)):
            self.m[i] = self.beta1 * self.m[i] + (1 - self.beta1)
               * g
            self.v[i] = self.beta2 * self.v[i] + (1 - self.beta2)
               * g**2

            m_hat = self.m[i] / (1 - self.beta1**self.t)
            v_hat = self.v[i] / (1 - self.beta2**self.t)

            p -= self.lr * m_hat / (np.sqrt(v_hat) + self.epsilon)
```

```
52
53          return params
54
55  # Comparaison sur la fonction de Rosenbrock
56  def rosenbrock(x, y):
57      return (1 - x)**2 + 100 * (y - x**2)**2
58
59  def rosenbrock_grad(x, y):
60      dx = -2*(1-x) - 400*x*(y - x**2)
61      dy = 200*(y - x**2)
62      return np.array([dx, dy])
63
64  # Test
65  x0, y0 = -1.5, 1.5
66  params_sgd = [np.array([x0]), np.array([y0])]
67  params_adam = [np.array([x0]), np.array([y0])]
68
69  sgd = SGD(lr=0.0001)
70  adam = Adam(lr=0.01)
71
72  for i in range(1000):
73      # SGD
74      g = rosenbrock_grad(params_sgd[0][0], params_sgd[1][0])
75      params_sgd = sgd.update(params_sgd, [g[:1], g[1:]])
76
77      # Adam
78      g = rosenbrock_grad(params_adam[0][0], params_adam[1][0])
79      params_adam = adam.update(params_adam, [g[:1], g[1:]])
80
81  print(f"SGD final: ({params_sgd[0][0]:.4f},
       {params_sgd[1][0]:.4f})")
82  print(f"Adam final: ({params_adam[0][0]:.4f},
       {params_adam[1][0]:.4f})")
83  print(f"Optimal: (1, 1)")
```

Listing 6.1 – Optimiseurs from scratch

Synthèse

Points clés du chapitre :

- Les problèmes **convexes** garantissent un optimum global
- **Adam** est l'optimiseur par défaut recommandé
- La **régularisation** (L1/L2) prévient l'overfitting
- Le **learning rate scheduling** améliore la convergence

— En haute dimension, les **points selle** sont le vrai défi

Troisième partie

Machine Learning

Chapitre 7

Introduction au Machine Learning

Objectifs du chapitre

Ce chapitre présente les concepts fondamentaux du Machine Learning : types d'apprentissage, workflow, évaluation, et les pièges à éviter. Il pose les bases pour tous les algorithmes qui suivront.

7.1 Qu'est-ce que le Machine Learning ?

Définition 7.1 (Machine Learning)**.** Le **Machine Learning** (apprentissage automatique) est un sous-domaine de l'IA où les systèmes **apprennent des patterns à partir de données** plutôt que d'être explicitement programmés.

> "Un programme informatique est dit apprendre de l'expérience E par rapport à une tâche T et une mesure de performance P, si sa performance sur T, mesurée par P, s'améliore avec l'expérience E." – Tom Mitchell, 1997

7.1.1 Programmation Traditionnelle vs Machine Learning

7.2 Types d'Apprentissage

7.2.1 Apprentissage Supervisé

Définition 7.2 (Apprentissage Supervisé)**.** On dispose de paires **(entrée, sortie attendue)** : $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$. L'objectif est d'apprendre une fonction $f : \boldsymbol{x} \to y$.

- **Régression** : $y \in \mathbb{R}$ (valeur continue)
  - Prédire le prix d'une maison
  - Estimer la température demain
- **Classification** : $y \in \{1, 2, \ldots, K\}$ (catégorie)

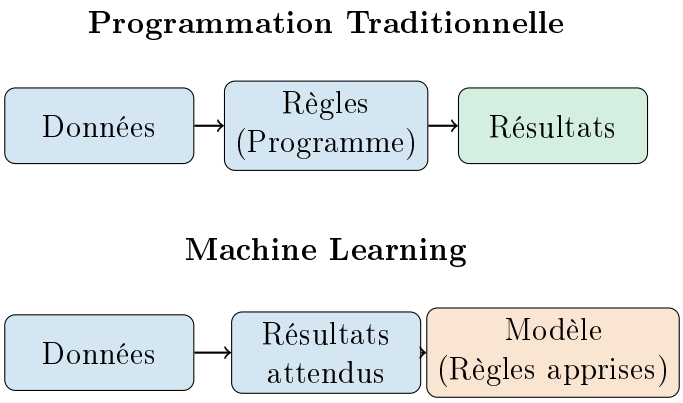


FIGURE 7.1 – Différence fondamentale entre programmation traditionnelle et ML

— Détecter un spam

— Reconnaître un chiffre manuscrit

7.2.2 Apprentissage Non Supervisé

Définition 7.3 (Apprentissage Non Supervisé)**.** On dispose uniquement d'entrées **sans labels** : $\{\boldsymbol{x}_i\}_{i=1}^n$. L'objectif est de découvrir des **structures cachées**.

— **Clustering** : Grouper des données similaires

— **Réduction dimensionnelle** : Compresser l'information

— **Détection d'anomalies** : Identifier les outliers

7.2.3 Apprentissage par Renforcement

Définition 7.4 (Apprentissage par Renforcement)**.** Un **agent** apprend à prendre des **actions** dans un **environnement** pour maximiser une **récompense cumulative**.

Applications : Jeux, robotique, trading automatique.

7.2.4 Autres Types

— **Semi-supervisé** : Peu de labels, beaucoup de données non labellées

— **Auto-supervisé** : Créer des labels à partir des données elles-mêmes

— **Transfer Learning** : Réutiliser un modèle pré-entraîné

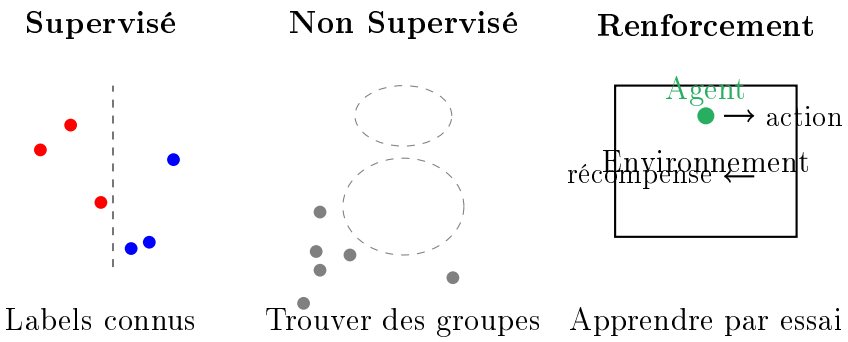


FIGURE 7.2 – Les trois principaux paradigmes d'apprentissage

TABLE 7.1 – Terminologie du Machine Learning

Terme	Description
Dataset	Ensemble de données utilisé pour l'apprentissage
Exemple / Instance	Un point de données $(\boldsymbol{x}, y)$
Feature / Attribut	Une caractéristique mesurée (x_j)
Label / Target	La sortie à prédire (y)
Modèle	La fonction apprise $f_{\boldsymbol{\theta}}$
Paramètres	Les variables ajustées pendant l'apprentissage $(\boldsymbol{\theta})$
Hyperparamètres	Les choix de design (learning rate, nb couches...)
Training	Phase d'apprentissage
Inference	Utilisation du modèle pour prédire

7.3 Terminologie et Notations

7.3.1 Vocabulaire Essentiel

7.3.2 Notations Standard

— n : nombre d'exemples (taille du dataset)

— p ou d : nombre de features (dimension)

— $\mathbf{X} \in \mathbb{R}^{n \times p}$: matrice de design

— $\boldsymbol{x}^{(i)} \in \mathbb{R}^p$: i-ème exemple

— x_j : j-ème feature

— $\boldsymbol{y} \in \mathbb{R}^n$: vecteur des labels

— $\hat{y}$: prédiction du modèle

— $\boldsymbol{\theta}$: paramètres du modèle

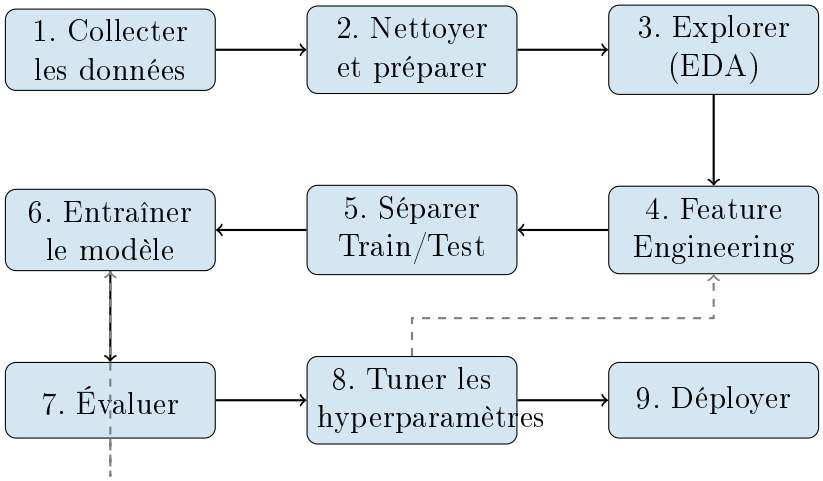


FIGURE 7.3 – Pipeline typique d'un projet ML

7.4 Le Workflow du Machine Learning

7.4.1 Étape 1-3 : Préparation des Données

1. **Collecte** : Sources de données, APIs, scraping, bases de données
2. **Nettoyage** :
   - Gestion des valeurs manquantes
   - Suppression des doublons
   - Correction des erreurs
3. **Exploration (EDA)** :
   - Statistiques descriptives
   - Visualisations
   - Détection d'anomalies

7.4.2 Étape 4 : Feature Engineering

Définition 7.5 (Feature Engineering)**.** L'art de créer, transformer et sélectionner des features pertinentes pour améliorer les performances du modèle.

Techniques courantes :

- **Normalisation** : $x' = \frac{x-\mu}{\sigma}$ (StandardScaler)
- **Min-Max** : $x' = \frac{x-x_{min}}{x_{max}-x_{min}}$
- **Encoding catégoriel** : One-Hot, Label Encoding
- **Création de features** : Interactions, polynômes
- **Sélection de features** : Importance, corrélation

7.4.3 Étape 5 : Séparation des Données

Important

Règle d'or : Le modèle ne doit **JAMAIS** voir les données de test pendant l'entraînement !

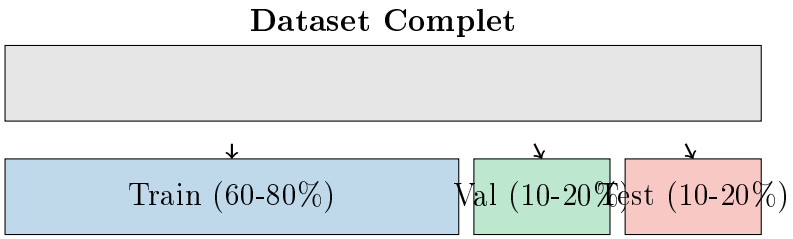


FIGURE 7.4 – Séparation Train/Validation/Test

— **Training set** : Pour entraîner le modèle
— **Validation set** : Pour tuner les hyperparamètres
— **Test set** : Pour l'évaluation finale (une seule fois !)

7.5 Biais et Variance : Le Compromis Fondamental

7.5.1 Définitions

Définition 7.6 (Biais)**.** Le **biais** mesure l'erreur due aux hypothèses simplificatrices du modèle. Un biais élevé signifie que le modèle est trop simple (**underfitting**).

Définition 7.7 (Variance)**.** La **variance** mesure la sensibilité du modèle aux fluctuations des données d'entraînement. Une variance élevée signifie que le modèle est trop complexe (**overfitting**).

7.5.2 Décomposition de l'Erreur

L'erreur de généralisation peut se décomposer :

$$\boxed{\text{Erreur} = \text{Biais}^2 + \text{Variance} + \text{Bruit irréductible}} \tag{7.1}$$

7.5.3 Diagnostic et Solutions

Contre l'underfitting :

— Augmenter la complexité du modèle
— Ajouter des features

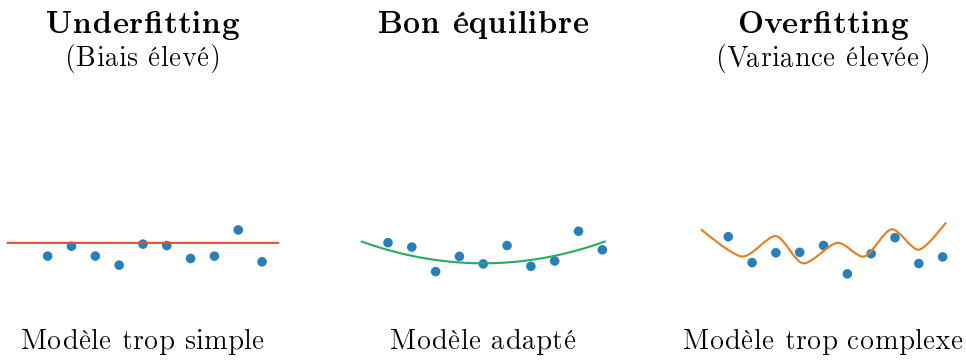


FIGURE 7.5 – Illustration du compromis biais-variance

TABLE 7.2 – Diagnostic biais-variance

	Erreur Train haute	**Erreur Train basse**
Erreur Val haute	Underfitting	Overfitting
Erreur Val basse	(rare)	Bon modèle

— Réduire la régularisation

— Entraîner plus longtemps

Contre l'overfitting :

— Simplifier le modèle

— Ajouter de la régularisation (L1, L2, Dropout)

— Augmenter la quantité de données

— Early stopping

— Data augmentation

7.6 Validation Croisée

Définition 7.8 (K-Fold Cross-Validation)**.** Diviser les données en K parties égales. Pour chaque fold k :

1. Utiliser le fold k comme validation
2. Entraîner sur les $K-1$ autres folds
3. Mesurer la performance

Moyenne des K performances = estimation de la généralisation.

Astuce

— $K = 5$ ou $K = 10$ sont les choix les plus courants

— Pour petits datasets : **Leave-One-Out** ($K = n$)

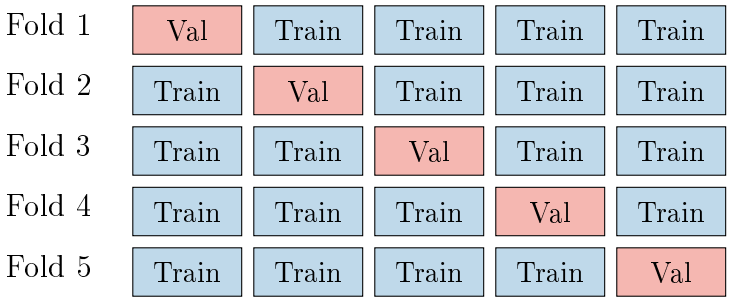


FIGURE 7.6 – 5-Fold Cross-Validation

— Pour séries temporelles : **Time Series Split** (respecter l'ordre temporel)
— Pour classification déséquilibrée : **Stratified K-Fold**

7.7 Métriques d'Évaluation

7.7.1 Métriques de Régression

— **MSE** (Mean Squared Error) :

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{7.2}$$

— **RMSE** (Root MSE) : $\text{RMSE} = \sqrt{\text{MSE}}$
— **MAE** (Mean Absolute Error) :

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{7.3}$$

— R^2 (Coefficient de détermination) :

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} \tag{7.4}$$

7.7.2 Métriques de Classification

Définition 7.9 (Matrice de Confusion).

		Prédit	
		Positif	Négatif
Réel	Positif	TP	FN
	Négatif	FP	TN

— **Accuracy** : $\frac{TP+TN}{TP+TN+FP+FN}$ (attention aux classes déséquilibrées !)
— **Precision** : $\frac{TP}{TP+FP}$ (parmi les prédits positifs, combien sont vrais ?)

— **Recall** (Sensibilité) : $\frac{TP}{TP+FN}$ (parmi les vrais positifs, combien détectés ?)

— **F1-Score** : $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

— **AUC-ROC** : Aire sous la courbe ROC

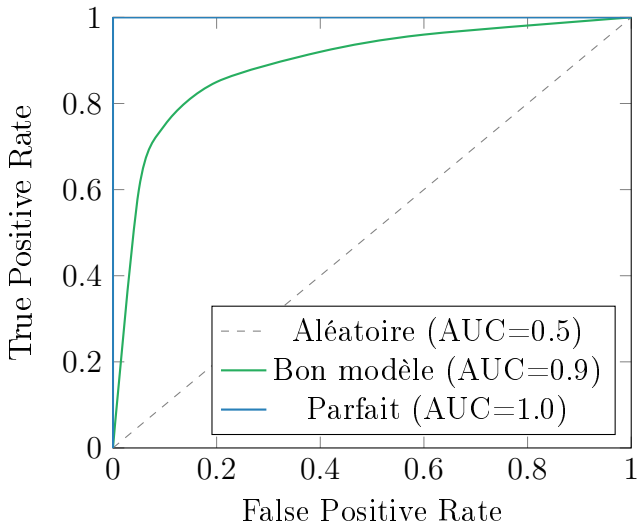


FIGURE 7.7 – Courbe ROC : plus l'aire est grande, meilleur est le modèle

7.8 Exercices

Exercice 7.1 (Types de Problèmes)**.** Classifiez les problèmes suivants (supervisé/non supervisé, régression/classification) :

1. Prédire si un email est un spam
2. Estimer le prix d'une action demain
3. Segmenter des clients par comportement d'achat
4. Reconnaître des émotions dans des photos
5. Détecter des transactions frauduleuses (avec historique)

Exercice 7.2 (Biais-Variance)**.** Un modèle a les performances suivantes :

— Erreur sur train : 2%

— Erreur sur validation : 15%

1. Quel est le problème ?
2. Proposez 3 solutions

Exercice 7.3 (Métriques)**.** Sur un test de 1000 patients pour une maladie rare (prévalence 1%) :

— 8 vrais positifs

— 2 faux négatifs

— 50 faux positifs

— 940 vrais négatifs

Calculez : Accuracy, Precision, Recall, F1-Score. Quelle métrique est la plus pertinente ici ?

7.9 Implémentation Python

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split,
    cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
    confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Charger les donnees (exemple avec Iris)
from sklearn.datasets import load_iris
data = load_iris()
X, y = data.data, data.target

# 2. Explorer
print(f"Shape: {X.shape}")
print(f"Classes: {np.unique(y)}")

# 3. Separer train/test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# 4. Preprocesser
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)  # IMPORTANT: pas de fit!

# 5. Entrainer
model = LogisticRegression(random_state=42)
model.fit(X_train_scaled, y_train)

# 6. Evaluer
y_pred = model.predict(X_test_scaled)
```

```python
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2%}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred,
    target_names=data.target_names))

# 7. Cross-validation
cv_scores = cross_val_score(model, X_train_scaled, y_train, cv=5)
print(f"\nCV Scores: {cv_scores}")
print(f"CV Mean: {cv_scores.mean():.2%} (+/-
    {cv_scores.std()*2:.2%})")

# 8. Matrice de confusion
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=data.target_names,
            yticklabels=data.target_names)
plt.xlabel('Predit')
plt.ylabel('Reel')
plt.title('Matrice de Confusion')
plt.show()
```

Listing 7.1 – Workflow ML avec scikit-learn

Synthèse

Points clés du chapitre :

- Le ML apprend des **patterns** à partir de **données**
- Trois types principaux : **supervisé**, **non supervisé**, **renforcement**
- Le compromis **biais-variance** est fondamental
- Toujours **séparer** les données (train/val/test)
- Choisir les **métriques** adaptées au problème
- La **validation croisée** donne une estimation robuste

Chapitre 8

Régression Linéaire

Objectifs du chapitre

La régression linéaire est l'algorithme fondamental du Machine Learning. Ce chapitre couvre en profondeur la théorie, les solutions analytiques et numériques, les variantes régularisées, et les diagnostics.

8.1 Introduction et Motivation

8.1.1 Le Problème de Régression

Définition 8.1 (Régression)**.** En **régression**, on cherche à prédire une variable continue $y \in \mathbb{R}$ à partir de variables explicatives $\boldsymbol{x} \in \mathbb{R}^p$.

Exemple 8.1. Exemples de problèmes de régression :

- Prédire le prix d'une maison (y) à partir de sa surface, nombre de chambres, localisation ($\boldsymbol{x}$)
- Estimer la consommation d'un véhicule à partir de son poids, puissance, aérodynamisme
- Prédire le chiffre d'affaires à partir des dépenses marketing

8.1.2 Hypothèse de Linéarité

La **régression linéaire** suppose que la relation entre $\boldsymbol{x}$ et y est linéaire :

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon \tag{8.1}$$

où ϵ est un terme d'erreur (bruit).

Forme vectorielle :

$$\boxed{y = \boldsymbol{w}^\top \boldsymbol{x} + b = \sum_{j=1}^{p} w_j x_j + b} \tag{8.2}$$

8.2 Régression Linéaire Simple

8.2.1 Modèle à Une Variable

Avec une seule feature x :

$$y = wx + b \tag{8.3}$$

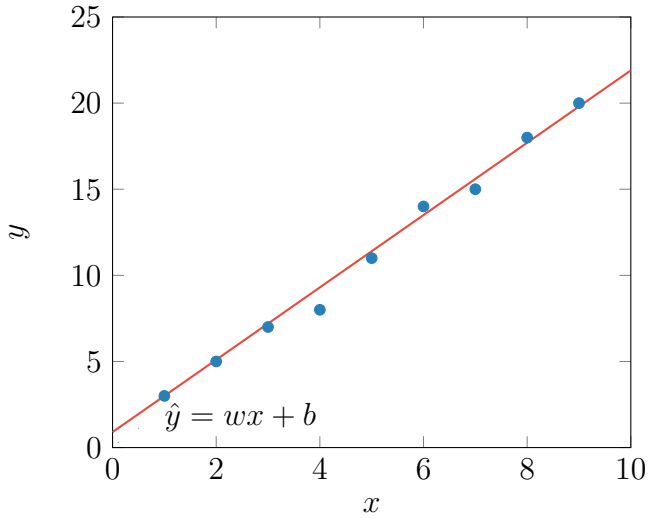


FIGURE 8.1 – Régression linéaire simple : trouver la meilleure droite

8.2.2 Fonction de Coût : Moindres Carrés

On cherche à minimiser la **somme des carrés des résidus** (SSR) :

$$J(w, b) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - (wx_i + b))^2 = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{8.4}$$

Intuition

Pourquoi les carrés ?

- Pénalise plus les grandes erreurs
- Différentiable partout (contrairement à la valeur absolue)
- Solution analytique simple
- Correspond au maximum de vraisemblance sous bruit gaussien

8.2.3 Solution Analytique (Cas Simple)

En annulant les dérivées partielles :

$$\frac{\partial J}{\partial w} = 0 \implies w = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2} = \frac{\mathrm{Cov}(x, y)}{\mathrm{Var}(x)} \tag{8.5}$$

$$\frac{\partial J}{\partial b} = 0 \implies b = \bar{y} - w\bar{x} \tag{8.6}$$

où $\bar{x} = \frac{1}{n}\sum_i x_i$ et $\bar{y} = \frac{1}{n}\sum_i y_i$.

Théorème 8.1 (Propriété de la Droite de Régression). *La droite de régression passe toujours par le point* $(\bar{x}, \bar{y})$.

8.3 Régression Linéaire Multiple

8.3.1 Formulation Matricielle

Avec n exemples et p features :

$$\mathbf{X} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_p^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_p^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & x_2^{(n)} & \cdots & x_p^{(n)} \end{pmatrix}_{n \times (p+1)}, \quad \boldsymbol{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{pmatrix} \tag{8.7}$$

La première colonne de $\mathbf{X}$ (tous des 1) capture l'intercept b.

Le modèle :

$$\hat{\boldsymbol{y}} = \mathbf{X}\boldsymbol{\beta} \tag{8.8}$$

où $\boldsymbol{\beta} = (b, w_1, w_2, \ldots, w_p)^\top$.

8.3.2 Fonction de Coût MSE

$$J(\boldsymbol{\beta}) = \frac{1}{2n} \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 = \frac{1}{2n}(\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta})^\top (\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}) \tag{8.9}$$

8.3.3 Solution Analytique : Équations Normales

Théorème 8.2 (Équations Normales). *Le minimum de* $J(\boldsymbol{\beta})$ *est atteint pour :*

$$\boxed{\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \boldsymbol{y}} \tag{8.10}$$

Démonstration. Le gradient de J est :

$$\nabla_{\boldsymbol{\beta}} J = \frac{1}{n}\mathbf{X}^\top(\mathbf{X}\boldsymbol{\beta} - \boldsymbol{y}) \tag{8.11}$$

En posant $\nabla J = \mathbf{0}$:

$$\mathbf{X}^\top\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^\top\boldsymbol{y} \tag{8.12}$$

$$\boldsymbol{\beta} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\boldsymbol{y} \tag{8.13}$$

□

Attention

La matrice $\mathbf{X}^\top\mathbf{X}$ doit être **inversible**. Elle ne l'est pas si :

— Il y a plus de features que d'exemples ($p > n$)

— Deux features sont parfaitement corrélées (multicolinéarité)

Solution : utiliser la **pseudo-inverse** ou la **régularisation**.

8.3.4 Interprétation Géométrique

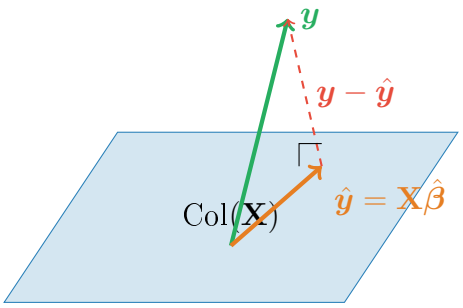


FIGURE 8.2 – La prédiction $\hat{\boldsymbol{y}}$ est la projection orthogonale de $\boldsymbol{y}$ sur l'espace des colonnes de $\mathbf{X}$

Théorème 8.3 (Projection Orthogonale). *La solution des moindres carrés $\hat{\boldsymbol{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$ est la **projection orthogonale** de $\boldsymbol{y}$ sur l'espace engendré par les colonnes de $\mathbf{X}$.*

*Le résidu $\boldsymbol{y} - \hat{\boldsymbol{y}}$ est **perpendiculaire** à cet espace.*

8.4 Descente de Gradient pour la Régression

Pour les grands datasets, la solution analytique devient coûteuse ($O(p^3)$ pour l'inversion). On utilise alors la descente de gradient.

8.4.1 Calcul du Gradient

$$\nabla_{\boldsymbol{\beta}} J = \frac{1}{n}\mathbf{X}^\top(\mathbf{X}\boldsymbol{\beta} - \boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})\boldsymbol{x}^{(i)} \tag{8.14}$$

8.4.2 Algorithmes

Algorithm 4 Batch Gradient Descent pour Régression Linéaire

Require: $\mathbf{X}, \boldsymbol{y}, \eta, n\_iter$

1: $\boldsymbol{\beta} \leftarrow \mathbf{0}$
2: **for** $t = 1$ to n_iter **do**
3: $\quad \hat{\boldsymbol{y}} \leftarrow \mathbf{X}\boldsymbol{\beta}$
4: $\quad \boldsymbol{g} \leftarrow \frac{1}{n}\mathbf{X}^\top(\hat{\boldsymbol{y}} - \boldsymbol{y})$
5: $\quad \boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \eta\boldsymbol{g}$
6: **end for**
7: **return** $\boldsymbol{\beta}$

TABLE 8.1 – Comparaison des méthodes de résolution

Méthode	Complexité	Quand l'utiliser
Équations normales	$O(np^2 + p^3)$	$p < 10000$
Décomposition SVD	$O(np^2)$	Plus stable
Batch GD	$O(np \times iter)$	Grands datasets
SGD	$O(p \times iter \times epochs)$	Très grands datasets

8.5 Hypothèses et Diagnostic

8.5.1 Hypothèses du Modèle Linéaire

1. **Linéarité** : La relation entre $\boldsymbol{x}$ et y est linéaire
2. **Indépendance** : Les observations sont indépendantes
3. **Homoscédasticité** : Variance constante des erreurs
4. **Normalité** : Les erreurs suivent une loi normale $\epsilon \sim \mathcal{N}(0, \sigma^2)$
5. **Pas de multicolinéarité** : Les features ne sont pas trop corrélées entre elles

Bon : Homoscédasticité **Problème** : Hétéroscédasticité **Problème** : Non-linéarité

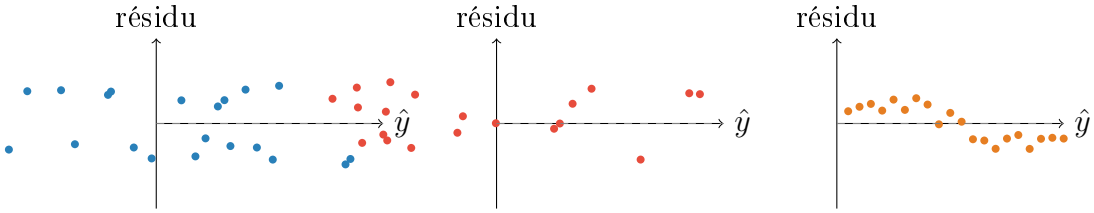


FIGURE 8.3 – Diagnostic par analyse des résidus

8.5.2 Diagnostic des Résidus

8.5.3 Coefficient de Détermination R^2

Définition 8.2 (R^2)**.** Le coefficient de détermination mesure la proportion de variance expliquée :

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} \tag{8.15}$$

- $R^2 = 1$: Le modèle explique parfaitement les données
- $R^2 = 0$: Le modèle n'est pas meilleur que la moyenne
- $R^2 < 0$: Le modèle est pire que la moyenne (possible sur le test set)

Attention

R^2 augmente **toujours** quand on ajoute des features, même inutiles. Utilisez le R^2 **ajusté** :

$$R^2_{adj} = 1 - (1 - R^2)\frac{n-1}{n-p-1} \tag{8.16}$$

8.6 Régularisation : Ridge et Lasso

8.6.1 Le Problème du Surapprentissage

Avec beaucoup de features, le modèle peut :

- **Overfitter** : Apprendre le bruit des données d'entraînement
- Avoir des coefficients très grands (instabilité)
- Être sensible à la multicolinéarité

Solution : Ajouter une pénalité sur la taille des coefficients.

8.6.2 Régression Ridge (L2)

Définition 8.3 (Ridge Regression)**.**

$$J_{Ridge}(\boldsymbol{\beta}) = \frac{1}{2n} \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_2^2 \tag{8.17}$$

Solution analytique :

$$\boxed{\hat{\boldsymbol{\beta}}_{Ridge} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \boldsymbol{y}} \tag{8.18}$$

- $\lambda = 0$: régression ordinaire
- $\lambda \to \infty$: tous les coefficients $\to 0$
- Le terme $\lambda\mathbf{I}$ rend la matrice **toujours inversible**
- Ne met pas de coefficients exactement à zéro

8.6.3 Régression Lasso (L1)

Définition 8.4 (Lasso Regression)**.**

$$J_{Lasso}(\boldsymbol{\beta}) = \frac{1}{2n} \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1 \tag{8.19}$$

- Pas de solution analytique (non différentiable en 0)
- Met certains coefficients **exactement à zéro**
- Effectue une **sélection de features** automatique
- Utile quand on suspecte que seules quelques features sont importantes

8.6.4 Elastic Net

Définition 8.5 (Elastic Net)**.** Combine L1 et L2 :

$$J_{EN}(\boldsymbol{\beta}) = \frac{1}{2n} \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda_1 \|\boldsymbol{\beta}\|_1 + \lambda_2 \|\boldsymbol{\beta}\|_2^2 \tag{8.20}$$

8.6.5 Choix de λ

Astuce

Le paramètre λ est un **hyperparamètre** à optimiser par validation croisée :

1. Tester une grille de valeurs de λ
2. Pour chaque λ, calculer l'erreur en CV
3. Choisir le λ qui minimise l'erreur de validation

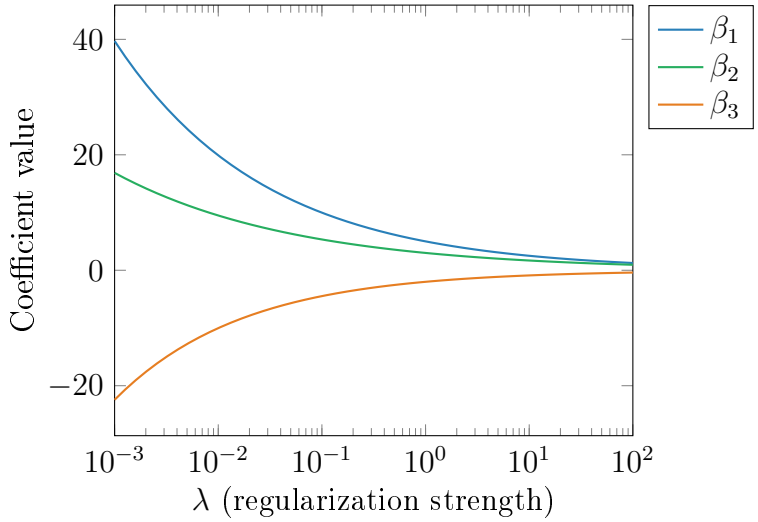


FIGURE 8.4 – Chemin de régularisation : coefficients en fonction de λ

8.7 Régression Polynomiale

8.7.1 Extension Non-Linéaire

Si la relation n'est pas linéaire, on peut créer des **features polynomiales** :

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_d x^d \tag{8.21}$$

Important

C'est toujours une régression **linéaire** ! Le modèle est linéaire en les **paramètres** $\boldsymbol{\beta}$, pas en x.

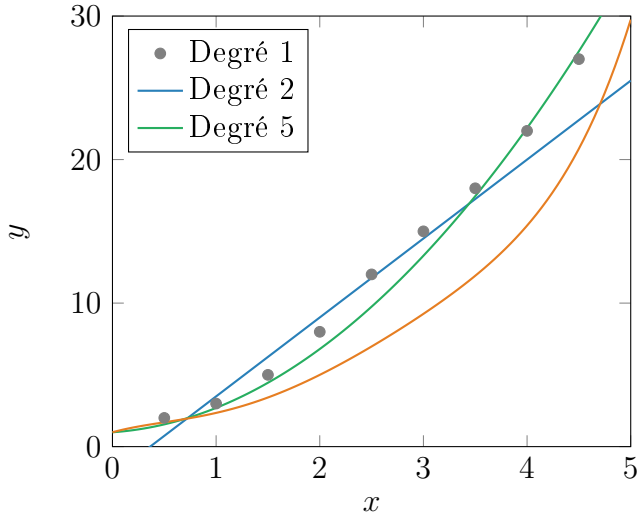


FIGURE 8.5 – Régression polynomiale de différents degrés

Attention

Un degré trop élevé mène à l'**overfitting** ! Utilisez la validation croisée pour choisir le degré optimal.

8.8 Exercices

Exercice 8.1 (Régression Simple)**.** On a les données suivantes :

x	1	2	3	4	5
y	2.1	3.9	6.2	7.8	10.1

1. Calculer $\bar{x}$, $\bar{y}$, $\text{Cov}(x, y)$, $\text{Var}(x)$
2. Trouver w et b pour la droite de régression
3. Calculer le R^2
4. Prédire y pour $x = 6$

Exercice 8.2 (Équations Normales)**.** Montrer que la Hessienne de $J(\boldsymbol{\beta}) = \frac{1}{2n} \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$ est $\frac{1}{n}\mathbf{X}^\top\mathbf{X}$, et qu'elle est semi-définie positive.

Exercice 8.3 (Ridge vs Lasso)**.**

1. Expliquer géométriquement pourquoi Lasso produit des coefficients exactement nuls mais pas Ridge
2. Dans quel cas préféreriez-vous Lasso ?
3. Dans quel cas préféreriez-vous Ridge ?

Exercice 8.4 (Projet : Boston Housing)**.** Utilisez le dataset Boston Housing pour :

1. Explorer les données (corrélations, distributions)
2. Implémenter la régression linéaire from scratch
3. Comparer avec scikit-learn
4. Appliquer Ridge et Lasso, comparer les coefficients
5. Analyser les résidus

8.9 Implémentation Python

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import PolynomialFeatures,
    StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, r2_score

```

```python
# ===== Implementation from scratch =====
class LinearRegressionScratch:
    """Regression lineaire par equations normales"""

    def __init__(self):
        self.coef_ = None
        self.intercept_ = None

    def fit(self, X, y):
        # Ajouter colonne de 1 pour l'intercept
        X_b = np.c_[np.ones(X.shape[0]), X]

        # Equations normales: beta = (X'X)^(-1) X'y
        self.theta = np.linalg.pinv(X_b.T @ X_b) @ X_b.T @ y

        self.intercept_ = self.theta[0]
        self.coef_ = self.theta[1:]
        return self

    def predict(self, X):
        return X @ self.coef_ + self.intercept_

# ===== Donnees de test =====
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X.flatten() + np.random.randn(100) * 0.5

# ===== Comparaison =====
# From scratch
model_scratch = LinearRegressionScratch()
model_scratch.fit(X, y)

# Scikit-learn
model_sklearn = LinearRegression()
model_sklearn.fit(X, y)

print("=== Comparaison ===")
print(f"Scratch:  w={model_scratch.coef_[0]:.4f},
   b={model_scratch.intercept_:.4f}")
print(f"Sklearn:  w={model_sklearn.coef_[0]:.4f},
   b={model_sklearn.intercept_:.4f}")

# ===== Ridge et Lasso =====
from sklearn.datasets import make_regression
X_multi, y_multi = make_regression(n_samples=100, n_features=20,
```

```
                                    n_informative=5, noise=10,
                                       random_state=42)

# Standardiser
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_multi)

# Comparer les coefficients
models = {
    'OLS': LinearRegression(),
    'Ridge': Ridge(alpha=1.0),
    'Lasso': Lasso(alpha=0.1)
}

plt.figure(figsize=(12, 4))
for i, (name, model) in enumerate(models.items()):
    model.fit(X_scaled, y_multi)

    plt.subplot(1, 3, i+1)
    plt.bar(range(20), model.coef_)
    plt.title(f'{name}\nNon-zero: {np.sum(np.abs(model.coef_) >
       0.01)}')
    plt.xlabel('Feature')
    plt.ylabel('Coefficient')

plt.tight_layout()
plt.show()

# ===== Validation croisee pour choisir alpha =====
alphas = np.logspace(-3, 3, 50)
ridge_scores = []
lasso_scores = []

for alpha in alphas:
    ridge = Ridge(alpha=alpha)
    lasso = Lasso(alpha=alpha)

    ridge_cv = cross_val_score(ridge, X_scaled, y_multi, cv=5,
       scoring='neg_mean_squared_error')
    lasso_cv = cross_val_score(lasso, X_scaled, y_multi, cv=5,
       scoring='neg_mean_squared_error')

    ridge_scores.append(-ridge_cv.mean())
    lasso_scores.append(-lasso_cv.mean())

```

```
plt.figure(figsize=(10, 4))
plt.semilogx(alphas, ridge_scores, label='Ridge')
plt.semilogx(alphas, lasso_scores, label='Lasso')
plt.xlabel('Alpha (regularization strength)')
plt.ylabel('MSE (CV)')
plt.legend()
plt.title('Choix du parametre de regularisation')
plt.show()

# Meilleur alpha
best_ridge_alpha = alphas[np.argmin(ridge_scores)]
best_lasso_alpha = alphas[np.argmin(lasso_scores)]
print(f"Best Ridge alpha: {best_ridge_alpha:.4f}")
print(f"Best Lasso alpha: {best_lasso_alpha:.4f}")
```

Listing 8.1 – Régression linéaire from scratch et avec scikit-learn

Synthèse

Points clés du chapitre :

- La régression linéaire modélise $y = \boldsymbol{w}^\top \boldsymbol{x} + b$
- Solution analytique : **équations normales** $\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \boldsymbol{y}$
- **Ridge** (L2) : stabilise, ne sélectionne pas
- **Lasso** (L1) : sélection de features automatique
- Toujours **vérifier les hypothèses** (résidus, multicolinéarité)
- Choisir λ par **validation croisée**

Chapitre 9

Régression Logistique et Classification

Objectifs du chapitre

Ce chapitre couvre la classification binaire et multiclasse, avec un focus sur la régression logistique. Nous dérivons le modèle depuis les probabilités, explorons la fonction de coût, et introduisons les métriques de classification.

9.1 Introduction à la Classification

9.1.1 Différence avec la Régression

TABLE 9.1 – Régression vs Classification

	Régression	**Classification**
Variable cible	Continue ($y \in \mathbb{R}$)	Catégorielle ($y \in \{0, 1, \ldots, K-1\}$)
Exemple	Prix, température	Spam/non-spam, diagnostic
Sortie du modèle	Valeur numérique	Probabilité de classe
Métrique typique	MSE, R^2	Accuracy, F1, AUC

9.1.2 Classification Binaire

Définition 9.1 (Classification Binaire)**.** Prédire une variable $y \in \{0, 1\}$ (ou $\{-1, +1\}$) à partir de features $\boldsymbol{x}$.

Exemple 9.1. Problèmes de classification binaire :

- Email : spam ou non-spam
- Image : chat ou chien
- Médical : malade ou sain
- Finance : défaut de paiement ou non

9.1.3 Pourquoi pas la Régression Linéaire ?

On pourrait tenter $y = \boldsymbol{w}^\top \boldsymbol{x} + b$, mais :

— Les prédictions peuvent être < 0 ou > 1

— Ce n'est pas interprétable comme une probabilité

— Sensible aux outliers

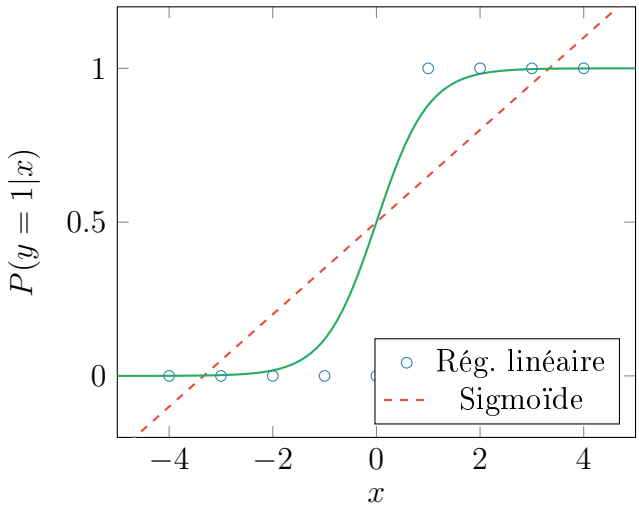


FIGURE 9.1 – La régression linéaire vs la fonction sigmoïde pour la classification

9.2 La Fonction Sigmoïde

9.2.1 Définition

Définition 9.2 (Fonction Sigmoïde / Logistique)**.**

$$\boxed{\sigma(z) = \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z}} \tag{9.1}$$

9.2.2 Propriétés

Théorème 9.1 (Propriétés de la Sigmoïde)**.**

1. ***Image*** *:* $\sigma : \mathbb{R} \to (0, 1)$
2. ***Symétrie*** *:* $\sigma(-z) = 1 - \sigma(z)$
3. ***Dérivée*** *:* $\sigma'(z) = \sigma(z)(1 - \sigma(z))$
4. ***Limites*** *:* $\lim_{z \to -\infty} \sigma(z) = 0$, $\lim_{z \to +\infty} \sigma(z) = 1$
5. ***Point central*** *:* $\sigma(0) = 0.5$

Preuve de la dérivée.

$$\sigma(z) = (1 + e^{-z})^{-1} \tag{9.2}$$

$$\sigma'(z) = -1 \cdot (1 + e^{-z})^{-2} \cdot (-e^{-z}) = \frac{e^{-z}}{(1 + e^{-z})^2} \tag{9.3}$$

On peut réécrire :

$$\sigma'(z) = \frac{1}{1 + e^{-z}} \cdot \frac{e^{-z}}{1 + e^{-z}} = \sigma(z) \cdot \frac{1 + e^{-z} - 1}{1 + e^{-z}} \tag{9.4}$$

$$= \sigma(z)(1 - \sigma(z)) \tag{9.5}$$

□

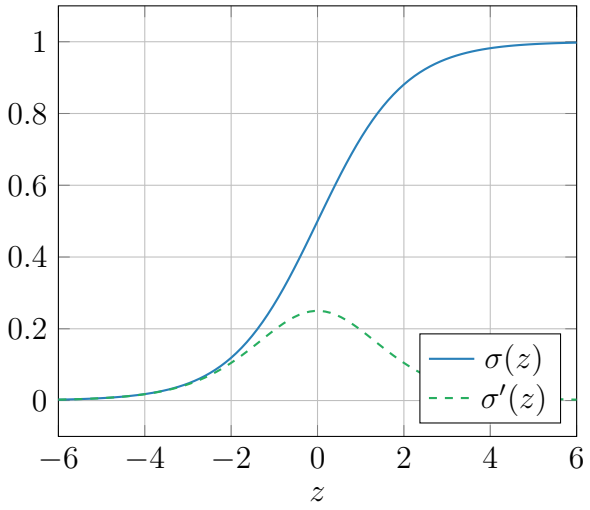


FIGURE 9.2 – La fonction sigmoïde et sa dérivée

9.3 Le Modèle de Régression Logistique

9.3.1 Formulation

Définition 9.3 (Régression Logistique)**.** Le modèle prédit la probabilité de la classe positive :

$$\boxed{P(y = 1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b) = \frac{1}{1 + e^{-(\boldsymbol{w}^\top \boldsymbol{x} + b)}}} \tag{9.6}$$

La prédiction finale :

$$\hat{y} = \begin{cases} 1 & \text{si } P(y = 1|\boldsymbol{x}) \geq 0.5 \\ 0 & \text{sinon} \end{cases} \tag{9.7}$$

9.3.2 Interprétation : Log-Odds

Définition 9.4 (Odds et Log-Odds)**.**

$$\text{Odds} = \frac{P(y=1)}{P(y=0)} = \frac{P(y=1)}{1-P(y=1)} \tag{9.8}$$

$$\text{Log-odds (logit)} = \log\left(\frac{P(y=1)}{P(y=0)}\right) = \boldsymbol{w}^\top \boldsymbol{x} + b \tag{9.9}$$

— Le modèle linéaire est appliqué aux **log-odds**
— Chaque feature multiplie les odds par e^{w_j} pour une augmentation unitaire
— Si $w_j > 0$: augmenter x_j augmente la probabilité de $y = 1$
— w_j représente le changement dans les log-odds pour une unité de x_j

9.4 Fonction de Coût : Cross-Entropy

9.4.1 Pourquoi pas MSE ?

Avec MSE :

$$J_{MSE}(\boldsymbol{w}, b) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i + b))^2 \tag{9.10}$$

Cette fonction est **non-convexe** (nombreux minima locaux).

9.4.2 Dérivation depuis le Maximum de Vraisemblance

Théorème 9.2 (Fonction de Coût Log-Loss)**.** *En maximisant la vraisemblance des données, on obtient la fonction de coût :*

$$\boxed{J(\boldsymbol{w}, b) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]} \tag{9.11}$$

où $\hat{y}_i = \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i + b)$.

Démonstration. La vraisemblance est :

$$L(\boldsymbol{w}, b) = \prod_{i=1}^{n} P(y_i|\boldsymbol{x}_i) = \prod_{i=1}^{n} \hat{y}_i^{y_i}(1-\hat{y}_i)^{1-y_i} \tag{9.12}$$

La log-vraisemblance :

$$\log L = \sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right] \tag{9.13}$$

On minimise $-\log L$ divisé par n pour obtenir J. □

9.4.3 Analyse de la Fonction de Coût

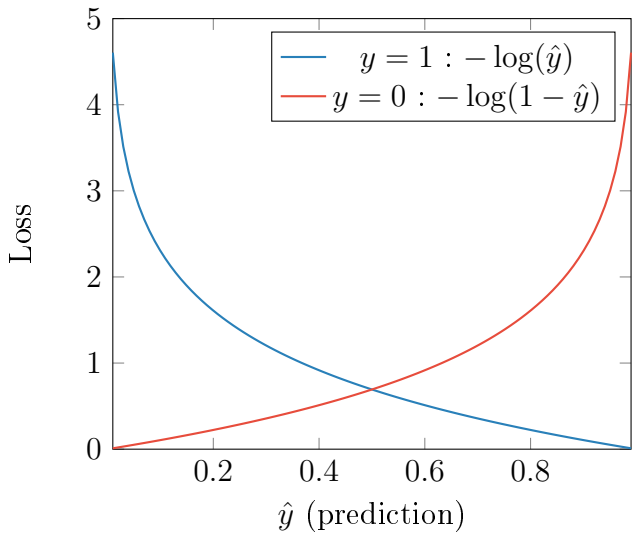


FIGURE 9.3 – Fonction de coût pour les deux classes

Intuition

- Si $y = 1$ et $\hat{y} \to 0$: pénalité $\to \infty$
- Si $y = 1$ et $\hat{y} \to 1$: pénalité $\to 0$
- La pénalité est plus forte pour les erreurs confiantes (mauvaise prédiction avec haute certitude)

9.5 Optimisation

9.5.1 Calcul du Gradient

Théorème 9.3 (Gradient de la Cross-Entropy)**.** *Le gradient est remarquablement simple :*

$$\frac{\partial J}{\partial w_j} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) x_j^{(i)} \tag{9.14}$$

$$\frac{\partial J}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) \tag{9.15}$$

Forme matricielle :

$$\boxed{\nabla_{\boldsymbol{w}} J = \frac{1}{n} \mathbf{X}^\top (\hat{\boldsymbol{y}} - \boldsymbol{y})} \tag{9.16}$$

Remarque 9.1. C'est la même forme que pour la régression linéaire ! Seule la définition de $\hat{y}$ change.

Démonstration. Pour un exemple (x, y) avec $\hat{y} = \sigma(z)$ où $z = \boldsymbol{w}^\top \boldsymbol{x} + b$:

$$\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_j} \tag{9.17}$$

$$= \left(-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}} \right) \cdot \hat{y}(1-\hat{y}) \cdot x_j \tag{9.18}$$

$$= \left(\frac{-y(1-\hat{y}) + (1-y)\hat{y}}{\hat{y}(1-\hat{y})} \right) \cdot \hat{y}(1-\hat{y}) \cdot x_j \tag{9.19}$$

$$= (-y + y\hat{y} + \hat{y} - y\hat{y}) \cdot x_j = (\hat{y} - y)x_j \tag{9.20}$$

□

9.5.2 Algorithme

Algorithm 5 Descente de Gradient pour Régression Logistique

Require: $\mathbf{X}, \boldsymbol{y}, \eta, n\_iter$

$\boldsymbol{w} \leftarrow \mathbf{0}$, $b \leftarrow 0$
for $t = 1$ to n_iter **do**
  $\boldsymbol{z} \leftarrow \mathbf{X}\boldsymbol{w} + b$
  $\hat{\boldsymbol{y}} \leftarrow \sigma(\boldsymbol{z})$ ▷ Application élément par élément
  $\boldsymbol{g}_w \leftarrow \frac{1}{n}\mathbf{X}^\top(\hat{\boldsymbol{y}} - \boldsymbol{y})$
  $g_b \leftarrow \frac{1}{n}\sum_i(\hat{y}_i - y_i)$
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{g}_w$
  $b \leftarrow b - \eta g_b$
end for
return $\boldsymbol{w}, b$

9.6 Frontière de Décision

9.6.1 Définition

Définition 9.5 (Frontière de Décision)**.** L'ensemble des points où $P(y = 1|\boldsymbol{x}) = 0.5$, soit :

$$\boldsymbol{w}^\top \boldsymbol{x} + b = 0 \tag{9.21}$$

C'est un **hyperplan** de dimension $p - 1$ dans l'espace à p dimensions.

9.6.2 Limites du Modèle

La régression logistique standard ne peut résoudre que des problèmes **linéairement séparables**.

Solutions : Features polynomiales, noyaux (SVM), réseaux de neurones.

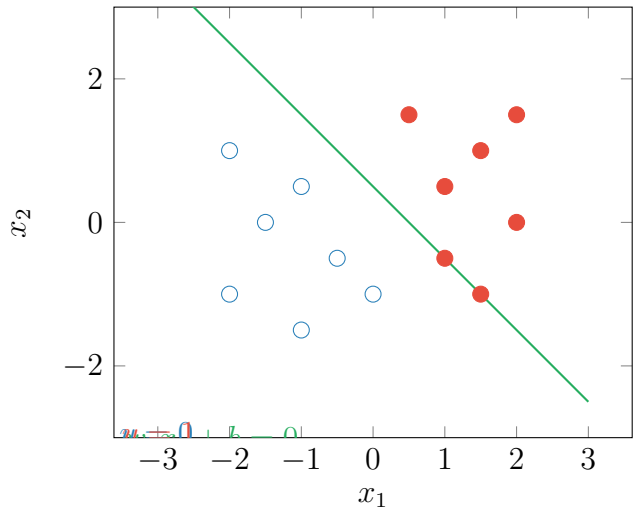


FIGURE 9.4 – Frontière de décision linéaire en 2D

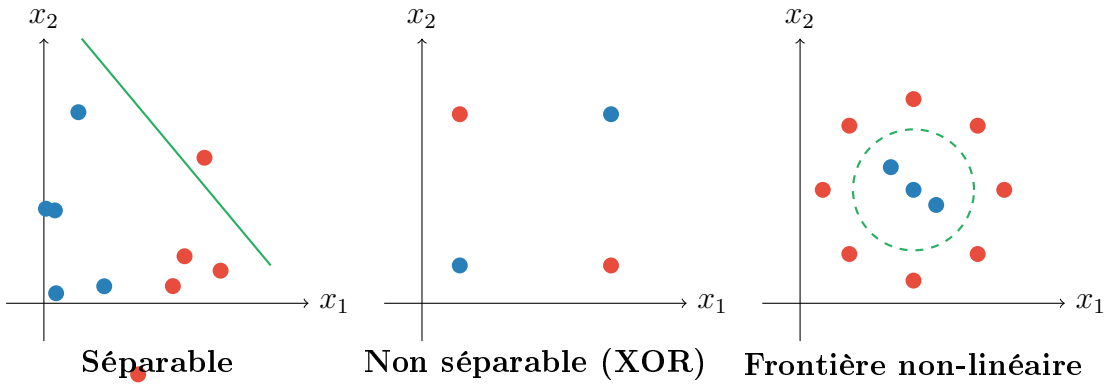


FIGURE 9.5 – Limites de la classification linéaire

9.7 Régularisation

9.7.1 L2 (Ridge)

$$J_{L2}(\boldsymbol{w}, b) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right] + \lambda \left\|\boldsymbol{w}\right\|_2^2 \tag{9.22}$$

9.7.2 L1 (Lasso)

$$J_{L1}(\boldsymbol{w}, b) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right] + \lambda \left\|\boldsymbol{w}\right\|_1 \tag{9.23}$$

Remarque 9.2. Dans scikit-learn, le paramètre est $C = 1/\lambda$ (régularisation inverse).

9.8 Classification Multiclasse

9.8.1 Stratégies One-vs-All et One-vs-One

One-vs-All (OvA) : K classifieurs binaires, chacun sépare une classe de toutes les autres

One-vs-One (OvO) : $\binom{K}{2} = \frac{K(K-1)}{2}$ classifieurs, chacun sépare deux classes

9.8.2 Softmax : Généralisation Multiclasse

Définition 9.6 (Fonction Softmax)**.** Pour K classes :

$$\boxed{P(y = k|\boldsymbol{x}) = \text{softmax}(\boldsymbol{z})_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}} \tag{9.24}$$

où $z_k = \boldsymbol{w}_k^\top \boldsymbol{x} + b_k$.

— $\sum_{k=1}^{K} P(y = k|\boldsymbol{x}) = 1$

— $0 < P(y = k|\boldsymbol{x}) < 1$ pour tout k

— Généralise la sigmoïde (cas $K = 2$)

9.8.3 Cross-Entropy Multiclasse

$$J(\mathbf{W}, \boldsymbol{b}) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} 1_{[y_i=k]} \log P(y = k|\boldsymbol{x}_i) \tag{9.25}$$

Avec one-hot encoding $y_{ik} = 1_{[y_i=k]}$:

$$J = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} \log \hat{y}_{ik} \tag{9.26}$$

9.9 Métriques d'Évaluation

9.9.1 Matrice de Confusion

Définition 9.7 (Matrice de Confusion)**.** Pour une classification binaire :

	Prédit Positif	**Prédit Négatif**
Réel Positif	TP (True Positive)	FN (False Negative)
Réel Négatif	FP (False Positive)	TN (True Negative)

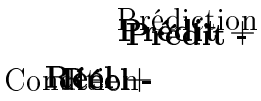


FIGURE 9.6 – Structure de la matrice de confusion

9.9.2 Métriques Dérivées

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{9.27}$$

$$\text{Precision} = \frac{TP}{TP + FP} = P(\text{réel } + \mid \text{prédit } +) \tag{9.28}$$

$$\text{Recall (Sensibilité)} = \frac{TP}{TP + FN} = P(\text{prédit } + \mid \text{réel } +) \tag{9.29}$$

$$\text{Specificity} = \frac{TN}{TN + FP} = P(\text{prédit } - \mid \text{réel } -) \tag{9.30}$$

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{9.31}$$

Important

L'**accuracy** peut être trompeuse avec des classes déséquilibrées !
Exemple : 99% de non-spam → Un modèle trivial "tout est non-spam" a 99% d'accuracy mais 0% de recall pour les spams.

9.9.3 Courbe ROC et AUC

Définition 9.8 (Courbe ROC)**.** La courbe **Receiver Operating Characteristic** trace le True Positive Rate (Recall) vs False Positive Rate pour différents seuils de classification.

$$TPR = \frac{TP}{TP + FN} = \text{Recall} \tag{9.32}$$

$$FPR = \frac{FP}{FP + TN} = 1 - \text{Specificity} \tag{9.33}$$

Définition 9.9 (AUC)**.** L'**Area Under the Curve** est l'aire sous la courbe ROC.

- AUC $= 1$: Classifieur parfait
- AUC $= 0.5$: Classifieur aléatoire
- AUC < 0.5 : Pire qu'aléatoire (inverser les prédictions !)

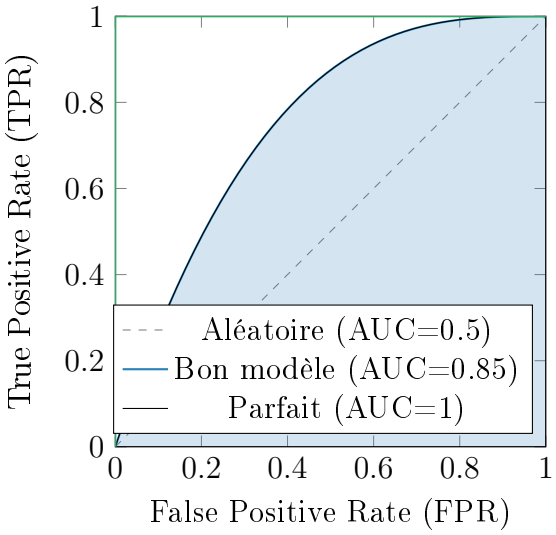


FIGURE 9.7 – Courbe ROC et AUC

9.9.4 Courbe Precision-Recall

Pour les données très déséquilibrées, la courbe **Precision-Recall** est plus informative.

9.10 Exercices

Exercice 9.1 (Dérivation du Gradient)**.** Montrer que pour la cross-entropy :

$$\frac{\partial J}{\partial w_j} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)x_j^{(i)}$$

Exercice 9.2 (Softmax)**.**

1. Montrer que pour $K = 2$, softmax se réduit à la sigmoïde
2. Calculer le gradient de la cross-entropy multiclasse

Exercice 9.3 (Métriques)**.** Un système de détection de fraude a :

	Prédit Fraude	Prédit Normal
Vraie Fraude	80	20
Vrai Normal	100	9800

Calculer : Accuracy, Precision, Recall, F1, Specificity.

Exercice 9.4 (Projet : Titanic)**.** Utilisez le dataset Titanic pour :

1. Prétraiter les données (valeurs manquantes, encoding)
2. Implémenter la régression logistique from scratch
3. Tracer la frontière de décision (2 features)
4. Calculer et interpréter toutes les métriques
5. Tracer la courbe ROC

9.11 Implémentation Python

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (confusion_matrix,
   classification_report,
                             roc_curve, auc,
                                precision_recall_curve)
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification

# ===== Implementation from scratch =====
class LogisticRegressionScratch:
    def __init__(self, lr=0.01, n_iter=1000):
        self.lr = lr
        self.n_iter = n_iter

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-np.clip(z, -500, 500)))

    def fit(self, X, y):
        n, p = X.shape
        self.w = np.zeros(p)
        self.b = 0

        self.losses = []
        for _ in range(self.n_iter):
            # Forward
            z = X @ self.w + self.b
            y_hat = self.sigmoid(z)

            # Loss (cross-entropy)
            loss = -np.mean(y * np.log(y_hat + 1e-8) +
                            (1 - y) * np.log(1 - y_hat + 1e-8))
            self.losses.append(loss)

            # Gradient
            error = y_hat - y
            grad_w = (1/n) * X.T @ error
            grad_b = (1/n) * np.sum(error)

            # Update
            self.w -= self.lr * grad_w
            self.b -= self.lr * grad_b
```

```
42
43          return self
44
45      def predict_proba(self, X):
46          z = X @ self.w + self.b
47          return self.sigmoid(z)
48
49      def predict(self, X, threshold=0.5):
50          return (self.predict_proba(X) >= threshold).astype(int)
51
52  # ===== Donnees =====
53  X, y = make_classification(n_samples=1000, n_features=2,
54                             n_redundant=0, n_informative=2,
55                             random_state=42,
                                n_clusters_per_class=1)
56  X_train, X_test, y_train, y_test = train_test_split(X, y,
        test_size=0.2, random_state=42)
57
58  # ===== Entrainement =====
59  model_scratch = LogisticRegressionScratch(lr=0.1, n_iter=1000)
60  model_scratch.fit(X_train, y_train)
61
62  model_sklearn = LogisticRegression()
63  model_sklearn.fit(X_train, y_train)
64
65  # ===== Evaluation =====
66  y_pred = model_scratch.predict(X_test)
67  y_proba = model_scratch.predict_proba(X_test)
68
69  print("=== Matrice de Confusion ===")
70  print(confusion_matrix(y_test, y_pred))
71
72  print("\n=== Rapport de Classification ===")
73  print(classification_report(y_test, y_pred))
74
75  # ===== Visualisation =====
76  fig, axes = plt.subplots(2, 2, figsize=(12, 10))
77
78  # 1. Frontiere de decision
79  ax = axes[0, 0]
80  xx, yy = np.meshgrid(np.linspace(X[:, 0].min()-1, X[:, 0].max()+1,
        100),
81                       np.linspace(X[:, 1].min()-1, X[:,
                            1].max()+1, 100))
82  Z = model_scratch.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
Z = Z.reshape(xx.shape)
ax.contourf(xx, yy, Z, alpha=0.3, cmap='RdYlBu')
ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='RdYlBu',
    edgecolors='k')
ax.set_title('Frontiere de Decision')
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')

# 2. Courbe de perte
ax = axes[0, 1]
ax.plot(model_scratch.losses)
ax.set_xlabel('Iteration')
ax.set_ylabel('Loss')
ax.set_title('Convergence de la Cross-Entropy')

# 3. Courbe ROC
ax = axes[1, 0]
fpr, tpr, _ = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)
ax.plot(fpr, tpr, label=f'AUC = {roc_auc:.3f}')
ax.plot([0, 1], [0, 1], 'k--')
ax.set_xlabel('False Positive Rate')
ax.set_ylabel('True Positive Rate')
ax.set_title('Courbe ROC')
ax.legend()

# 4. Courbe Precision-Recall
ax = axes[1, 1]
precision, recall, _ = precision_recall_curve(y_test, y_proba)
ax.plot(recall, precision)
ax.set_xlabel('Recall')
ax.set_ylabel('Precision')
ax.set_title('Courbe Precision-Recall')

plt.tight_layout()
plt.show()
```

Listing 9.1 – Régression logistique et métriques

Synthèse

Points clés du chapitre :

— La régression logistique modélise $P(y = 1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)$

— Fonction de coût : **cross-entropy** (dérivée du MLE)

— Gradient simple : $\nabla J = \frac{1}{n}\mathbf{X}^\top(\hat{\boldsymbol{y}} - \boldsymbol{y})$

— **Softmax** généralise au cas multiclasse

— Métriques essentielles : Accuracy, Precision, Recall, F1, AUC

— Attention aux **classes déséquilibrées**

Chapitre 10

Arbres de Décision et Méthodes d'Ensemble

Objectifs du chapitre

Ce chapitre explore les arbres de décision et leurs extensions puissantes : Random Forest, Gradient Boosting, XGBoost. Ces méthodes dominent les compétitions de ML sur données tabulaires.

10.1 Arbres de Décision

10.1.1 Intuition

Définition 10.1 (Arbre de Décision)**.** Un **arbre de décision** partitionne l'espace des features par une série de questions binaires, formant une structure arborescente.

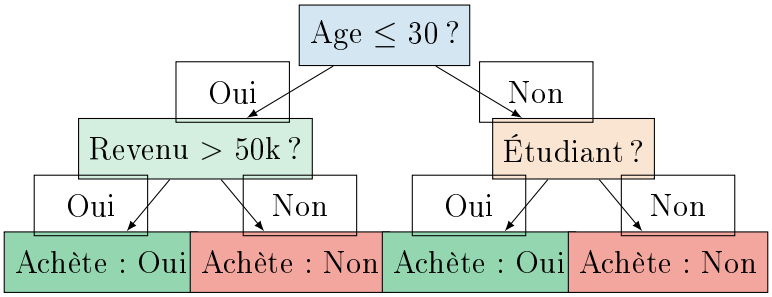


FIGURE 10.1 – Exemple d'arbre de décision pour prédire un achat

10.1.2 Terminologie

— **Nœud racine** : Premier nœud de l'arbre

— **Nœud interne** : Nœud avec une condition de split

— **Feuille (leaf)** : Nœud terminal avec une prédiction

— **Profondeur** : Distance depuis la racine

— **Split** : Division d'un nœud en deux enfants

10.1.3 Avantages et Inconvénients

TABLE 10.1 – Avantages et inconvénients des arbres de décision

Avantages	**Inconvénients**
Interprétable	Tendance à l'overfitting
Pas de normalisation requise	Instable (haute variance)
Gère variables mixtes	Frontières orthogonales
Capture interactions	Biaisé vers features à nombreuses valeurs
Rapide à entraîner	Performance limitée seul

10.2 Construction d'un Arbre : Critères de Split

10.2.1 Impureté d'un Nœud

On cherche à créer des nœuds **purs** (une seule classe).

Définition 10.2 (Entropie)**.** Pour un nœud avec proportions de classes $p_1, p_2, \ldots, p_K$:

$$H = -\sum_{k=1}^{K} p_k \log_2(p_k) \tag{10.1}$$

Convention : $0 \cdot \log(0) = 0$.

Définition 10.3 (Indice de Gini)**.**

$$G = 1 - \sum_{k=1}^{K} p_k^2 = \sum_{k=1}^{K} p_k(1 - p_k) \tag{10.2}$$

10.2.2 Gain d'Information

Définition 10.4 (Gain d'Information)**.** Pour un split divisant le nœud parent P en enfants L (gauche) et R (droite) :

$$\text{Gain} = H(P) - \frac{n_L}{n_P} H(L) - \frac{n_R}{n_P} H(R) \tag{10.3}$$

On choisit le split qui **maximise le gain** (ou minimise l'impureté pondérée).

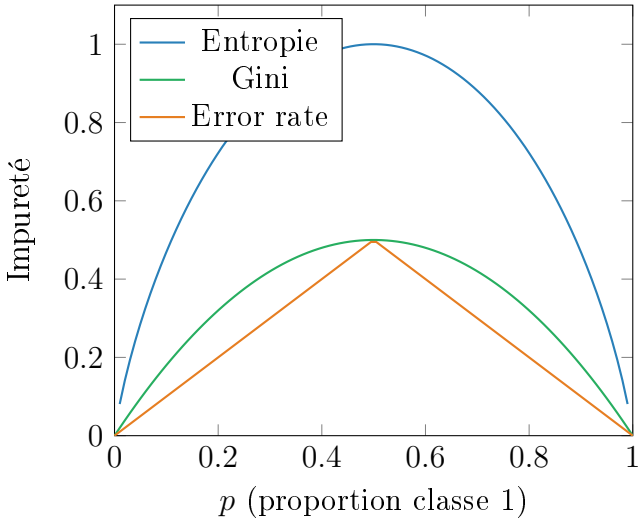


FIGURE 10.2 – Comparaison des mesures d'impureté (classification binaire)

10.2.3 Critère pour la Régression

Définition 10.5 (Variance Reduction)**.** Pour la régression, on minimise la variance :

$$\text{MSE}(S) = \frac{1}{|S|} \sum_{i \in S} (y_i - \bar{y}_S)^2 \tag{10.4}$$

La prédiction dans chaque feuille est la **moyenne** des y dans cette feuille.

10.2.4 Algorithme CART

Algorithm 6 CART (Classification and Regression Trees)

```
procedure BuildTree(S, depth)
    if stopping criterion met then
        return Leaf(majority class or mean)
    end if
    best_gain ← 0
    for each feature j do
        for each threshold t in values of feature j do
            Split S into S_L = {x : x_j ≤ t} and S_R = {x : x_j > t}
            gain ← ComputeGain(S, S_L, S_R)
            if gain > best_gain then
                best_gain, best_j, best_t ← gain, j, t
            end if
        end for
    end for
    S_L, S_R ← Split(S, best_j, best_t)
    left ← BuildTree(S_L, depth + 1)
    right ← BuildTree(S_R, depth + 1)
    return Node(best_j, best_t, left, right)
end procedure
```

10.2.5 Critères d'Arrêt et Élagage

Pré-élagage (early stopping) :

- Profondeur maximale atteinte
- Nombre minimum d'échantillons dans un nœud
- Gain minimum pour split
- Nombre maximum de feuilles

Post-élagage :

1. Construire l'arbre complet
2. Élaguer les branches qui n'améliorent pas la performance en validation
3. Méthode : Cost-Complexity Pruning (paramètre α)

10.3 Random Forest

10.3.1 Le Problème de la Variance

Un arbre unique a une **haute variance** : un petit changement dans les données peut produire un arbre très différent.

Solution : Construire plusieurs arbres et les agréger.

10.3.2 Bagging (Bootstrap Aggregating)

Définition 10.6 (Bagging). 1. Créer B échantillons bootstrap (tirage avec remise)

2. Entraîner un arbre sur chaque échantillon
3. Agréger les prédictions :
   — Classification : Vote majoritaire
   — Régression : Moyenne

Théorème 10.1 (Réduction de Variance par Bagging). *Si les arbres sont identiquement distribués avec variance σ^2 et corrélation ρ :*

$$Var\left(\frac{1}{B}\sum_{b=1}^{B} T_b\right) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \tag{10.5}$$

Pour réduire la variance, il faut :

— *Augmenter B (plus d'arbres)*
— *Réduire ρ (décorréler les arbres)*

10.3.3 Random Forest : Décorrélation

Définition 10.7 (Random Forest). Comme le Bagging, mais avec une modification :

À chaque split, on ne considère qu'un **sous-ensemble aléatoire** de features.

— Classification : $m = \sqrt{p}$ features
— Régression : $m = p/3$ features

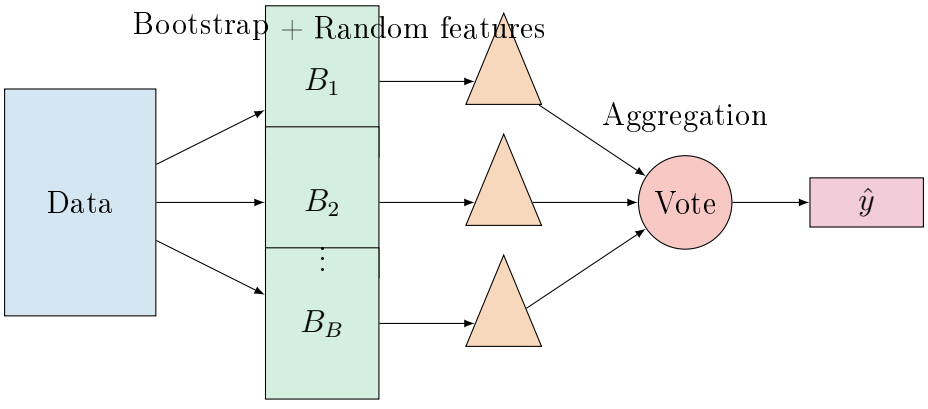


FIGURE 10.3 – Architecture du Random Forest

10.3.4 Out-of-Bag Error

Définition 10.8 (OOB Error). Chaque échantillon bootstrap laisse environ 37% des données de côté.

L'**erreur OOB** évalue chaque exemple avec les arbres qui ne l'ont pas vu.

C'est une estimation non biaisée de l'erreur de généralisation, **sans validation set** !

10.3.5 Importance des Features

Deux méthodes principales :

1. **Importance par impureté (MDI)** :

$$\text{Importance}(j) = \sum_{\text{splits sur } j} \frac{n_{\text{node}}}{n_{\text{total}}} \cdot \Delta\text{impurity} \tag{10.6}$$

2. **Importance par permutation (MDA)** :
   (a) Calculer l'erreur OOB de base
   (b) Permuter aléatoirement la feature j
   (c) Mesurer l'augmentation d'erreur

10.4 Gradient Boosting

10.4.1 Boosting vs Bagging

TABLE 10.2 – Comparaison Bagging vs Boosting

	Bagging	**Boosting**
Arbres	Indépendants en parallèle	Séquentiels
Objectif	Réduire variance	Réduire biais
Poids	Égaux	Adaptés aux erreurs
Overfitting	Peu probable	Possible

10.4.2 Principe du Gradient Boosting

Définition 10.9 (Gradient Boosting)**.** Construire un **modèle additif** :

$$F_M(\boldsymbol{x}) = F_0(\boldsymbol{x}) + \sum_{m=1}^{M} \gamma_m h_m(\boldsymbol{x}) \tag{10.7}$$

où chaque h_m est un arbre qui corrige les erreurs des précédents.

Algorithm 7 Gradient Boosting

Require: $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, fonction de perte L, nombre d'itérations M

1: $F_0(\boldsymbol{x}) \leftarrow \arg\min_\gamma \sum_{i=1}^n L(y_i, \gamma)$ ▷ Initialisation
2: **for** $m = 1$ to M **do**
3: $\quad r_i \leftarrow -\left[\frac{\partial L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right]_{F=F_{m-1}}$ ▷ Pseudo-résidus
4: $\quad$ Fit tree h_m to $\{(\boldsymbol{x}_i, r_i)\}$
5: $\quad \gamma_m \leftarrow \arg\min_\gamma \sum_{i=1}^n L(y_i, F_{m-1}(\boldsymbol{x}_i) + \gamma h_m(\boldsymbol{x}_i))$
6: $\quad F_m(\boldsymbol{x}) \leftarrow F_{m-1}(\boldsymbol{x}) + \eta\gamma_m h_m(\boldsymbol{x})$ ▷ η = learning rate
7: **end for**
8: **return** F_M

10.4.3 Interprétation : Descente de Gradient dans l'Espace des Fonctions

Intuition

À chaque itération :

1. On calcule le gradient de la perte par rapport aux prédictions
2. L'arbre h_m **approxime** ce gradient (direction de descente)
3. On fait un pas dans cette direction

C'est une **descente de gradient fonctionnelle**.

Pour MSE : $L = \frac{1}{2}(y - F)^2$, le pseudo-résidu est $r = y - F$ (le résidu classique).

10.4.4 Hyperparamètres Clés

— `n_estimators` : Nombre d'arbres M

— `learning_rate` (η) : Taux de shrinkage (0.01 à 0.3)

— `max_depth` : Profondeur des arbres (souvent 3-8)

— `subsample` : Fraction des données par arbre

— `colsample` : Fraction des features par arbre

Astuce

Règle du pouce : Learning rate plus petit = plus d'arbres nécessaires.

— $\eta = 0.1$ avec 100 arbres

— $\eta = 0.01$ avec 1000 arbres

10.5 XGBoost, LightGBM, CatBoost

10.5.1 XGBoost

Innovations clés :

1. **Régularisation** explicite dans l'objectif :

$$\mathcal{L} = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{m=1}^{M} \Omega(h_m) \tag{10.8}$$

   où $\Omega(h) = \gamma T + \frac{1}{2}\lambda \|\boldsymbol{w}\|_2^2$ (T = nombre de feuilles)

2. **Approximation de Taylor** d'ordre 2 :

$$L(y, \hat{y} + h) \approx L(y, \hat{y}) + g \cdot h + \frac{1}{2} H \cdot h^2 \tag{10.9}$$

   où $g = \frac{\partial L}{\partial \hat{y}}$ et $H = \frac{\partial^2 L}{\partial \hat{y}^2}$

3. **Parallélisation** des splits
4. **Histograms** pour splits rapides
5. Gestion native des **valeurs manquantes**

10.5.2 LightGBM

Différences avec XGBoost :

— **Leaf-wise** growth (vs level-wise)
— **GOSS** (Gradient-based One-Side Sampling)
— **EFB** (Exclusive Feature Bundling)
— Plus rapide sur grands datasets

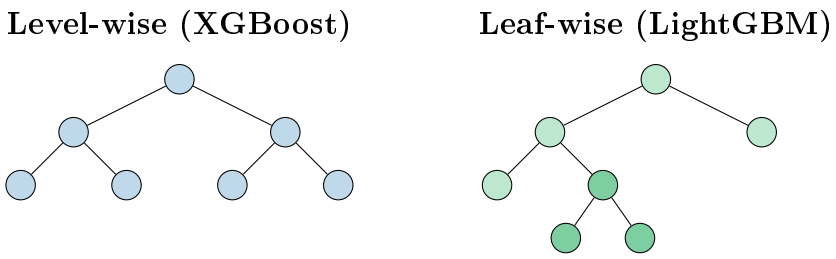


FIGURE 10.4 – Level-wise vs Leaf-wise tree growth

10.5.3 CatBoost

Spécialités :

— Gestion native des **features catégorielles**

— **Ordered boosting** pour réduire le target leakage

— Bon par défaut (moins de tuning)

10.6 Comparaison et Choix

TABLE 10.3 – Comparaison des méthodes d'ensemble

	Random Forest	**XGBoost**	**LightGBM**	**CatBoost**
Vitesse	Moyenne	Rapide	Très rapide	Rapide
Catégorielles	Non natif	Non natif	Partiel	Natif
Valeurs manquantes	Non	Natif	Natif	Natif
Overfitting	Robuste	À surveiller	À surveiller	Robuste
Tuning requis	Peu	Moyen	Moyen	Peu

Astuce

Règles de sélection :

— **Petit dataset** : Random Forest ou CatBoost

— **Grand dataset** : LightGBM

— **Beaucoup de catégorielles** : CatBoost

— **Compétition Kaggle** : XGBoost ou LightGBM avec tuning

— **Production** : CatBoost (peu de tuning, robuste)

10.7 Exercices

Exercice 10.1 (Construction d'un Arbre)**.** Données :

Météo	Température	Jouer Tennis
Soleil	Chaud	Non
Soleil	Froid	Oui
Nuageux	Chaud	Oui
Pluie	Froid	Non
Pluie	Chaud	Oui
Nuageux	Froid	Oui

1. Calculer l'entropie initiale
2. Calculer le gain d'information pour chaque feature
3. Quel split choisir en premier ?

Exercice 10.2 (Random Forest vs Single Tree)**.** Expliquez pourquoi la variance d'un Random Forest est inférieure à celle d'un arbre unique, même si les arbres individuels ne sont pas meilleurs.

Exercice 10.3 (Gradient Boosting)**.**

1. Pour MSE, montrer que le pseudo-résidu est le résidu classique
2. Pourquoi utilise-t-on des arbres peu profonds dans le boosting ?

Exercice 10.4 (Projet : Classification Multiclasse)**.** Utilisez le dataset Iris pour :

1. Entraîner et visualiser un arbre de décision
2. Comparer Random Forest, XGBoost, et LightGBM
3. Analyser l'importance des features
4. Optimiser les hyperparamètres par grid search

10.8 Implémentation Python

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier,
    GradientBoostingClassifier
from sklearn.datasets import make_classification, load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
import xgboost as xgb
import lightgbm as lgb

# ===== Donnees =====
X, y = make_classification(n_samples=1000, n_features=10,
                           n_informative=5, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2)

# ===== Arbre de Decision =====
tree = DecisionTreeClassifier(max_depth=4, random_state=42)
tree.fit(X_train, y_train)
print(f"Decision Tree: {tree.score(X_test, y_test):.4f}")

# Visualisation
plt.figure(figsize=(20, 10))
plot_tree(tree, filled=True, feature_names=[f'x{i}' for i in
    range(10)])
plt.show()

# ===== Random Forest =====
```

```
rf = RandomForestClassifier(n_estimators=100, max_depth=10,
                            random_state=42, oob_score=True)
rf.fit(X_train, y_train)
print(f"Random Forest: {rf.score(X_test, y_test):.4f}")
print(f"OOB Score: {rf.oob_score_:.4f}")

# Importance des features
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 6))
plt.bar(range(10), importances[indices])
plt.xticks(range(10), [f'x{i}' for i in indices])
plt.title('Feature Importance (Random Forest)')
plt.show()

# ===== Gradient Boosting (sklearn) =====
gb = GradientBoostingClassifier(n_estimators=100,
    learning_rate=0.1,
                                max_depth=3, random_state=42)
gb.fit(X_train, y_train)
print(f"Gradient Boosting: {gb.score(X_test, y_test):.4f}")

# ===== XGBoost =====
xgb_model = xgb.XGBClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=3,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42
)
xgb_model.fit(X_train, y_train)
print(f"XGBoost: {xgb_model.score(X_test, y_test):.4f}")

# ===== LightGBM =====
lgb_model = lgb.LGBMClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=3,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42,
    verbose=-1
)
```

```
lgb_model.fit(X_train, y_train)
print(f"LightGBM: {lgb_model.score(X_test, y_test):.4f}")

# ===== Hyperparameter Tuning =====
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2]
}

grid_search = GridSearchCV(
    xgb.XGBClassifier(random_state=42),
    param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)
grid_search.fit(X_train, y_train)

print(f"\nBest params: {grid_search.best_params_}")
print(f"Best CV score: {grid_search.best_score_:.4f}")
print(f"Test score: {grid_search.score(X_test, y_test):.4f}")

# ===== Courbe d'apprentissage =====
train_scores = []
test_scores = []
n_trees = range(10, 201, 10)

for n in n_trees:
    model = xgb.XGBClassifier(n_estimators=n, random_state=42,
        learning_rate=0.1)
    model.fit(X_train, y_train)
    train_scores.append(model.score(X_train, y_train))
    test_scores.append(model.score(X_test, y_test))

plt.figure(figsize=(10, 6))
plt.plot(n_trees, train_scores, label='Train')
plt.plot(n_trees, test_scores, label='Test')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Learning Curve - XGBoost')
plt.show()
```

Listing 10.1 – Arbres et méthodes d'ensemble

Synthèse

Points clés du chapitre :

- Les arbres de décision sont interprétables mais instables
- **Random Forest** : Bagging + random features → réduit la variance
- **Gradient Boosting** : Modèle additif séquentiel → réduit le biais
- **XGBoost/LightGBM/CatBoost** : Implémentations optimisées de boosting
- Les méthodes d'ensemble dominent sur les données tabulaires
- Hyperparamètres clés : nombre d'arbres, profondeur, learning rate

Chapitre 11

Machines à Vecteurs de Support (SVM)

Objectifs du chapitre

Les SVM sont des classificateurs puissants basés sur la théorie de l'optimisation convexe. Ce chapitre couvre la marge maximale, les SVM à marge souple, l'astuce du noyau, et les extensions.

11.1 Introduction : L'Hyperplan à Marge Maximale

11.1.1 Motivation

Pour un problème linéairement séparable, il existe une infinité d'hyperplans séparateurs. Lequel choisir ?

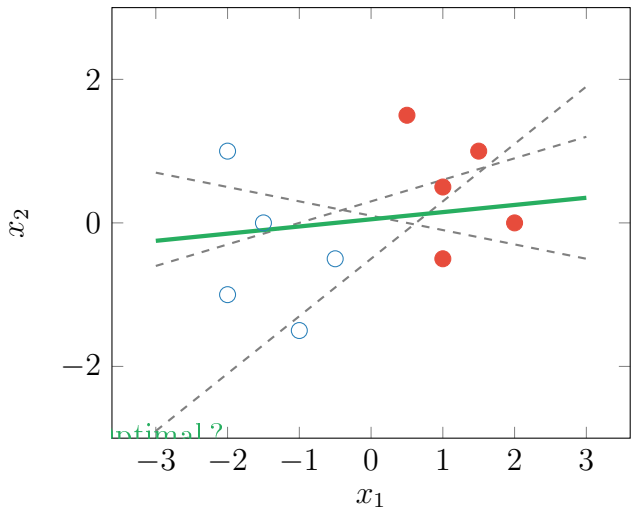


FIGURE 11.1 – Plusieurs hyperplans séparateurs possibles

Intuition : L'hyperplan le plus "sûr" est celui qui maximise la **marge** — la distance aux points les plus proches.

11.1.2 Définitions Géométriques

Définition 11.1 (Hyperplan)**.** Un hyperplan en dimension p est défini par :

$$H = \{\boldsymbol{x} \in \mathbb{R}^p : \boldsymbol{w}^\top \boldsymbol{x} + b = 0\} \tag{11.1}$$

où $\boldsymbol{w}$ est le vecteur normal et b le biais.

Définition 11.2 (Distance d'un Point à un Hyperplan)**.** La distance signée d'un point $\boldsymbol{x}_0$ à l'hyperplan est :

$$d(\boldsymbol{x}_0, H) = \frac{\boldsymbol{w}^\top \boldsymbol{x}_0 + b}{\|\boldsymbol{w}\|} \tag{11.2}$$

Définition 11.3 (Marge)**.** La **marge** est la distance minimale entre l'hyperplan et les points les plus proches :

$$\gamma = \min_i \frac{y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)}{\|\boldsymbol{w}\|} \tag{11.3}$$

où $y_i \in \{-1, +1\}$ est la classe.

11.2 SVM à Marge Dure (Hard Margin)

11.2.1 Formulation du Problème

Objectif : Maximiser la marge sous contrainte de classification correcte.

$$\max_{\boldsymbol{w},b} \gamma \quad \text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq \gamma \|\boldsymbol{w}\|, \ \forall i \tag{11.4}$$

En normalisant $\|\boldsymbol{w}\|$ tel que la marge fonctionnelle soit 1 :

[SVM Hard Margin - Formulation Primale]

$$\boxed{\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 \quad \text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1, \ \forall i} \tag{11.5}$$

Intuition

Minimiser $\|\boldsymbol{w}\|$ revient à maximiser la marge $\gamma = 1/\|\boldsymbol{w}\|$.

11.2.2 Vecteurs de Support

Définition 11.4 (Vecteurs de Support)**.** Les **vecteurs de support** sont les points qui satisfont l'égalité :

$$y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) = 1 \tag{11.6}$$

Ce sont les points les plus proches de l'hyperplan, et ils **définissent** entièrement la solution.

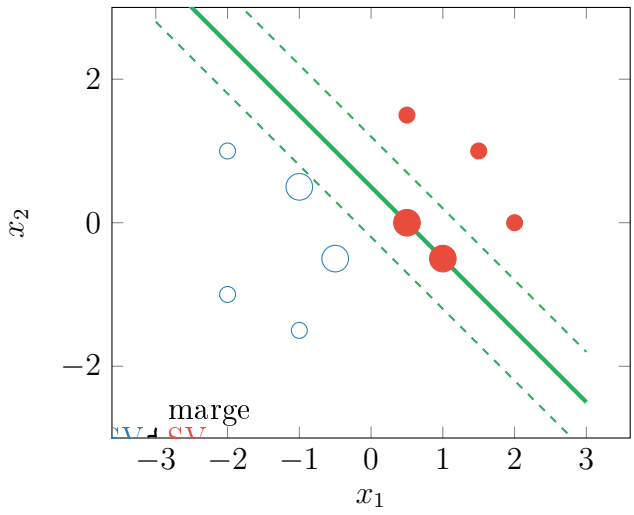


FIGURE 11.2 – Vecteurs de support et hyperplan à marge maximale

11.2.3 Formulation Duale

La formulation duale est cruciale pour l'astuce du noyau.

Théorème 11.1 (Problème Dual SVM)**.**

$$\boxed{\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j} \tag{11.7}$$

$$s.t. \quad \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{11.8}$$

Esquisse de preuve. Le Lagrangien est :

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) - 1] \tag{11.9}$$

Conditions KKT :

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = 0 \implies \boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i \tag{11.10}$$

$$\frac{\partial \mathcal{L}}{\partial b} = 0 \implies \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{11.11}$$

Substitution dans $\mathcal{L}$ donne le dual. □

Corollaire 11.1 (Expression de $\boldsymbol{w}$). *Le vecteur optimal est une combinaison linéaire des vecteurs de support :*

$$\boldsymbol{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \boldsymbol{x}_i \tag{11.12}$$

où $\alpha_i^* > 0$ uniquement pour les vecteurs de support.

11.3 SVM à Marge Souple (Soft Margin)

11.3.1 Le Problème des Données Non-Séparables

En pratique, les données sont rarement parfaitement séparables. On introduit des **variables de slack** ξ_i.

[SVM Soft Margin]

$$\boxed{\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i} \tag{11.13}$$

$$\text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \tag{11.14}$$

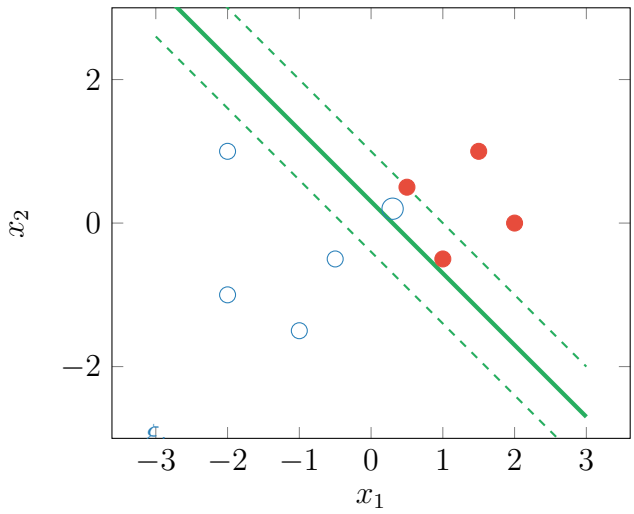


FIGURE 11.3 – SVM à marge souple avec variable de slack ξ_i

11.3.2 Interprétation de ξ_i et C

- $\xi_i = 0$: Point correctement classé, à distance $\geq$ marge
- $0 < \xi_i < 1$: Point correctement classé mais dans la marge
- $\xi_i = 1$: Point sur l'hyperplan
- $\xi_i > 1$: Point mal classé

Le paramètre C contrôle le **trade-off** :

— C grand : Pénalise fortement les erreurs → marge étroite

— C petit : Tolère les erreurs → marge large

11.3.3 Formulation Duale Soft Margin

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j \tag{11.15}$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{11.16}$$

La seule différence : $\alpha_i \leq C$ au lieu de α_i non borné.

11.4 L'Astuce du Noyau (Kernel Trick)

11.4.1 Motivation : Données Non-Linéairement Séparables

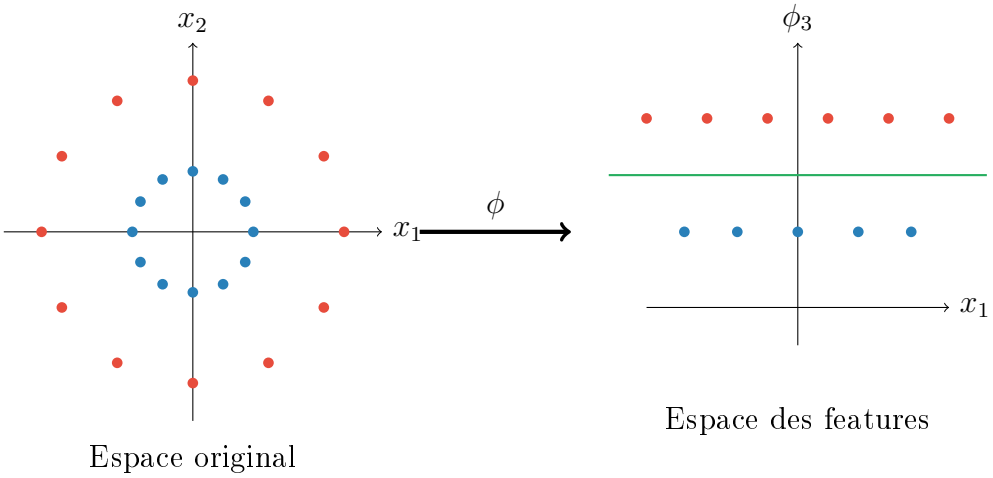


FIGURE 11.4 – Transformation vers un espace où les données sont linéairement séparables

11.4.2 Transformation ϕ

On mappe les données vers un espace de plus grande dimension :

$$\phi : \mathbb{R}^p \to \mathcal{H} \quad \text{(espace de Hilbert)} \tag{11.17}$$

Exemple 11.1. En 2D, avec $\boldsymbol{x} = (x_1, x_2)$:

$$\phi(\boldsymbol{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2, \sqrt{2c}x_1, \sqrt{2c}x_2, c) \tag{11.18}$$

Un cercle dans $\mathbb{R}^2$ devient un hyperplan dans $\mathbb{R}^6$!

11.4.3 Le Kernel Trick

Dans la formulation duale, les données n'apparaissent qu'à travers leurs **produits scalaires** $\boldsymbol{x}_i^\top \boldsymbol{x}_j$.

Définition 11.5 (Fonction Noyau)**.** Une fonction noyau K calcule le produit scalaire dans l'espace des features sans calculer explicitement ϕ :

$$\boxed{K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^\top \phi(\boldsymbol{x}_j)} \tag{11.19}$$

Théorème 11.2 (Mercer)**.** *Une fonction K est un noyau valide si et seulement si la matrice de Gram $\mathbf{K}$ avec $K_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ est **semi-définie positive** pour toutes données.*

11.4.4 Noyaux Classiques

TABLE 11.1 – Principaux noyaux pour SVM

Noyau	Formule	Usage
Linéaire	$K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\top \boldsymbol{x}'$	Données linéairement séparables
Polynomial	$K(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\top \boldsymbol{x}' + c)^d$	Frontières polynomiales
RBF (Gaussien)	$K(\boldsymbol{x}, \boldsymbol{x}') = e^{-\gamma \|\boldsymbol{x} - \boldsymbol{x}'\|^2}$	Le plus polyvalent
Sigmoïde	$K(\boldsymbol{x}, \boldsymbol{x}') = \tanh(\gamma \boldsymbol{x}^\top \boldsymbol{x}' + c)$	Similaire aux réseaux de neurones

11.4.5 Le Noyau RBF en Détail

$$K_{RBF}(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\gamma \left\|\boldsymbol{x} - \boldsymbol{x}'\right\|^2\right) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2\sigma^2}\right) \tag{11.20}$$

où $\gamma = \frac{1}{2\sigma^2}$.

- $K(\boldsymbol{x}, \boldsymbol{x}) = 1$ (auto-similarité maximale)
- $K \to 0$ quand $\|\boldsymbol{x} - \boldsymbol{x}'\| \to \infty$
- Espace des features de dimension **infinie** !

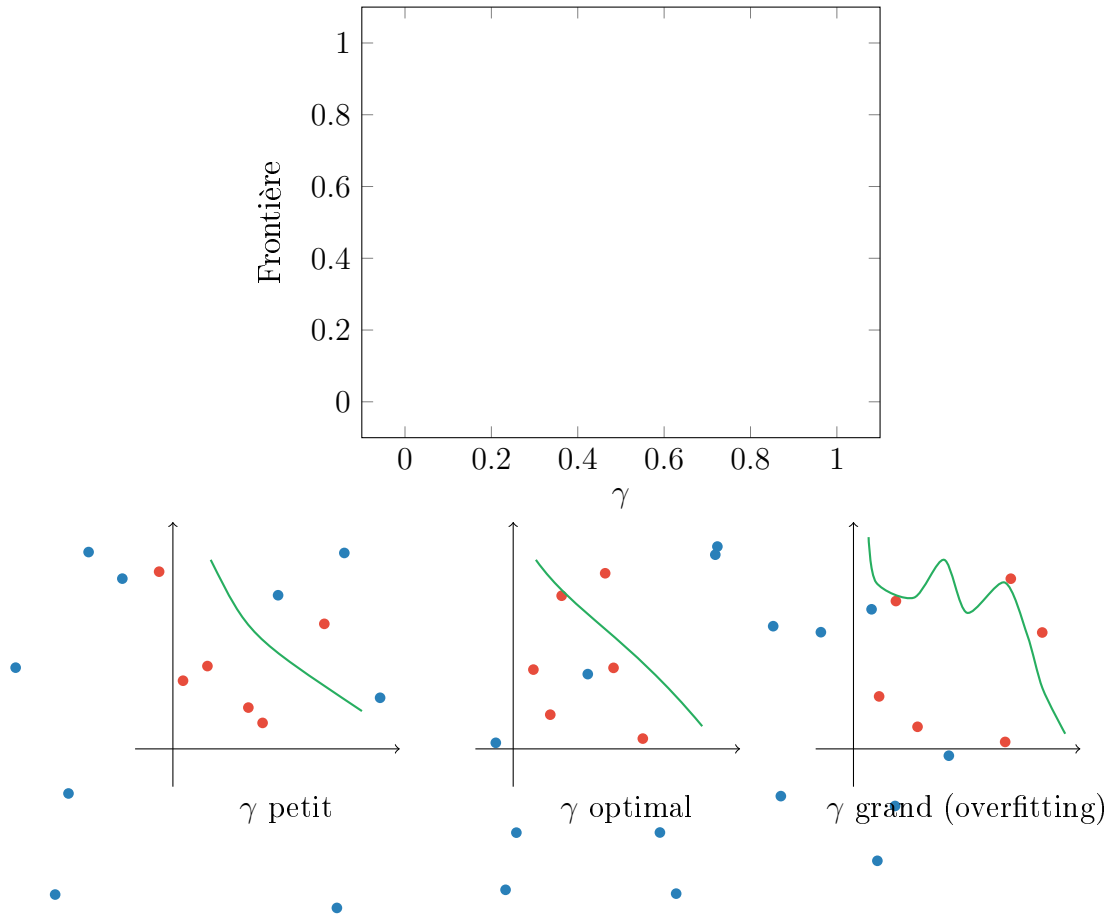


FIGURE 11.5 – Effet du paramètre γ sur la frontière de décision

11.5 Prédiction avec SVM

11.5.1 Fonction de Décision

$$f(\boldsymbol{x}) = \sum_{i \in SV} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b \tag{11.21}$$

$$\hat{y} = \text{sign}(f(\boldsymbol{x})) \tag{11.22}$$

Remarque 11.1. La prédiction ne dépend que des **vecteurs de support**, pas de tous les points d'entraînement. C'est une forme de **compression** des données.

11.5.2 Calcul du Biais b

Pour tout vecteur de support $\boldsymbol{x}_s$ avec $0 < \alpha_s < C$:

$$b = y_s - \sum_{i \in SV} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}_s) \tag{11.23}$$

En pratique, on moyenne sur tous ces vecteurs de support.

11.6 SVM Multi-Classes

Les SVM sont naturellement binaires. Pour K classes :

One-vs-All (OvA) : K classifieurs, chacun sépare une classe des autres

One-vs-One (OvO) : $\binom{K}{2}$ classifieurs, vote majoritaire

	OvA	**OvO**
Nombre de classifieurs	K	$K(K-1)/2$
Taille du problème	Plus grand	Plus petit
Temps d'entraînement	Plus rapide	Plus lent (si K grand)
Déséquilibre	Problématique	Moins problématique

11.7 SVM pour la Régression (SVR)

Définition 11.6 (Support Vector Regression)**.** Au lieu de maximiser la marge autour d'un hyperplan, on définit un **tube** ϵ autour de la fonction de régression.

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \tag{11.24}$$

$$\text{s.t.} \quad y_i - (\boldsymbol{w}^\top \boldsymbol{x}_i + b) \leq \epsilon + \xi_i \tag{11.25}$$

$$(\boldsymbol{w}^\top \boldsymbol{x}_i + b) - y_i \leq \epsilon + \xi_i^* \tag{11.26}$$

11.8 Choix des Hyperparamètres

11.8.1 Grid Search pour C et γ

```python
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': ['scale', 'auto', 0.01, 0.1, 1]
}

grid = GridSearchCV(SVC(kernel='rbf'), param_grid, cv=5,
   scoring='accuracy')
```

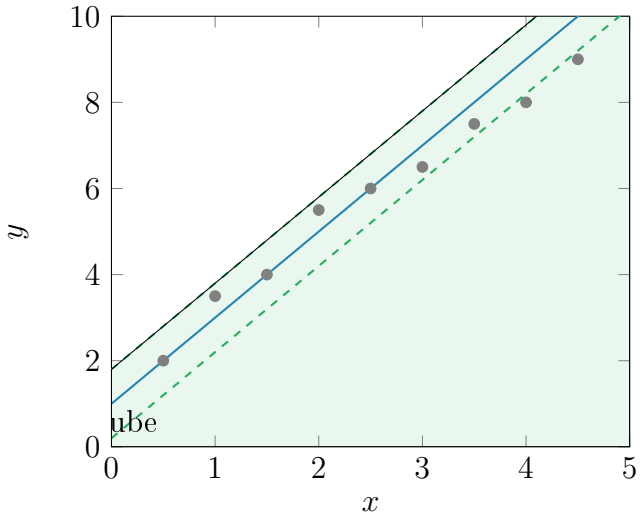


FIGURE 11.6 – SVR avec tube ϵ

```
grid.fit(X_train, y_train)

print(f"Best params: {grid.best_params_}")
```

Listing 11.1 – Grid search pour SVM

11.8.2 Règles Empiriques

— Commencer avec $C = 1$ et $\gamma = 1/(n_{features} \cdot \text{Var}(X))$ (default 'scale')

— **Standardiser** les données (crucial pour SVM !)

— Utiliser validation croisée

— C et γ interagissent : chercher sur une grille

11.9 Exercices

Exercice 11.1 (Marge)**.** Pour les points $(1, 1, +1)$, $(2, 2, +1)$, $(-1, -1, -1)$, $(-2, -2, -1)$:

1. Trouver l'hyperplan optimal
2. Calculer la marge
3. Identifier les vecteurs de support

Exercice 11.2 (Noyau Polynomial)**.** Montrer que le noyau polynomial $K(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\top \boldsymbol{x}' + 1)^2$ correspond à la transformation :

$$\phi(\boldsymbol{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$$

pour $\boldsymbol{x} = (x_1, x_2)$.

Exercice 11.3 (Kernel Trick). Soit $K(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\|\boldsymbol{x} - \boldsymbol{x}'\|^2)$.

1. Montrer que $K(\boldsymbol{x}, \boldsymbol{x}) = 1$
2. Montrer que $0 < K(\boldsymbol{x}, \boldsymbol{x}') \leq 1$
3. Pourquoi ce noyau correspond-il à un espace de dimension infinie ?

11.10 Implémentation Python

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC, SVR
from sklearn.datasets import make_circles, make_moons
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.pipeline import Pipeline

# ===== Donnees non-lineaires =====
X, y = make_circles(n_samples=300, noise=0.1, factor=0.5,
    random_state=42)

# ===== Visualisation de l'effet du noyau =====
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

kernels = ['linear', 'poly', 'rbf', 'sigmoid']
for ax, kernel in zip(axes.flat, kernels):
    clf = SVC(kernel=kernel, gamma='auto', C=1)
    clf.fit(X, y)

    # Frontiere de decision
    xx, yy = np.meshgrid(np.linspace(-2, 2, 100), np.linspace(-2,
        2, 100))
    Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    ax.contourf(xx, yy, Z, levels=[-1, 0, 1], alpha=0.3,
        colors=['blue', 'white', 'red'])
    ax.contour(xx, yy, Z, levels=[-1, 0, 1], colors='k',
        linestyles=['--', '-', '--'])
    ax.scatter(X[:, 0], X[:, 1], c=y, cmap='RdYlBu',
        edgecolors='k')

    # Vecteurs de support
    ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:,
        1],
```

```
                s=100, facecolors='none', edgecolors='green',
                   linewidths=2)

    ax.set_title(f'Kernel: {kernel}\nSupport vectors:
       {len(clf.support_vectors_)}')

plt.tight_layout()
plt.show()

# ===== Pipeline avec standardisation =====
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('svm', SVC(kernel='rbf'))
])

param_grid = {
    'svm__C': [0.1, 1, 10, 100],
    'svm__gamma': [0.001, 0.01, 0.1, 1]
}

grid = GridSearchCV(pipe, param_grid, cv=5, scoring='accuracy')
grid.fit(X, y)

print(f"Best params: {grid.best_params_}")
print(f"Best score: {grid.best_score_:.4f}")

# ===== Visualisation C vs gamma =====
C_range = np.logspace(-2, 2, 10)
gamma_range = np.logspace(-3, 1, 10)
scores = np.zeros((len(C_range), len(gamma_range)))

for i, C in enumerate(C_range):
    for j, gamma in enumerate(gamma_range):
        clf = SVC(kernel='rbf', C=C, gamma=gamma)
        clf.fit(X, y)
        scores[i, j] = clf.score(X, y)

plt.figure(figsize=(10, 8))
plt.imshow(scores, interpolation='nearest', cmap='viridis')
plt.xlabel('gamma')
plt.ylabel('C')
plt.colorbar(label='Accuracy')
plt.xticks(np.arange(len(gamma_range)), [f'{g:.3f}' for g in
   gamma_range], rotation=45)
plt.yticks(np.arange(len(C_range)), [f'{c:.2f}' for c in C_range])
```

```
plt.title('Grille de recherche C vs gamma')
plt.show()

# ===== SVR (Regression) =====
np.random.seed(42)
X_reg = np.sort(5 * np.random.rand(100, 1), axis=0)
y_reg = np.sin(X_reg).ravel() + np.random.randn(100) * 0.1

svr_rbf = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=0.1)
svr_rbf.fit(X_reg, y_reg)

X_plot = np.linspace(0, 5, 200).reshape(-1, 1)
y_pred = svr_rbf.predict(X_plot)

plt.figure(figsize=(10, 6))
plt.scatter(X_reg, y_reg, c='k', label='Data')
plt.plot(X_plot, y_pred, 'r-', label='SVR prediction')
plt.fill_between(X_plot.ravel(), y_pred - 0.1, y_pred + 0.1,
                 alpha=0.3, color='r', label='epsilon-tube')
plt.legend()
plt.title('Support Vector Regression')
plt.show()
```

Listing 11.2 – SVM avec scikit-learn

Synthèse

Points clés du chapitre :

- SVM cherche l'hyperplan à **marge maximale**
- **Soft margin** : tolère les erreurs via paramètre C
- **Kernel trick** : travaille dans un espace de grande dimension sans le calculer
- Noyau RBF : le plus versatile, paramètre γ
- **Standardisation** des données est cruciale
- Seuls les **vecteurs de support** comptent pour la prédiction

Chapitre 12

Apprentissage Non-Supervisé

Objectifs du chapitre

Ce chapitre couvre les principales techniques d'apprentissage non-supervisé : clustering (K-Means, DBSCAN, hiérarchique), réduction de dimensionnalité (PCA, t-SNE), et leurs applications.

12.1 Introduction

12.1.1 Qu'est-ce que l'Apprentissage Non-Supervisé ?

Définition 12.1 (Apprentissage Non-Supervisé)**.** Apprentissage à partir de données **non étiquetées**. L'objectif est de découvrir des structures cachées dans les données.

TABLE 12.1 – Supervisé vs Non-Supervisé

	Supervisé	**Non-Supervisé**
Labels	Oui	Non
Objectif	Prédire y	Découvrir la structure
Évaluation	Facile (métrique)	Difficile
Exemples	Classification, Régression	Clustering, PCA

12.1.2 Applications

— **Segmentation client** : Grouper les clients par comportement

— **Compression** : Réduire la dimensionnalité des images

— **Détection d'anomalies** : Identifier les transactions frauduleuses

— **Visualisation** : Projeter en 2D/3D des données haute dimension

— **Prétraitement** : Réduire le bruit, feature engineering

12.2 Clustering : K-Means

12.2.1 L'Algorithme K-Means

Définition 12.2 (K-Means)**.** Partitionner n points en K clusters en minimisant l'inertie :

$$\boxed{J = \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in C_k} \|\boldsymbol{x}_i - \boldsymbol{\mu}_k\|^2} \tag{12.1}$$

où $\boldsymbol{\mu}_k$ est le centroïde du cluster C_k.

Algorithm 8 K-Means (Lloyd's Algorithm)

Require: $\mathbf{X}$, nombre de clusters K

1: Initialiser aléatoirement K centroïdes $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$

2: **repeat**

3: **Affectation** : Assigner chaque point au centroïde le plus proche

$$c_i = \arg\min_k \|\boldsymbol{x}_i - \boldsymbol{\mu}_k\|^2$$

4: **Mise à jour** : Recalculer les centroïdes

$$\boldsymbol{\mu}_k = \frac{1}{|C_k|} \sum_{\boldsymbol{x}_i \in C_k} \boldsymbol{x}_i$$

5: **until** convergence (affectations stables)

6: **return** Clusters $C_1, \ldots, C_K$ et centroïdes $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$

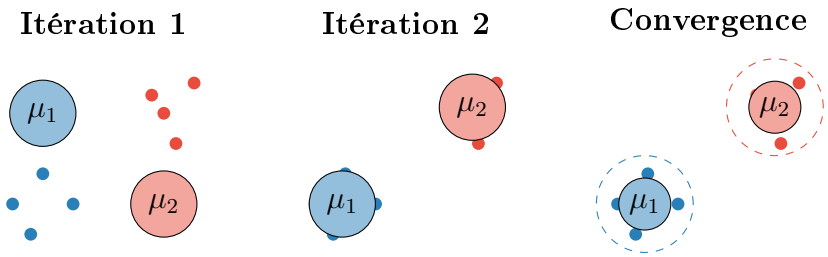


FIGURE 12.1 – Convergence de K-Means

12.2.2 Propriétés et Limites

Théorème 12.1 (Convergence de K-Means)**.** *K-Means converge toujours vers un minimum local en un nombre fini d'itérations.*

Attention

Limites de K-Means :

- Sensible à l'initialisation → K-Means++ améliore
- Suppose des clusters **sphériques**
- Nécessite de spécifier K à l'avance
- Sensible aux outliers

12.2.3 Choix de K : Méthode du Coude

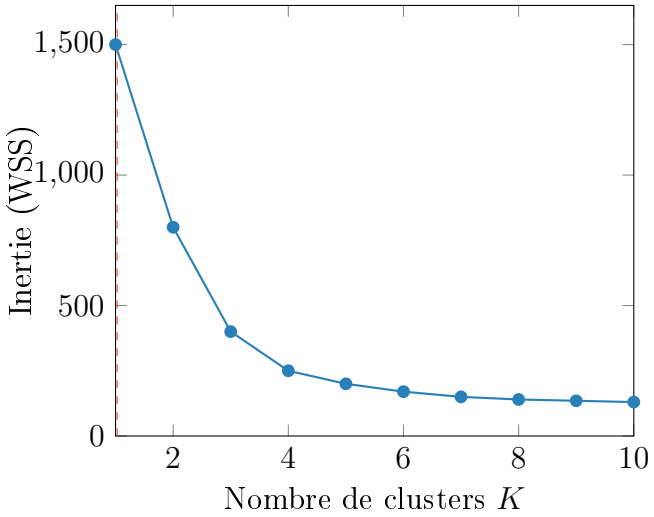


FIGURE 12.2 – Méthode du coude : choisir K au "coude" de la courbe

12.2.4 Score Silhouette

Définition 12.3 (Coefficient de Silhouette)**.** Pour un point i :

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{12.2}$$

où $a(i)$ = distance moyenne aux points du même cluster, $b(i)$ = distance moyenne au cluster le plus proche.

- $s \approx 1$: Bien assigné
- $s \approx 0$: À la frontière
- $s < 0$: Probablement mal assigné

12.3 Clustering Hiérarchique

12.3.1 Principe

Le clustering hiérarchique construit une hiérarchie de clusters (dendrogramme).

Agglomératif (bottom-up) : Chaque point est un cluster, puis fusion progressive

Divisif (top-down) : Tous les points dans un cluster, puis division

12.3.2 Algorithme Agglomératif

Algorithm 9 Clustering Hiérarchique Agglomératif

```
Initialiser : chaque point est un cluster
while nombre de clusters > 1 do
    Trouver les deux clusters les plus proches
    Fusionner ces deux clusters
    Mettre à jour la matrice de distances
end while
```

12.3.3 Critères de Liaison (Linkage)

Comment mesurer la distance entre deux clusters A et B ?

TABLE 12.2 – Méthodes de liaison

Méthode	**Distance** $d(A, B)$
Single linkage	$\min_{a \in A, b \in B} d(a, b)$
Complete linkage	$\max_{a \in A, b \in B} d(a, b)$
Average linkage	$\frac{1}{\lvert A \rvert \lvert B \rvert} \sum_{a \in A} \sum_{b \in B} d(a, b)$
Ward	Augmentation de l'inertie intra-cluster

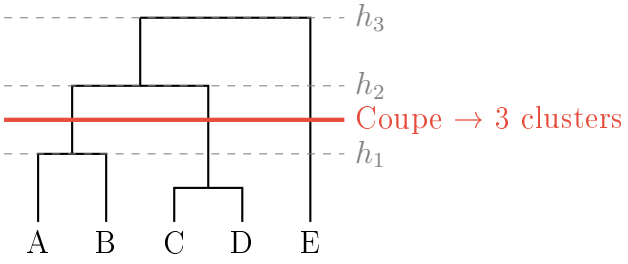


FIGURE 12.3 – Dendrogramme et coupe horizontale

12.4 DBSCAN

12.4.1 Motivation

K-Means et le clustering hiérarchique supposent des clusters convexes. DBSCAN trouve des clusters de forme arbitraire.

Définition 12.4 (DBSCAN (Density-Based Spatial Clustering))**.** Deux paramètres : ϵ (rayon) et minPts (nombre minimum de voisins).

Types de points :

— **Core point** : $\geq$ minPts voisins dans un rayon ϵ

— **Border point** : Voisin d'un core point mais pas core lui-même

— **Noise point** : Ni core ni border

Algorithm 10 DBSCAN

```
for chaque point p non visité do
    Marquer p comme visité
    N ← voisins de p dans un rayon ε
    if |N| < minPts then
        Marquer p comme bruit
    else
        Créer un nouveau cluster C
        Étendre le cluster à partir de p
    end if
end for
```

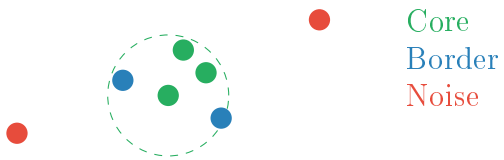


FIGURE 12.4 – Types de points dans DBSCAN

12.4.2 Avantages et Inconvénients

Avantages	Inconvénients
Trouve des clusters de forme arbitraire	Sensible à ϵ et minPts
Ne nécessite pas de spécifier K	Difficulté avec densités variables
Robuste aux outliers	Problèmes en haute dimension

12.5 Réduction de Dimensionnalité : PCA

12.5.1 Motivation

Données haute dimension → difficile à visualiser, computationnellement coûteux, curse of dimensionality.

Objectif : Projeter les données sur un sous-espace de dimension $k < p$ en préservant le maximum d'information (variance).

12.5.2 Analyse en Composantes Principales (PCA)

Définition 12.5 (PCA)**.** Trouver les directions de **variance maximale** dans les données.

Les **composantes principales** sont les vecteurs propres de la matrice de covariance, ordonnés par valeur propre décroissante.

Théorème 12.2 (PCA comme Maximisation de Variance)**.** *La première composante principale $\boldsymbol{u}_1$ maximise :*

$$\boldsymbol{u}_1 = \arg\max_{\|\boldsymbol{u}\|=1} Var(\mathbf{X}\boldsymbol{u}) = \arg\max_{\|\boldsymbol{u}\|=1} \boldsymbol{u}^\top \boldsymbol{\Sigma} \boldsymbol{u} \tag{12.3}$$

où $\boldsymbol{\Sigma}$ est la matrice de covariance.

Solution : $\boldsymbol{u}_1$ est le vecteur propre associé à la plus grande valeur propre de $\boldsymbol{\Sigma}$.

12.5.3 Algorithme PCA

Algorithm 11 PCA

Require: $\mathbf{X} \in \mathbb{R}^{n \times p}$, dimension cible k

1: Centrer les données : $\mathbf{X}_c = \mathbf{X} - \bar{\boldsymbol{x}}$
2: Calculer la matrice de covariance : $\boldsymbol{\Sigma} = \frac{1}{n-1} \mathbf{X}_c^\top \mathbf{X}_c$
3: Calculer les valeurs/vecteurs propres : $\boldsymbol{\Sigma} \boldsymbol{u}_j = \lambda_j \boldsymbol{u}_j$
4: Sélectionner les k premiers vecteurs propres : $\mathbf{U}_k = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k]$
5: Projeter : $\mathbf{Z} = \mathbf{X}_c \mathbf{U}_k$
6: **return** $\mathbf{Z} \in \mathbb{R}^{n \times k}$

Remarque 12.1. En pratique, on utilise la **SVD** : $\mathbf{X}_c = \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^\top$. Les colonnes de $\mathbf{V}$ sont les composantes principales.

12.5.4 Choix du Nombre de Composantes

Définition 12.6 (Variance Expliquée)**.** La proportion de variance expliquée par les k premières composantes :

$$\text{VE}_k = \frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{p} \lambda_j} \tag{12.4}$$

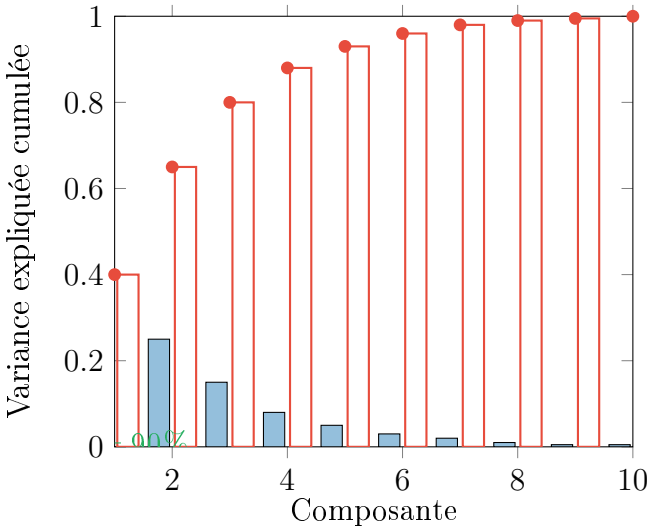


FIGURE 12.5 – Scree plot : choisir k pour atteindre 90% de variance expliquée

12.5.5 Interprétation Géométrique

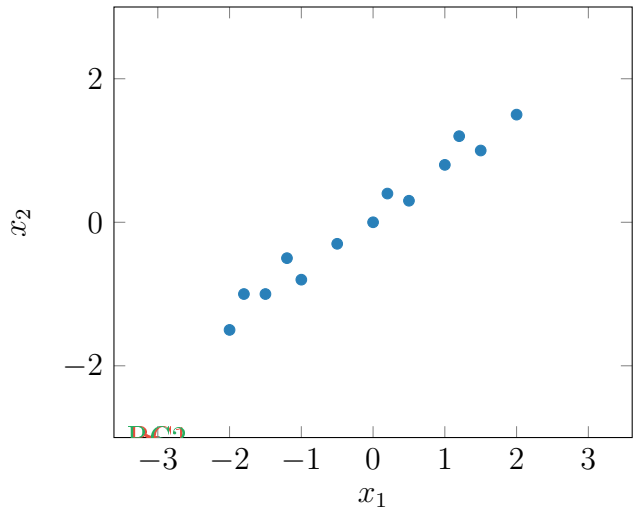


FIGURE 12.6 – PCA : les composantes principales sont les axes de l'ellipsoïde de données

12.6 t-SNE et UMAP

12.6.1 t-SNE (t-Distributed Stochastic Neighbor Embedding)

PCA préserve la structure **globale** (grandes distances). t-SNE préserve la structure **locale** (voisinages).

Définition 12.7 (t-SNE). 1. Calculer les similarités dans l'espace original (gaussienne)

2. Calculer les similarités dans l'espace réduit (Student-t)

3. Minimiser la divergence KL entre les deux distributions

$$p_{j|i} = \frac{\exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2 / 2\sigma_i^2)} \tag{12.5}$$

$$q_{ij} = \frac{(1 + \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|\boldsymbol{y}_k - \boldsymbol{y}_l\|^2)^{-1}} \tag{12.6}$$

$$KL(P\|Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{12.7}$$

12.6.2 UMAP (Uniform Manifold Approximation and Projection)

Plus rapide que t-SNE, préserve mieux la structure globale.

TABLE 12.3 – Comparaison PCA, t-SNE, UMAP

	PCA	**t-SNE**	**UMAP**
Linéaire	Oui	Non	Non
Structure préservée	Globale	Locale	Les deux
Scalabilité	Excellente	Limitée	Bonne
Reproductibilité	Oui	Dépend seed	Dépend seed
Nouveaux points	Facile	Difficile	Possible

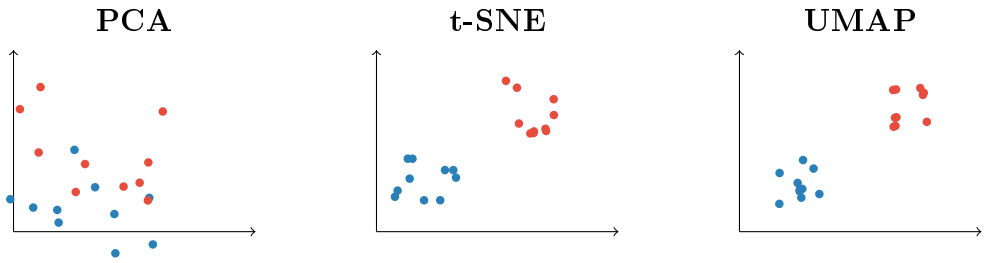


FIGURE 12.7 – Comparaison visuelle des méthodes de réduction de dimensionnalité

12.7 Exercices

Exercice 12.1 (K-Means)**.** Données : $(1,1), (1,2), (2,1), (8,8), (8,9), (9,8)$.

1. Appliquer K-Means avec $K = 2$, en initialisant $\mu_1 = (1,1)$ et $\mu_2 = (2,1)$
2. Combien d'itérations jusqu'à convergence ?
3. Calculer l'inertie finale

Exercice 12.2 (PCA).
1. Montrer que les composantes principales sont orthogonales
2. Pourquoi centre-t-on les données avant PCA ?
3. Si les deux premières composantes expliquent 95% de la variance, combien de dimensions garder ?

Exercice 12.3 (DBSCAN). Expliquer pourquoi DBSCAN peut trouver des clusters non-sphériques alors que K-Means ne le peut pas.

12.8 Implémentation Python

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.datasets import make_blobs, make_moons
from sklearn.metrics import silhouette_score
from scipy.cluster.hierarchy import dendrogram, linkage

# ===== K-Means =====
X, y_true = make_blobs(n_samples=300, centers=4, random_state=42)

# Methode du coude
inertias = []
silhouettes = []
K_range = range(2, 10)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X)
    inertias.append(kmeans.inertia_)
    silhouettes.append(silhouette_score(X, kmeans.labels_))

fig, axes = plt.subplots(1, 2, figsize=(12, 4))

axes[0].plot(K_range, inertias, 'bo-')
axes[0].set_xlabel('K')
axes[0].set_ylabel('Inertie')
axes[0].set_title('Methode du Coude')

axes[1].plot(K_range, silhouettes, 'go-')
axes[1].set_xlabel('K')
axes[1].set_ylabel('Score Silhouette')
axes[1].set_title('Score Silhouette')
```

```
35
36 plt.tight_layout()
37 plt.show()
38
39 # K-Means final
40 kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
41 labels = kmeans.fit_predict(X)
42
43 plt.figure(figsize=(8, 6))
44 plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
45 plt.scatter(kmeans.cluster_centers_[:, 0],
    kmeans.cluster_centers_[:, 1],
46              c='red', marker='X', s=200, label='Centroides')
47 plt.title('K-Means Clustering')
48 plt.legend()
49 plt.show()
50
51 # ===== DBSCAN pour donnees non-spheriques =====
52 X_moons, _ = make_moons(n_samples=200, noise=0.1, random_state=42)
53
54 fig, axes = plt.subplots(1, 3, figsize=(15, 4))
55
56 # K-Means (echec)
57 kmeans_moons = KMeans(n_clusters=2, random_state=42, n_init=10)
58 axes[0].scatter(X_moons[:, 0], X_moons[:, 1],
    c=kmeans_moons.fit_predict(X_moons))
59 axes[0].set_title('K-Means (echec)')
60
61 # DBSCAN (succes)
62 dbscan = DBSCAN(eps=0.2, min_samples=5)
63 axes[1].scatter(X_moons[:, 0], X_moons[:, 1],
    c=dbscan.fit_predict(X_moons))
64 axes[1].set_title('DBSCAN (succes)')
65
66 # Hierarchique
67 agg = AgglomerativeClustering(n_clusters=2, linkage='single')
68 axes[2].scatter(X_moons[:, 0], X_moons[:, 1],
    c=agg.fit_predict(X_moons))
69 axes[2].set_title('Hierarchique (single)')
70
71 plt.tight_layout()
72 plt.show()
73
74 # ===== PCA =====
75 from sklearn.datasets import load_digits
```

```
76
77 digits = load_digits()
78 X_digits = digits.data
79
80 pca = PCA()
81 pca.fit(X_digits)
82
83 # Variance expliquee
84 plt.figure(figsize=(10, 4))
85 plt.subplot(1, 2, 1)
86 plt.bar(range(1, 65), pca.explained_variance_ratio_)
87 plt.xlabel('Composante')
88 plt.ylabel('Variance expliquee')
89
90 plt.subplot(1, 2, 2)
91 plt.plot(range(1, 65), np.cumsum(pca.explained_variance_ratio_))
92 plt.axhline(y=0.9, color='r', linestyle='--', label='90%')
93 plt.xlabel('Nombre de composantes')
94 plt.ylabel('Variance cumulee')
95 plt.legend()
96 plt.tight_layout()
97 plt.show()
98
99 # Projection 2D
100 pca_2d = PCA(n_components=2)
101 X_pca = pca_2d.fit_transform(X_digits)
102
103 plt.figure(figsize=(10, 8))
104 scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=digits.target,
    cmap='tab10')
105 plt.colorbar(scatter)
106 plt.xlabel('PC1')
107 plt.ylabel('PC2')
108 plt.title('PCA sur MNIST Digits')
109 plt.show()
110
111 # ===== t-SNE =====
112 tsne = TSNE(n_components=2, perplexity=30, random_state=42)
113 X_tsne = tsne.fit_transform(X_digits)
114
115 plt.figure(figsize=(10, 8))
116 scatter = plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=digits.target,
    cmap='tab10')
117 plt.colorbar(scatter)
118 plt.xlabel('t-SNE 1')
```

```
plt.ylabel('t-SNE 2')
plt.title('t-SNE sur MNIST Digits')
plt.show()

# ===== Dendrogramme =====
X_small, _ = make_blobs(n_samples=30, centers=3, random_state=42)

Z = linkage(X_small, method='ward')

plt.figure(figsize=(12, 5))
dendrogram(Z, leaf_rotation=90, leaf_font_size=10)
plt.title('Dendrogramme - Methode Ward')
plt.xlabel('Echantillon')
plt.ylabel('Distance')
plt.show()
```

Listing 12.1 – Clustering et réduction de dimensionnalité

Synthèse

Points clés du chapitre :

- **K-Means** : Simple mais suppose des clusters sphériques
- **DBSCAN** : Trouve des formes arbitraires, détecte les outliers
- **Hiérarchique** : Produit un dendrogramme, pas besoin de K a priori
- **PCA** : Réduction linéaire, préserve la variance globale
- **t-SNE/UMAP** : Réduction non-linéaire, préserve les voisinages
- Évaluation difficile → utiliser silhouette, inertie, visualisation

Chapitre 13

Sélection de Modèles et Validation

Objectifs du chapitre

Ce chapitre final couvre les techniques essentielles pour évaluer, sélectionner et optimiser les modèles de Machine Learning : validation croisée, recherche d'hyperparamètres, et bonnes pratiques.

13.1 Le Problème du Surapprentissage

13.1.1 Biais vs Variance : Rappel

$$\mathbb{E}[(y - \hat{f}(\boldsymbol{x}))^2] = \underbrace{\text{Biais}^2[\hat{f}(\boldsymbol{x})]}_{\text{Erreur d'approximation}} + \underbrace{\text{Var}[\hat{f}(\boldsymbol{x})]}_{\text{Instabilité}} + \underbrace{\sigma^2}_{\text{Bruit irréductible}} \tag{13.1}$$

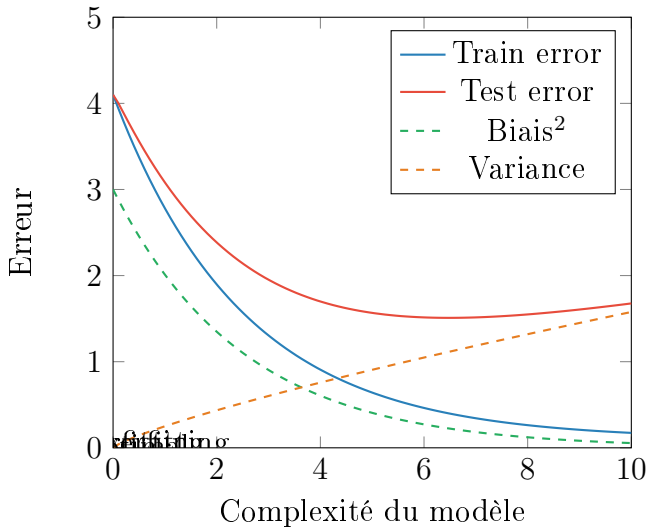


FIGURE 13.1 – Compromis biais-variance et erreurs train/test

13.1.2 Détecter le Surapprentissage

TABLE 13.1 – Diagnostic du modèle

Situation	Train Error	Test Error	Problème
Underfitting	Élevée	Élevée	Biais élevé
Overfitting	Basse	Élevée	Variance élevée
Bon modèle	Basse	Basse (proche train)	Équilibré

13.2 Techniques de Validation

13.2.1 Hold-Out (Train/Test Split)

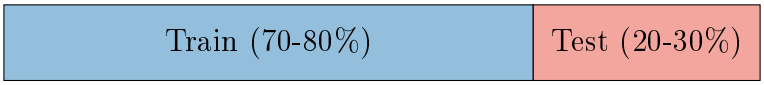


FIGURE 13.2 – Simple train/test split

Attention

Problèmes du hold-out :

— Dépend de la split aléatoire

— Perte de données d'entraînement

— Variance élevée de l'estimation

13.2.2 K-Fold Cross-Validation

Définition 13.1 (K-Fold CV).

1. Diviser les données en K folds égaux
2. Pour chaque fold k :
   — Entraîner sur les $K-1$ autres folds
   — Évaluer sur le fold k
3. Moyenner les K scores

$$\text{CV Score} = \frac{1}{K}\sum_{k=1}^{K}\text{Score}_k \pm \frac{\sigma}{\sqrt{K}} \tag{13.2}$$

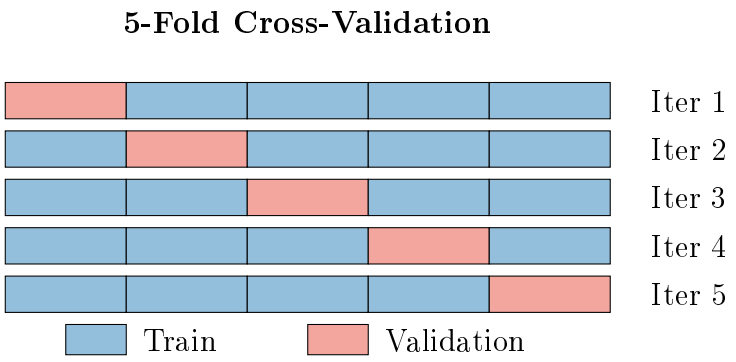


FIGURE 13.3 – 5-Fold Cross-Validation

TABLE 13.2 – Types de cross-validation

Méthode	Usage
K-Fold	Standard, $K = 5$ ou $K = 10$
Stratified K-Fold	Classification déséquilibrée (préserve proportions)
Leave-One-Out (LOO)	Petits datasets ($K = n$)
Leave-P-Out	Variante de LOO
Time Series Split	Données temporelles (pas de leak futur)
Group K-Fold	Données groupées (même groupe jamais dans train et test)

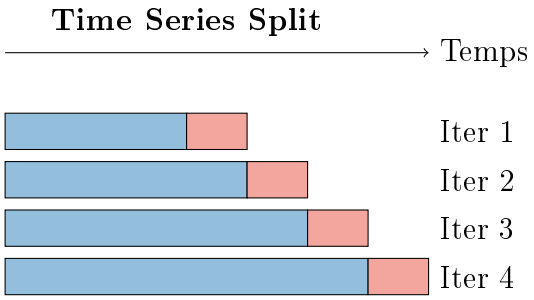


FIGURE 13.4 – Cross-validation pour séries temporelles

13.2.3 Variantes de Cross-Validation

13.2.4 Train / Validation / Test

Important

Pour l'optimisation d'hyperparamètres, il faut **trois** ensembles :

— **Train** : Entraîner le modèle

— **Validation** : Choisir les hyperparamètres

— **Test** : Évaluation finale (jamais vu pendant le développement)

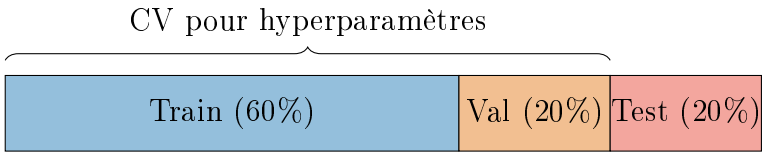


FIGURE 13.5 – Découpage train/validation/test

13.3 Recherche d'Hyperparamètres

13.3.1 Grid Search

Définition 13.2 (Grid Search)**.** Tester **toutes les combinaisons** d'une grille d'hyperparamètres.

```python
from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [0.001, 0.01, 0.1, 1],
    'kernel': ['rbf', 'poly']
}

grid = GridSearchCV(SVC(), param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)

print(f"Best params: {grid.best_params_}")
print(f"Best score: {grid.best_score_:.4f}")
```

Listing 13.1 – Grid Search avec scikit-learn

Inconvénient : Complexité $O(n^k)$ pour k hyperparamètres avec n valeurs chacun.

13.3.2 Random Search

Définition 13.3 (Random Search)**.** Échantillonner aléatoirement les hyperparamètres depuis des distributions.

Théorème 13.1 (Bergstra & Bengio, 2012)**.** *Random Search est plus efficace que Grid Search car :*

— *Les hyperparamètres n'ont pas tous la même importance*

— *Random Search explore plus de valeurs pour les hyperparamètres importants*

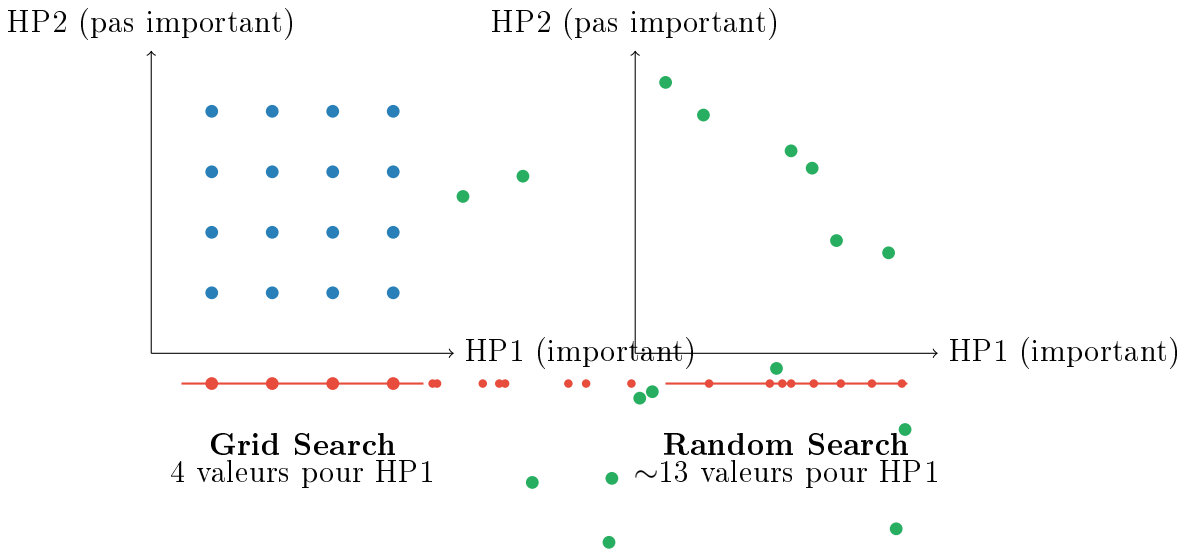


FIGURE 13.6 – Grid Search vs Random Search : exploration de l'espace

13.3.3 Recherche Bayésienne

Définition 13.4 (Bayesian Optimization)**.** Construire un modèle probabiliste de la fonction objectif et utiliser ce modèle pour choisir intelligemment les prochains points à évaluer.

Algorithm 12 Bayesian Optimization

1: **for** $t = 1$ to T **do**
2: Fit un modèle surrogate (GP) sur les observations
3: Trouver $\boldsymbol{\theta}_{next}$ qui maximise l'acquisition function
4: Évaluer $f(\boldsymbol{\theta}_{next})$
5: Ajouter $(\boldsymbol{\theta}_{next}, f(\boldsymbol{\theta}_{next}))$ aux observations
6: **end for**

Librairies : `optuna`, `hyperopt`, `scikit-optimize`.

13.3.4 Comparaison des Méthodes

TABLE 13.3 – Comparaison des méthodes de recherche d'hyperparamètres

Méthode	Efficacité	Implémentation	Parallélisation
Grid Search	Faible	Simple	Facile
Random Search	Moyenne	Simple	Facile
Bayesian	Élevée	Complexe	Difficile

13.4 Nested Cross-Validation

Attention

Utiliser la même CV pour choisir les hyperparamètres ET évaluer le modèle final → **biais optimiste** !

Définition 13.5 (Nested CV). — **Boucle externe** : Évalue la performance de généralisation

— **Boucle interne** : Sélectionne les meilleurs hyperparamètres

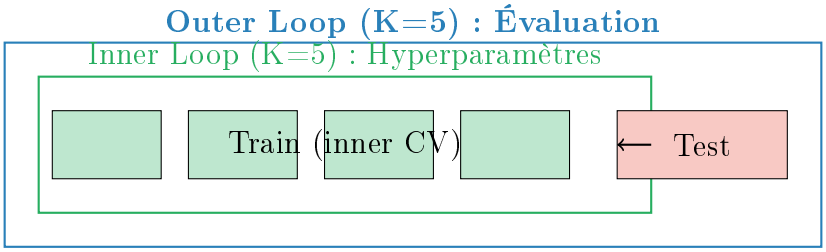


FIGURE 13.7 – Nested Cross-Validation

```python
from sklearn.model_selection import cross_val_score, GridSearchCV

# Inner CV pour hyperparametres
inner_cv = KFold(n_splits=5, shuffle=True, random_state=42)
grid = GridSearchCV(SVC(), param_grid, cv=inner_cv,
    scoring='accuracy')

# Outer CV pour evaluation
outer_cv = KFold(n_splits=5, shuffle=True, random_state=42)
nested_scores = cross_val_score(grid, X, y, cv=outer_cv)

print(f"Nested CV score: {nested_scores.mean():.4f} +/-
    {nested_scores.std():.4f}")
```

Listing 13.2 – Nested CV avec scikit-learn

13.5 Courbes d'Apprentissage et de Validation

13.5.1 Courbe d'Apprentissage

Montre comment la performance évolue avec la **taille des données d'entraînement**.

Interprétation :

— Si les courbes convergent avec un gap → plus de données aidera

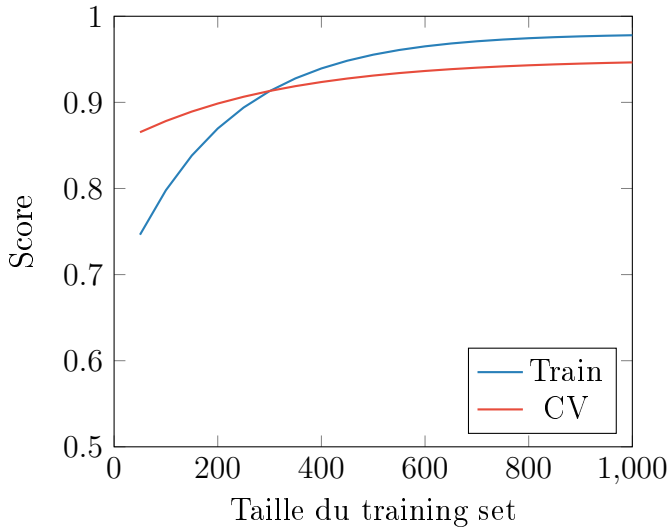


FIGURE 13.8 – Courbe d'apprentissage : plus de données améliore la généralisation

— Si les courbes sont proches mais performance faible → modèle trop simple

— Si grand gap persistant → overfitting, régulariser

13.5.2 Courbe de Validation

Montre comment la performance évolue avec un **hyperparamètre**.

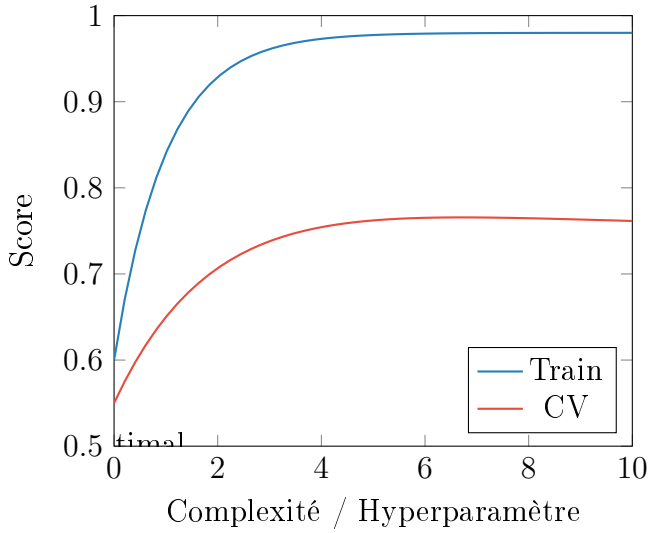


FIGURE 13.9 – Courbe de validation : trouver l'hyperparamètre optimal

13.6 Bonnes Pratiques

13.6.1 Pipeline ML Robuste

1. **Séparer le test set** dès le début (ne jamais y toucher)
2. **EDA** sur le train set uniquement
3. **Preprocessing** : fit sur train, transform sur train et test
4. **CV** pour choisir modèle et hyperparamètres
5. **Évaluation finale** sur le test set
6. **Réentraîner** sur toutes les données (train + val) avec les meilleurs hyperparamètres

```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV

# 1. Separation initiale
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)

# 2. Pipeline avec preprocessing
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('model', RandomForestClassifier())
])

# 3. Grid search avec CV
param_grid = {
    'model__n_estimators': [50, 100, 200],
    'model__max_depth': [None, 10, 20]
}

grid = GridSearchCV(pipe, param_grid, cv=5, scoring='f1')
grid.fit(X_train, y_train)

# 4. Evaluation finale
print(f"Test score: {grid.score(X_test, y_test):.4f}")

# 5. Reentrainement final (optionnel)
best_pipe = grid.best_estimator_
best_pipe.fit(np.vstack([X_train, X_test]), np.hstack([y_train,
    y_test]))
```

Listing 13.3 – Pipeline ML complet

13.6.2 Data Leakage

Définition 13.6 (Data Leakage)**.** Information du test set qui "fuite" vers le training, donnant une estimation optimiste.

Sources courantes :

— Normaliser sur tout le dataset avant de split

— Feature engineering utilisant des infos futures

— Duplication d'observations entre train et test

— CV mal implémentée pour séries temporelles

13.6.3 Checklist avant Déploiement

☐ Nested CV utilisée ou test set vraiment isolé

☐ Pas de data leakage

☐ Performance stable sur différents splits

☐ Métriques adaptées au problème métier

☐ Baseline simple comparée

☐ Modèle interprétable si nécessaire

☐ Test sur données récentes / hors distribution

13.7 Exercices

Exercice 13.1 (Cross-Validation)**.**

1. Pourquoi K-Fold CV est-elle préférable au simple hold-out ?
2. Quand utiliser Stratified K-Fold ?
3. Quand utiliser LOO-CV ?

Exercice 13.2 (Grid vs Random)**.** Vous avez 10 hyperparamètres à optimiser avec un budget de 100 évaluations.

1. Combien de valeurs par hyperparamètre avec Grid Search ?
2. Pourquoi Random Search est-il préférable ici ?

Exercice 13.3 (Data Leakage)**.** Identifiez le data leakage dans ce code :

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)  # BUG: fit sur tout X

X_train, X_test = train_test_split(X_scaled, ...)
model.fit(X_train, y_train)
score = model.score(X_test, y_test)
```

Comment corriger ?

13.8 Implémentation Complète

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import (train_test_split,
    GridSearchCV,
                                     RandomizedSearchCV,
                                        cross_val_score,
                                     learning_curve,
                                        validation_curve)
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from scipy.stats import randint, uniform

# ===== Chargement des donnees =====
data = load_breast_cancer()
X, y = data.data, data.target

# Separation initiale
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print(f"Train: {X_train.shape}, Test: {X_test.shape}")

# ===== Pipeline =====
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', SVC())
])

# ===== Grid Search =====
param_grid = {
    'clf__C': [0.1, 1, 10],
    'clf__gamma': ['scale', 0.01, 0.1],
    'clf__kernel': ['rbf', 'linear']
}

```

```
grid = GridSearchCV(pipe, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid.fit(X_train, y_train)

print(f"\n=== Grid Search ===")
print(f"Best params: {grid.best_params_}")
print(f"Best CV score: {grid.best_score_:.4f}")
print(f"Test score: {grid.score(X_test, y_test):.4f}")

# ===== Random Search =====
param_distributions = {
    'clf__C': uniform(0.1, 100),
    'clf__gamma': uniform(0.001, 1),
    'clf__kernel': ['rbf', 'linear', 'poly']
}

random_search = RandomizedSearchCV(
    pipe, param_distributions, n_iter=50, cv=5,
    scoring='accuracy', random_state=42, n_jobs=-1
)
random_search.fit(X_train, y_train)

print(f"\n=== Random Search ===")
print(f"Best params: {random_search.best_params_}")
print(f"Best CV score: {random_search.best_score_:.4f}")
print(f"Test score: {random_search.score(X_test, y_test):.4f}")

# ===== Nested CV =====
from sklearn.model_selection import cross_val_score, KFold

outer_cv = KFold(n_splits=5, shuffle=True, random_state=42)
inner_cv = KFold(n_splits=3, shuffle=True, random_state=42)

# GridSearch comme "modele" dans la CV externe
grid_nested = GridSearchCV(pipe, param_grid, cv=inner_cv, scoring='accuracy')
nested_scores = cross_val_score(grid_nested, X, y, cv=outer_cv, scoring='accuracy')

print(f"\n=== Nested CV ===")
print(f"Nested CV score: {nested_scores.mean():.4f} +/- {nested_scores.std():.4f}")

# ===== Courbe d'apprentissage =====
train_sizes, train_scores, test_scores = learning_curve(
```

```
    grid.best_estimator_, X_train, y_train,
    cv=5, train_sizes=np.linspace(0.1, 1.0, 10),
    scoring='accuracy', n_jobs=-1
)

plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.plot(train_sizes, train_scores.mean(axis=1), 'o-',
   label='Train')
plt.fill_between(train_sizes,
                 train_scores.mean(axis=1) -
                    train_scores.std(axis=1),
                 train_scores.mean(axis=1) +
                    train_scores.std(axis=1), alpha=0.1)
plt.plot(train_sizes, test_scores.mean(axis=1), 'o-', label='CV')
plt.fill_between(train_sizes,
                 test_scores.mean(axis=1) -
                    test_scores.std(axis=1),
                 test_scores.mean(axis=1) +
                    test_scores.std(axis=1), alpha=0.1)
plt.xlabel('Training set size')
plt.ylabel('Accuracy')
plt.title('Learning Curve')
plt.legend()
plt.grid(True)

# ===== Courbe de validation =====
param_range = np.logspace(-3, 2, 10)
train_scores, test_scores = validation_curve(
    pipe, X_train, y_train, param_name='clf__C',
    param_range=param_range, cv=5, scoring='accuracy', n_jobs=-1
)

plt.subplot(1, 2, 2)
plt.semilogx(param_range, train_scores.mean(axis=1), 'o-',
   label='Train')
plt.semilogx(param_range, test_scores.mean(axis=1), 'o-',
   label='CV')
plt.fill_between(param_range,
                 train_scores.mean(axis=1) -
                    train_scores.std(axis=1),
                 train_scores.mean(axis=1) +
                    train_scores.std(axis=1), alpha=0.1)
plt.fill_between(param_range,
```

```
                    test_scores.mean(axis=1) -
                        test_scores.std(axis=1),
                    test_scores.mean(axis=1) +
                        test_scores.std(axis=1), alpha=0.1)
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.title('Validation Curve')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

# ===== Comparaison de modeles =====
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier

models = {
    'Logistic': LogisticRegression(max_iter=1000),
    'SVM': SVC(),
    'RandomForest': RandomForestClassifier(n_estimators=100),
    'GradientBoosting': GradientBoostingClassifier()
}

print("\n=== Comparaison des modeles (5-Fold CV) ===")
for name, model in models.items():
    pipe = Pipeline([('scaler', StandardScaler()), ('clf', model)])
    scores = cross_val_score(pipe, X_train, y_train, cv=5,
        scoring='accuracy')
    print(f"{name:20s}: {scores.mean():.4f} +/-
        {scores.std():.4f}")
```

Listing 13.4 – Pipeline complet de validation et optimisation

Synthèse

Points clés du chapitre :

- **Cross-validation** : Estimation robuste de la performance
- **Nested CV** : Évite le biais optimiste lors de l'optimisation des hyperparamètres
- **Grid Search** : Exhaustif mais coûteux
- **Random Search** : Plus efficace pour beaucoup d'hyperparamètres
- **Bayesian Optimization** : Le plus efficace mais complexe
- Toujours garder un **test set** intouché pour l'évaluation finale

— Attention au **data leakage** !

Quatrième partie

Annexes

Annexe A

Notations Mathématiques

Cette annexe récapitule toutes les notations mathématiques utilisées dans ce guide.

A.1 Ensembles et Nombres

Notation	**Description**	**Exemple**
$\mathbb{N}$	Entiers naturels	$\{0, 1, 2, 3, \ldots\}$
$\mathbb{Z}$	Entiers relatifs	$\{\ldots, -2, -1, 0, 1, 2, \ldots\}$
$\mathbb{R}$	Nombres réels	$\pi, \sqrt{2}, -3.14$
$\mathbb{R}^n$	Espace réel de dimension n	$\mathbb{R}^3$ (espace 3D)
$\mathbb{R}^{m \times n}$	Matrices réelles $m \times n$	$\mathbb{R}^{3 \times 4}$
$[a, b]$	Intervalle fermé	$[0, 1]$
(a, b)	Intervalle ouvert	$(0, 1)$
$\{x : P(x)\}$	Ensemble par compréhension	$\{x : x > 0\}$
$\in$	Appartenance	$x \in \mathbb{R}$
$\subset$	Inclusion stricte	$\mathbb{N} \subset \mathbb{Z}$
$\subseteq$	Inclusion large	$A \subseteq B$
$\cup$	Union	$A \cup B$
$\cap$	Intersection	$A \cap B$
$\emptyset$	Ensemble vide	
$\lvert A \rvert$	Cardinal de A	$\lvert\{1, 2, 3\}\rvert = 3$

TABLE A.1 – Notations pour les ensembles

Notation	Description	Exemple
$\boldsymbol{x}, \boldsymbol{a}$	Vecteur (gras minuscule)	$\boldsymbol{x} = (x_1, x_2, x_3)^\top$
$\mathbf{A}, \mathbf{X}$	Matrice (gras majuscule)	$\mathbf{A} \in \mathbb{R}^{m \times n}$
x_i	i-ème composante de $\boldsymbol{x}$	x_3
A_{ij}	Élément ligne i, colonne j	A_{23}
$\mathbf{A}^\top$	Transposée de $\mathbf{A}$	$(A^\top)_{ij} = A_{ji}$
$\mathbf{A}^{-1}$	Inverse de $\mathbf{A}$	$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$
$\mathbf{I}_n$	Matrice identité $n \times n$	$\mathbf{I}_3$
$\mathbf{0}$	Matrice nulle	
$\text{diag}(\boldsymbol{d})$	Matrice diagonale	$\text{diag}(1, 2, 3)$
$\text{tr}(\mathbf{A})$	Trace de $\mathbf{A}$	$\sum_i A_{ii}$
$\det(\mathbf{A})$	Déterminant de $\mathbf{A}$	$\lvert\mathbf{A}\rvert$
$\text{rang}(\mathbf{A})$	Rang de $\mathbf{A}$	

TABLE A.2 – Notations pour les vecteurs et matrices

Notation	Description	Formule
$\boldsymbol{x} \cdot \boldsymbol{y}$	Produit scalaire	$\sum_i x_i y_i$
$\boldsymbol{x}^\top \boldsymbol{y}$	Produit scalaire (notation matricielle)	$\sum_i x_i y_i$
$\mathbf{AB}$	Produit matriciel	$(\mathbf{AB})_{ij} = \sum_k A_{ik} B_{kj}$
$\boldsymbol{x} \otimes \boldsymbol{y}$	Produit tensoriel (outer product)	$\boldsymbol{x}\boldsymbol{y}^\top$
$\odot$	Produit de Hadamard (élément par élément)	$(A \odot B)_{ij} = A_{ij} B_{ij}$
$\lVert\boldsymbol{x}\rVert_1$	Norme L^1 (Manhattan)	$\sum_i \lvert x_i\rvert$
$\lVert\boldsymbol{x}\rVert_2$	Norme L^2 (Euclidienne)	$\sqrt{\sum_i x_i^2}$
$\lVert\boldsymbol{x}\rVert_p$	Norme L^p	$(\sum_i \lvert x_i\rvert^p)^{1/p}$
$\lVert\boldsymbol{x}\rVert_\infty$	Norme infinie	$\max_i \lvert x_i\rvert$
$\lVert\mathbf{A}\rVert_F$	Norme de Frobenius	$\sqrt{\sum_{i,j} A_{ij}^2}$

TABLE A.3 – Produits et normes

A.2 Vecteurs et Matrices

A.3 Produits et Normes

A.4 Calcul Différentiel

A.5 Probabilités et Statistiques

A.6 Machine Learning

A.7 Opérateurs et Fonctions Usuelles

A.8 Conventions Typographiques

— **Scalaires** : lettres minuscules italiques (x, y, α)

Notation	Description	Exemple
$\frac{df}{dx}$	Dérivée de f par rapport à x	$\frac{d(x^2)}{dx} = 2x$
$f'(x)$	Dérivée de f	$f'(x) = 2x$
$\frac{\partial f}{\partial x}$	Dérivée partielle	$\frac{\partial (xy)}{\partial x} = y$
∇f	Gradient de f	$\left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)^\top$
$\nabla^2 f$	Hessienne de f	$\left(\frac{\partial^2 f}{\partial x_i \partial x_j}\right)$
$\mathbf{J}_f$	Jacobienne de f	$\left(\frac{\partial f_i}{\partial x_j}\right)$
$\frac{\partial \mathbf{L}}{\partial \mathbf{W}}$	Gradient matriciel	

TABLE A.4 – Notations du calcul différentiel

Notation	Description	Exemple
$P(A)$	Probabilité de l'événement A	$P(\text{pile}) = 0.5$
$P(A\|B)$	Probabilité conditionnelle	$P(A \text{ sachant } B)$
$p(x)$	Densité de probabilité	$p(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$
$X \sim \mathcal{D}$	X suit la distribution $\mathcal{D}$	$X \sim \mathcal{N}(0, 1)$
$\mathcal{N}(\mu, \sigma^2)$	Loi normale	
$\mathcal{U}(a, b)$	Loi uniforme	
$\text{Ber}(p)$	Loi de Bernoulli	
$\mathbb{E}[X]$	Espérance de X	$\mathbb{E}[X] = \int x\, p(x) dx$
$\text{Var}(X)$	Variance de X	$\mathbb{E}[(X - \mu)^2]$
$\text{Cov}(X, Y)$	Covariance	$\mathbb{E}[(X - \mu_X)(Y - \mu_Y)]$
σ	Écart-type	$\sqrt{\text{Var}(X)}$
ρ_{XY}	Coefficient de corrélation	$\frac{\text{Cov}(X,Y)}{\sigma_X \sigma_Y}$

TABLE A.5 – Notations probabilistes

— **Vecteurs** : lettres minuscules grasses ($\boldsymbol{x}$, $\boldsymbol{w}$, $\boldsymbol{\theta}$)

— **Matrices** : lettres majuscules grasses ($\mathbf{A}$, $\mathbf{X}$, $\mathbf{W}$)

— **Ensembles** : lettres majuscules calligraphiques ($\mathcal{D}$, $\mathcal{L}$)

— **Variables aléatoires** : lettres majuscules (X, Y)

— **Réalisations** : lettres minuscules (x, y)

Notation	Description	Contexte
$\mathcal{D}$	Dataset	$\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$
n	Nombre d'échantillons	Taille du dataset
d ou p	Dimension des features	Nombre de variables
$\mathbf{X}$	Matrice des features	$\mathbf{X} \in \mathbb{R}^{n \times d}$
$\boldsymbol{y}$	Vecteur des cibles	$\boldsymbol{y} \in \mathbb{R}^n$ (régression)
$\hat{y}$	Prédiction	$\hat{y} = f(\boldsymbol{x})$
$\boldsymbol{\theta}$, $\boldsymbol{w}$	Paramètres du modèle	Poids à apprendre
b	Biais (intercept)	
$\mathcal{L}$	Fonction de perte (loss)	MSE, Cross-entropy
$J(\boldsymbol{\theta})$	Fonction de coût	$J = \frac{1}{n}\sum_i \mathcal{L}$
λ	Paramètre de régularisation	Ridge, Lasso
η, α	Taux d'apprentissage	Learning rate
K	Nombre de classes	Classification multiclasse
$h_\theta(\boldsymbol{x})$	Hypothèse du modèle	Fonction apprise

TABLE A.6 – Notations Machine Learning

Notation	Description	Formule
$\arg\min_x f(x)$	Argument du minimum	$x^* : f(x^*) \leq f(x)\, \forall x$
$\arg\max_x f(x)$	Argument du maximum	$x^* : f(x^*) \geq f(x)\, \forall x$
$\sum_{i=1}^n$	Somme	$\sum_{i=1}^3 i = 1+2+3 = 6$
$\prod_{i=1}^n$	Produit	$\prod_{i=1}^3 i = 1 \times 2 \times 3 = 6$
$\log(x)$	Logarithme naturel	$\ln(x)$
$\exp(x)$	Exponentielle	e^x
$\sigma(x)$	Fonction sigmoïde	$\frac{1}{1+e^{-x}}$
$\text{softmax}(\boldsymbol{z})_i$	Fonction softmax	$\frac{e^{z_i}}{\sum_j e^{z_j}}$
$\text{ReLU}(x)$	Rectified Linear Unit	$\max(0, x)$
$\tanh(x)$	Tangente hyperbolique	$\frac{e^x - e^{-x}}{e^x + e^{-x}}$
$\mathbb{1}_A$	Fonction indicatrice	1 si A vrai, 0 sinon
$\text{sign}(x)$	Fonction signe	± 1 selon signe de x

TABLE A.7 – Opérateurs et fonctions usuelles

Annexe B

Aide-Mémoire Python pour le ML

Cette annexe fournit un récapitulatif des commandes Python les plus utilisées en Data Science et Machine Learning.

B.1 NumPy - Cheat Sheet

B.1.1 Création d'Arrays

```python
import numpy as np

# Depuis une liste
a = np.array([1, 2, 3])
A = np.array([[1, 2], [3, 4]])

# Arrays speciaux
np.zeros((3, 4))          # Matrice de zeros 3x4
np.ones((2, 3))           # Matrice de uns 2x3
np.eye(4)                 # Matrice identite 4x4
np.diag([1, 2, 3])        # Matrice diagonale

# Sequences
np.arange(0, 10, 2)       # [0, 2, 4, 6, 8]
np.linspace(0, 1, 5)      # 5 points entre 0 et 1

# Aleatoire
np.random.rand(3, 3)      # Uniforme [0, 1]
np.random.randn(3, 3)     # Normale N(0, 1)
np.random.randint(0, 10, (3, 3))  # Entiers
np.random.seed(42)        # Reproductibilite
```

Listing B.1 – Création d'arrays NumPy

B.1.2 Opérations sur les Arrays

```
# Proprietes
a.shape                    # Dimensions
a.dtype                    # Type de donnees
a.ndim                     # Nombre de dimensions
a.size                     # Nombre total d'elements

# Reshape
a.reshape(3, 4)            # Redimensionner
a.flatten()                # Aplatir en 1D
a.T                        # Transposee

# Operations element par element
a + b                      # Addition
a * b                      # Multiplication (Hadamard)
a ** 2                     # Puissance
np.sqrt(a)                 # Racine carree
np.exp(a)                  # Exponentielle
np.log(a)                  # Logarithme

# Agregations
np.sum(a)                  # Somme totale
np.sum(a, axis=0)          # Somme par colonne
np.mean(a)                 # Moyenne
np.std(a)                  # Ecart-type
np.max(a), np.min(a)       # Max, Min
np.argmax(a)               # Indice du max
```

Listing B.2 – Opérations NumPy

B.1.3 Algèbre Linéaire

```
# Produit matriciel
A @ B                      # Recommande (Python 3.5+)
np.dot(A, B)               # Equivalent
np.matmul(A, B)            # Equivalent

# Decompositions
np.linalg.inv(A)           # Inverse
np.linalg.det(A)           # Determinant
np.linalg.eig(A)           # Valeurs/vecteurs propres
np.linalg.svd(A)           # SVD

# Systemes lineaires
```

```
np.linalg.solve(A, b)    # Resout Ax = b
np.linalg.lstsq(A, b)    # Moindres carres

# Normes
np.linalg.norm(x)        # Norme L2
np.linalg.norm(x, ord=1)  # Norme L1
np.linalg.norm(A, 'fro')  # Norme Frobenius
```

Listing B.3 – Algèbre linéaire NumPy

B.2 Pandas - Cheat Sheet

B.2.1 Création et Chargement

```
import pandas as pd

# Creer un DataFrame
df = pd.DataFrame({
    'col1': [1, 2, 3],
    'col2': ['a', 'b', 'c']
})

# Charger des fichiers
df = pd.read_csv('fichier.csv')
df = pd.read_excel('fichier.xlsx')
df = pd.read_json('fichier.json')

# Sauvegarder
df.to_csv('output.csv', index=False)
df.to_excel('output.xlsx', index=False)
```

Listing B.4 – Pandas - Création et chargement

B.2.2 Exploration

```
# Apercu
df.head(10)              # 10 premieres lignes
df.tail(5)               # 5 dernieres lignes
df.sample(5)             # 5 lignes aleatoires

# Informations
df.shape                 # (n_lignes, n_colonnes)
df.columns               # Noms des colonnes
df.dtypes                # Types de donnees
df.info()                # Resume complet
```

```
df.describe()            # Stats descriptives

# Valeurs manquantes
df.isnull().sum()        # Compte par colonne
df.dropna()              # Supprimer lignes avec NaN
df.fillna(0)             # Remplacer NaN par 0
df.fillna(df.mean())     # Remplacer par moyenne
```

Listing B.5 – Pandas - Exploration

B.2.3 Sélection et Filtrage

```
# Selection de colonnes
df['col']                # Une colonne (Series)
df[['col1', 'col2']]     # Plusieurs colonnes

# Selection par position
df.iloc[0]               # Premiere ligne
df.iloc[0:5]             # Lignes 0 a 4
df.iloc[0, 2]            # Element (0, 2)
df.iloc[:, 0:3]          # Toutes lignes, col 0-2

# Selection par label
df.loc[0, 'col']         # Element avec index 0
df.loc[:, ['a', 'b']]    # Colonnes a et b

# Filtrage conditionnel
df[df['age'] > 30]       # Age > 30
df[df['ville'] == 'Paris']
df[(df['a'] > 0) & (df['b'] < 10)]   # Conditions multiples
df[df['col'].isin(['A', 'B'])]       # Dans une liste
```

Listing B.6 – Pandas - Sélection

B.2.4 Transformation et Agrégation

```
# Nouvelles colonnes
df['new'] = df['a'] + df['b']
df['category'] = df['value'].apply(lambda x: 'high' if x > 50 else
    'low')

# Renommer
df.rename(columns={'old': 'new'})

# Trier
```

```
df.sort_values('col', ascending=False)
df.sort_index()

# GroupBy
df.groupby('category')['value'].mean()
df.groupby('cat').agg({'a': 'sum', 'b': 'mean'})

# Pivot
pd.pivot_table(df, values='val', index='row', columns='col')

# Merge / Join
pd.merge(df1, df2, on='key', how='inner')
df1.join(df2, on='key')
pd.concat([df1, df2], axis=0)  # Empiler
```

Listing B.7 – Pandas - Transformation

B.3 Matplotlib - Cheat Sheet

```
import matplotlib.pyplot as plt

# Configuration
plt.figure(figsize=(10, 6))
plt.style.use('seaborn')

# Graphique lineaire
plt.plot(x, y, label='courbe', color='blue', linestyle='--',
    linewidth=2)

# Nuage de points
plt.scatter(x, y, c=colors, s=sizes, alpha=0.7)

# Histogramme
plt.hist(data, bins=30, edgecolor='black', alpha=0.7)

# Barres
plt.bar(categories, values)
plt.barh(categories, values)  # Horizontal

# Boxplot
plt.boxplot([data1, data2, data3])

# Heatmap
plt.imshow(matrix, cmap='viridis')
```

```
plt.colorbar()

# Annotations
plt.xlabel('X label')
plt.ylabel('Y label')
plt.title('Titre')
plt.legend()
plt.grid(True, alpha=0.3)

# Subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
axes[0, 0].plot(x, y)
axes[0, 1].scatter(x, y)

# Sauvegarder
plt.savefig('figure.png', dpi=300, bbox_inches='tight')
plt.show()
```

Listing B.8 – Matplotlib - Graphiques de base

B.4 Seaborn - Cheat Sheet

```
import seaborn as sns

# Configuration
sns.set_theme(style='whitegrid')

# Distribution
sns.histplot(data=df, x='col', hue='category', kde=True)
sns.kdeplot(data=df, x='col', hue='category')
sns.boxplot(data=df, x='cat', y='val')
sns.violinplot(data=df, x='cat', y='val')

# Relations
sns.scatterplot(data=df, x='x', y='y', hue='cat', size='val')
sns.lineplot(data=df, x='x', y='y', hue='cat')
sns.regplot(data=df, x='x', y='y')  # Avec regression

# Matrices
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0)
sns.pairplot(df, hue='category')
sns.clustermap(data)

# Categoriques
```

```
sns.countplot(data=df, x='category')
sns.barplot(data=df, x='cat', y='val', hue='group')
```

Listing B.9 – Seaborn - Visualisations statistiques

B.5 Scikit-learn - Cheat Sheet

B.5.1 Pipeline de Base

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, mean_squared_error

# 1. Separer train/test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y  # stratify pour classification
)

# 2. Normaliser
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)  # Pas fit !

# 3. Entrainer
model.fit(X_train_scaled, y_train)

# 4. Predire
y_pred = model.predict(X_test_scaled)
y_proba = model.predict_proba(X_test_scaled)  # Classification

# 5. Evaluer
score = model.score(X_test_scaled, y_test)
```

Listing B.10 – Scikit-learn - Pipeline standard

B.5.2 Modèles de Régression

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
```

```
6
7  # Regression lineaire
8  model = LinearRegression()
9
10 # Avec regularisation
11 model = Ridge(alpha=1.0)
12 model = Lasso(alpha=1.0)
13 model = ElasticNet(alpha=1.0, l1_ratio=0.5)
14
15 # Arbres
16 model = DecisionTreeRegressor(max_depth=5)
17 model = RandomForestRegressor(n_estimators=100, max_depth=10)
18 model = GradientBoostingRegressor(n_estimators=100,
       learning_rate=0.1)
19
20 # Autres
21 model = SVR(kernel='rbf', C=1.0)
22 model = KNeighborsRegressor(n_neighbors=5)
```

Listing B.11 – Modèles de régression

B.5.3 Modèles de Classification

```
1  from sklearn.linear_model import LogisticRegression
2  from sklearn.tree import DecisionTreeClassifier
3  from sklearn.ensemble import RandomForestClassifier,
       GradientBoostingClassifier
4  from sklearn.svm import SVC
5  from sklearn.neighbors import KNeighborsClassifier
6  from sklearn.naive_bayes import GaussianNB
7
8  # Regression logistique
9  model = LogisticRegression(C=1.0, max_iter=1000)
10
11 # Arbres
12 model = DecisionTreeClassifier(max_depth=5, criterion='gini')
13 model = RandomForestClassifier(n_estimators=100, max_depth=10)
14 model = GradientBoostingClassifier(n_estimators=100,
       learning_rate=0.1)
15
16 # Autres
17 model = SVC(kernel='rbf', C=1.0, probability=True)
18 model = KNeighborsClassifier(n_neighbors=5)
19 model = GaussianNB()
```

Listing B.12 – Modèles de classification

B.5.4 Clustering

```python
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.mixture import GaussianMixture

# K-Means
kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X)
centers = kmeans.cluster_centers_

# DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
labels = dbscan.fit_predict(X)

# Hierarchique
hc = AgglomerativeClustering(n_clusters=3, linkage='ward')
labels = hc.fit_predict(X)

# GMM
gmm = GaussianMixture(n_components=3)
labels = gmm.fit_predict(X)
```

Listing B.13 – Modèles de clustering

B.5.5 Prétraitement

```python
from sklearn.preprocessing import (
    StandardScaler, MinMaxScaler, RobustScaler,
    LabelEncoder, OneHotEncoder,
    PolynomialFeatures
)
from sklearn.impute import SimpleImputer

# Normalisation
scaler = StandardScaler()          # Moyenne=0, std=1
scaler = MinMaxScaler()            # [0, 1]
scaler = RobustScaler()            # Robuste aux outliers

# Encodage
le = LabelEncoder()
y_encoded = le.fit_transform(y)
```

```
16
17 ohe = OneHotEncoder(sparse=False)
18 X_encoded = ohe.fit_transform(X[['cat_col']])
19
20 # Valeurs manquantes
21 imputer = SimpleImputer(strategy='mean')  # ou 'median',
       'most_frequent'
22 X_imputed = imputer.fit_transform(X)
23
24 # Features polynomiales
25 poly = PolynomialFeatures(degree=2, include_bias=False)
26 X_poly = poly.fit_transform(X)
```

Listing B.14 – Prétraitement

B.5.6 Validation Croisée et Hyperparamètres

```
1 from sklearn.model_selection import (
2     cross_val_score, GridSearchCV, RandomizedSearchCV,
3     KFold, StratifiedKFold
4 )
5
6 # Validation croisee simple
7 scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
8 print(f"Accuracy: {scores.mean():.3f} (+/- {scores.std()*2:.3f})")
9
10 # Grid Search
11 param_grid = {
12     'n_estimators': [50, 100, 200],
13     'max_depth': [3, 5, 10, None],
14     'min_samples_split': [2, 5, 10]
15 }
16 grid = GridSearchCV(model, param_grid, cv=5, scoring='accuracy',
       n_jobs=-1)
17 grid.fit(X_train, y_train)
18 print(f"Best params: {grid.best_params_}")
19 print(f"Best score: {grid.best_score_:.3f}")
20 best_model = grid.best_estimator_
21
22 # Randomized Search
23 from scipy.stats import randint, uniform
24 param_dist = {
25     'n_estimators': randint(50, 500),
26     'max_depth': randint(3, 20),
27     'min_samples_split': randint(2, 20)
```

```
}
random_search = RandomizedSearchCV(model, param_dist, n_iter=100,
    cv=5)
```

Listing B.15 – Validation et tuning

B.5.7 Métriques

```
from sklearn.metrics import (
    # Regression
    mean_squared_error, mean_absolute_error, r2_score,
    # Classification
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, classification_report,
    roc_auc_score, roc_curve
)

# Regression
mse = mean_squared_error(y_true, y_pred)
rmse = mean_squared_error(y_true, y_pred, squared=False)
mae = mean_absolute_error(y_true, y_pred)
r2 = r2_score(y_true, y_pred)

# Classification
acc = accuracy_score(y_true, y_pred)
prec = precision_score(y_true, y_pred, average='weighted')
rec = recall_score(y_true, y_pred, average='weighted')
f1 = f1_score(y_true, y_pred, average='weighted')

# Matrice de confusion
cm = confusion_matrix(y_true, y_pred)
print(classification_report(y_true, y_pred))

# ROC-AUC
auc = roc_auc_score(y_true, y_proba)
fpr, tpr, thresholds = roc_curve(y_true, y_proba)
```

Listing B.16 – Métriques d'évaluation

Annexe C

Exercices Corrigés

Cette annexe fournit les solutions détaillées des exercices proposés dans les différents chapitres.

C.1 Chapitre 1 : Introduction à l'IA

Exercice 1.1 - QCM Compréhension

1. **Réponse : b)** Le Machine Learning est un sous-ensemble de l'Intelligence Artificielle.

   Explication : La hiérarchie correcte est : IA ⊃ ML ⊃ DL. Le Machine Learning est une branche de l'IA qui permet aux machines d'apprendre à partir de données.

2. **Réponse : c)** L'apprentissage par renforcement.

   Explication : Quand un agent apprend en jouant contre lui-même, il reçoit des récompenses (gagner) et des pénalités (perdre), ce qui caractérise l'apprentissage par renforcement.

3. **Réponse : c)** Clustering.

   Explication : Le clustering est une technique d'apprentissage non supervisé qui regroupe des données similaires.

C.2 Chapitre 2 : Python pour la Data Science

Exercice 2.1 - NumPy : Opérations matricielles

```
import numpy as np

# 1. Creer la matrice A
A = np.array([[1, 2, 3],
              [4, 5, 6],
```

```
                [7, 8, 9]])

# 2. Somme de chaque ligne et colonne
somme_lignes = np.sum(A, axis=1)     # [6, 15, 24]
somme_colonnes = np.sum(A, axis=0)   # [12, 15, 18]

# 3. Normaliser chaque colonne
mean_cols = np.mean(A, axis=0)
std_cols = np.std(A, axis=0)
A_normalized = (A - mean_cols) / std_cols

# 4. Calculer la trace
trace = np.trace(A)   # 1 + 5 + 9 = 15

# 5. Valeurs propres
eigenvalues, eigenvectors = np.linalg.eig(A)
# Environ : [16.12, -1.12, 0]
```

Listing C.1 – Solution Exercice 2.1

Exercice 2.2 - Pandas : Analyse Titanic

```
import pandas as pd
import seaborn as sns

# Charger le dataset
df = sns.load_dataset('titanic')

# 1. Exploration
print(df.shape)            # (891, 15)
print(df.isnull().sum())   # Valeurs manquantes

# 2. Taux de survie
taux_global = df['survived'].mean()  # ~38%
taux_classe = df.groupby('pclass')['survived'].mean()
# Classe 1: ~63%, Classe 2: ~47%, Classe 3: ~24%

# 3. FamilySize
df['FamilySize'] = df['sibsp'] + df['parch'] + 1

# 4-5. Visualisations
sns.boxplot(data=df, x='pclass', y='age')
sns.heatmap(df.corr(), annot=True)
```

Listing C.2 – Solution Exercice 2.2

C.3 Chapitre 3 : Algèbre Linéaire

Exercice 3.1 - Produit scalaire et norme

Soit $\boldsymbol{u} = (1, 2, 3)^\top$ et $\boldsymbol{v} = (4, -1, 2)^\top$.

1. Produit scalaire :

$$\boldsymbol{u} \cdot \boldsymbol{v} = 1 \times 4 + 2 \times (-1) + 3 \times 2 = 4 - 2 + 6 = \boxed{8} \tag{C.1}$$

2. Normes :

$$\|\boldsymbol{u}\|_2 = \sqrt{1 + 4 + 9} = \sqrt{14} \approx 3.74 \tag{C.2}$$

$$\|\boldsymbol{v}\|_2 = \sqrt{16 + 1 + 4} = \sqrt{21} \approx 4.58 \tag{C.3}$$

3. Angle :

$$\cos\theta = \frac{8}{\sqrt{14} \times \sqrt{21}} \approx 0.467 \implies \theta \approx \boxed{62.2} \tag{C.4}$$

Exercice 3.2 - Multiplication matricielle

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \mathbf{B} = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

$$\mathbf{AB} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix} \tag{C.5}$$

$$\mathbf{BA} = \begin{pmatrix} 23 & 34 \\ 31 & 46 \end{pmatrix} \tag{C.6}$$

Attention

$\mathbf{AB} \neq \mathbf{BA}$: le produit matriciel n'est pas commutatif !

C.4 Chapitre 4 : Calcul Différentiel

Exercice 4.1 - Gradient

$f(x, y) = x^2 + 3xy + y^2$

Gradient :

$$\nabla f = \begin{pmatrix} 2x + 3y \\ 3x + 2y \end{pmatrix} \tag{C.7}$$

Au point $(1, 2)$ **:**

$$\nabla f(1,2) = \begin{pmatrix} 2+6 \\ 3+4 \end{pmatrix} = \begin{pmatrix} 8 \\ 7 \end{pmatrix} \tag{C.8}$$

Hessienne :

$$\mathbf{H} = \begin{pmatrix} 2 & 3 \\ 3 & 2 \end{pmatrix} \tag{C.9}$$

C.5 Chapitre 5 : Probabilités

Exercice 5.1 - Théorème de Bayes

Test médical : Sensibilité $= 99\%$, Spécificité $= 95\%$, Prévalence $= 1\%$

Question : $P(\text{Malade}|\text{Test}^+)$?

$$P(T^+) = 0.99 \times 0.01 + 0.05 \times 0.99 = 0.0594 \tag{C.10}$$

$$P(M|T^+) = \frac{0.99 \times 0.01}{0.0594} \approx \boxed{16.7\%} \tag{C.11}$$

Important

Même avec un test fiable, si la maladie est rare, un test positif ne garantit pas la maladie !

C.6 Chapitre 6 : Régression Linéaire

Exercice 6.1 - Régression simple

Données : $\{(1,2), (2,4), (3,5), (4,4), (5,5)\}$

Coefficients :

$$\bar{x} = 3, \quad \bar{y} = 4 \tag{C.12}$$

$$w_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2} = \frac{4}{10} = 0.4 \tag{C.13}$$

$$w_0 = \bar{y} - w_1\bar{x} = 4 - 0.4 \times 3 = 2.8 \tag{C.14}$$

Équation : $\boxed{\hat{y} = 2.8 + 0.4x}$

R^2 **:**

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{2.8}{6} = \boxed{0.533} \tag{C.15}$$

C.7 Chapitre 7 : Classification

Exercice 7.1 - Propriétés de la sigmoïde

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

1. Montrer $\sigma(-z) = 1 - \sigma(z)$:

$$\sigma(-z) = \frac{1}{1+e^{z}} = \frac{e^{-z}}{e^{-z}+1} = 1 - \frac{1}{1+e^{-z}} = 1 - \sigma(z) \quad \checkmark \tag{C.16}$$

2. Dérivée :

$$\frac{d\sigma}{dz} = \sigma(z)(1-\sigma(z)) \tag{C.17}$$

Cette propriété est cruciale pour la rétropropagation !

C.8 Chapitre 8 : Arbres et Ensembles

Exercice 8.1 - Gain d'information

Dataset : 6 positifs, 4 négatifs. Division : Gauche (4+, 1-), Droite (2+, 3-)

Entropie initiale :

$$H(S) = -0.6\log_2(0.6) - 0.4\log_2(0.4) = 0.971 \tag{C.18}$$

Entropie après division :

$$H(G) = -0.8\log_2(0.8) - 0.2\log_2(0.2) = 0.722 \tag{C.19}$$
$$H(D) = -0.4\log_2(0.4) - 0.6\log_2(0.6) = 0.971 \tag{C.20}$$
$$H(S|A) = 0.5 \times 0.722 + 0.5 \times 0.971 = 0.847 \tag{C.21}$$

Gain :

$$IG(A) = 0.971 - 0.847 = \boxed{0.124} \tag{C.22}$$

C.9 Chapitre 9 : SVM

Exercice 9.1 - Marge

$\boldsymbol{w} = (3,4)^\top$

$$\|\boldsymbol{w}\| = 5 \implies \gamma = \frac{2}{5} = \boxed{0.4} \tag{C.23}$$

C.10 Chapitre 10 : K-Means

Exercice 10.1 - K-Means manuel

Points : $A(1,1)$, $B(1,2)$, $C(4,4)$, $D(5,4)$

Centres initiaux : $\mu_1 = (1,1)$, $\mu_2 = (5,4)$

Itération 1 :

— Cluster 1 : $\{A, B\}$ (plus proches de μ_1)

— Cluster 2 : $\{C, D\}$ (plus proches de μ_2)

Nouveaux centres :

$$\mu_1 = \frac{(1,1)+(1,2)}{2} = (1, 1.5) \tag{C.24}$$

$$\mu_2 = \frac{(4,4)+(5,4)}{2} = (4.5, 4) \tag{C.25}$$

L'algorithme converge car les clusters ne changent plus.

Bibliographie

Livres de Référence

— **Mathematics for Machine Learning** – Deisenroth, Faisal, Ong

— **Pattern Recognition and Machine Learning** – Christopher Bishop

— **The Elements of Statistical Learning** – Hastie, Tibshirani, Friedman

— **Hands-On Machine Learning with Scikit-Learn** – Aurélien Géron

— **Deep Learning** – Goodfellow, Bengio, Courville

Ressources en Ligne

— Coursera : Machine Learning (Andrew Ng)

— Stanford CS229 : Machine Learning

— 3Blue1Brown : Essence of Linear Algebra

— StatQuest : Statistics and Machine Learning