

Intelligence Artificielle

Finance et Gestion



Objectifs pédagogiques du cours



- Comprendre les concepts et domaines de l'IA (IA, ML, DL, NLP)
 - Maîtriser les méthodes clés (supervisé, non supervisé, deep learning)
 - Appliquer l'IA à la finance et à la gestion (prédiction, classification, optimisation, NLP)
 - Développer des solutions reproductibles en Python (pandas, scikit-learn, keras)
 - Évaluer les enjeux éthiques, les biais et la conformité (RGPD, AI Act)
 - Savoir dialoguer avec les équipes data/IT et piloter des projets IA
-

Plan du cours (12 chapitres)



- 1 Introduction à l'IA
 - 2 Fondements mathématiques
 - 3 Apprentissage supervisé
 - 4 Apprentissage non supervisé
 - 5 Réseaux de neurones & Deep Learning
 - 6 Traitement du langage (NLP)
 - 7 IA explicable (XAI)
 - 8 IA en Finance
 - 9 IA en Gestion
 - 10 Éthique & Réglementation
 - 11 MLOps & Déploiement
 - 12 Projets pratiques
-

Cours IA – Finance & Gestion (4e année)

Modalités d'évaluation et prérequis



Modalités d'évaluation

Transparence du barème

40% Contrôle continu

QCM, mini-projets, participation

30% Projet pratique

Étude de cas, code, rapport, soutenance

30% Examen final

Théorie + étude(s) de cas

■ Barème et critères de notation communiqués dès le début

Rubriques détaillées: exactitude, rigueur, clarté, interprétabilité

Prérequis

Connaissances de base

- Statistiques descriptive et inférentielle
- Algèbre linéaire (vecteurs, matrices)
- Notions de programmation Python

Outils à utiliser

■ Jupyter Notebook

■ pandas

■ scikit-learn

■ matplotlib

Compétences métier

- Finance & gestion (KPI, risques, ROI)
- Excel / BI (visualisation, reporting)

Votre enseignant et le cadre du cours



[Nom Prénom]

[Prof./Maître de conf.] – ENCG Settat

Spécialités

- IA appliquée à la finance
- NLP (traitement du langage)
- IA explicable (XAI)
- MLOps & déploiement



Recherche & Projets récents

- ✓ Scoring crédit
- ✓ Optimisation de portefeuille
- ✓ Détection de fraude
- ✓ NLP & assistants conversationnels



Disponibilités & bureau

Créneaux [créneaux]

Bureau [N°/Bâtiment]



Contacts

@ [email]

#[Teams/Slack]



Politique d'assiduité et d'intégrité académique

Présence requise aux séances et respect des délais. Tout plagiat ou fraude est interdit et sanctionné selon le règlement de l'ENCG.

Définitions clés (IA / ML / DL / Data Science)



Intelligence Artificielle (IA)

Techniques visant à créer des systèmes capables d'exécuter des tâches nécessitant l'intelligence humaine (raisonner, percevoir, décider).



Machine Learning (ML)

Apprentissage à partir de données pour prédire/agir : supervisé, non supervisé, et par renforcement.



Deep Learning (DL)

Réseaux de neurones profonds pour modéliser des patterns complexes : CNN (images), RNN/LSTM (séries), Transformers (texte).



Data Science

Extraction de connaissances à partir des données : statistiques, ML, data visualisation et ingénierie des données.



Types d'apprentissage

Supervisé (données étiquetées) • Non supervisé (structure sans étiquettes) • Renforcement (agent, récompenses).

Objectif: apprendre un comportement ou des regroupements à partir des données/retours.



Jeux de données: train/val/test

Entraînement (apprendre) • Validation (régler hyperparamètres) • Test (évaluer généralisation). Séparation indispensable pour éviter la fuite d'information et mesurer la performance réelle.



Caractéristiques, étiquettes, cible

Caractéristiques (features): variables explicatives.

Étiquettes (labels): valeurs connues (supervisé).

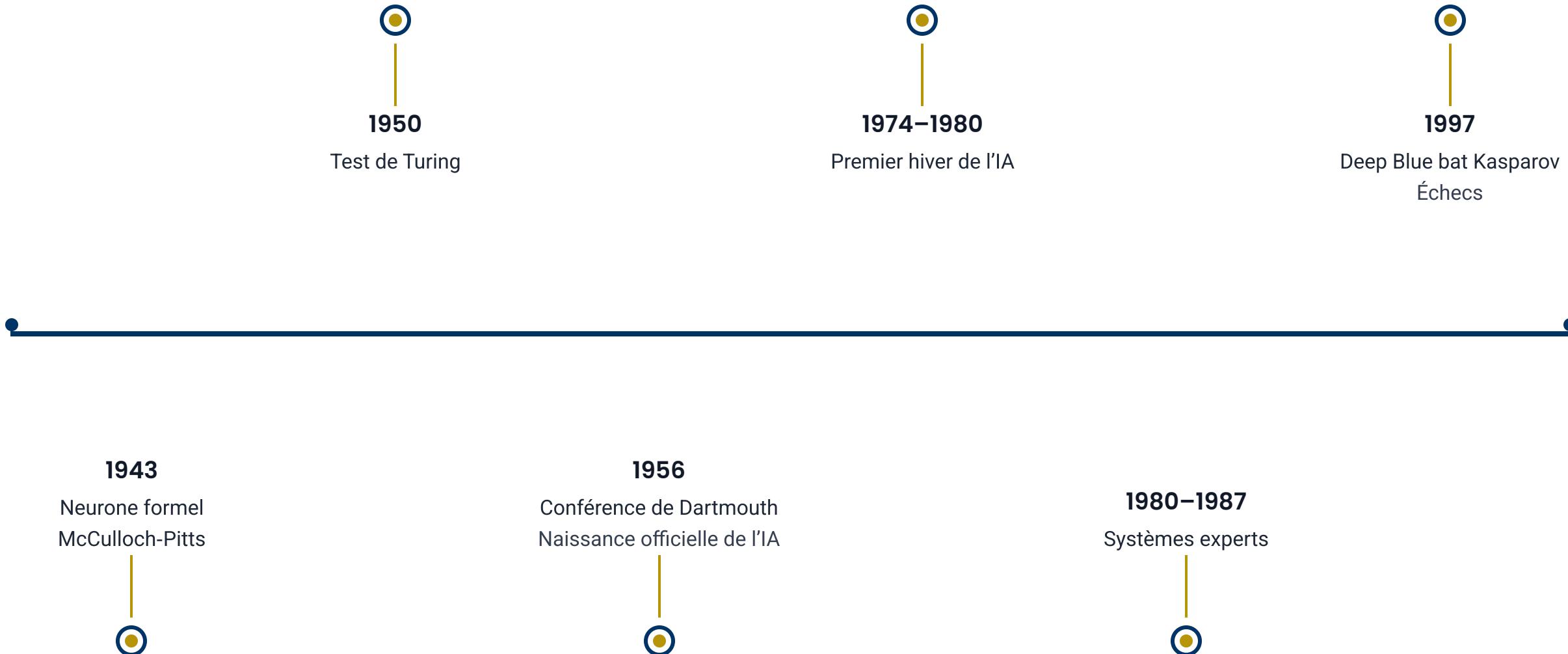
Cible (target): variable à prédire (ex.: défaut = 0/1, prix, segment).



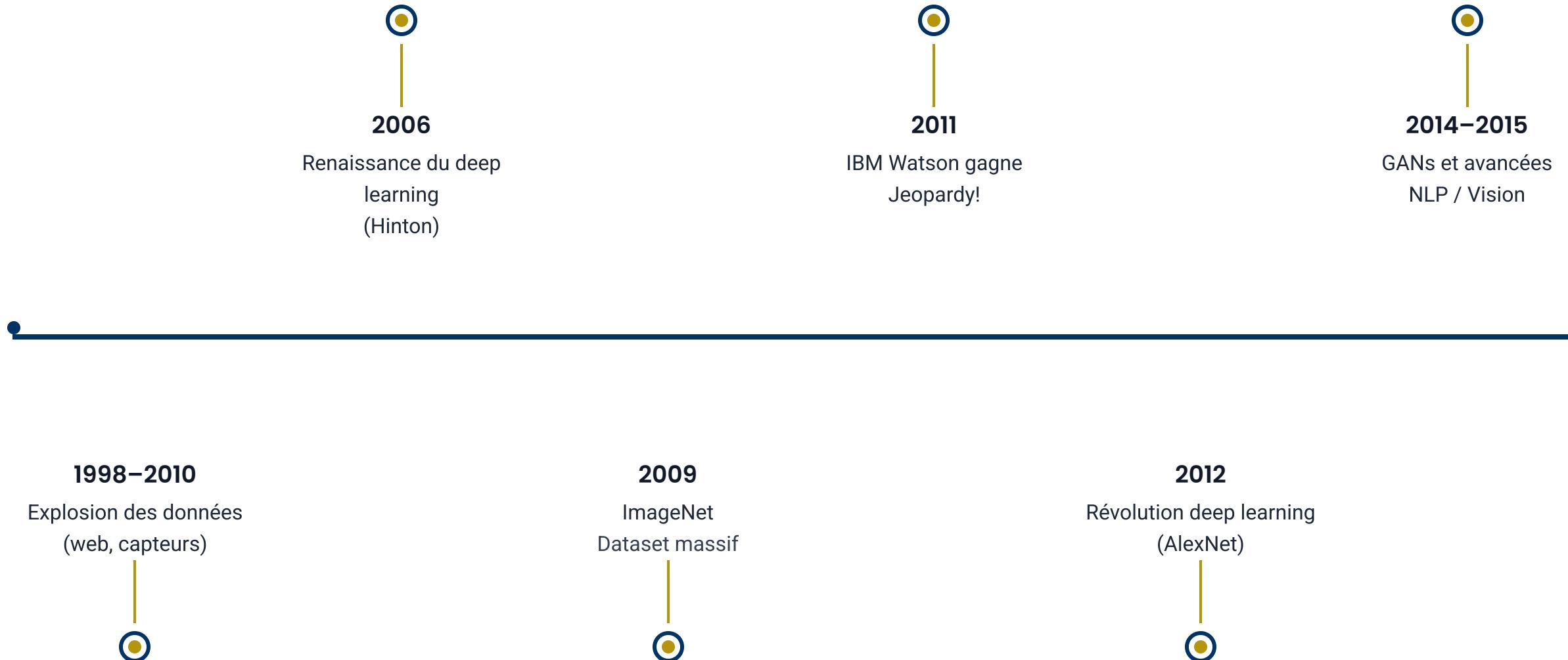
Surapprentissage vs sous-apprentissage

Surapprentissage: modèle trop complexe, mémorise le train, généralise mal. Sous-apprentissage: trop simple. Remèdes: plus de données, régularisation, validation croisée, ajuster la complexité du modèle.

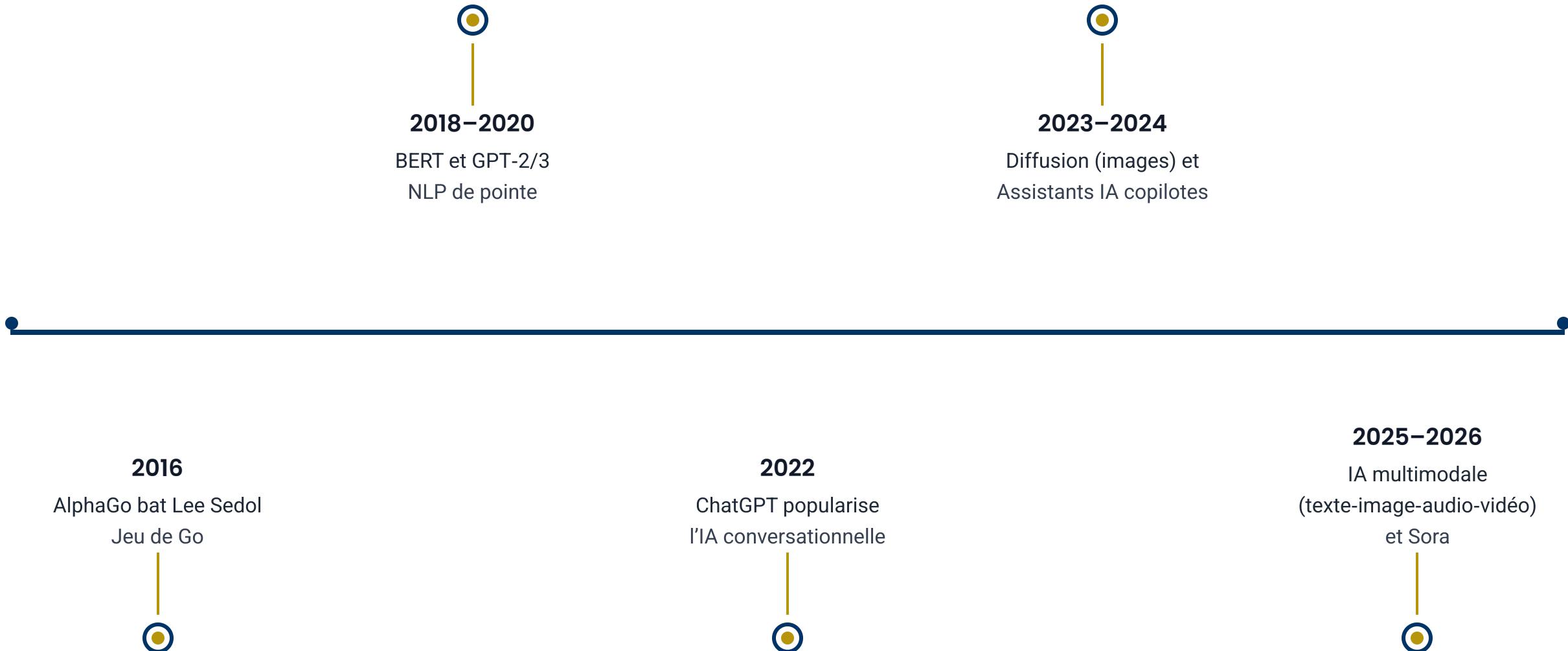
Timeline IA (1943–1997) – Des fondations aux victoires



Timeline IA (1998–2015) – Internet, données et puissance



Timeline IA (2016–2026) – Deep Learning, IA générative et multimodalité



Types d'IA – Faible (Narrow) • Générale (AGI) • Forte (ASI)



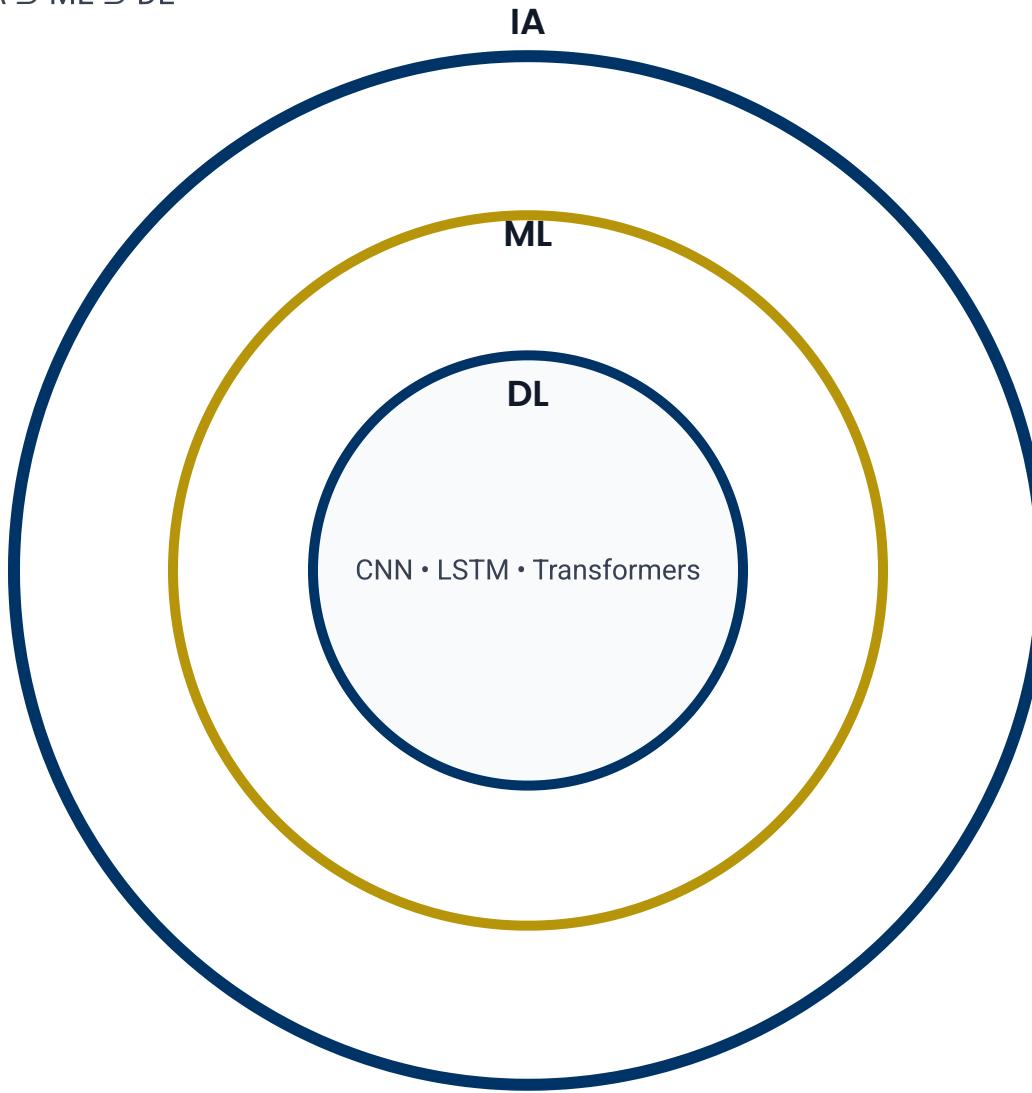
| Critère | IA Faible (Narrow) | IA Générale (AGI) | IA Forte (ASI) |
|-------------------------|--|---|---|
| Définition | Conçue pour une tâche spécifique bien délimitée. | Polyvalente, capacité de transfert entre tâches variées. | Dépasse les performances humaines dans la plupart des domaines. |
| Capacités | Optimise une tâche donnée selon des objectifs définis. | Raisonnement général, apprentissage flexible et adaptation. | Capacités supérieures, potentielle auto-amélioration. |
| Exemples | Recommandation, reconnaissance d'images, traduction automatique. | Aucun exemple réel à ce jour (hypothétique). | Aucune instance existante (théorique/spéculatif). |
| Maturité | Mature et largement déployée aujourd'hui. | Hypothétique (non atteinte). | Théorique (concept non réalisé). |
| Risques | Circonscrits à la tâche (erreurs locales, biais). | Systémiques (mauvaise généralisation, impacts sociaux). | Gouvernance critique, perte potentielle de contrôle. |
| Horizon temporel | Présent (usage courant). | Long terme / indéterminé. | Inconnu / spéculatif. |

Types d'IA – Comparaison par usages et limites

| Critère | IA Faible (Narrow) | IA Générale (AGI) | IA Forte (ASI) |
|-------------------------|--|---|--|
| Polyvalence | Faible (une tâche ciblée, peu de transfert). | Maximale (apprentissage et adaptation multi-tâches). | Extrême (dépassement des capacités humaines). |
| Interprétabilité | Plutôt élevée (+ interprétable). | Limitée (modèles potentiellement opaques et complexes). | Très faible/opaque. |
| Données requises | Modérées, annotées pour la tâche visée. | Massives et variées (domaines multiples). | Inconnues/spéculatives (hypothétiques). |
| Besoins calcul | Modérés (selon algorithme et taille). | Très élevés (entraînement et inférence). | Potentiellement gigantesques. |
| Gouvernance | Locale, risques limités. | Complexe (impacts sociétaux, conformité). | Enjeux éthiques majeurs, gouvernance critique. |

IA, ML, DL – Relations (diagramme de Venn)

Inclusion: IA ⊇ ML ⊇ DL



● IA hors ML

- Systèmes experts (règles)

- Logique floue

● ML hors DL

- Arbres de décision, Forêts aléatoires
- SVM, Régressions (linéaire/logistique)

● DL (au sein du ML)

- CNN (images)
- LSTM/GRU (séries temporelles)
- Transformers (texte)

Idée clé: Tout DL ⊂ ML ⊂ IA.



LLM – Modèles de langage

Modèles auto-régressifs pour la génération et la synthèse de texte.

Exemples d'usages :

- Génération / résumé de documents
- RAG (Retrieval-Augmented Generation)



Diffusion – Images

Apprentissage par débruitage progressif pour générer des visuels réalistes.

Exemples d'usages :

- Génération d'images produits/maquettes
- Storytelling data & communication visuelle



Multimodal – Texte ↔ Image/Audio/Vidéo

Compréhension et génération sur plusieurs modalités, coordination par agents.

Exemples d'usages :

- Agents & copilotes métiers
- Analyse de documents (texte+image) et vidéos



ChatGPT

Assistant conversationnel pour texte et code.

Usages :

- Questions / réponses et assistance
- Synthèse de documents
- Génération et révision de code



DALL-E / Midjourney

Génération d'images de qualité à partir de texte.

Usages :

- Création d'images et maquettes
- Data storytelling visuel



Sora

Génération vidéo à partir de texte (texte → vidéo).

Usages :

- Prototypage créatif
- Publicité et communication

IA générative – Cas d'usage: Finance & Gestion



Focus usages métier concrets et bonnes pratiques

Finance

Exemples concrets

- Génération de rapports risques & conformité
Synthèse automatisée des expositions, stress tests, seuils d'alerte.
- Assistants KYC/AML
Aide à la collecte/validation des pièces, résumés d'alertes.
- RAG sur politiques de crédit
Réponses appuyées par documents internes et sources tracées.

Gestion

- Assistants service client
Réponses contextualisées, handover fluide vers agents humains.
- Rédaction d'emails marketing
Variations A/B, ton adapté au segment, respect de la charte.
- Synthèse de réunions RH
Compte-rendu, actions, responsabilités et échéances.

Garde-fous indispensables

Validation humaine (human-in-the-loop)
Relecture et approbation avant envoi.

Filtres de sécurité
Blocage contenu sensible, conformité RGPD.

Traçabilité des sources
Références, RAG, horodatage des réponses.

Applications IA en Finance – Vue d'ensemble



Scoring crédit

Estimer la probabilité de défaut (PD) pour accorder un prêt.

KPI : AUC

TPR @ FPR contrôlé



Détection de fraude

Repérer des anomalies en temps réel dans les transactions.

KPI : Precision@K

Latence faible



Trading algorithmique

Générer des signaux et optimiser l'exécution des ordres.

KPI : Sharpe

Drawdown



Robo-advisors

Allocation personnalisée selon le profil de risque.

Profil risque

KYC & conformité



Pipelines types

Chaîne de valeur d'un modèle IA en production.

Ingestion données Features Modèle Calibration
Monitoring



KPI

Mesures de performance modèle et business.

AUC/ROC F1-score Precision@K (fraude) Sharpe/Sortino
Coût du risque ROI



Contraintes

Exigences techniques et réglementaires clés.

Données déséquilibrées Latence Conformité (RGPD/Bâle)



Outils

Écosystème pour entraîner, suivre et déployer.

scikit-learn XGBoost MLflow Monitoring (drift, alertes)

Applications IA en Gestion – Vue d'ensemble



CRM & personnalisation

Next-best-action et recommandation de produits pour booster l'engagement.

Recommandation NBA



Supply chain

Prévision de la demande et optimisation des stocks pour réduire les ruptures.

Forecast Stocks



Ressources humaines

Prédiction du turnover, screening de CV et mobilité interne assistée.

Turnover ATS



Marketing

Segmentation, attribution multi-canale et optimisation budgétaire.

Segmentation Attribution

Applications IA en Gestion – Impacts et ROI



Indicateurs

Impacts opérationnels mesurés par les KPI métiers.

Churn -15 à -30%

Conversion +5 à +12%

Stocks -10 à -20%

Time-to-hire -20 à -35%



Enablers

Conditions de succès pour industrialiser et adopter l'IA.

Gouvernance data

MLOps

XAI & adoption

Formation équipes



Risques

Points de vigilance à maîtriser avant le déploiement.

Biais

Dette technique

Drift des données



Investissement

Ressources et leviers pour passer à l'échelle.

Cloud

Talents

Outils

Cas d'usage sectoriels détaillés



Banque

Scoring crédit, détection de fraude et KYC automatisé pour accélérer la décision et réduire le risque.

Scoring

Fraude

KYC



Assurance

Évaluation des risques, fraude aux sinistres et chatbots pour indemnisation rapide et expérience client.

Tarification

Fraude sinistres

Chatbots



Retail

Recommandation produits, prévision de la demande et prix dynamiques pour booster conversion et marge.

Reco

Demande

Pricing



Santé

Diagnostic assisté, analyse d'imagerie et prédition des réadmissions pour améliorer les parcours de soins.

Diagnostic

Imagerie

Réadmissions

Cas d'usage sectoriels (suite) – ROI et indicateurs



Manufacturing

Maintenance prédictive et contrôle qualité par vision.

↓ 20–30% coûts

Qualité ↑



Télécoms

Prédiction du churn et optimisation du réseau.

Churn ↓ 15–25%

QoS ↑



Immobilier

Valorisation automatique et ciblage de prospects.

± 5% précision

Leads qualifiés ↑



Transport & Logistique

Optimisation des tournées et prévision de la demande.

↓ 10–15% km

Service level ↑

Quiz interactif — Chapitre 1



1

Différence IA faible vs IA forte

Quelle est la différence entre IA faible (spécialisée) et IA forte (générale/super-intelligente) ?

2

Naissance officielle de l'IA

Quelle année marque la naissance officielle de l'IA ?
(Réponse : 1956 – Conférence de Dartmouth)

3

Applications en finance

Citez 3 applications de l'IA en finance (ex. scoring crédit, détection de fraude, trading algorithmique).

4

IA, ML et DL – Relations

Quelle est la relation entre IA, Machine Learning et Deep Learning (ensembles imbriqués) ?

5

IA générative – Définition et exemples

Qu'est-ce que l'IA générative et citez 2 outils (ex. ChatGPT, DALL·E, Midjourney, Sora).

Pourquoi des mathématiques en IA ?



- Modéliser les phénomènes (incertitude, relations linéaires/non linéaires)
- Optimiser des fonctions de coût pour apprendre des paramètres
- Prédire et quantifier les risques/incertitudes
- Domaines clés : statistiques, algèbre linéaire, calcul différentiel, optimisation
- Niveau requis : révisions Bac+2, approche intuitive + rigueur



Mesures de tendance centrale

Moyenne

$$\bar{x} = (1/n) \sum x_i$$

Moyenne arithmétique des n observations.

Médiane

Valeur centrale

Valeur qui coupe l'échantillon en deux parts égales.

Mode

Valeur la plus fréquente

La valeur observée avec la fréquence maximale.



Remarque – Sensibilité aux valeurs extrêmes

La moyenne est sensible aux valeurs extrêmes (outliers), alors que la médiane y est robuste. Utiliser la médiane lorsque la distribution est asymétrique ou contient des valeurs aberrantes.

Statistiques descriptives – Dispersion

Équations (dispersion)

Variance

$$\sigma^2 = (1/n) \sum (x_i - \bar{x})^2$$

Mesure la dispersion des observations autour de la moyenne (en unités au carré).

Écart-type

$$\sigma = \sqrt{\sigma^2}$$

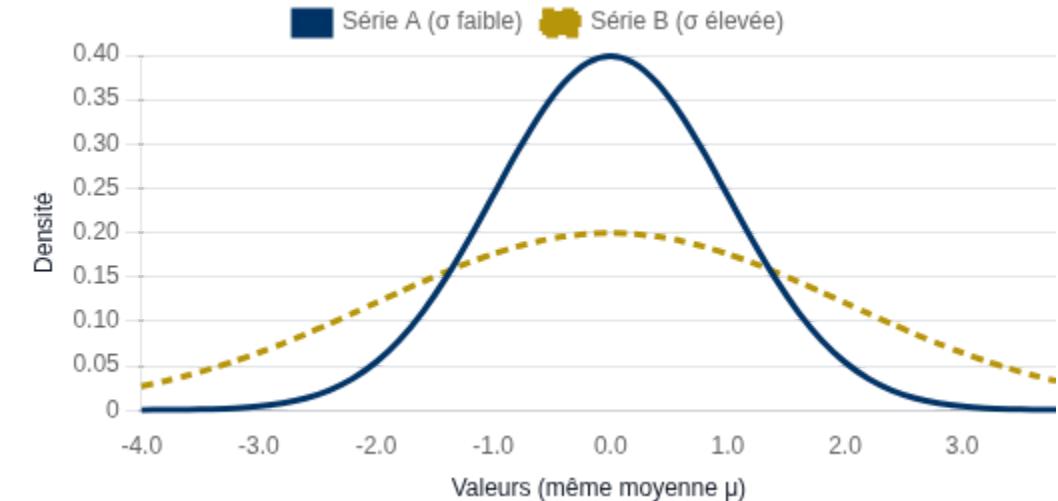
Mesure de dispersion exprimée dans les mêmes unités que la variable.

Règle pratique

Plus σ est grand, plus la série est volatile/incertaine.

Visuel et interprétation

Deux séries avec même moyenne mais dispersions différentes



Lecture

Série A (σ faible) concentrée;
Série B (σ élevée) plus étalée.

Interprétation managériale

En finance, la dispersion reflète le **risque/volatilité** des rendements.

Exemple financier – Rendements mensuels (calcul détaillé)



Données et objectifs

Données (rendements mensuels en %)

+1,2 +0,8 -0,5 +2,1 +0,3 -1,0



Calculs pas à pas

Étape 1 – Moyenne

$$\bar{r} = (1/n) \sum r_i$$

Résultat (approx.) : $\bar{r} \approx 0,82\%$

Interprétation : rendement moyen mensuel positif.

Étape 2 – Volatilité (écart-type)

$$\sigma = \sqrt{(1/n) \sum (r_i - \bar{r})^2}$$

Résultat (approx.) : $\sigma_r \approx 1,04\%$

Interprétation : dispersion modérée autour de la moyenne.



Lecture

Rendement moyen positif et **volatilité modérée** : le profil risque/rendement est compatible avec un portefeuille diversifié à sensibilité moyenne.

Statistiques en Python (pandas)



Extrait de code – Calculs de base avec pandas

```
# Importer les bibliothèques
import pandas as pd
import numpy as np

# Série de rendements mensuels (en %)
r = pd.Series([1.2, 0.8, -0.5, 2.1, 0.3, -1.0])

# Moyenne et écart-type (population: ddof=0)
print(r.mean(), r.std(ddof=0))

# Statistiques descriptives (min, quartiles, max...)
print(r.describe())
```

Astuce: utiliser `ddof=0` pour l'écart-type population, `ddof=1` pour l'échantillon.

Lecture des résultats

Moyenne (μ) $\approx 0.82\%$ – rendement moyen.

Écart-type (σ) $\approx 1.04\%$ – volatilité.

describe() résume
count/mean/std/min/quartiles/max.

Bonnes pratiques

- Vérifier unités (% vs décimales).
- Documenter les hypothèses (population/échantillon).



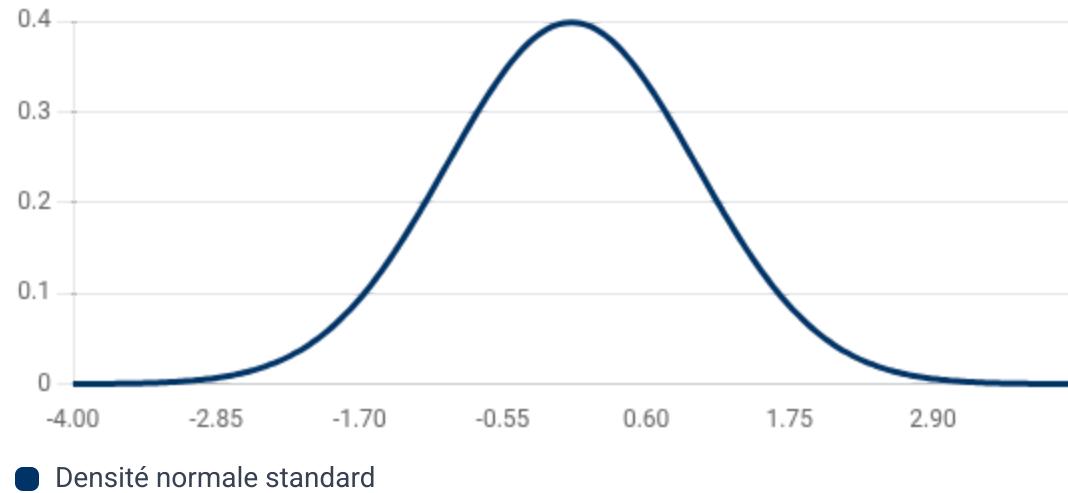
Densité et paramètres

$$f(x) = 1 / (\sigma\sqrt{2\pi}) \cdot \exp(- (x - \mu)^2 / (2\sigma^2))$$

Avec : μ = moyenne, σ = écart-type

- Distribution **symétrique** autour de μ
- Forme en **cloche** (Gauss), contrôlée par σ

Visualisation ($\mu = 0, \sigma = 1$)

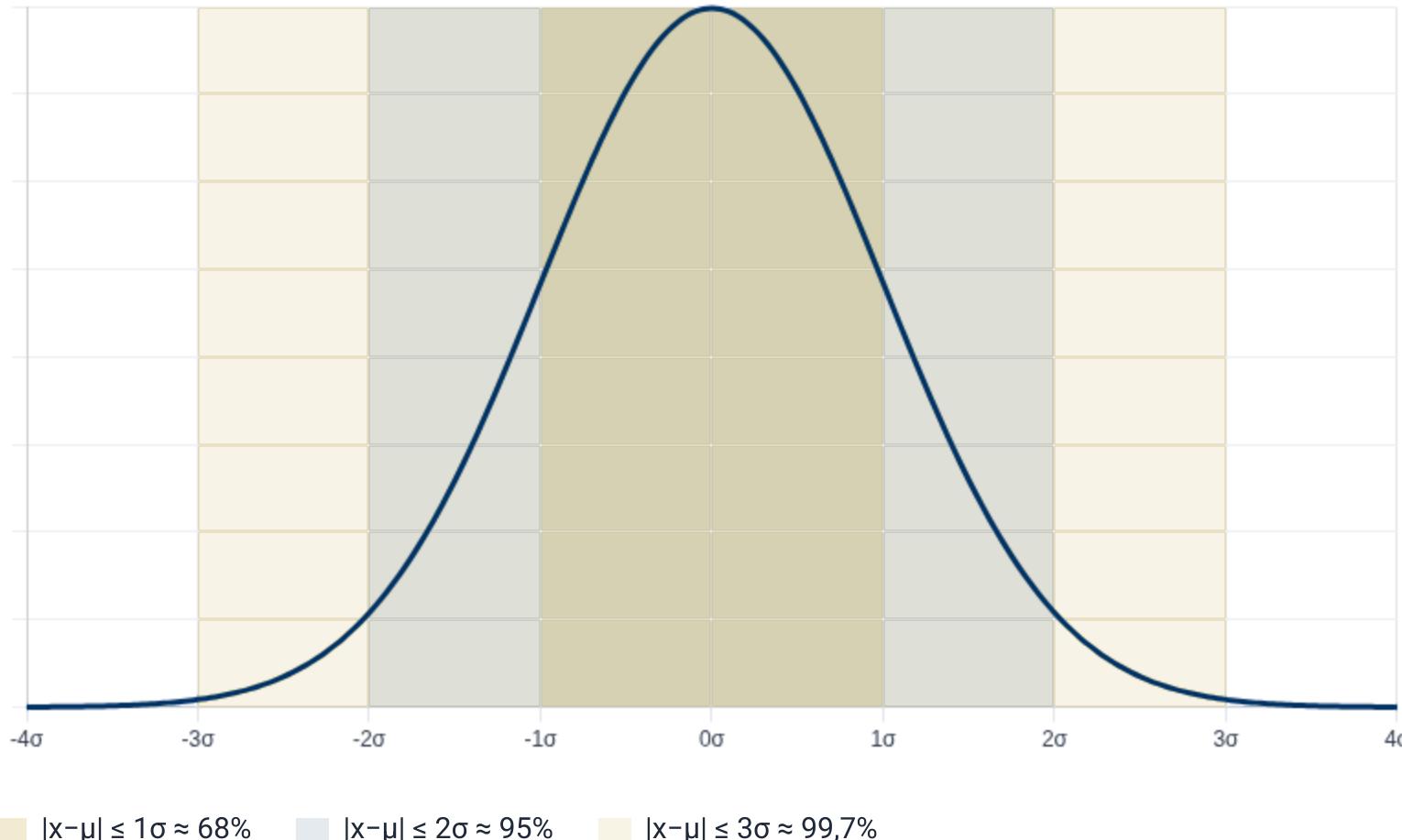


Interprétation

La probabilité se concentre autour de μ . En pratique, $\sim 68\%$ des observations se situent dans $[\mu \pm \sigma]$, $\sim 95\%$ dans $[\mu \pm 2\sigma]$, $\sim 99.7\%$ dans $[\mu \pm 3\sigma]$.

Propriétés de $N(\mu, \sigma^2)$ – Loi normale

Courbe en cloche avec zones colorées: $|x-\mu| \leq 1\sigma, 2\sigma$ et 3σ



Propriétés clés

1) Règle 68–95–99,7

$P(|X-\mu| \leq \sigma) \approx 0,68$; $P(|X-\mu| \leq 2\sigma) \approx 0,95$; $P(|X-\mu| \leq 3\sigma) \approx 0,997$.

2) Standardisation

$z = (x - \mu) / \sigma$. Si $X \sim N(\mu, \sigma^2)$ alors $Z = (X - \mu) / \sigma \sim N(0, 1)$.

3) Additivité (indépendance)

Si $X \sim N(\mu_1, \sigma_1^2)$, $Y \sim N(\mu_2, \sigma_2^2)$ indép., alors $X+Y \sim N(\mu_1+\mu_2, \sigma_1^2+\sigma_2^2)$.

Symétrie autour de μ et queue fine
(distributions sous-gaussiennes).

Couleurs ENCG : Bleu #003366 • Doré #b7950b



Applications (théorie → pratique)

- VaR (Value-at-Risk) : perte maximale attendue à un niveau de confiance (ex. 95%).
- Modélisation des rendements : hypothèse de normalité pour estimer risque et volatilité.
- Approximation Black–Scholes : rendements log-normaux pour le pricing d'options.
- Hypothèses à vérifier : stationnarité, queues épaisses, corrélations.

Code Python – VaR 95%

NumPy · SciPy

```
import numpy as np
import scipy.stats as st

# Hypothèses: rendements ~ N( $\mu$ ,  $\sigma^2$ )
mu, sigma = 0.005, 0.02 # 0.5% moyen, 2% volatilité mensuelle

# VaR à 95% (quantile 5%)
var95 = mu + sigma * st.norm.ppf(0.05)
print("VaR 95%:", var95)
```

Interprétation : avec 95% de confiance, la perte mensuelle ne dépassera pas la VaR (sous l'hypothèse de normalité).

Formule et interprétation

$$P(A | B) = [P(B | A) \cdot P(A)] / P(B)$$

Mise à jour d'une croyance initiale $P(A)$ après observation de l'évidence B .

Classification

$$P(y | x) \propto P(x | y) \cdot P(y)$$

On combine la vraisemblance des données $P(x | y)$ avec l'a priori de la classe $P(y)$.

Intuition

Le théorème de Bayes réalise une **mise à jour** cohérente de nos croyances à partir d'une nouvelle information (évidence).

Si l'évidence est fréquente sous A ($P(B|A)$ élevée), la croyance en A augmente.

Applications

- Filtrage de spam (email)
- Scoring crédit (probabilité de défaut)
- Diagnostic (détection de fraude, médical)



Attention au taux de base (base rate)

Lorsque l'événement A est rare, $P(A|B)$ peut rester faible même si $P(B|A)$ est élevée. Toujours considérer $P(A)$.

Exemple pas à pas – Détection de fraude (Bayes)



Données (probabilités)

$$P(\text{Fraude}) = 0,001 \cdot P(\text{Transaction suspecte} | \text{Fraude}) = 0,95 \cdot P(\text{Transaction suspecte}) = 0,01$$

1 Numérateur

$$0,95 \times 0,001 = 0,00095$$

Interprétation: probabilité conjointe d'avoir une fraude et un signal suspect.

Formule: $P(S \wedge F) = P(S | F) \times P(F)$

2 Division par l'évidence

$$0,00095 / 0,01 = 0,095$$

On conditionne par la probabilité d'une transaction suspecte.

Formule: $P(F | S) = P(S \wedge F) / P(S)$

3 Posterior

$$P(\text{Fraude} | \text{Suspecte}) = 0,095 = 9,5\%$$

Attention au **faible taux de base** (prévalence faible) qui limite la probabilité a posteriori.

Formule de Bayes: $P(F | S) = [P(S | F) P(F)] / P(S)$

Rappel – **Théorème de Bayes**: $P(A | B) = [P(B | A) \times P(A)] / P(B)$. Ici A=Fraude, B=Transaction suspecte.

Naïve Bayes (sklearn) – Démo



Extrait de code – Gaussian Naïve Bayes (classification binaire)

```
# 1) Importer le classifieur
from sklearn.naive_bayes import GaussianNB

# 2) Jeu de données jouet (2 features) et étiquettes binaires
X = [[35, 0.6],
      [52, 0.2],
      [28, 0.7]]
y = [1, 0, 1]

# 3) Entraîner le modèle
nb = GaussianNB().fit(X, y)

# 4) Prédire les probabilités pour une nouvelle observation
proba = nb.predict_proba([[40, 0.5]])
print(proba) # sortie: [[P(classe=0) P(classe=1)]] (ex.: [[0.32 0.68]])
```

Ce que montre la démo

Apprentissage supervisé binaire avec GaussianNB.

Sortie: probabilités pour chaque classe [0, 1].

Exemple de features: [âge, ratio] (e.g., ratio dette).

Bonnes pratiques

- Vérifier les échelles et l'hypothèse gaussienne par classe.
- Comparer avec une baseline (Logistic, Arbre) et valider par CV.

Astuce: Naïve Bayes gaussien suppose l'indépendance conditionnelle des features et une distribution normale par classe.



Définition (variable aléatoire discrète X)

$$H(X) = - \sum p_i \log_2 (p_i)$$

Avec \log_2 en base 2, l'unité est le **bit**. Convention : $0 \cdot \log(0) = 0$.



Interprétation

L'entropie mesure l'**incertitude moyenne** d'une source aléatoire : plus la distribution est étalée, plus $H(X)$ est élevée. Elle quantifie l'information attendue par observation.

En IA/ML : utilisée pour les **arbres de décision** (entropie et gain d'information) et pour l'**optimisation** (pertes de type entropie croisée).



Cas extrêmes

Uniforme (max)

Si $p_i = 1/K$ pour $i=1 \dots K$, alors

$$H(X) = \log_2(K)$$

Incultitude maximale.

Dégénérée (min)

Si $p_{i_0} = 1$ et 0 sinon,

$$H(X) = 0$$

Aucune incertitude.

Gain d'information (arbres de décision)



Formule du gain d'information

$$\text{Gain}(S, A) = H(S) - \sum(|S_v| / |S|) \cdot H(S_v)$$

Où $H(S)$ est l'entropie du jeu S , et S_v les sous-ensembles induits par les valeurs v de l'attribut A .

1 Avant le split

Label binaire 50/50 $\rightarrow H(S) = 1$ bit.

2 Split parfait

Noeuds fils purs $\rightarrow H(S_v) = 0$ pour chaque v .

3 Gain obtenu

$\text{Gain}(S, A) = 1 - 0 = 1$ bit.



Interprétation pour arbres de décision

- Plus le **gain d'information** est élevé, plus la question (attribut A) **réduit l'incertitude** \rightarrow bon candidat pour la racine ou un noeud.
- Critères alternatifs proches : **indice de Gini** (approxime l'entropie), ou **gain normalisé** (C4.5) pour éviter le biais vers les attributs très arisés.
- Attention au **surapprentissage** : limiter la profondeur, exiger un nombre minimal d'échantillons par noeud, ou élaguer l'arbre.

Notations et formules

Vecteur colonne

$$v = (v_1, \dots, v_n)^T$$

Représente n composantes; orientation colonne pour le calcul matriciel.

Produit scalaire

$$u \cdot v = \sum u_i v_i$$

Mesure l'alignement entre deux vecteurs de même dimension.

Norme euclidienne

$$\|v\| = \sqrt{(\sum v_i^2)}$$

Longueur du vecteur; utilisée pour distances et régularisation.

Cosine (similarité)

$$\cos \theta = (u \cdot v) / (\|u\| \|v\|)$$

Indique la proximité directionnelle; compris entre -1 et 1.

Applications en IA

- Similarité cosinus pour comparer textes (NLP, recherche d'information).
- Distances et normes pour clustering (K-means) et KNN.
- Représentation des features en vecteurs pour modèles ML.



Opérations matricielles et notation

Produit matriciel

$$(AB)_{ij} = \sum_k a_{ik} b_{kj}$$

Dimensions compatibles : $(m \times p) \cdot (p \times n) \rightarrow (m \times n)$.

Transposée

$$A^T$$

Propriété clé : $(AB)^T = B^T A^T$.

Inverse (si inversible)

$$A A^{-1} = I$$

Exige $\det(A) \neq 0$. Sinon utiliser pseudo-inverse.

Système linéaire

$$A x = b \rightarrow x = A^{-1}b$$

En pratique : résoudre numériquement (LU/QR) plutôt qu'inverser explicitement.



Bonnes pratiques et notations

Ne pas calculer A^{-1} pour résoudre $A x = b$: utiliser des solveurs numériques stables. Indices i, j, k désignent respectivement ligne, colonne et somme sur la dimension commune.

Données comme matrice X



→ Colonnes = variables (âge, revenu, score)

| Observation | Âge | Revenu | Score |
|-------------|-----|--------|-------|
| Client 1 | 34 | 3 200 | 650 |
| Client 2 | 48 | 4 600 | 720 |
| Client 3 | 27 | 2 500 | 590 |

↓ Lignes = observations (clients)

Notation : $X \in \mathbb{R}^{n \times p}$ avec $X = [x_{ij}]$, $i = 1 \dots n$ (observations), $p =$ nombre de variables.

Structure et usage

Chaque ligne représente un client, chaque colonne une variable explicative utilisée par les algorithmes (X) pour prédire une cible y.

Bonnes pratiques

Normalisation Encodage Gestion des NA

- Normaliser/standardiser pour comparer les échelles (ex. âge vs revenu).
- Encoder les variables catégorielles (One-Hot, ordinal si pertinent).
- Traiter les valeurs manquantes (imputation, suppression raisonnée).

Astuce : conserver une **feature store** et documenter les transformations pour la reproductibilité (MLOps).

NumPy – Opérations matricielles



Extrait de code – Produits, transposée, inverse et résolution

```
# Import des bibliothèques
import numpy as np

# Définir deux matrices 2x2
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

# 1) Produit matriciel (équivalent à np.matmul(A,B))
print("A @ B =\n", A @ B)

# 2) Transposée de A
print("A.T =\n", A.T)

# 3) Inverse de A (si det(A) ≠ 0)
print("inv(A) =\n", np.linalg.inv(A))

# 4) Résoudre Ax = b sans inverser A (plus stable numériquement)
b = np.array([1, 0])
print("solve(A, b) = ", np.linalg.solve(A, b))
```

Lecture des résultats

A @ B calcule le produit matriciel (composition de transformations).

A.T échange lignes/colonnes (utile pour produits scalaires/matriciels).

inv(A) existe si $\det(A) \neq 0$ (attention au conditionnement).

solve(A,b) donne x tel que $Ax=b$ (méthode recommandée).

Bonnes pratiques

- Utiliser `@` pour les produits; `*` est élément-par-élément.
- Éviter l'inversion explicite pour résoudre des systèmes.
- Vérifier les dimensions compatibles avant les opérations.

Astuce: préférez `np.linalg.solve(A, b)` à `np.linalg.inv(A) @ b` pour plus de stabilité et de performance.



Définition et exemple

Définition

$$\frac{\partial f}{\partial x_i} = \lim_{h \rightarrow 0} [f(x_1, \dots, x_i+h, \dots, x_n) - f(x_1, \dots, x_i, \dots, x_n)] / h$$

Taux de variation de f lorsqu'on fait varier une seule variable x_i et que les autres restent constantes.

Exemple

$$f(x,y) = x^2 + x y$$

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

$$\frac{\partial f}{\partial y} = x$$

Lecture : la pente locale par rapport à x dépend de x et y , tandis que la pente par rapport à y est x .



Interprétation géométrique

La dérivée partielle mesure la pente de la surface $z = f(x,y)$ dans la direction d'un axe (x ou y). Sur une carte de niveaux (courbes de niveau), elle indique la variation de z quand on se déplace horizontalement ($\frac{\partial f}{\partial x}$) ou verticalement ($\frac{\partial f}{\partial y}$) en gardant l'autre variable constante.



Définition et propriétés

$$\nabla f(x) = (\partial f / \partial x_1, \dots, \partial f / \partial x_n)^T$$

Le gradient rassemble les dérivées partielles et indique la direction de plus forte croissance de f .

Propriété et descente

- Direction de plus forte croissance: suivre ∇f
- Descente (minimisation): aller vers $-\nabla f$

Idée clé: pour réduire f , on se déplace à l'opposé du gradient.

Exemple – Régression linéaire (MSE)

$$\partial J / \partial \theta = (1/m) X^T (X\theta - y)$$

Avec X : matrice des caractéristiques, y : cibles, m : nombre d'observations.



Mise à jour (descente de gradient) – $\theta \leftarrow \theta - \alpha \nabla J(\theta)$, avec α le taux d'apprentissage.

Fonction de coût – Visualisation 3D

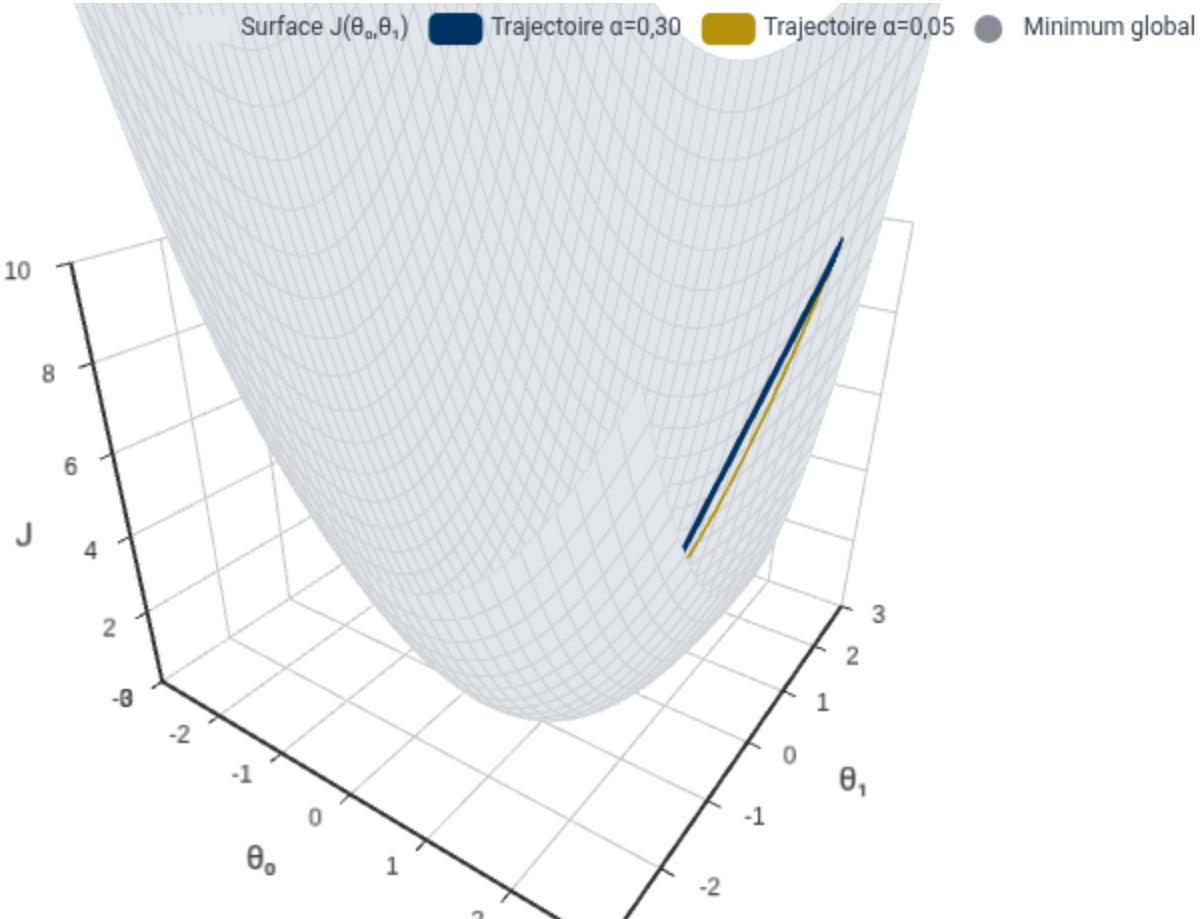
Surface $J(\theta_0, \theta_1)$ convexe (régression linéaire)

- $J(\theta_0, \theta_1)$ est une surface en « bol » (paraboloïde) avec un **minimum global**.
- La **descente de gradient** suit des trajectoires vers ce minimum.
- **Taux d'apprentissage α** : trop petit \rightarrow convergence lente, trop grand \rightarrow oscillations/divergence.

Mise à jour des paramètres :

$$\theta \leftarrow \theta - \alpha \cdot \nabla J(\theta)$$

- Trajectoire ($\alpha = 0,30$) – rapide et stable
- Trajectoire ($\alpha = 0,05$) – lente
- Point du minimum global



Formule d'actualisation des paramètres

$$\theta \leftarrow \theta - \alpha \nabla J(\theta)$$

Paramètres

- θ – vecteur des paramètres du modèle (poids à apprendre).
- α – taux d'apprentissage (*learning rate*), contrôle l'amplitude de la mise à jour.
- $\nabla J(\theta)$ – gradient de la fonction de coût (direction de plus forte croissance de J).

Principe et critère d'arrêt

Mettre à jour θ itérativement dans la direction opposée au gradient afin de minimiser $J(\theta)$.

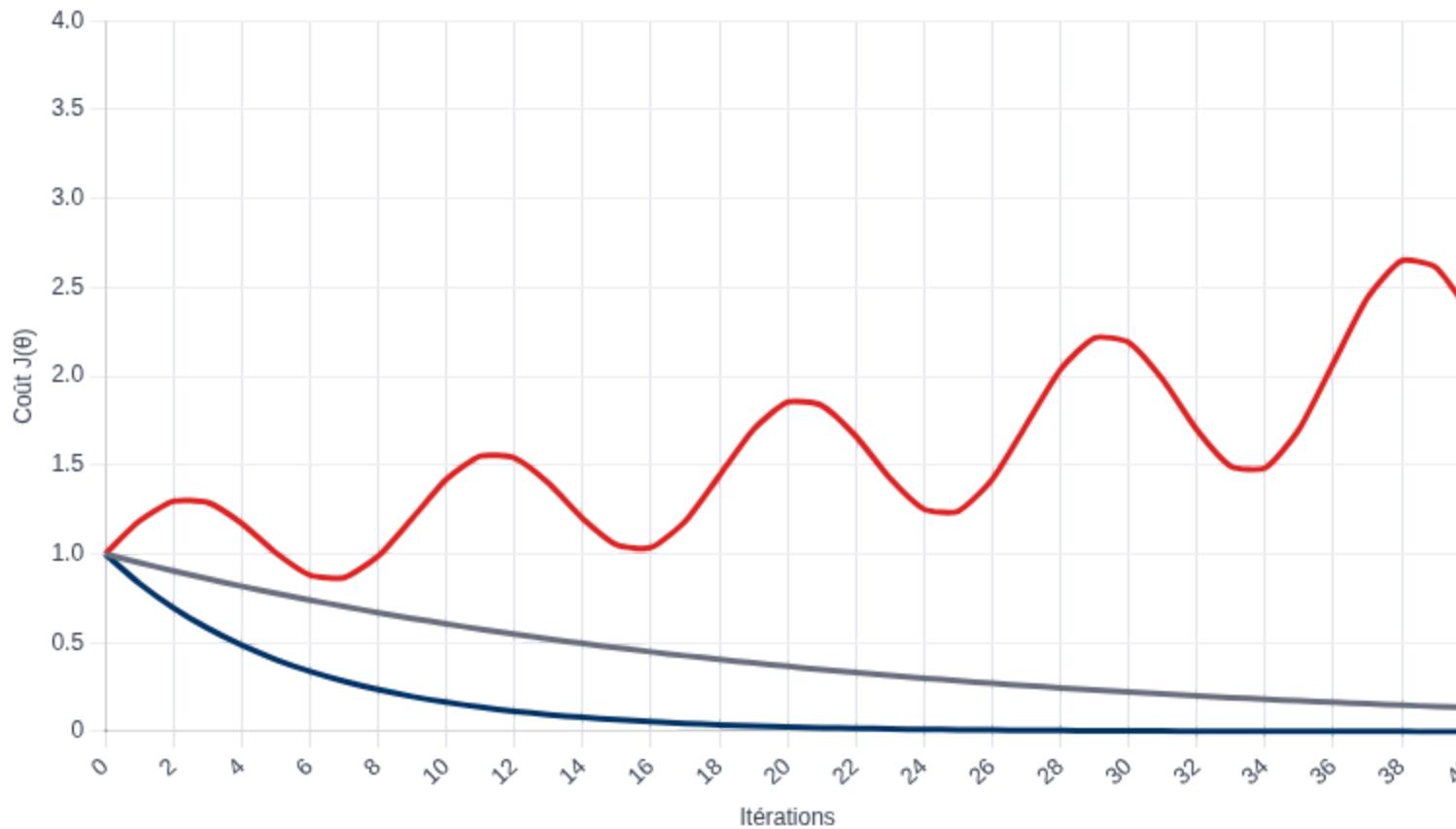
- Répéter jusqu'à convergence :

Arrêt si $|\nabla J(\theta)| < \epsilon$ (seuil) ou si le **nombre max d'itérations** est atteint.

Choisir α avec soin : trop petit \Rightarrow lent ; trop grand \Rightarrow instable (oscillations/divergence).

Learning rate α – Impact sur la convergence

Coût $J(\theta)$ en fonction des itérations pour trois valeurs de α



Axe X : Itérations • Axe Y : Coût $J(\theta)$

Couleurs ENCG : Bleu #003366 • Doré #b7950b

Effet de α sur la descente de gradient

- α trop petit**
Convergence lente (beaucoup d'itérations), coût reste élevé longtemps.
- α optimal**
Convergence rapide et stable (sweet spot), diminution régulière de $J(\theta)$.
- α trop grand**
Oscillations voire divergence (instabilité), incapacité à atteindre le minimum.

Rappel : mise à jour $\theta \leftarrow \theta - \alpha \nabla J(\theta)$, arrêter quand convergence (critère ϵ ou itérations max).

Types de descente de gradient – Batch • SGD • Mini-batch



| Critère | Batch GD | SGD | Mini-batch GD |
|--------------------------|---|--|---|
| Données utilisées | Tout le dataset à chaque mise à jour. | 1 échantillon par mise à jour. | Lot de 32–256 échantillons. |
| Vitesse/itération | Lente (calcul complet). | Très rapide (mise à jour fréquente). | Compromis (accélère avec GPU). |
| Convergence | Stable, vers minimum (convexe), mais lente. | Bruitée, peut osciller autour du minimum. | Plus lisse que SGD, rapide et fiable en pratique. |
| Mémoire requise | Élevée (tout le dataset). | Faible (un seul point). | Modérée (taille du lot). |
| Usage recommandé | Petits datasets, entraînement hors-ligne. | Apprentissage en ligne/flux très volumineux. | Pratique courante (réseaux neuronaux, GPU). |

Code Python – Descente de gradient ($f(x) = (x-3)^2$)

Implémentation from scratch – suivi de la convergence

```
# 1) Imports
import numpy as np
import matplotlib.pyplot as plt

# 2) Fonction objectif et dérivée
def f(x): return (x - 3)**2
def df_dx(x): return 2 * (x - 3)

# 3) Hyperparamètres et initialisation
x = 0.0                      # point de départ
alpha = 0.1                    # learning rate
iterations = 20                # nombre d'itérations
xs, fs = [x], [f(x)]          # historique pour visualisation

print(f"it=0: x={x:.3f}, f={f(x):.3f}")

# 4) Boucle de descente de gradient
for i in range(1, iterations+1):
    grad = df_dx(x)           # calcul du gradient
    x = x - alpha * grad       # mise à jour : x := x - α∇f
    xs.append(x); fs.append(f(x))  # sauvegarder trajectoire
    print(f"it={i}: x={x:.3f}, f={fs[-1]:.3f}")

# 5) Visualisation de la convergence
X = np.linspace(-1, 6, 100)
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.plot(X, f(X), 'b-', label='f(x)=(x-3)^2')
plt.plot(xs, [f(v) for v in xs], 'ro-', label='Chemin GD')
plt.xlabel('x'); plt.ylabel('f(x)')
plt.title('Descente de gradient'); plt.legend()

plt.subplot(1, 2, 2)
plt.plot(range(len(fs)), fs, 'ro-')
plt.xlabel('Itérations'); plt.ylabel('f(x)')
plt.title('Convergence')
plt.tight_layout(); plt.show()
```

Paramètres clés

Point de départ : $x=0.0$

Learning rate : $\alpha=0.1$

Itérations : 20

Lecture

- La trajectoire xs montre la progression vers $x=3$.
- La courbe $f(x)$ diminue de façon monotone.

Bonnes pratiques

- Essayer plusieurs α (trop grand → divergences).
- Tracer la convergence pour diagnostiquer.

Astuce: le minimum de $f(x)$ se situe en $x=3$; la descente de gradient s'en approche de manière itérative.

Quiz interactif — Chapitre 2



1

Moyenne et variance

Calculez la moyenne et la variance de la série [10, 15, 20, 25, 30]. (Réponse : moyenne = 20, variance = 50)

2

Théorème de Bayes

Donnez la formule de Bayes et son interprétation (mise à jour d'une croyance avec une nouvelle évidence).

3

Gradient — Définition

Qu'est-ce que le gradient d'une fonction et quelle est sa propriété géométrique (direction de plus forte croissance) ?

4

Learning rate α

Quel est le rôle du learning rate α dans la descente de gradient (vitesse de convergence, stabilité) ?

5

Batch GD vs SGD

Quelle est la différence entre Batch Gradient Descent (toutes les données, plus stable) et Stochastic Gradient Descent (un échantillon, plus bruité mais rapide) ?

Apprentissage supervisé – Définition et workflow



Objectif : apprendre $f : X \rightarrow y$ en minimisant l'erreur de généralisation (bonne performance sur de **nouvelles** données).

1 Entrées étiquetées (X, y)

Dataset avec **features** (X) et **labels** (y).

Ex. scoring crédit : X = profil client, y = défaut (0/1).

2 Entraînement

L'algorithme **apprend des patterns** à partir de (X, y) .

Ajuste les paramètres pour réduire l'erreur.

3 Modèle

Fonction apprise f (ex. régression, arbre, réseau).

Prête à être utilisée sur de nouvelles données.

4 Prédictions

Application de f sur **nouvelles** observations X' .

Sorties : y' (valeurs ou classes).

5 Évaluation

Mesure des performances sur un **jeu de test**.

Métriques : MSE/R², Accuracy, F1, AUC...

6 Déploiement

Mise en production (API, batch, temps réel).

Surveillance, **drift**, réentraînement.

Rappel : bien séparer **train / validation / test** pour estimer l'erreur de généralisation et éviter la fuite de données.

Division des données (train/val/test)



Principes

Répartition recommandée

Entraînement 70%

Validation 15%

Test 15%

- Stratification (classification) pour respecter les proportions de classes
- **random_state** pour la reproductibilité
- Mélange des données (**shuffle**) avant le split
- Adapter les proportions selon la taille et l'objectif (ex. 80/10/10)

Bonnes pratiques

- Pas de fuite de données (le test ne doit jamais influencer le train/val)
Ex.: éviter de normaliser sur l'ensemble complet avant le split
- Normaliser/standardiser **après** le split (fit sur train, transform sur val/test)
- Utiliser la validation croisée (**K-fold**) si dataset de petite taille
- Garder le set **test** intouché jusqu'à l'évaluation finale

Astuce: pour séries temporelles, préférer un split chronologique (no shuffle)

Surapprentissage (Overfitting) – Causes et symptômes



Causes et symptômes

- Modèle trop complexe (trop de paramètres)
- Trop de features vs peu de données
- Données bruitées ou présence d'*outliers*
- Score **train** élevé mais score **test** faible
- Courbes d'apprentissage qui divergent

Remèdes

- Régularisation (L1/L2) et pénalisation de la complexité
- Plus de données / *data augmentation*
- *Early stopping* et validation croisée
- *Dropout / bagging* (pour modèles complexes)
- Simplifier le modèle et sélectionner les features

Sous-apprentissage (Underfitting) – Causes et symptômes



Causes et symptômes

- Modèle trop simple pour capturer les patterns
- Features insuffisantes ou peu informatives
- Scores faibles sur train et test
- Biais élevé (erreurs systématiques)
- Courbes d'apprentissage qui convergent bas

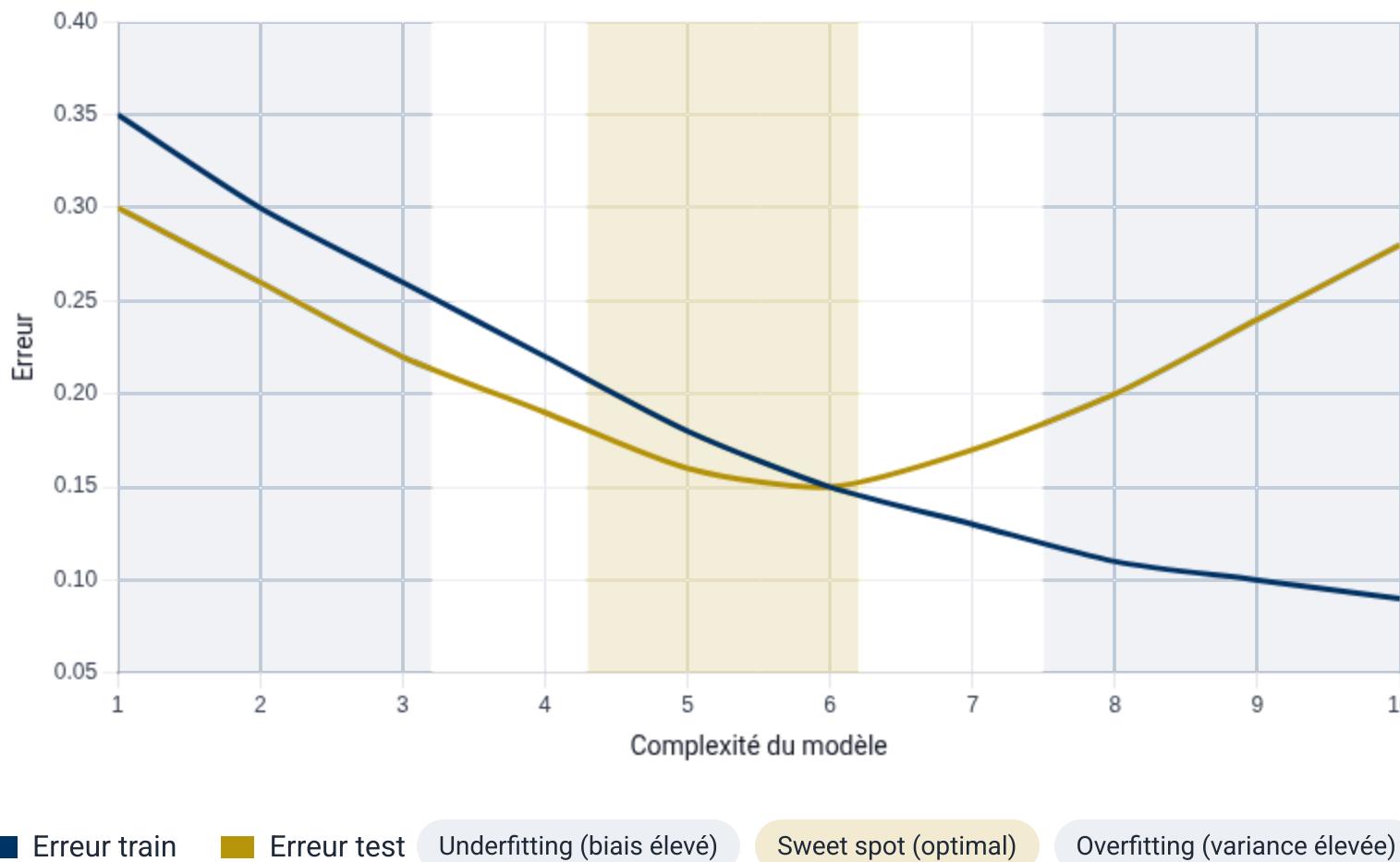
Remèdes

- Choisir un modèle plus riche/complexé
- Faire du feature engineering pertinent
- Réduire la régularisation (λ) si trop pénalisante
- Augmenter le nombre d'itérations/epochs

Indice pratique: vérifier les **courbes d'apprentissage** pour confirmer un biais élevé.

Biais–Variance – Trade-off

Axe X : Complexité du modèle • Axe Y : Erreur (train/test)



Concepts clés

Erreur de généralisation
≈ Biais² + Variance + Bruit

Zones

Gauche : **underfitting** (biais élevé). Droite : **overfitting** (variance élevée).

Objectif

Minimiser l'erreur **test** (point optimal).

Bonnes pratiques

Validation croisée, régularisation, early stopping, choix de complexité.

Couleurs ENCG : Bleu #003366 • Doré #b7950b

x¹

Formulations du modèle de régression

Modèle simple

$$y = \theta_0 + \theta_1 x + \varepsilon$$

Une variable explicative x pour expliquer y . ε = bruit aléatoire (erreur).

Modèle multiple

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n + \varepsilon$$

Plusieurs variables x_j (features) contribuent linéairement à y .

Forme vectorielle

$$\hat{y} = X \theta$$

X : matrice des features ($m \times (n+1)$), θ : paramètres, \hat{y} : prédictions.



Hypothèses (Gauss–Markov) pour une estimation valide

- ✓ Linéarité: la relation entre y et les x est (approximativement) linéaire.
- ✓ Homoscedasticité: variance constante des erreurs ($\text{Var}(\varepsilon) = \sigma^2$).
- ✓ Indépendance des erreurs: pas d'autocorrélation des résidus.
- ✓ Normalité des résidus (souvent utile) pour inférences et tests.

Remarque: en présence de violation d'hypothèses, envisager transformations, features non linéaires, régularisation, ou modèles alternatifs.



Fonction de coût (erreur quadratique moyenne)

$$J(\theta) = (1 / 2m) \cdot \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

où $h_{\theta}(x) = \theta^T x$ et m = nombre d'exemples.

Gradient

$$\frac{\partial J}{\partial \theta} = (1/m) \cdot X^T(X\theta - y)$$

Direction de plus forte croissance; on descend vers $-\nabla J$ pour minimiser $J(\theta)$.



Optimisation

- Descente de gradient (itératif): $\theta \leftarrow \theta - \alpha \nabla J(\theta)$
- Équations normales (analytique): $\theta = (X^T X)^{-1} X^T y$

Utiliser l'analytique si $X^T X$ est inversible et n est modéré; sinon préférer l'itératif.



Remarques

- Ajouter un terme d'interception (colonne de 1 dans X).
- Standardiser X si les échelles varient fortement.
- Surveiller l'overfitting via validation croisée.

Régression linéaire – Code Python (prix immobilier)



Exemple complet – Prédire le prix en fonction de la surface

```
# Imports
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Données (surface en m², prix en €)
df = pd.DataFrame({
    'Surface': [50, 70, 90, 110, 130],
    'Prix': [100000, 140000, 180000, 220000, 260000]
})

# Features (X) et cible (y)
X = df[['Surface']]
y = df['Prix']

# Split entraînement/test (80/20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Entraînement du modèle
model = LinearRegression().fit(X_train, y_train)

# Évaluation
y_pred = model.predict(X_test)
print(f"Coefficient: {model.coef_[0]:.2f} €/m²")
print(f"Intercept: {model.intercept_:.2f} €")
print(f"R²: {r2_score(y_test, y_pred):.3f}")
```

Astuce: séparer d'abord train/test, puis appliquer tout preprocessing sur X_train et réutiliser les mêmes paramètres sur X_test.

Ce que montre le code

Données jouet pour illustrer la relation linéaire Surface → Prix.

Entraînement d'un LinearRegression et calcul de R².

Interprétation du coefficient en €/m² et de l'intercept.

Bonnes pratiques

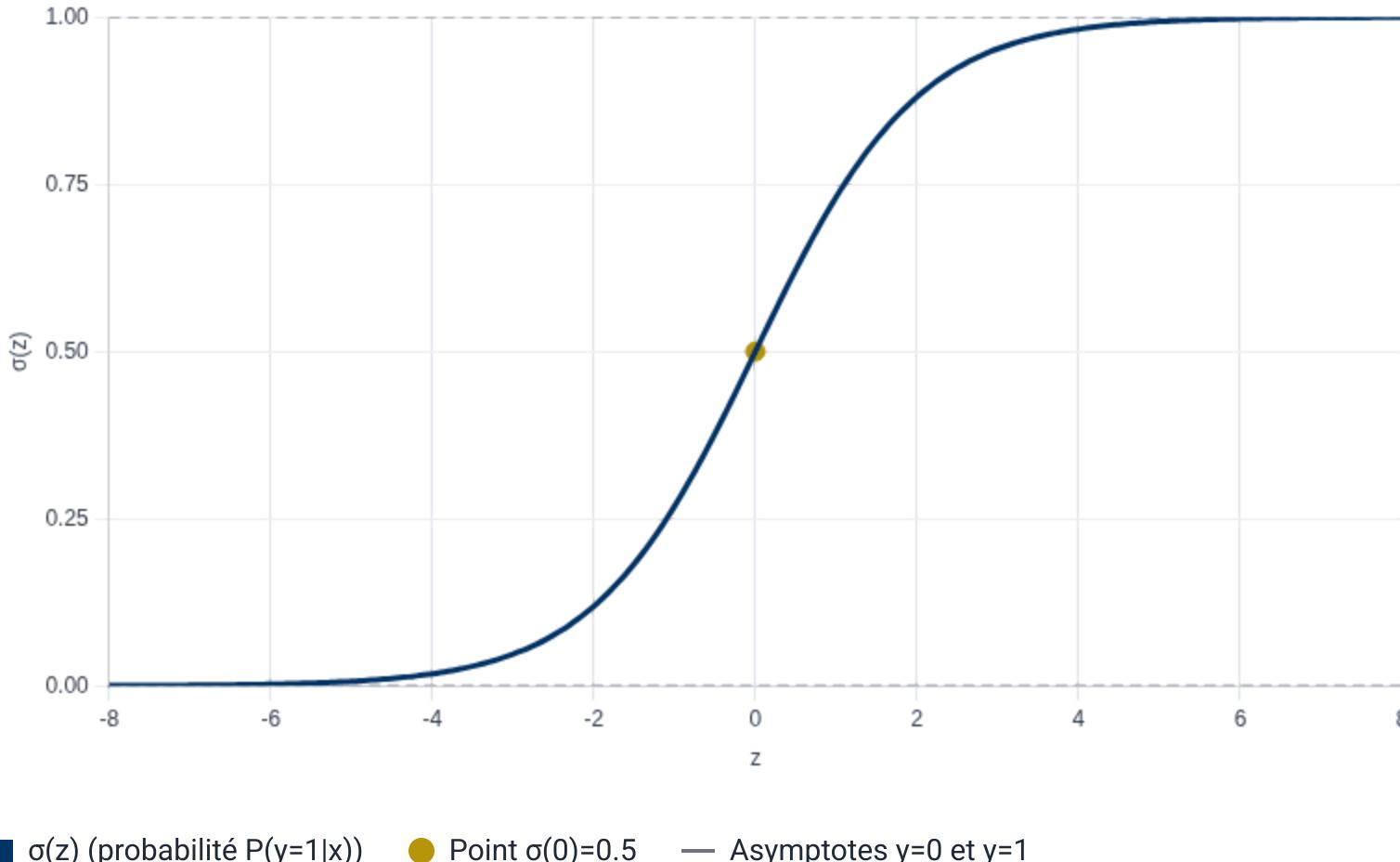
- Vérifier linéarité et résidus (hétéroscédasticité).
- Éviter la fuite de données; fixer random_state.

Métriques régression – MSE • MAE • R²

| Critère | MSE | MAE | R ² |
|-----------------------------|---|--|--|
| Définition | Moyenne des carrés des erreurs. | Moyenne des valeurs absolues des erreurs. | Part de variance expliquée par le modèle. |
| Formule | $MSE = (1/m) \sum (y - \hat{y})^2$ | $MAE = (1/m) \sum y - \hat{y} $ | $R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$ |
| Sensibilité outliers | Élevée (pénalise très fortement les grandes erreurs). | Faible (robuste aux outliers). | Moyenne (impact indirect via erreurs). |
| Interprétation | Pénalise fortement les grandes erreurs; utile pour optimisation analytique. | Erreur typique (médiane proche si asymétrique). | Entre 0 et 1 (plus proche de 1 = meilleur ajustement). |
| Usage recommandé | Quand les grandes erreurs doivent être fortement pénalisées. | Quand robustesse aux outliers est souhaitée (coût L1). | Comparer des modèles sur un même dataset (qualité d'ajustement). |

Fonction sigmoïde – Forme et propriétés

Courbe en S: $\sigma(z) = 1 / (1 + e^{-z})$ – asymptotes $y=0$ et $y=1$, point central $\sigma(0)=0.5$



Propriétés clés

Définition

$\sigma(z) = 1 / (1 + e^{-z})$; $\sigma(z) \in (0,1)$.

Point central

$\sigma(0) = 0.5$ (seuil naturel de décision).

Dérivée

$\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z))$ (max au voisinage de $z=0$).

Interprétation

Modélise une probabilité: $P(y=1|x)$. Quand $z \rightarrow +\infty$, $\sigma \rightarrow 1$; quand $z \rightarrow -\infty$, $\sigma \rightarrow 0$.

Couleurs ENCG : Bleu #003366 • Doré #b7950b

Régression logistique – Classification binaire



- Hypothèse: $P(y=1|x) = \sigma(\theta^T x)$ où σ est la fonction sigmoïde
 $z = \theta^T x$ = combinaison linéaire des variables (features)
- Décision: $\hat{y} = 1$ si $p \geq$ seuil (par défaut 0,5), sinon $\hat{y} = 0$
- Fonction de coût (log-loss, entropie croisée)
 $J(\theta) = -(1/m) \sum [y \log(h) + (1-y) \log(1-h)]$
avec $h = \sigma(\theta^T x)$
- Optimisation: descente de gradient (et variantes)
Mise à jour typique: $\theta \leftarrow \theta - \alpha \nabla J(\theta)$
- Applications clés
Scoring crédit, détection de spam, diagnostic médical
- Note pratique
Ajuster le seuil si classes déséquilibrées (ex.: fraude, défaut)

Scoring crédit – Logistic Regression (code Python)



Exemple minimal – Pipeline standardisation + régression logistique

```
# Imports
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Données simulées : [Âge, Revenu, Ratio_dette]
X = np.array([[35, 3000, 0.3],
              [52, 5000, 0.1],
              [28, 2000, 0.7],
              [45, 4500, 0.2]])
y = np.array([1, 0, 1, 0]) # 1 = défaut, 0 = pas défaut

# Split train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42, stratify=y)

# Standardisation (nécessaire pour LR stable et comparable)
scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Modèle de scoring (régression logistique)
clf = LogisticRegression(random_state=42).fit(X_train_scaled, y_train)

# Évaluation
print(f"Accuracy: {clf.score(X_test_scaled, y_test):.2%}")
print(classification_report(y_test,
                            clf.predict(X_test_scaled),
                            zero_division=0))
```

Astuce: utiliser `stratify=y` pour préserver le ratio de classes; ajuster le seuil de décision selon l'objectif métier (ex. contrôle du FPR).

Lecture rapide

StandardScaler rend les features comparables.

La LR renvoie **probabilités** et classes.

Accuracy utile si classes équilibrées; sinon préférer F1/ROC-AUC.

Bonnes pratiques

- Pipeline `scaler` + modèle pour éviter fuites.
- Évaluer sur **test** jamais vu.
- Suivre aussi **Precision/Recall** selon le risque.

Matrice de confusion – TP / FP / TN / FN

| | | Prédit |
|---------|--|---|
| Réel | Positif | Négatif |
| Positif | TP Vrais Positifs Ex.: 45 | FN Faux Négatifs Ex.: 15 |
| Négatif | FP Faux Positifs Ex.: 5 | TN Vrais Négatifs Ex.: 135 |

TP FN FP TN

Indicateurs dérivés

$$\text{Précision} = \text{TP} / (\text{TP} + \text{FP})$$

$$\text{Rappel} = \text{TP} / (\text{TP} + \text{FN})$$

$$\text{Spécificité} = \text{TN} / (\text{TN} + \text{FP})$$

Exemple chiffré (200 cas)

TP = 45

$$FP = 5$$

FN = 15

TN = 135

Précision = 45 / 50 = 90%

Rappel = 45 / 60 = 75%

Spécificité = 135 / 140 = **96,4%**

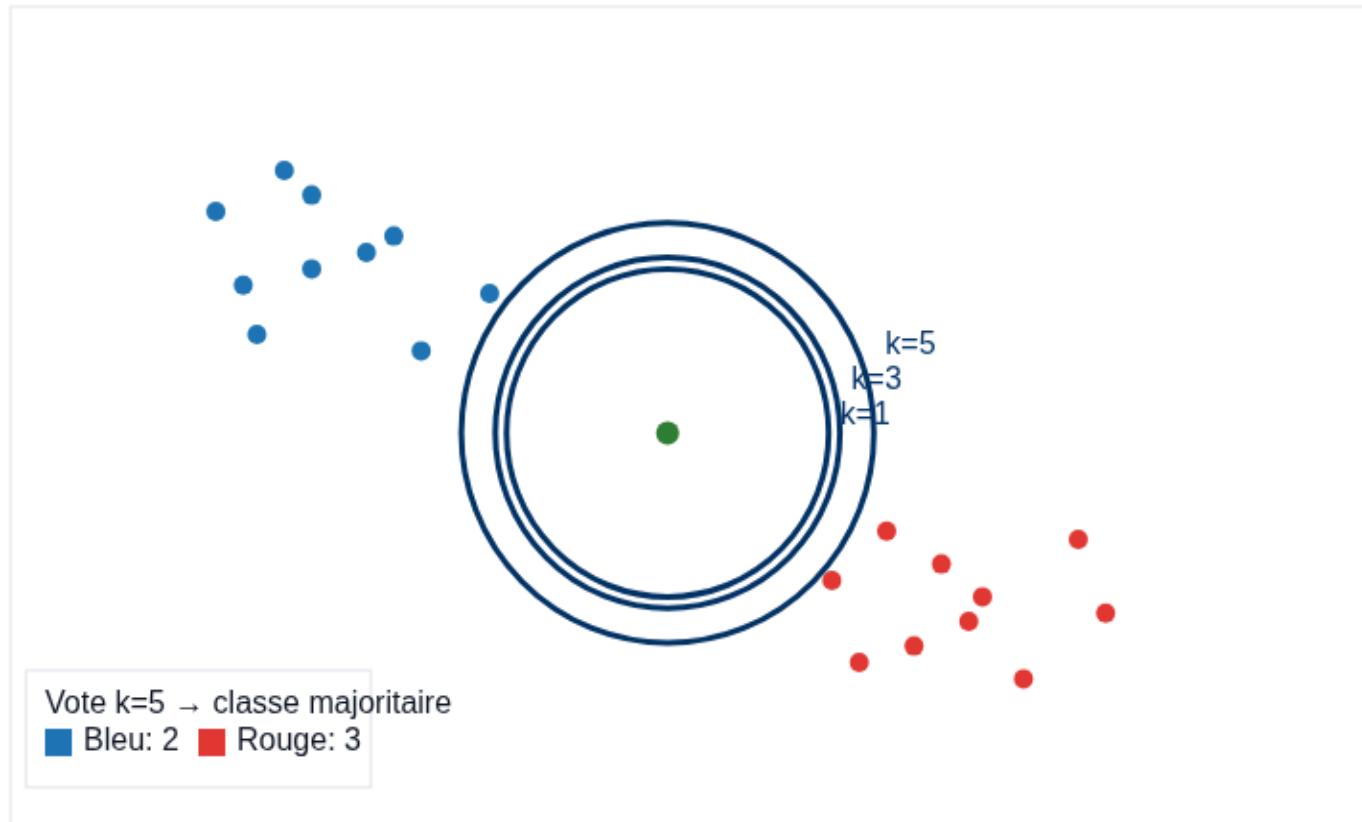
Interprétation métier

Précision élevée \Rightarrow peu de faux positifs. Rappel moyen \Rightarrow des positifs manqués.

En **détection de fraude**, on privilégie souvent un rappel élevé pour limiter les fraudes non détectées, quitte à gérer plus d'alertes (FP) via revue humaine.

KNN – Principe (intuition géométrique)

Nuage de points 2D – classes et voisins autour du point à classer



● Classe 0 (bleu) ● Classe 1 (rouge) ● Nouveau point (vert) ● Cercles $k=1,3,5$

Idée clé

Décision par vote des k plus proches voisins autour du point à classer. La classe majoritaire l'emporte.

Distances

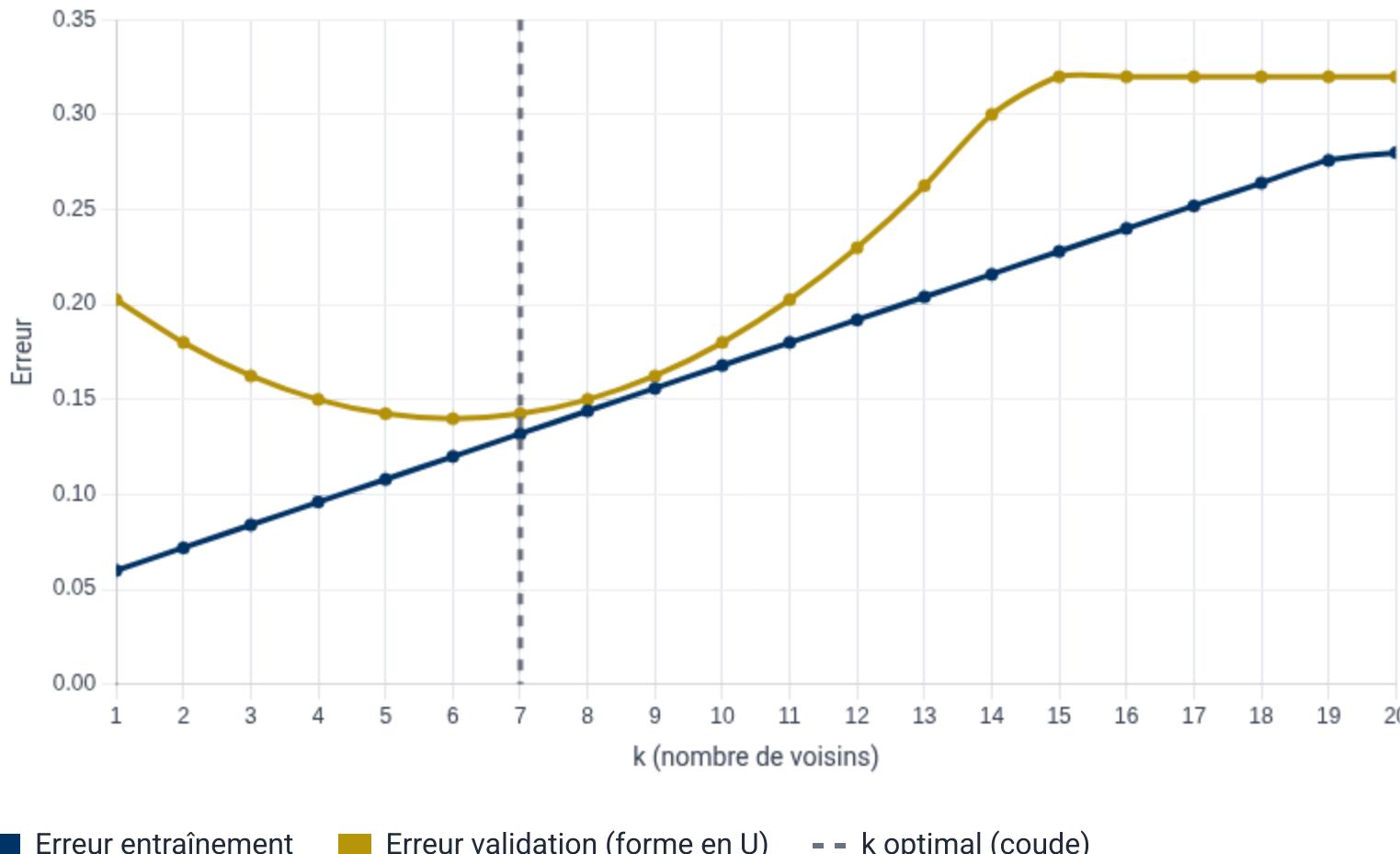
- Euclidienne: $\sqrt{((x_1-x_2)^2 + (y_1-y_2)^2)}$
- Manhattan: $|x_1-x_2| + |y_1-y_2|$

Bonnes pratiques

- Choisir k via validation (ex.: courbe erreur vs k).
- **Normaliser** les features (échelles comparables).
- Poids par distance (option) pour voisins proches.

KNN – Choix de k (erreur train/validation)

Courbes d'erreur en fonction de k (1 à 20). Méthode du « coude »: minimiser l'erreur de validation.



Guides de choix de k

k petit (p.ex. 1–3)

Variance élevée, surapprentissage. Erreur train très basse, validation plus élevée.

k grand (p.ex. ≥ 15)

Biais élevé, sous-apprentissage. Les deux erreurs convergent vers un niveau plus haut.

Méthode du coude

Choisir le k minimisant l'erreur de validation (souvent forme en U).

Recommandation pratique

Préférer des k impairs (3, 5, 7) pour éviter les égalités de vote.

Axes: X = k (1–20) • Y = Erreur (0–1)

KNN – Code Python (classification)



Extrait de code – Pipeline KNN avec normalisation

```
# Imports
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import numpy as np

# Jeu de données jouet (exemple)
X = np.array([[1.2, 0.7], [0.3, 1.1], [1.0, 0.8], [0.1, 0.2], [0.5, 0.4], [1.3, 1.0]])
y = np.array([1, 1, 1, 0, 0, 1])

# Division des données
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42, stratify=y)

# Normalisation (obligatoire pour KNN)
scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Modèle KNN (k=5, distance euclidienne)
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean').fit(X_train_scaled, y_train)

# Prédictions et évaluation
y_pred = knn.predict(X_test_scaled)
print(f'Accuracy: {accuracy_score(y_test, y_pred):.2%}')
print(classification_report(y_test, y_pred, zero_division=0))
```

Astuce: appliquer StandardScaler après le train_test_split et seulement avec fit sur X_train.

Points clés

Normalisation indispensable (KNN dépend des distances).

Choisir un **k** impair (3/5/7) pour éviter les égalités.

Distances: euclidienne, Manhattan; tester au besoin.

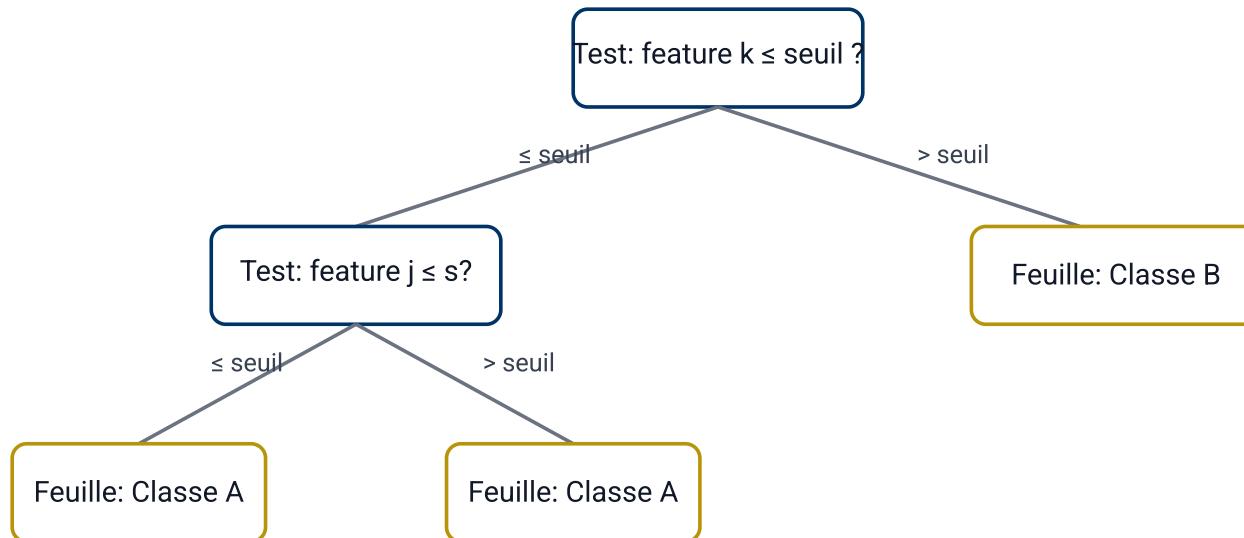
Évaluer avec **accuracy** + **classification_report**.

Bonnes pratiques

- Utiliser validation croisée pour choisir k.
- Gérer le déséquilibre de classes (poids, métriques adaptées).

Arbre de décision – Structure

Schéma hiérarchique – nœud racine, branches (\leq seuil / $>$ seuil), nœuds internes, feuilles (prédictions)



Nœud de décision (test sur une feature)

Feuille (prédiction finale)

— Branche avec condition (\leq seuil / $>$ seuil)

Règles interprétables (IF-THEN)

- IF feature $k \leq$ seuil AND feature $j > s$ THEN Classe A.
- Chaque chemin de la racine à une feuille définit une règle métier.

Profondeur & critères d'arrêt

- **max_depth** : limite la profondeur de l'arbre (contrôle la complexité).
- **min_samples_split** : nb. min. d'échantillons pour scinder un nœud.
- **min_samples_leaf** : nb. min. d'échantillons par feuille.

Bonnes pratiques

- Privilégier l'**interprétabilité** en contexte métier (finance/gestion).
- Éviter l'**overfitting** via contraintes et **élagage (pruning)**.



Critères de pureté

Indice de Gini

$$\text{Gini}(S) = 1 - \sum^i p^i 2$$

où p^i est la proportion de la classe i dans S .

Références rapides

- Gini = 0 → nœud **pur**
- Gini = 0,5 → binaire 50/50

Entropie de Shannon

$$H(S) = - \sum^i p^i \log^2(p^i)$$

- $H = 0 \rightarrow$ nœud **pur**
- $H = 1 \rightarrow$ binaire équilibré

Échelle en **bits**.



Gain d'information (choix du meilleur split)

$$\text{Gain}(S, A) = H(S) - \sum^v (|S^v| / |S|) \cdot H(S^v)$$

Objectif : **maximiser** le gain pour obtenir des nœuds plus **purs**.

Guidelines

- Gini → calcul **rapide**
- Entropie → plus **sensible** aux changements



Conseil pratique

Dans la pratique, Gini et Entropie conduisent souvent à des splits similaires. Choisir l'un ou l'autre selon la vitesse souhaitée et la sensibilité voulue.

Avantages

- Interprétables et visuels
Règles IF/THEN faciles à communiquer.
- Gèrent relations non-linéaires naturellement
Découpage récursif de l'espace des features.
- Peu de preprocessing requis
Pas besoin de normalisation; gèrent NA par stratégies.
- Numériques et catégorielles
Fonctionnent avec types mixtes (via splits).
- Déetectent interactions automatiquement
Les splits successifs modélisent des interactions.

Limites

- Overfitting si non contraints (profondeur)
Nécessite *max_depth*, *min_samples_leaf*.
- Instables
Petite variation des données → arbre différent.
- Performances faibles seuls
Meilleurs en ensembles (Random Forest, Boosting).
- Biais vers features à nombreuses valeurs
Préférer critères ou pénalités adaptés.
- Zones de décision rectangulaires
Frontières axis-aligned, parfois trop rigides.

Arbre de décision – Code Python

Décision binaire avec contraintes anti-overfitting

```
# Imports principaux
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Hypothèse : X (features, DataFrame) et y (labels, Series) existent déjà
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

# Modèle avec contraintes pour limiter l'overfitting
clf = DecisionTreeClassifier(
    max_depth=4, min_samples_split=20,
    min_samples_leaf=10, criterion='gini', random_state=42
).fit(X_train, y_train)

# Évaluation
print(f'Accuracy train: {clf.score(X_train, y_train):.2%}')
print(f'Accuracy test: {clf.score(X_test, y_test):.2%}')

# Visualisation (optionnel)
# plt.figure(figsize=(12, 8))
# plot_tree(clf, feature_names=X.columns, class_names=['0', '1'], filled=True)
# plt.show()

# Importance des variables
print(clf.feature_importances_)
```

Astuce: fixer random_state pour la reproductibilité et limiter la max_depth pour réduire l'overfitting.

Bonnes pratiques

- Contraindre l'arbre: max_depth, min_samples_leaf.
- Comparer Accuracy train vs test (détection overfitting).
- Interpréter via feature_importances_ et l'arbre.
- Stratifier le split si classes déséquilibrées.

Quand l'utiliser ?

Données tabulaires, besoin d'interprétabilité rapide, baseline avant ensembles (Random Forest, Gradient Boosting).

Random Forest – Principe du bagging



Principe en bref

Échantillonner (bootstrap) → Entraîner plusieurs arbres en parallèle → Agréger (vote/moyenne).

1 Bootstrap

Créer des **échantillons avec remise à part** du dataset d'origine.

Chaque échantillon contient des duplications et omet des points (out-of-bag).

But: injecter de la diversité entre les modèles.

2 Entraîner des arbres

Former **B arbres indépendants en parallèle** (typ. 100–500).

Sous-échantillonnage de **features** à chaque split pour plus de diversité.

Clés: n_estimators, max_features, profondeur, min_samples_leaf.

3 Agrégation

Vote majoritaire (classification) ou **moyenne** (régression).

Combine les prédictions des arbres pour stabiliser le résultat.

OOB score: évalue la performance sans set de validation séparé.

Bénéfices – réduction de la **variance**, robustesse au bruit, moins de surapprentissage que l'arbre seul.

Différence vs boosting – apprentissage **parallèle** (bagging) et non **séquentiel**.

Random Forest – Architecture d'ensemble

Schéma de flux



Points clés (hyperparamètres)

- **n_estimators** : nombre d'arbres (B).
- **max_features** : nb. de features à chaque split. → Vote majoritaire (clf) /
- **max_depth / min_samples_leaf** contrôlent la profondeur et l'overfitting.
- **bootstrap=True** : active l'échantillonage avec remise.

Bonnes pratiques

Stratifier si classes déséquilibrées, calibrer les probabilités si nécessaire.

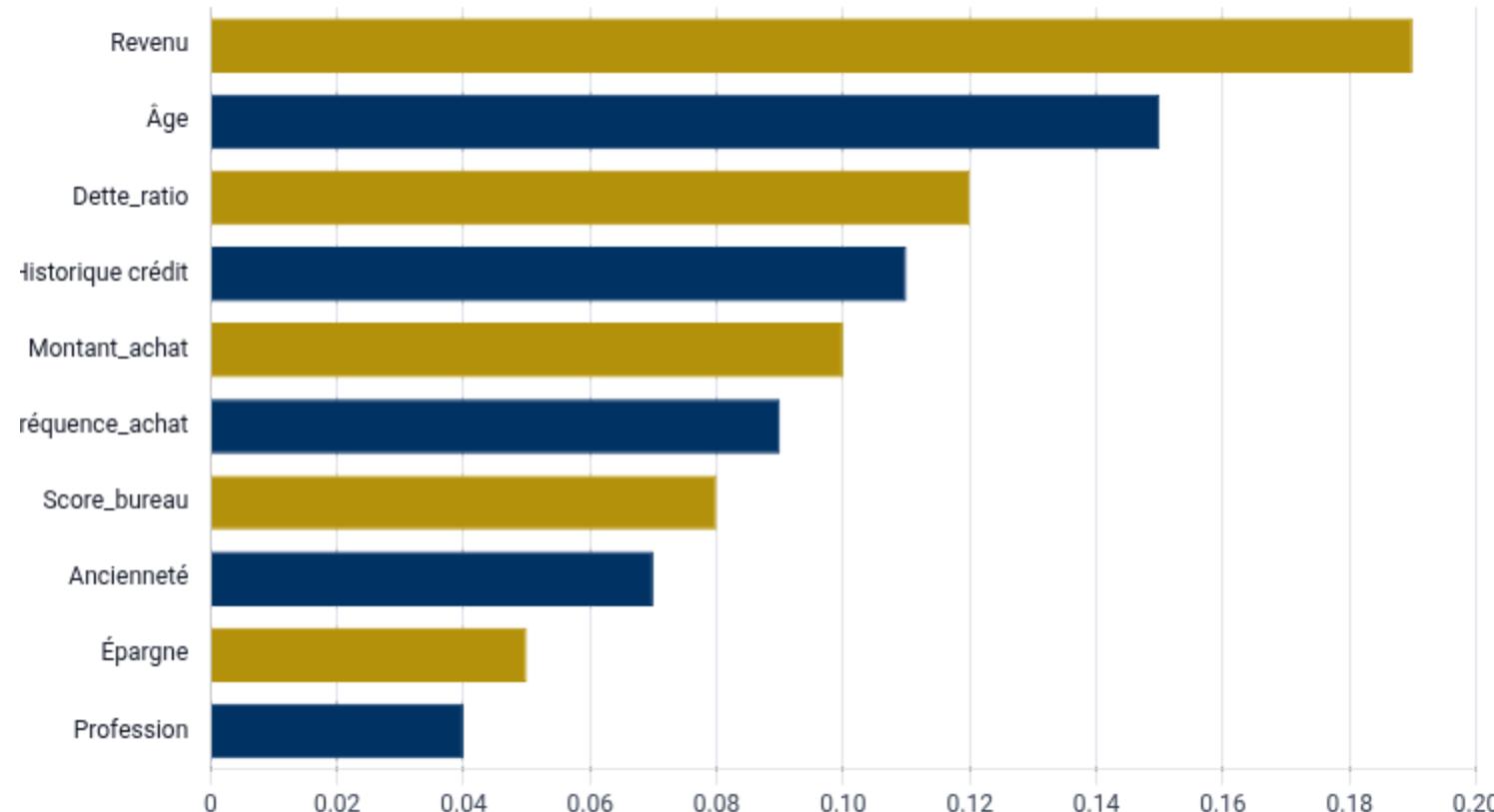
Résumé

Random Forest = **bagging** d'arbres + **sous-échantillonage** de features → robuste et performant sur données tabulaires.

Random Forest – Importance des variables

Top 10 variables les plus importantes

Méthode affichée : Importance Gini



Lecture : plus la barre est longue, plus la variable contribue à réduire l'impureté moyenne des noeuds (sur l'ensemble des arbres).

Astuce: pour éviter une lecture biaisée, vérifier aussi **l'importance par permutation** (sensibilité du score lorsque l'on permute une feature).

Types d'importance

- **Gini (réduction d'impureté)** : somme des gains d'impureté pondérés sur tous les splits.
- **Permutation** : baisse du score (ex. AUC) quand on permute une feature – *plus fiable avec corrélations*.

Mises en garde

- Biais possible vers les variables à **forte cardinalité** (beaucoup de modalités).
- Effets des **corrélations** entre variables (importances “partagées”).
- Toujours **valider métier** et comparer Gini vs permutation.

Bonnes pratiques

- Évaluer l'importance sur **validation croisée**.
- Utiliser des **métriques adaptées** (AUC, F1, Precision@K).
- Compléter par des méthodes XAI (PDP, SHAP).

Random Forest – Code Python (comparaison)



Comparaison Arbre de décision vs Random Forest (classification)

```
# Imports
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Supposons X, y déjà préparés (features/labels) ; split train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# 1) Arbre simple (risque d'overfitting si non contraint)
clf_tree = DecisionTreeClassifier(max_depth=4, random_state=42).fit(X_train, y_train)

# 2) Random Forest (bagging + sous-échantillonnage de features)
rf = RandomForestClassifier(
    n_estimators=300, max_depth=None, max_features='sqrt',
    min_samples_leaf=5, random_state=42
).fit(X_train, y_train)

# Évaluation comparée
print(f'Arbre simple - Train: {clf_tree.score(X_train, y_train):.2%}')
print(f'Arbre simple - Test: {clf_tree.score(X_test, y_test):.2%}')
print(f'Random Forest - Train: {rf.score(X_train, y_train):.2%}')
print(f'Random Forest - Test: {rf.score(X_test, y_test):.2%}')

# Importances de variables (top 5)
idx_top5 = rf.feature_importances_.argsort()[-5:][::-1]
print('Top 5 features:', idx_top5)
```

Note: assurez-vous que X, y et le train_test_split sont définis en amont.

À retenir

RF réduit la **variance** vs un arbre unique.

Bon **baseline** pour données tabulaires.

Interprétation via feature_importances_.

Hyperparamètres clés

- n_estimators, max_depth, max_features
- min_samples_leaf, random_state

XGB

Idée générale

Somme de **faibles apprenants** (arbres) ajustés **séquentiellement** pour corriger les erreurs des modèles précédents.



Forces de XGBoost

- ✓ Régularisation **L1/L2** intégrée (contrôle complexité).
- ✓ **Parallélisme** efficace et hautes performances.
- ✓ Gestion native des **valeurs manquantes** (NA).
- ✓ **Early stopping** pour éviter l'overfitting.

Comparaison rapide

Bagging (Random Forest)
→ apprentissage **parallèle**, réduit la **variance**.

Boosting (XGBoost) → apprentissage **séquentiel**, réduit le **biais**.



Hyperparamètres clés

- n_estimators** — nombre d'arbres.
- max_depth** — profondeur des arbres (3–10).
- colsample_bytree** — fraction des features.
- Astuce: commencer avec **learning_rate** bas et **n_estimators** plus élevé; surveiller la courbe de validation (**early_stopping_rounds**).
- learning_rate** — shrinkage (≈ 0.01 – 0.3).
- subsample** — fraction des données (0.5–1).
- À ajuster via validation croisée.

LGBM

Idée générale

Croissance **leaf-wise**: ajoute la feuille au **gain maximal** (vs *level-wise*), ce qui accélère l'apprentissage et améliore le score avec moins d'arbres.



Forces de LightGBM

- ✓ **Très rapide** sur grands datasets (leaf-wise + optimisations).
- ✓ Excellentes performances sur **données tabulaires**.
- ✓ **Mémoire efficace** grâce à GOSS/EFB.
- ⚠ **Attention**: risque d'**overfitting leaf-wise** si non contraint.

Techniques clés

GOSS (Gradient-based One-Side Sampling): échantillonne préférentiellement les points aux plus grands résidus pour **réduire les données** tout en conservant l'information.

EFB (Exclusive Feature Bundling): regroupe des **features mutuellement exclusives** afin de **réduire la dimension** sans perte majeure.



Hyperparamètres clés

- | | |
|--|--|
| num_leaves – nb max de feuilles ($\approx 31-255$). | max_depth – profondeur (-1 = illimité). |
| learning_rate – 0.01–0.1. | n_estimators – nombre d'arbres. |
| min_child_samples – min. d'exemples par feuille. | Ajuster via validation croisée . |
- Astuces: combiner **learning_rate** bas + **n_estimators** élevé, contraindre **num_leaves/max_depth**, utiliser **early_stopping** sur un set de validation.

Idée générale

CAT

CatBoost est un algorithme de boosting par arbres optimisé pour les **variables catégorielles**: encodage natif (ordered target statistics) et **ordered boosting** pour éviter la fuite de cible.

Pourquoi CatBoost ?

Catégorielles natives sans preprocessing lourd.

Anti-leakage via ordered boosting.



Forces de CatBoost

- ✓ **Encodage catégoriel natif** (ordered target statistics).
- ✓ **Ordered boosting**: réduit la **fuite de cible**.
- ✓ Bonnes performances **sans réglages fins**.
- ✓ **Haute cardinalité** gérée automatiquement.



Hyperparamètres clés

depth – profondeur arbres (4–10).

learning_rate – 0.01–0.3.

iterations – 100–1000.

l2_leaf_reg – régularisation L2.

Usage recommandé: **datasets riches en variables catégorielles**, peu de feature engineering requis. Ajuster **learning_rate** et **iterations** via validation croisée.

Idée générale



Construire une **somme de faibles apprenants** (petits arbres) **séquentiellement**, chaque nouveau modèle corrigant les erreurs des précédents.

1 Initialiser

Modèle faible initial (ex. **arbre peu profond** ou **moyenne**).

Point de départ de l'ensemble.

2 Calculer les erreurs

Résidus / pseudo-gradients du modèle courant (ce qu'il n'explique pas).

Cible d'apprentissage du prochain arbre.

3 Ajuster un nouvel arbre

Apprendre un **arbre faible** pour prédire ces résidus.

Il « corrige » les erreurs précédentes.

4 Combiner (shrinkage)

Mettre à jour la prédiction avec un **learning_rate** (ex. 0,01–0,30) pour **pondérer** l'apport du nouvel arbre.

Régularisation par *shrinkage*.

5 Répéter

Répéter les étapes 2–4 pour une **séquence d'arbres** jusqu'à convergence ou **early stopping**.

Nombre d'arbres = $n_estimators$.

✓ Effets attendus

Réduction du biais (amélioration progressive) et **contrôle de la variance** via régularisation et *learning_rate*.

Vs. Bagging

Boosting = apprentissage **séquentiel** (correction d'erreurs) • **Bagging** = apprentissage **parallèle** (moyenne/vote).

Boosting – Code Python comparatif



Comparer XGBoost, LightGBM et CatBoost (classification binaire)

```
# Installation (si nécessaire) :  
# pip install xgboost lightgbm catboost  
  
from xgboost import XGBClassifier  
from lightgbm import LGBMClassifier  
from catboost import CatBoostClassifier  
from sklearn.model_selection import train_test_split  
from sklearn.metrics import roc_auc_score  
  
# Hypothèse : X, y déjà préparés (features et cible binaire)  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)  
  
# Définir les modèles de boosting  
models = {  
    'XGBoost': XGBClassifier(n_estimators=500, learning_rate=0.05, max_depth=4,  
                            subsample=0.8, colsample_bytree=0.8, random_state=42, eval_metric='auc'),  
    'LightGBM': LGBMClassifier(n_estimators=500, learning_rate=0.05, num_leaves=31, random_state=42),  
    'CatBoost': CatBoostClassifier(iterations=500, learning_rate=0.05, depth=6, verbose=0, random_state=42)  
}  
  
# Entraînement et évaluation AUC  
for name, model in models.items():  
    model.fit(X_train, y_train)  
    y_pred_proba = model.predict_proba(X_test)[:, 1]  
    auc = roc_auc_score(y_test, y_pred_proba)  
    print(f'{name} - AUC: {auc:.4f}')
```

Astuce: utiliser learning_rate faible + n_estimators élevé avec early stopping pour éviter l'overfitting.

Points clés

Mesure: AUC ROC pour classes déséquilibrées.

XGBoost: robuste, réglages fins.

LightGBM: très rapide (leaf-wise).

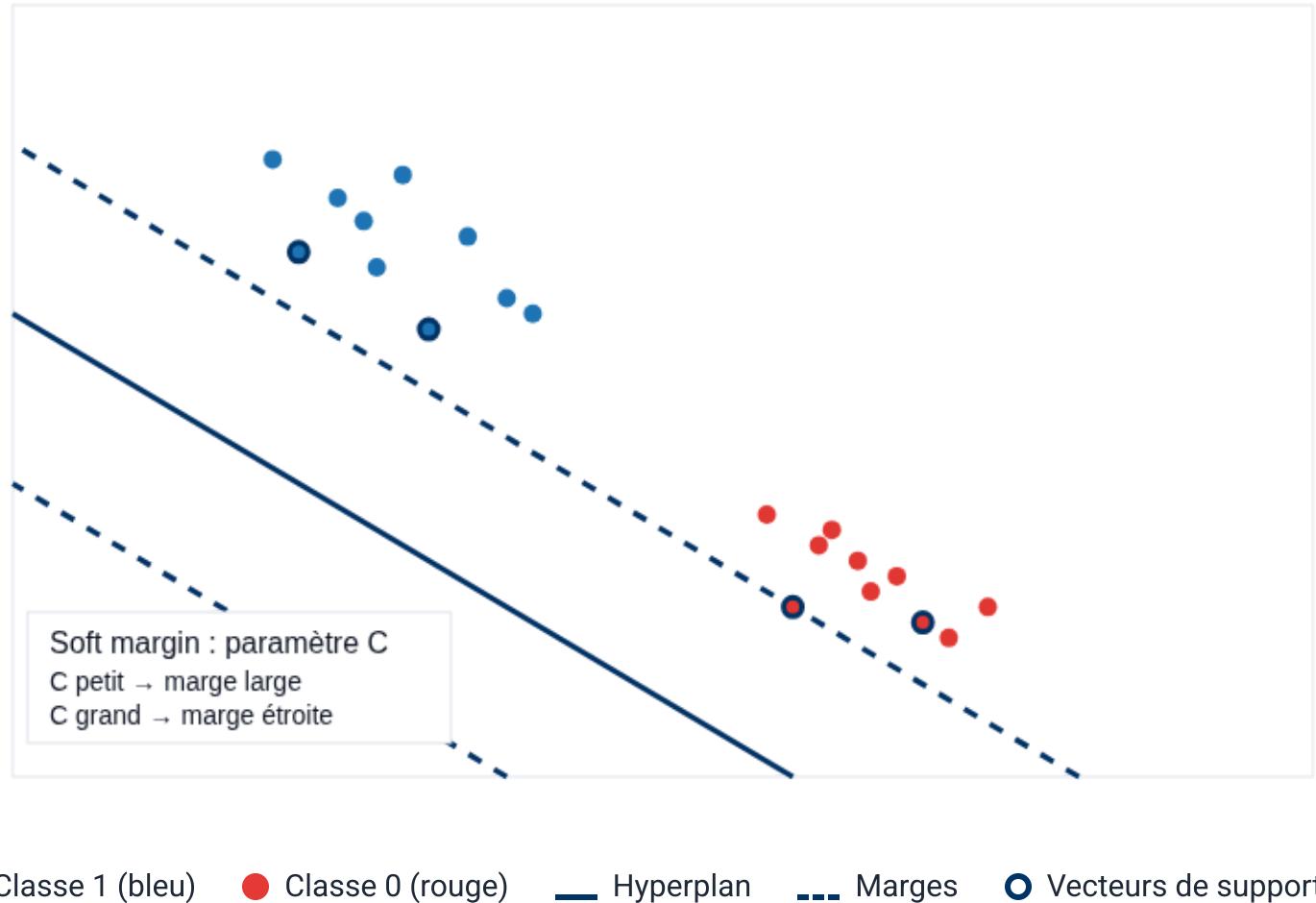
CatBoost: gère bien les catégorielles.

Bonnes pratiques

- Standardiser/encoder avant fit (si nécessaire).
- Gérer déséquilibre: scale_pos_weight / class_weight.
- Validation croisée + grille/opt. bayésienne.

SVM – Hyperplan et marge maximale

Hyperplan séparateur, marges parallèles et vecteurs de support (2 classes)



Concept

Trouver l'hyperplan qui **maximise la marge** entre les classes. Les **vecteurs de support** (points les plus proches) déterminent la frontière.

Soft margin (paramètre C)

- C petit \rightarrow marge **large**, plus tolérante aux erreurs.
- C grand \rightarrow marge **étroite**, moins d'erreurs mais risque d'overfitting.

Repères visuels

- Ligne pleine = **hyperplan**.
- Pointillés = **marges** parallèles.
- Cercles épais = **vecteurs de support**.



Kernel Linéaire

Formule: $K(x, x') = x \cdot x'$

Usage: données linéairement séparables; rapide et interprétable.



Kernel Polynomial

Formule: $K(x, x') = (\gamma x \cdot x' + r)^d$

Paramètres: degré $d = 2, 3$ (interactions non linéaires).



Kernel RBF (Gaussien)

Formule: $K(x, x') = \exp(-\gamma \|x - x'\|^2)$

Rôle de γ : contrôle le rayon d'influence (plus γ est grand, plus la zone est réduite).



Kernel Sigmoïde

Formule: $K(x, x') = \tanh(\gamma x \cdot x' + r)$

Lien: s'apparente aux réseaux de neurones (activation \tanh).

Bonnes pratiques

Standardiser les features (moyenne 0, σ 1) avant SVM.

Choisir le kernel selon la complexité/non-linéarité des données.

Grid search/validation croisée pour C , γ (RBF) et d (polynomial).



Formules clés (classification binaire)

Accuracy (Exactitude)

$$\text{Accuracy} = (\text{TP} + \text{TN}) / (\text{TP} + \text{FP} + \text{TN} + \text{FN})$$

Taux global de bonnes classifications.

Precision (Précision)

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

Fiabilité des prédictions positives.

Recall (Rappel, Sensibilité)

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

Couverture des vrais positifs détectés.



Interprétations métier et cas de données déséquilibrées

Accuracy = qualité globale, mais trompeuse sur jeux déséquilibrés.

Ex.: fraude = 1% → prédire "pas fraude" donne 99% d'accuracy mais 0 détection.

Conclusion: sur données déséquilibrées, se concentrer sur Precision/Recall (et F1), pas seulement sur l'Accuracy.

Precision = coût des faux positifs (alertes inutiles, surcharge opérationnelle).

À privilégier pour limiter fausses alertes (ex. fraude, prospection).

Recall = coût des faux négatifs (cas manqués, risques non détectés).

À privilégier quand manquer un positif est coûteux (ex. dépistage, conformité).

Métriques – F1-Score et trade-off

Formules

Moyenne harmonique

F1-Score

$$F1 = 2 \cdot (\text{Précision} \cdot \text{Rappel}) / (\text{Précision} + \text{Rappel})$$

Mesure unique équilibrant Précision et Rappel (utile en données déséquilibrées).

F- β (pondération du rappel)

$$F\beta = (1+\beta^2) \cdot (\text{Précision} \cdot \text{Rappel}) / (\beta^2 \cdot \text{Précision} + \text{Rappel})$$

$\beta < 1$ favorise la Précision • $\beta > 1$ favorise le Rappel

- Choisir β selon le coût métier des faux positifs (FP) vs faux négatifs (FN).

Pratique métier

Ajuster le seuil de décision

Adapter le seuil en fonction des coûts FP/FN et des objectifs (alertes, priorisation).

Courbe Precision-Recall

Identifier le « meilleur » compromis pour un rappel cible et une précision acceptable.

Calibration des probabilités

Platt scaling ou régression isotone pour des probabilités bien calibrées.

Fraude

Privilégier la Précision (limiter FP) et utiliser F1/F β avec $\beta < 1$.

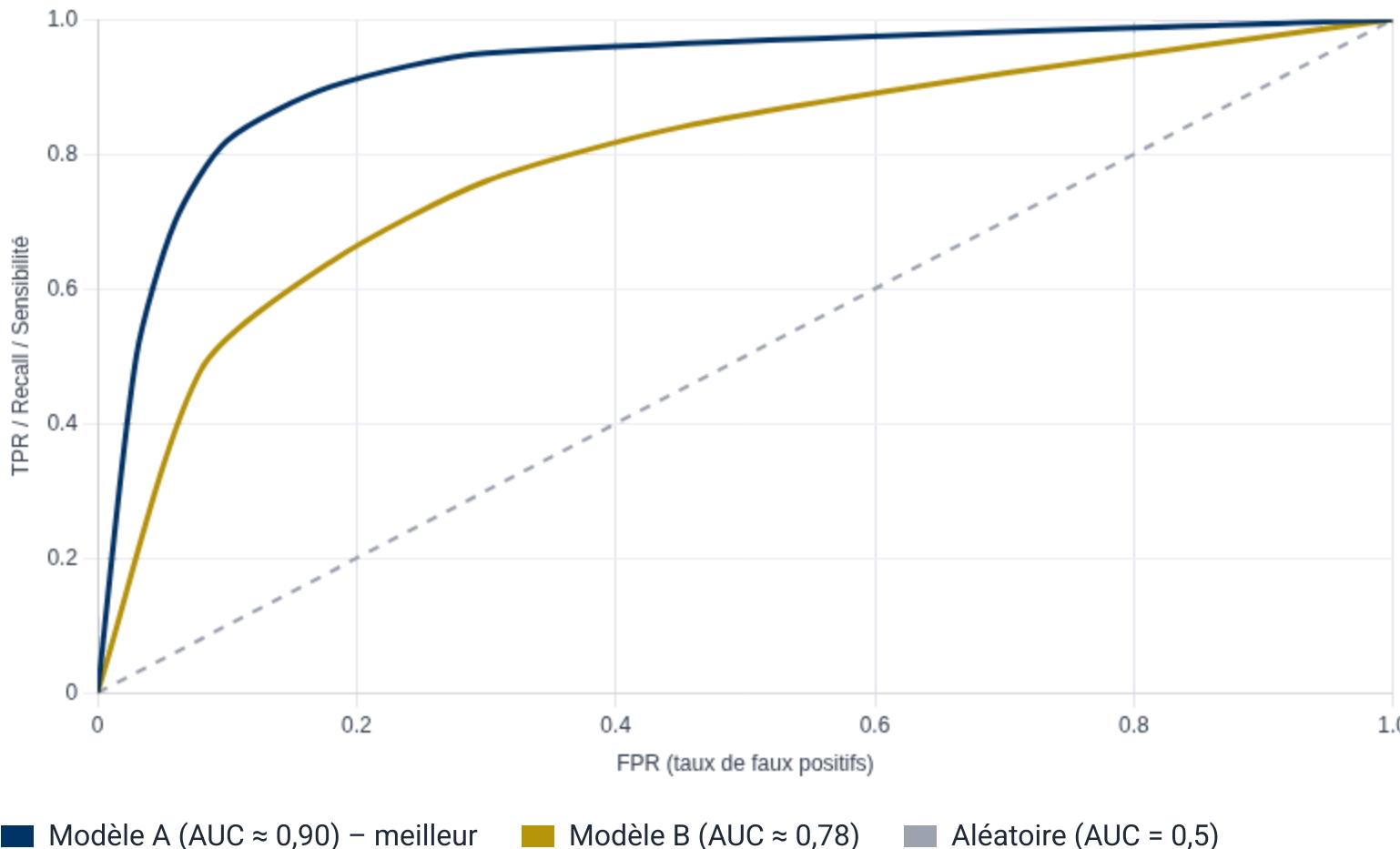
Médical (dépistage)

Privilégier le Rappel (limiter FN) et utiliser F β avec $\beta > 1$.

- À retenir : un F1 = 63% est souvent plus informatif que l'Accuracy en données déséquilibrées.

ROC & AUC – Interprétation

Axes : TPR (Recall, Sensibilité) sur Y • FPR (taux faux positifs) sur X



Lecture et bonnes pratiques

Courbe ROC

Performance à tous les seuils (0 → 1).

Diagonale = modèle aléatoire.

AUC (Area Under Curve)

Probabilité qu'un positif soit classé au-dessus d'un négatif.

Échelle: 1 parfait • 0,9–1 excellent • 0,8–0,9 bon • 0,7–0,8 acceptable • <0,7 faible

Choix métier

Comparer par AUC puis choisir le seuil selon le trade-off TPR/FPR (coûts d'alerte vs cas manqués).

Couleurs ENCG : Bleu #003366 • Doré #b7950b

Quand utiliser quelle métrique ?



Choix guidé par les coûts des faux positifs/faux négatifs et les objectifs métiers



Fraude bancaire ($\approx 0,1\%$)

Jeu très déséquilibré: prioriser les alertes pertinentes dans le top K.

Precision@K

Recall@K

AUC

Seuil selon coûts



Diagnostic médical (dépistage)

Minimiser les faux négatifs; accepter des faux positifs pour examens.

Recall (Sensibilité)

Seuil bas

PR curve



Scoring crédit

Discriminer et calibrer; seuil selon l'appétit au risque.

ROC-AUC

KS

Brier score

Seuil risque



Système de recommandation

Pertinence top-K et diversité des résultats.

MAP@K

NDCG

Recall@K

Validation croisée K-fold – Schéma (ex. K = 5)

Rotation des plis (5 itérations)

Entraînement Validation



Score final

Moyenne des **K** scores \pm écart-type (robuste et reproduitible).

Bonnes pratiques (K-fold)

- Utiliser la **stratification** pour classes déséquilibrées (classification).
- Éviter la **fuite de données** : normaliser/encoder **après** le split (fit sur train uniquement).
- Activer le **shuffle** et fixer **random_state** pour la reproductibilité.
- Sélectionner une **métrique adaptée** (ex. ROC-AUC, F1, RMSE selon le cas d'usage).

Rappel sur les rôles

Le K-fold est utilisé sur l'**ensemble d'entraînement** pour sélectionner/évaluer les modèles. Le **test set** reste intouché jusqu'à l'évaluation finale.

Validation croisée – Code sklearn



Extrait de code – K-fold stratifié, AUC et métriques multiples

```
# Imports
from sklearn.model_selection import cross_val_score, StratifiedKFold, cross_validate
from sklearn.ensemble import RandomForestClassifier
import numpy as np

# Stratégie K-fold STRATIFIÉE
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Modèle
rf = RandomForestClassifier(n_estimators=300, max_depth=10, random_state=42)

# Cross-validation avec une métrique (AUC ROC)
scores_auc = cross_val_score(rf, X, y, cv=skf, scoring='roc_auc')
print(f'AUC moyenne: {scores_auc.mean():.4f} ± {scores_auc.std():.4f} ')

# Cross-validation avec plusieurs métriques
scoring = {'auc': 'roc_auc', 'f1': 'f1', 'precision': 'precision'}
results = cross_validate(rf, X, y, cv=skf, scoring=scoring)

for metric, vals in results.items():
    if metric.startswith('test_'):
        print(f'{metric}: {np.mean(vals):.4f}'')
```

Points clés

AUC moyenne $\pm \sigma$ résume la performance globale.

Plusieurs métriques via `cross_validate`.

Standardiser/encoder **après** le split (pipeline recommandé).

Bonnes pratiques

- Stratification pour préserver les proportions de classes.
- Répéter CV (différents seeds) pour stabilité.
- Sélectionner la **métrique métier** (ex. AUC pour scoring).

Astuce: utilisez `StratifiedKFold` pour classes déséquilibrées et évitez toute fuite de données.

Échelles de features – formules et usages

↔ MinMaxScaler

$$x' = (x - \min(x)) / (\max(x) - \min(x)) \in [0,1]$$

Préserve la forme de la distribution, sensible aux outliers. Usage: réseaux de neurones (sigmoïde/tanh).

.StandardScaler (Z-score)

$$z = (x - \mu) / \sigma$$

Moyenne 0, écart-type 1. Robuste si distribution \sim normale. Usage: KNN, SVM, régression régularisée (L1/L2).

RobustScaler

Utilise médiane et IQR (Q3-Q1)

Moins sensible aux outliers. Utile pour distributions asymétriques ou avec valeurs extrêmes.

✓ Quand normaliser ?

- Algos sensibles à l'échelle: KNN, SVM, réseaux de neurones
- Régularisation L1/L2 (régressions pénalisées)

- Quand pas nécessaire ?

- Arbres de décision, Random Forest, Gradient Boosting

⚠ Attention – Data leakage

Toujours **fit** le scaler sur le **train uniquement**, puis **transform** train/validation/test. Intégrer le scaler dans un *pipeline* (sklearn) pour éviter toute fuite.

Encodage des variables catégorielles



Techniques

One-Hot Encoding

Créer des colonnes binaires (0/1) pour chaque modalité; augmente la dimension.

Label / Ordinal Encoding

Assigner des entiers (0,1,2, ...); risque d'introduire un ordre artificiel entre catégories.

Frequency Encoding

Remplacer chaque modalité par sa fréquence d'apparition dans les données.

Target Encoding

Remplacer par la moyenne de la cible par modalité; puissant mais attention au surapprentissage.

Bonnes pratiques

Encoder après le split train/test

Éviter toute fuite de données (data leakage) depuis le test vers l'entraînement.

Régulariser le Target Encoding

Utiliser K-fold ou smoothing pour limiter l'overfitting lors du calcul des moyennes cibles.

Gérer la forte cardinalité

Privilégier frequency/target encoding ou regrouper les modalités rares.

Choisir la méthode selon le contexte

One-Hot si < 10–15 modalités; éviter Label si catégories non ordonnées (ordre artificiel).

Outils: scikit-learn *OneHotEncoder*, *LabelEncoder*, *category_encoders* (*TargetEncoder*).

Étude de cas – Scoring crédit (problématique & données)



Objectif métier

Estimer la probabilité de défaut (PD) pour décider l'octroi; minimiser les pertes tout en maximisant les acceptations rentables.

PD

ROI



Données

German Credit (Kaggle/UCI), 1000 clients, 20 variables: revenus, emploi, dettes, historique crédit, âge, durée du prêt. ≈30% défaut.

1000 clients

20 features

30% défaut



Contraintes

Classes déséquilibrées (30/70); équité et non-discrimination (attributs protégés); explicabilité RGPD (droit à l'explication); calibration des probabilités pour le pricing.

Équité

RGPD

Déséquilibre



KPI métier

ROC-AUC; KS statistic; Recall@FPR=5%; Brier score (calibration); coût attendu = pertes évitées – opportunités manquées.

AUC

KS

Recall@5% FPR

Brier

Pipeline complet scoring crédit – De l'EDA au déploiement



1 EDA (analyse exploratoire)

- Distributions, corrélations
- Valeurs manquantes, outliers
- Déséquilibre des classes

2 Prétraitement

- Encodage cat. One-Hot/Target
- Normalisation **StandardScaler**
- Feature eng. (ratios, interactions)
- Imputation des NA

3 Modélisation

- Baseline: Logistic Regression
- RF, XGBoost, LightGBM
- Tuning: Grid/Random Search

4 Évaluation

- ROC-AUC, KS, Brier (calibration)
- Learning curves, confusion
- Recall@FPR 5% (fraude/risque)

5 Interprétation

- SHAP: importance des variables
- Analyse FP/FN (erreurs)
- Fairness: attributs protégés

6 Déploiement

- API REST (Flask), seuils
- Monitoring drift, A/B testing
- Re-entraînement périodique



Résultats (exemple) – **AUC=0,78** • **KS=0,42** • **Recall@FPR5%=65%** • **Brier=0,12**

Quiz interactif – Chapitre 3



1

Overfitting vs Underfitting

Quelle est la différence entre surapprentissage et sous-apprentissage ? (Indications : overfitting = modèle trop complexe mémorise le train; underfitting = modèle trop simple ne capte pas les patterns)

2

Precision vs Recall

Dans quel contexte privilégier Precision plutôt que Recall, et inversement ? (Indications : Precision si coût FP élevé ex. spam; Recall si coût FN élevé ex. dépistage médical)

3

Principe du bagging (Random Forest)

Expliquez le bagging et ses bénéfices. (Indications : bootstrap avec remise, arbres en parallèle, vote majoritaire, réduit la variance)

4

Random Forest vs Gradient Boosting

Quelle différence de principe et d'effet (biais/variance) ? (Indications : RF = bagging parallèle ↓ variance; GB = boosting séquentiel corrige erreurs ↓ biais)

5

Pourquoi normaliser pour KNN et SVM ?

Justifiez la normalisation avant KNN/SVM. (Indications : algorithmes sensibles aux échelles; sinon, grandes valeurs dominent distance/marge; normaliser équilibre les features)

Introduction à l'apprentissage non supervisé



- Objectif : découvrir des structures cachées sans étiquettes (cible y inconnue)
- Différence vs supervisé : pas de cible → regroupements, détection d'anomalies, réduction de dimension
- Données d'entrée : X uniquement (features numériques/catégorielles)
- Sorties typiques : clusters (groupes), composantes latentes, scores d'anomalie
- Usages métier : segmentation clients, exploration des données, préparation de features, compression

Cas d'usage concrets du clustering



Segmentation clients

Marketing personnalisé, next-best-offer, pricing différencié par segment.

Personnalisation

NBO



Détection d'anomalies

Fraudes, pannes d'équipements, comportements atypiques.

Fraude

Maintenance



Compression / Réduction

Visualisation haute dimension, accélération des algorithmes, débruitage.

PCA / t-SNE

Vitesse



Organisation de catalogues

Regrouper produits similaires, recommandations par similarité, taxonomie automatique.

Similarité

Recommandation

K-Means – Algorithme (étapes)



Objectif du K-Means : **regrouper** les points en **K clusters** en minimisant la somme des distances intra-cluster (inertie).

1 Initialiser K centroïdes

Choisir K positions de départ (aléatoire ou **k-means++** recommandé pour une meilleure convergence).

Paramètre : **K** (nombre de clusters).

2 Affecter les points

Pour chaque point, trouver le centroïde le plus proche selon la **distance euclidienne** (affectation au cluster).

Mesure : distance L2 minimale.

3 Mettre à jour les centroïdes

Recalculer chaque centroïde comme la **moyenne** des points appartenant au cluster.

Optimise l'inertie intra-cluster.

4 Répéter jusqu'à convergence

Répéter les étapes 2 → 3 jusqu'à **stabilisation** (peu de changements d'affectation) ou itérations maximales.

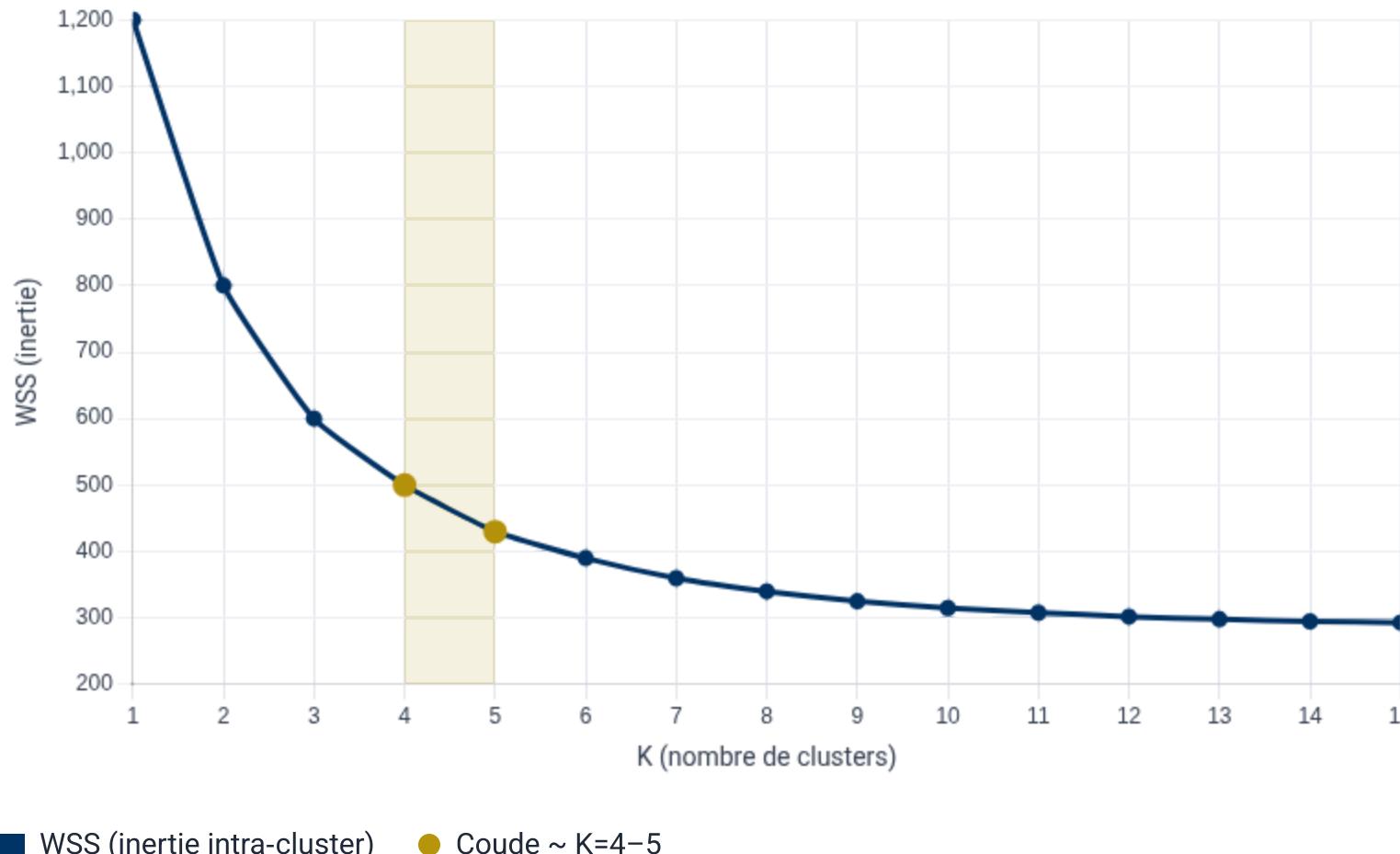
Critères : variation faible d'inertie ou itérations atteintes.

Bonnes pratiques

- **Standardiser** les features (échelles comparables).
- Tester plusieurs valeurs de **K** (méthode du coude, silhouette).
- Utiliser plusieurs initialisations **n_init** (10–50) pour éviter un minimum local.

Choisir K – Méthode du coude

Inertie (WSS) vs K (1 à 15). Courbe décroissante avec « coude » visible vers K \approx 4–5.



Définitions

Inertie (WSS)

Somme des distances intra-cluster au carré.
Décroît quand K augmente.

Méthode du coude

Choisir K au point de cassure où le gain marginal diminue fortement.

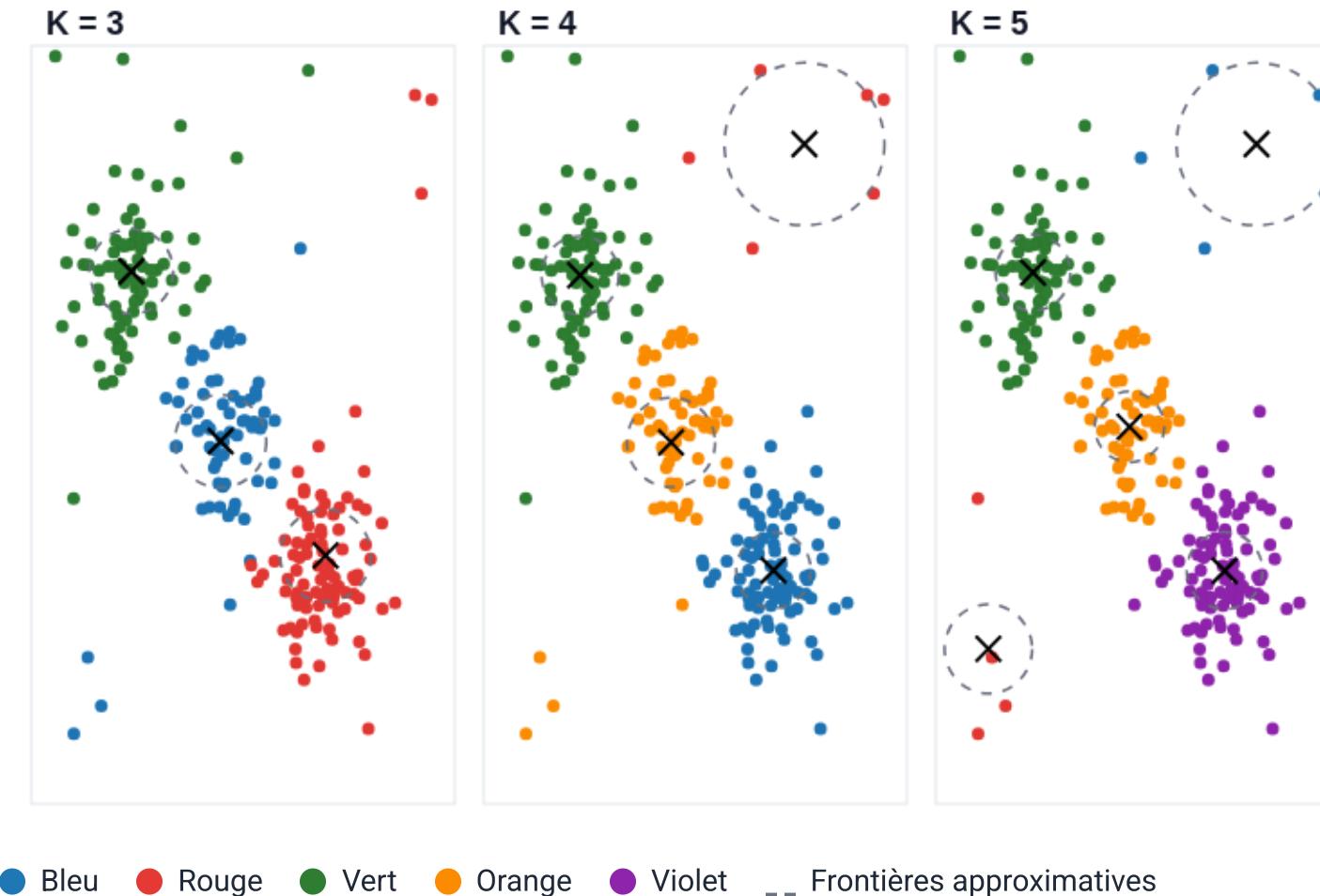
Alternatives

- Silhouette moyenne $\in [-1,1]$, viser $> 0,5$
- Calinski–Harabasz (variance inter/intra), maximiser
- Davies–Bouldin (similarité clusters), minimiser

Bonnes pratiques : standardiser les features, essayer plusieurs initialisations (n_init).

Visualisation des clusters (K=3,4,5)

Trois segmentations côte à côté – K=3, K=4, K=5 (centroïdes « \times » noirs, frontières approximatives en pointillé)



Lecture des visuels

- **K=3** : clusters larges et bien séparés.
- **K=4** : segmentation plus fine, souvent équilibrée.
- **K=5** : risque de *sur-segmentation* (clusters petits).

Compacité & séparation

Compacité = cohésion intra-cluster. Séparation = distance inter-clusters. Chercher un compromis (méthode du coude, silhouette).

Attention à l'échelle

Normalisation requise (features sur mêmes échelles) pour éviter qu'une variable domine la distance euclidienne.

Code Python – K-Means (segmentation clients bancaires)

Extrait de code – Segmentation K-Means (clients banque)

```
# 1) Imports
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# 2) Données clients (âge, revenu, fréquence d'achat, panier moyen)
df = pd.DataFrame({
    'age': [25,35,45,55,30,40,50,60],
    'revenu': [2000,3500,5000,6500,2500,4000,5500,7000],
    'freq_achat': [2,5,8,3,4,7,6,2],
    'panier_moyen': [50,120,200,80,90,150,180,70]
})

# 3) Sélection des features et standardisation (obligatoire pour K-Means)
X = df[['age', 'revenu', 'freq_achat', 'panier_moyen']].values
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4) Entraînement K-Means avec K=4 segments
km = KMeans(n_clusters=4, n_init=10, random_state=42).fit(X_scaled)

# 5) Affectation cluster par client
df['cluster'] = km.labels_

# 6) Centroïdes (re-projetés dans l'espace d'origine) et inertie
centroides = scaler.inverse_transform(km.cluster_centers_)
print('Centroïdes par cluster:')
print(pd.DataFrame(centroides,
    columns=['age', 'revenu', 'freq_achat', 'panier_moyen']))
print('Inertie:', km.inertia_)
```

Astuce: standardiser les variables et tester K via coude/silhouette; utiliser $n_init \geq 10$.

Lecture rapide

Clusters (0–3) : groupes de clients aux profils proches.

Centroïdes : profils moyens par segment (âge, revenu, etc.).

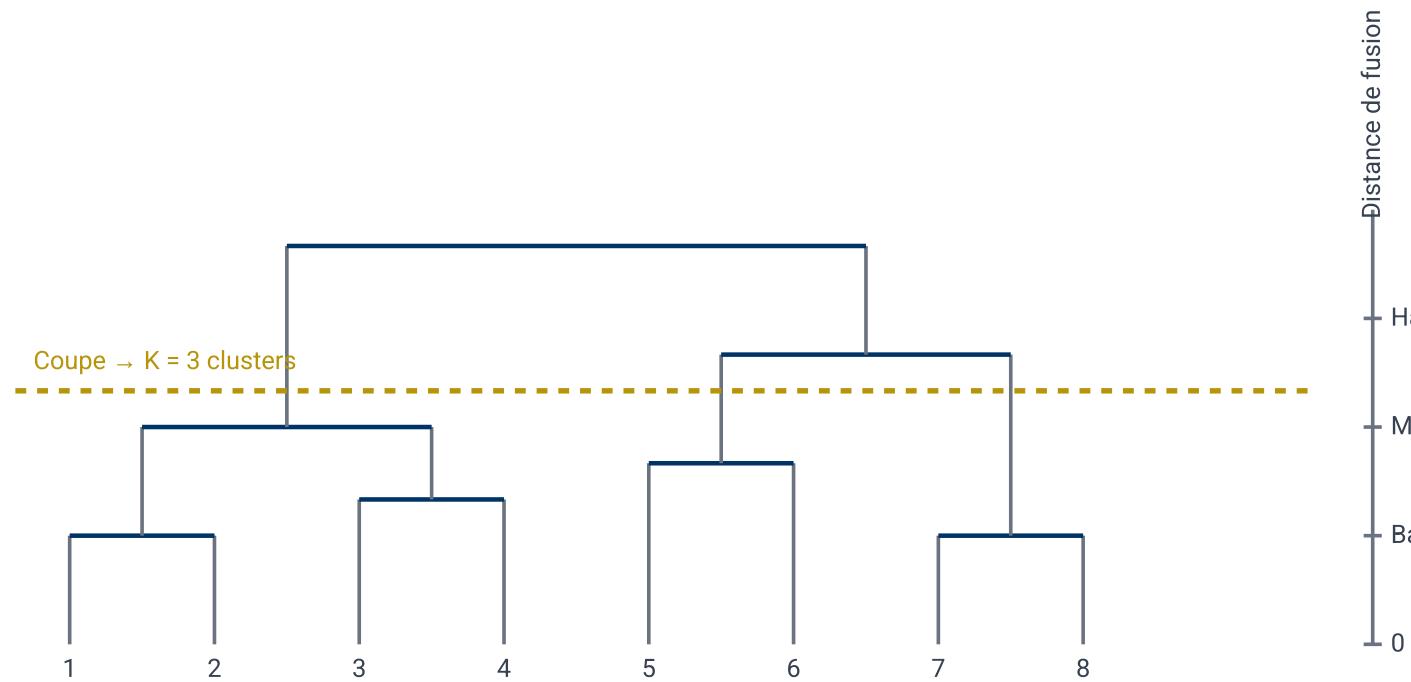
Inertie : compacité intra-cluster (plus petit = mieux).

Bonnes pratiques

- Standardiser avant K-Means.
- Choisir K (coude/silhouette).
- Fixer `random_state` pour reproductibilité.

Clustering hiérarchique – Principe et dendrogramme

Dendrogramme vertical (8 feuilles) – hauteur des branches = distance de fusion



Algorithme agglomératif (bottom-up)

- Départ : chaque point est un **cluster individuel**.
- Fusion itérative des **deux clusters les plus proches** selon une distance choisie.
- Arrêt quand il reste 1 cluster (ou selon un **seuil**).

Dendrogramme & coupe

- Arbre de fusion représentant la **hiérarchie**.
- Hauteur des branches = **distance de fusion**.
- Une **coupe horizontale** donne **K clusters finaux**.
- Avantage : **K non requis a priori**, hiérarchie visualisable.

Distances & liaisons (linkage)

- Distances : euclidienne, Manhattan, *cosine*.
- Liaisons : **single** (min), **complete** (max), **average** (moyenne), **Ward** (variance intra).

Méthodes de liaison (linkage) – Single • Complete • Average • Ward



| Single | Complete | Average | Ward |
|--|--|--|---|
| Définition | | | |
| Distance minimale entre points de deux clusters. | Distance maximale entre points des clusters. | Distance moyenne entre tous les couples de points. | Fusion minimisant l'augmentation de variance intra-cluster. |
| Distance calculée | | | |
| $\min d(x, y)$ | $\max d(x, y)$ | $\text{mean } d(x, y)$ | $\Delta \text{ SSE}$ (variance intra) minimale |
| Avantages | | | |
| Déetecte des structures en « chaînes ». | Clusters compacts, bien séparés. | Compromis équilibré (stabilité). | Groupes équilibrés, souvent pertinents en pratique. |
| Limites | | | |
| Sensible au bruit/chaînage. | Sensible aux outliers extrêmes. | Peut lisser des structures fines. | Suppose des formes proches du sphérique. |
| Cas d'usage | | | |
| Formes allongées, données chaînées. | Clusters bien séparés et compacts. | Usage général quand aucun a priori fort. | Segments marketing/clients – RECOMMANDÉ. |



Paramètres clés

`eps` = rayon du voisinage (ϵ). `min_samples` = nb. minimal de voisins dans ϵ pour être « cœur ».



Points « cœur »

Au moins `min_samples` voisins dans ϵ : initient et forment les clusters (densité suffisante).



Points « bord »

Dans le voisinage d'un cœur mais pas assez de voisins : rattachés au cluster le plus proche.



Bruit / Outliers

Ni cœur ni bord : étiquetés `-1` (non assignés). Indiquent des points atypiques.

Forces vs K-Means

- Déetecte des formes non sphériques.
- Assez robuste au bruit (outliers).
- Pas besoin de spécifier K.

Notation : cœur \rightarrow densité suffisante • bord \rightarrow voisin d'un cœur • bruit \rightarrow `-1`

Limites

- Très sensible à ϵ et `min_samples`.
- Difficultés si densités variables dans un même jeu.

Comparaison K-Means vs DBSCAN – Code rapide



Extrait de code – DBSCAN (bruit) vs K-Means (tous points en clusters)

```
# Imports
from sklearn.cluster import DBSCAN, KMeans
from sklearn.preprocessing import StandardScaler
import numpy as np

# Données simulées (2 clusters + bruit/outliers)
np.random.seed(42)
X = np.vstack([
    np.random.randn(100, 2),
    np.random.randn(100, 2) + 5,
    np.random.uniform(-3, 8, size=(10, 2))
])

# Standardisation (recommandée pour les distances)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# DBSCAN (déetecte le bruit : label -1)
db = DBSCAN(eps=0.5, min_samples=10).fit(X_scaled)
print('Clusters DBSCAN:', set(db.labels_)) # -1 = bruit
print('Nombre outliers:', (db.labels_ == -1).sum())

# K-Means (force tous les points dans k clusters)
km = KMeans(n_clusters=2, random_state=42).fit(X_scaled)
print('Clusters K-Means:', set(km.labels_)) # {0,1}
print('Inertie K-Means:', km.inertia_)

# Conclusion : DBSCAN est robuste au bruit et peut détecter des formes non sphériques ;
# K-Means est sensible aux outliers et suppose des clusters plutôt sphériques.
```

Lecture rapide

DBSCAN étiquette le bruit en -1 et ne nécessite pas K.

K-Means assigne tous les points à un cluster et retourne l'inertie.

Choix métier: formes *non sphériques* ou *bruit* → DBSCAN; segments compacts → K-Means.

Paramètres clés

- DBSCAN: eps, min_samples.
- K-Means: n_clusters, n_init.

Astuce: standardiser les features avant d'utiliser des algorithmes à base de distances.

Étapes et formules clés de l'ACP (PCA)

Centrer les données

$$X_c = X - \text{mean}(X)$$

Chaque feature a une moyenne nulle (pré-requis PCA).

Matrice de covariance

$$S = (1/m) \cdot X_c^T X_c$$

Capture les covariances entre variables.

Décomposition en valeurs propres

$$S = V \Lambda V^T$$

V = vecteurs propres (axes principaux), Λ = $\text{diag}(\lambda_i)$.

Projection sur k composantes

$$Z = X_c \cdot V_k$$

V_k = k premiers vecteurs propres ($k < d$).

Variance expliquée

$$\text{ratio}_i = \lambda_i / \sum_j \lambda_j$$

Part de variance portée par la composante i.

Variance cumulée

$$\text{cumVar}(k) = \sum_{i=1..k} \lambda_i / \sum_j \lambda_j$$

Choisir k tel que $\text{cumVar}(k) \approx 90\text{--}95\%$.

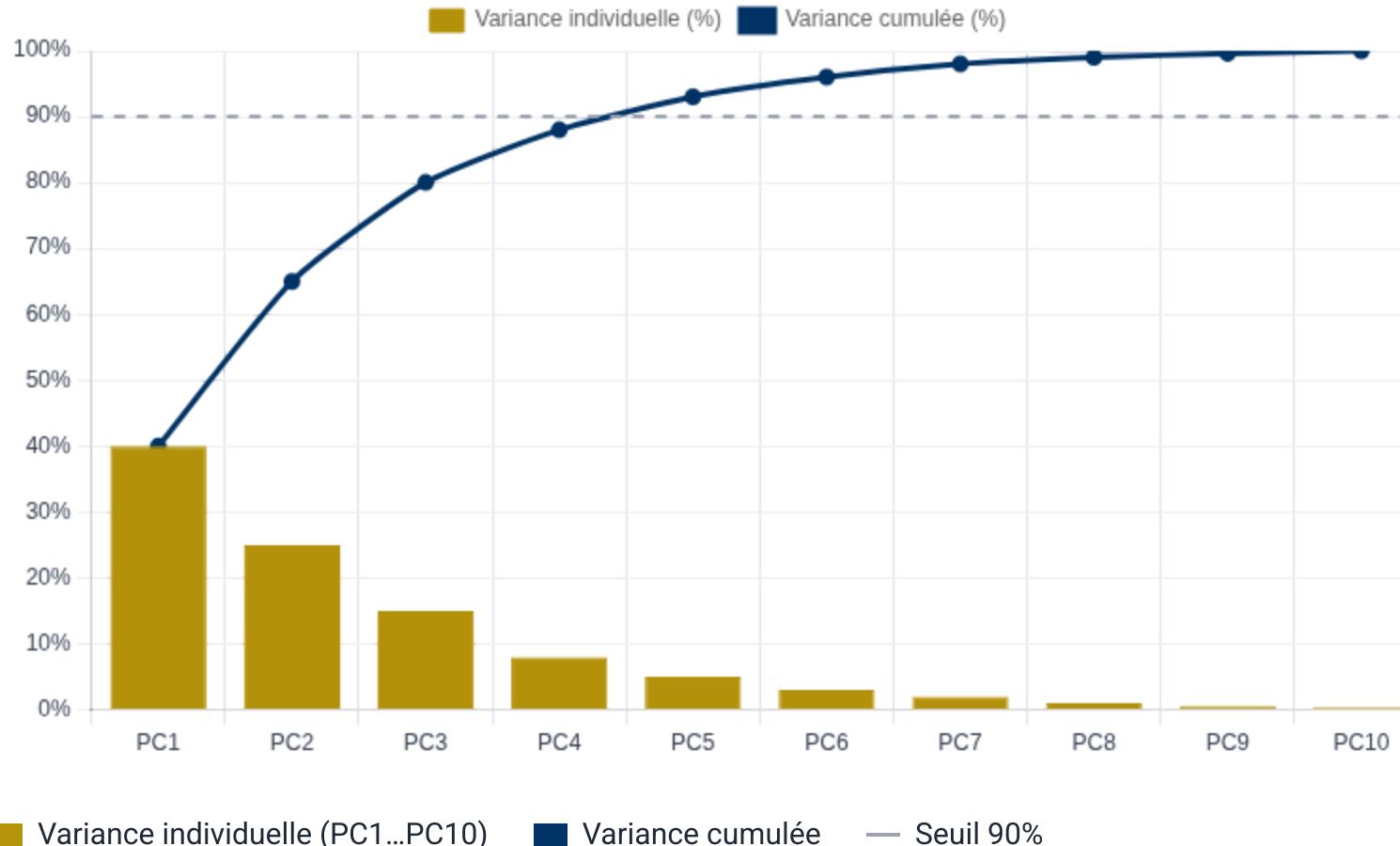


Objectif

Réduire la dimension tout en conservant un maximum de variance (typiquement 90–95%) pour simplifier les modèles, accélérer l'entraînement et améliorer la visualisation.

PCA – Variance expliquée (Scree plot)

Scree plot: variance expliquée par composante (barres) et cumulée (courbe, %)



Interprétation & sélection de k

Axes

X: composantes principales (PC1–PC10). Y: variance expliquée (%) individuelle ou cumulée.

Méthodes

- Seuil fixe: viser 90–95% cumulés.
- Coude: cassure de la courbe individuelle.
- Règle de Kaiser: valeurs propres > 1 .

Exemple

PC1=40%, PC2=25%, PC3=15% \rightarrow cumul=80%
(3 composantes suffisent souvent pour visualiser).

Couleurs ENCG : Bleu #003366 • Doré #b7950b



Concepts clés (côté métier & data)

- Réduction de dimension haute (10D, 100D) → basse (2D, 3D) pour une meilleure visualisation des clusters et des patterns.
 - Conserver l'essentiel de la variance ($\geq 90\%$) avec moins de dimensions via des composantes principales orthogonales.
 - Applications : explorer les données, détecter des outliers, visualiser des groupes, preprocessing avant clustering/modélisation.
- Bonnes pratiques : standardiser les variables (échelles comparables) et choisir le nombre de composantes par variance expliquée cumulée (ex. 90–95%).

Code Python – PCA 10D → 2D

scikit-learn • matplotlib

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Standardiser d'abord
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 10 features → 2 composantes
pca = PCA(n_components=2, random_state=42)
Z = pca.fit_transform(X_scaled)

# Variance expliquée
```

Interprétation : PC1 et PC2 maximisent la variance projetée, facilitant l'exploration visuelle (groupes, outliers) dans un plan 2D.

PCA – Applications



- Compression de données et accélération d'entraînement (réduire les features de 100 à 20 → temps de calcul divisé, stockage réduit)
- Visualisation de clusters latents (projeter la haute dimension vers 2D/3D pour explorer et détecter des patterns visuels)
- Débruitage et élimination des features redondantes (composantes à faible variance = bruit; conserver les principales = signal)
- Orthogonalisation des features (composantes principales décorrélées, utile pour modèles sensibles à la colinéarité)
- Prétraitement pour le clustering (K-Means sur composantes principales plus stable)
- Feature extraction (créer de nouvelles features synthétiques combinant les variables d'origine)

t-SNE — Principe et bonnes pratiques



- Preserve les voisinages locaux (méthode non linéaire) : maintient les similarités locales plutôt que la structure globale.
- Paramètre clé : perplexité ($\approx 5-50$) pour l'équilibre local/global – tester plusieurs valeurs.
- Stochastique et non déterministe : résultats sensibles à l'initialisation – fixer `random_state`.
- Pas d'application out-of-sample : pas de transform direct sur nouvelles données (`fit_transform` seulement).
- Usage : visualisation exploratoire (pas un modèle prédictif) – interpréter les distances avec prudence.
- Standardiser et échantillonner les grands jeux de données : coût $\approx O(n^2)$, viser 5 000–10 000 points max.
- Itérations suffisantes : $\ge 1\,000$ pour une convergence stable (vérifier la stabilité des motifs).

Bonnes pratiques t-SNE : standardiser • ajuster perplexité • fixer `random_state` • limiter la taille

ENCG Settat • Charte: Bleu #003366 • Doré #b7950b

PCA vs t-SNE vs UMAP – Comparaison



| Critère | PCA | t-SNE | UMAP |
|--------------------------------|-------------------------------------|---------------------------------------|---|
| Conservation | Structure globale (variance). | Voisinages locaux. | Local + global équilibré. |
| Type transformation | Linéaire (combinaisons). | Non linéaire (probabiliste). | Non linéaire (topologique). |
| Vitesse / Scalabilité | Très rapide $\sim O(n \cdot d^2)$. | Lent $\sim O(n^2)$. | Rapide $\sim O(n \cdot \log n)$. |
| Paramètres clés | <code>n_components</code> . | <code>perplexity, iterations</code> . | <code>n_neighbors, min_dist</code> . |
| Out-of-sample transform | Oui (direct). | Non. | Oui (approximatif). |
| Cas d'usage | Prétraitement, compression. | Visualisation, exploration. | Visualisation + analyse rapide (grands jeux). |
| Interprétation | Axes = combinaisons linéaires. | Distances locales relatives. | Structure topologique préservée. |

Isolation Forest – Détection d'anomalies



Idée clé : les anomalies sont plus faciles à isoler que les points « normaux » (chemins d'isolation plus courts).

1 Échantillonnage

Échantillonne aléatoirement des **features** et des **points** (bootstrap) pour créer des arbres diversifiés.

2 Arbres d'isolation

Construit des arbres par **coupes aléatoires** (feature aléatoire, seuil entre min et max).

3 Isolation rapide

Les **outliers** s'isolent en peu de coupes → **chemins courts** dans l'arbre.

4 Score d'anomalie

Score basé sur la **longueur de chemin moyenne** (sur la forêt) : chemins **courts** → anomalie **-1**, chemins **longs** → normal **+1**.

Paramètres clés

`n_estimators` (100–200 arbres) • **contamination** (proportion attendue 0,01–0,10) • `max_samples` (256 typique pour chaque arbre)

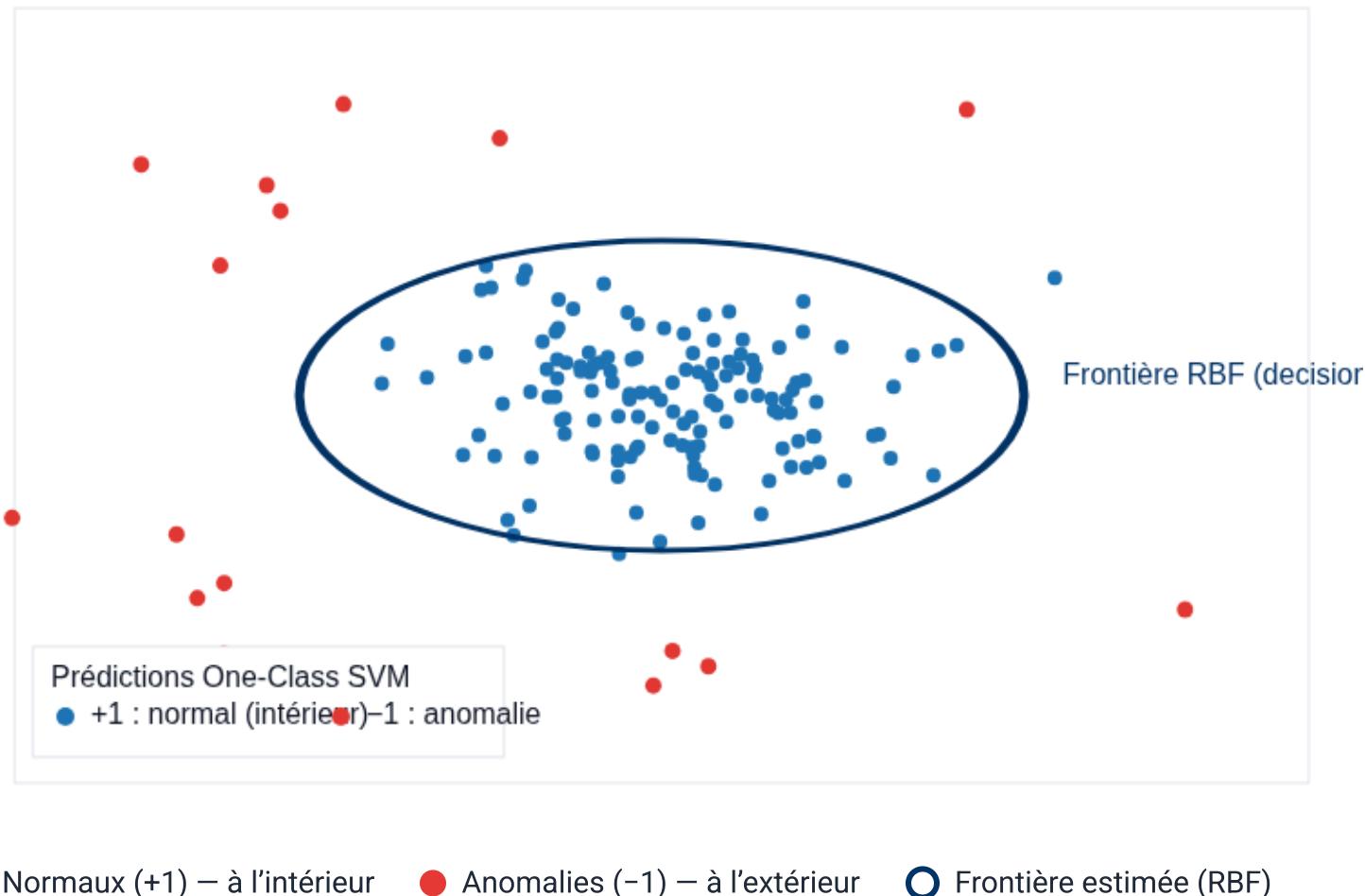
Avantages

Rapide et **scalable** • Peu sensible à la dimension • Efficace pour **anomalies globales** sans supervision.

One-Class SVM – Frontière de normalité (kernel RBF)



Nuage 2D – zone normale (RBF) et anomalies



Principe

Apprendre la **frontière de normalité** entourant la masse des points « normaux ». Les points en dehors sont prédits comme **anomalies**.

Paramètres clés

- **kernel** : RBF gaussien (recommandé pour formes complexes)
- **nu** $\in [0.01-0.2]$: fraction max d'anomalies attendues, contrôle la souplesse
- **gamma** : largeur RBF (petit = frontière large, grand = étroite)

Sorties & usages

- **decision_function** : distance signée (négatif = anomalie)
- **Prédictions** : {+1 normal, -1 anomalie}
- Données non linéaires, **frontières complexes**

Code Python – LOF et Isolation Forest



Détection d'anomalies – LOF (local) et Isolation Forest (global)

```
# Imports
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import LocalOutlierFactor
from sklearn.ensemble import IsolationForest

# Données simulées : 200 points "normaux" + 10 anomalies
np.random.seed(42)
X_normal = np.random.randn(200, 2)
X_anomalies = np.random.uniform(-4, 4, size=(10, 2))
X = np.vstack([X_normal, X_anomalies])

# Standardisation (souvent indispensable)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# LOF = Local Outlier Factor (anomalies "locales")
lof = LocalOutlierFactor(n_neighbors=20, contamination=0.05)
y_lof = lof.fit_predict(X_scaled) # -1 = anomalie, 1 = normal
print('LOF - Anomalies détectées:', (y_lof == -1).sum())
print('LOF - Taux anomalies:', (y_lof == -1).mean())

# Isolation Forest = anomalies "globales"
iso = IsolationForest(n_estimators=200, contamination=0.05, random_state=42)
iso.fit(X_scaled)
y_iso = iso.predict(X_scaled) # -1 = anomalie, 1 = normal
print('IF - Anomalies détectées:', (y_iso == -1).sum())
print('IF - Taux anomalies:', (y_iso == -1).mean())

# Scores d'anomalie : plus négatif = plus "anormal"
scores_iso = iso.decision_function(X_scaled)
print('Top 5 anomalies (scores les plus négatifs):', np.argsort(scores_iso)[:5])
```

Astuce: labels de sortie = -1 (anomalie) / 1 (normal) dans LOF/IsolationForest.

Lecture des résultats

LOF détecte des anomalies locales (voisinage).

Isolation Forest isole rapidement les outliers (global).

Un score plus **négatif** = plus anormal.

Points clés

- Standardiser les features avant LOF/SVM.
- contamination ≈ part attendue d'anomalies.
- Comparer méthodes pour robustesse.

Étude de cas – Segmentation clientèle bancaire



Objectif & Données

Objectif métier

- Identifier des segments clients actionnables pour des offres personnalisées **next-best-offer (NBO)**.

Sources de données

- RFM : Récence, Fréquence, Montant.
- Socio-démographiques : âge, revenu, patrimoine.
- Interactions : canal préféré, réponse aux campagnes.
- Produits détenus : comptes, cartes, crédits, épargne.

Périmètre : clients particuliers actifs • Horizon d'analyse : 12 mois glissants.

Méthodologie & KPI

- Prétraitement : standardisation (StandardScaler), encodage des catégorielles.
- Choix de K : méthode du coude / silhouette (segments typiques banque : K=4–6).
- Profilage : analyser moyennes par cluster et nommer les segments (VIP, Actifs, Dormants, Entrants).

KPI business (adoption et valeur)

- Taille segments équilibrée (\approx 15–30% chacun).
- Stabilité temporelle (cohérence d'affectation).
- CLV / rentabilité par segment.
- Adoption métier (utilisation par les commerciaux).

Livrables : description segments, personas et recommandations NBO par segment.

Résultats segmentation clientèle bancaire – 4 segments



VIP Premium (18%)

45–60 ans, revenu > 6000 €, patrimoine important, multi-équipé (comptes, cartes, crédits, épargne). CLV élevé.

Offres premium

Wealth mgmt

Conseiller dédié



Actifs Engagés (28%)

30–45 ans, revenu 3500–6000 €, 5–10 transactions/mois, usage digital intensif.

Upselling digital

Notif. personnalisées

Programmes fidélité



Dormants à Réactiver (32%)

50+ ans, revenu 2500–4000 €, < 2 transactions/mois, potentiel sous-exploité.

Réactivation ciblée

Offres spéciales

Contact téléphonique



Entrants Jeunes (22%)

20–35 ans, revenu 1500–3000 €, début de relation, forte croissance potentielle.

Onboarding digital

Éducation financière

Produits starter

Quiz interactif — Chapitre 4



1

Supervisé vs non supervisé

Quelle est la différence entre apprentissage supervisé et non supervisé ? (Indication : supervisé = labels y, non supervisé = structures cachées sans labels)

2

Choisir K pour K-Means

Comment choisir le K optimal pour K-Means ? (Indication : coude WSS, silhouette moyenne, tester plusieurs K)

3

DBSCAN vs K-Means

Dans quels cas utiliser DBSCAN plutôt que K-Means ?
(Indication : formes non sphériques, bruit/outliers, pas de K à fixer)

4

Rôle de la PCA

À quoi sert la PCA et comment choisir le nombre de composantes ? (Indication : réduction dimension, 90–95% variance cumulée / scree plot)

5

PCA vs t-SNE

Quelle est la différence entre PCA et t-SNE ? (Indication : PCA linéaire globale et rapide; t-SNE non linéaire, voisinages locaux, visualisation exploratoire)

Introduction au deep learning – pourquoi maintenant ?



- Définition : apprentissage automatique fondé sur des réseaux de neurones profonds (plusieurs couches cachées)
- 2012 (ImageNet/AlexNet) : rupture de performance + essor GPU → révolution du deep learning
- Enablers : données massives, calcul parallèle (GPU/TPU), bibliothèques (TensorFlow/Keras, PyTorch)
- Applications phares : vision (OCR), NLP (BERT/GPT), séries temporelles (LSTM), recommandation
- Intérêt business : automatisation de tâches complexes, amélioration des KPI (qualité, coûts, délais)
- Conditions de succès : données étiquetées de qualité, capacité de calcul, MLOps pour l'industrialisation

Deep Learning vs ML traditionnel – comparaison



| Critère | ML traditionnel | Deep Learning | Cas d'usage |
|-------------------------------|--|---|--|
| Ingénierie de features | Manuelle (experts métier/feature engineering). | Automatique (apprentissage de représentations). | Images, texte non structuré, audio/vidéo. |
| Données requises | Modérées (tableaux, features agrégées). | Très volumineuses (big data, labels nombreux). | Big data (vision/NLP à grande échelle). |
| Calcul | CPU suffisant, entraînement rapide. | GPU/TPU recommandés, entraînements lourds. | Jobs distribués, pipelines d'entraînement. |
| Performance | Peut plafonner sur tâches complexes. | Surpasse souvent sur vision/NLP/séries. | Vision, NLP, audio, perception complexe. |
| Interprétabilité | Élevée (arbres, régressions, règles). | Plus faible (boîte noire) → XAI requis. | XAI pour conformité (finance, santé). |
| Déploiement | Simple (services classiques, CPU). | MLOps dédié (serving GPU, monitoring). | Pipelines DL (CI/CD, traçabilité, drift). |

Composants

- Entrées $x_1 \dots x_d$ (features)
- Poids $w_1 \dots w_d$ et biais b
- Somme pondérée
$$z = \sum_{i=1..d} w_i x_i + b$$
- Activation
$$a = f(z)$$
 (ex. ReLU, sigmoïde, tanh)

Sortie de couche: $a^{(l)} = f(W^{(l)}a^{(l-1)} + b^{(l)})$

Intuitions

- Le neurone calcule une combinaison linéaire puis applique une non-linéarité
- Les poids et le biais sont appris pour minimiser une fonction de coût (descente de gradient)
- Inspiration biologique: analogie neurone (soma, dendrites, axone) mais modèle mathématique simplifié

Astuce: sans non-linéarité, plusieurs couches se réduisent à un modèle linéaire → incapacité à modéliser des relations complexes.

Équations du neurone

Somme pondérée

$$z = \sum_{i=1}^d w_i x_i + b$$

Combinaison linéaire des entrées avec biais.

Activation générique

$$a = f(z)$$

Introduction d'une non-linéarité pour modéliser des relations complexes.

Sortie de couche (notation matricielle)

$$a^{(l)} = f(W^{(l)} \cdot a^{(l-1)} + b^{(l)})$$

$W^{(l)}$: matrice des poids, $b^{(l)}$: biais, $a^{(l-1)}$: activations de la couche précédente.

Exemples d'activation

Sigmoïde

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

Sortie (0,1) – utile en sortie binaire.

Tanh

$$\tanh(z)$$

Centrée en 0 – souvent meilleure en couches cachées.

ReLU

$$\text{ReLU}(z) = \max(0, z)$$

Simple et rapide – atténue les gradients ténus.

Perceptron simple – Algorithme et limites



Principe

Classificateur linéaire: prédire $\hat{y} = 1$ si $w \cdot x + b > 0$, sinon 0. Mise à jour en ligne avec un **taux d'apprentissage η** .

1 Initialiser

Choisir w et b (aléatoires petits ou nuls).

Paramètre: η (learning rate).

2 Prédire

Pour chaque (x, y) :
 $\hat{y} = 1$ si $w \cdot x + b > 0$, sinon 0.

3 Mettre à jour

Si **erreur** ($y \neq \hat{y}$) :
 $w \leftarrow w + \eta (y - \hat{y}) x$
 $b \leftarrow b + \eta (y - \hat{y})$

4 Répéter

Répéter les passes jusqu'à **convergence** (plus d'erreurs ou nb itérations max).

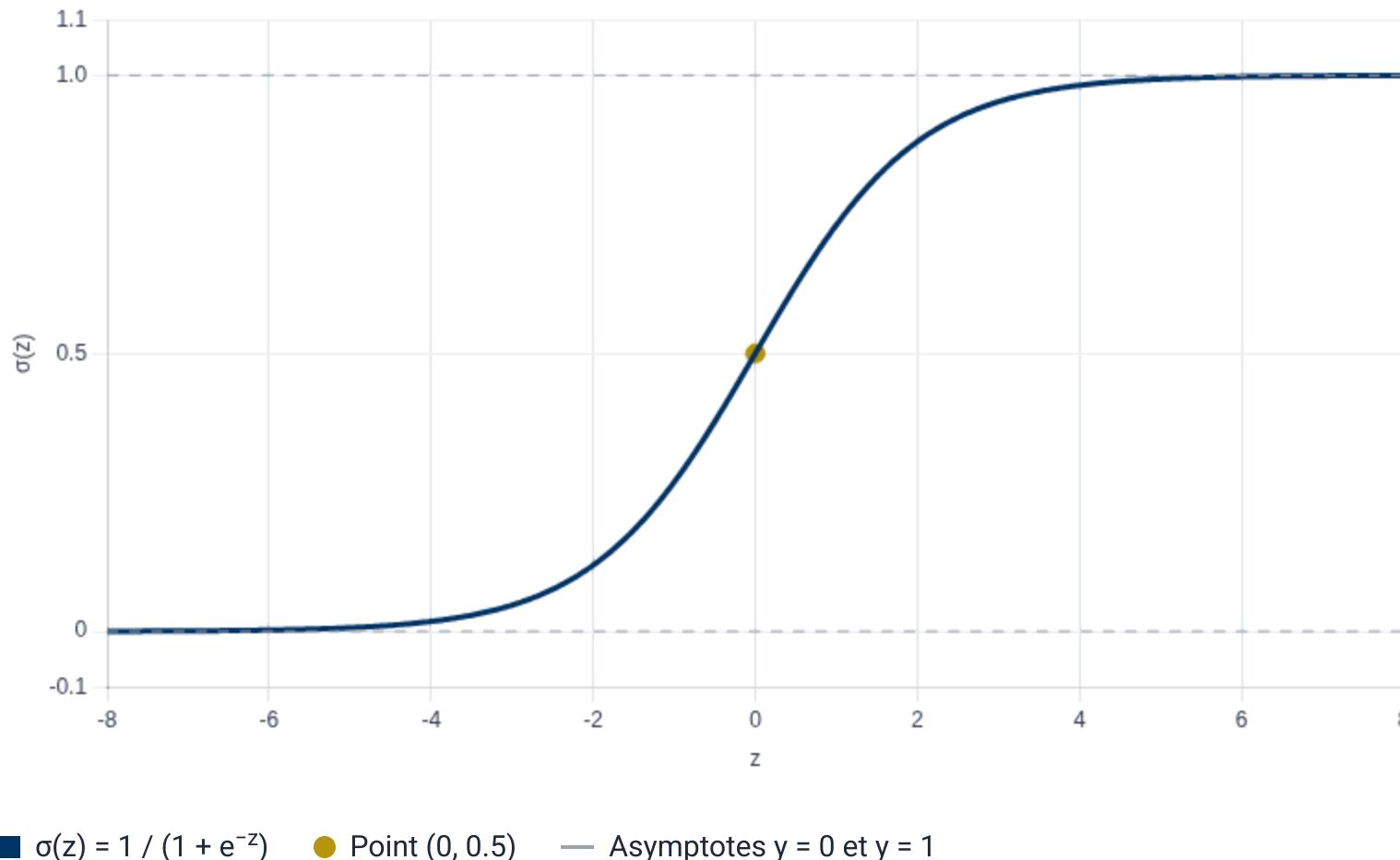
Converge si données **linéairement séparables**.

Limite fondamentale : les problèmes **non linéaires** (ex. **XOR**) ne sont pas séparables par un hyperplan – nécessité d'**ajouter des couches cachées** (MLP) et des activations non linéaires.

Fonction d'activation – Sigmoïde



Courbe en S de $\sigma(z)$, point $(0, 0.5)$ et asymptotes horizontales $y = 0$ et $y = 1$



Formules

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

$$\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z))$$

Propriétés

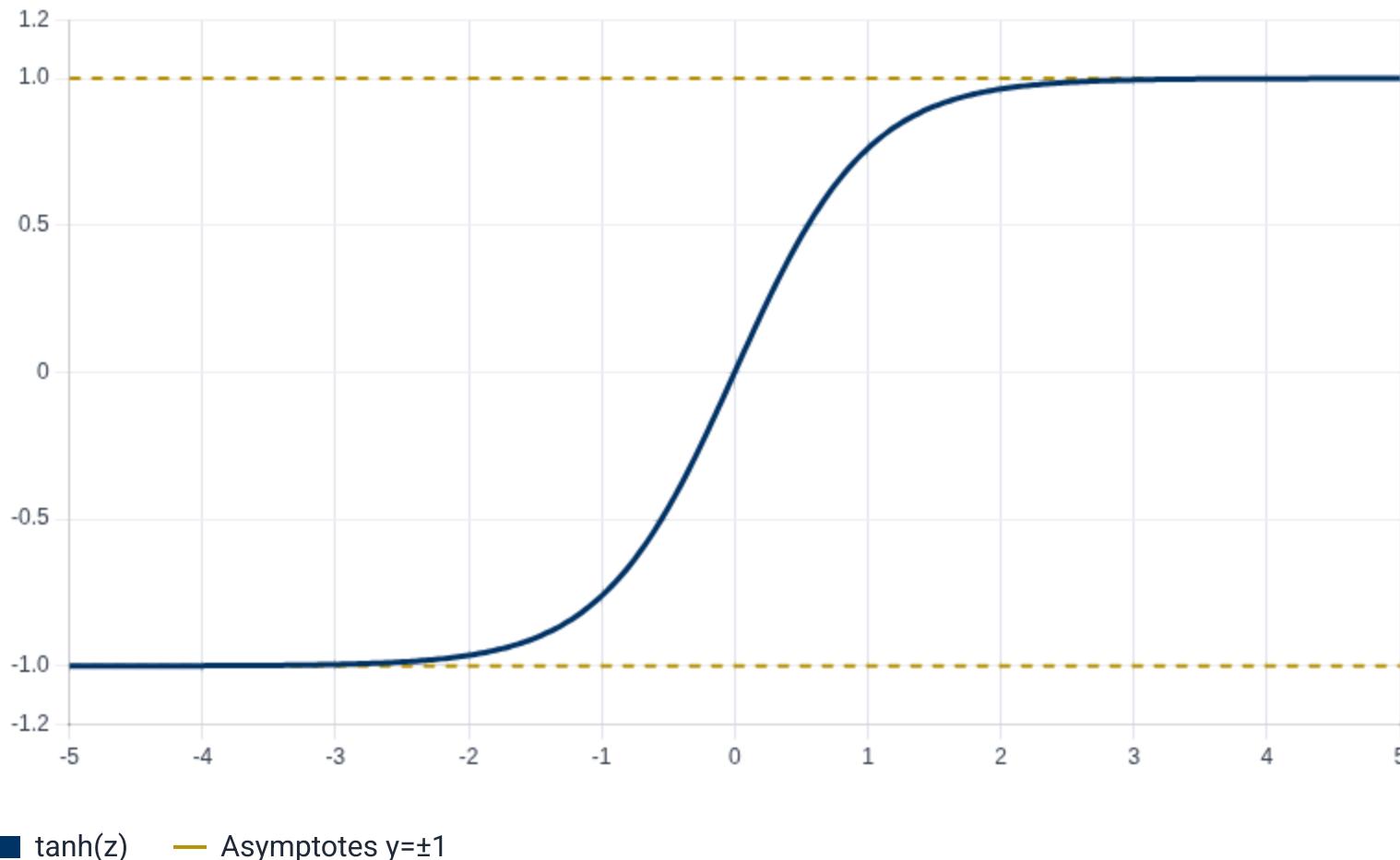
- Sortie dans $(0, 1)$ interprétable comme probabilité.
- Saturations pour $z \rightarrow \pm\infty \rightarrow$ gradients faibles.
- Usage : sortie binaire et logits calibrés.

Couleurs ENCG : Bleu #003366 • Doré #b7950b

Fonction d'activation – Tanh



Courbe en S centrée (0,0), asymptotes horizontales à $y=-1$ et $y=1$, symétrie par l'origine



Formule et dérivée

$$\tanh(z) = (e^z - e^{-z}) / (e^z + e^{-z})$$

$$\text{Dérivée : } 1 - \tanh^2(z)$$

Propriétés

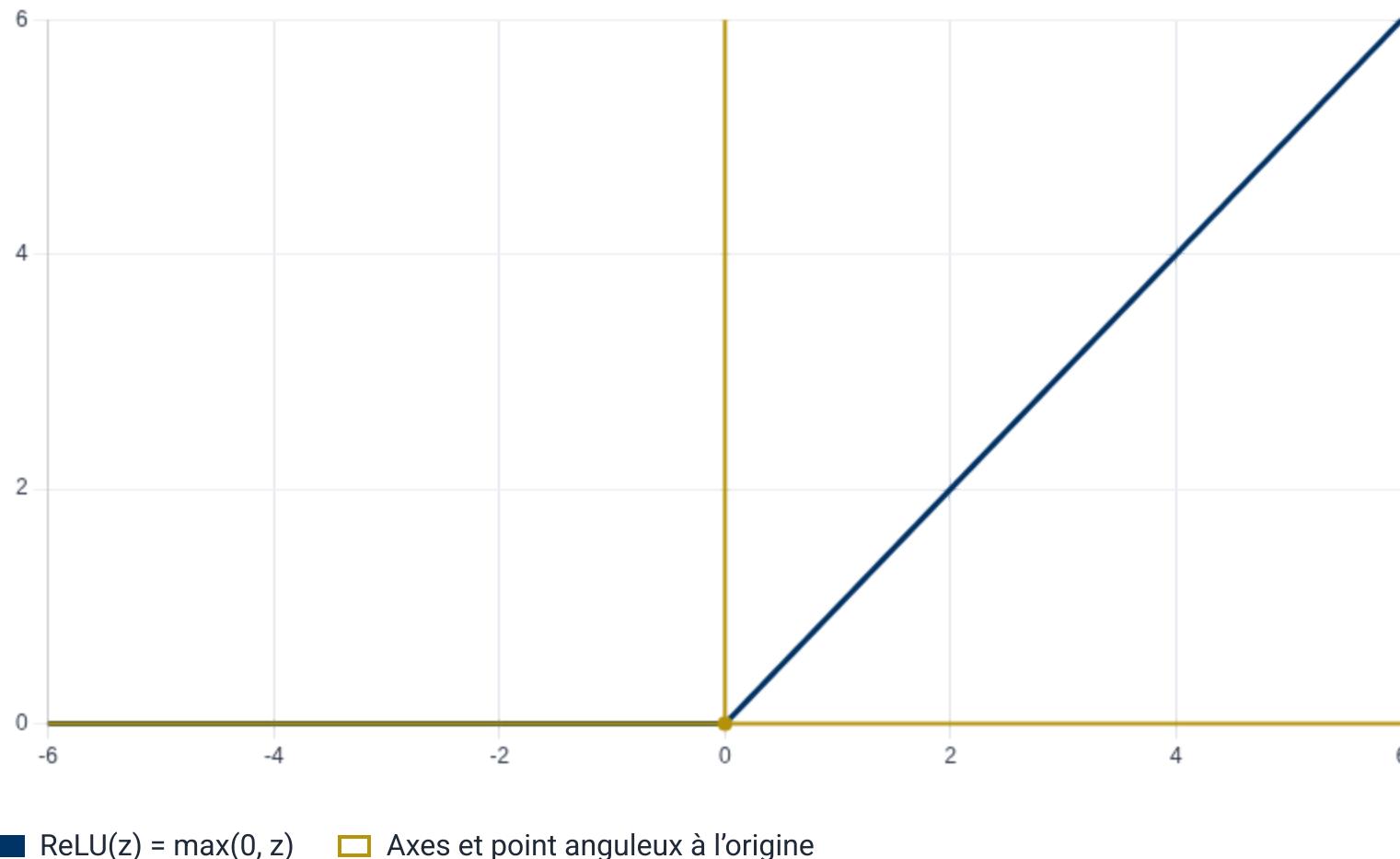
- Sortie dans $(-1, 1)$, centrée en 0
- Souvent préférable à la sigmoïde en couches cachées (gradients plus forts, centrage)
- Symétrie impaire : $\tanh(-z) = -\tanh(z)$

Couleurs ENCG : Bleu #003366 • Doré #b7950b

Fonction d'activation – ReLU



Courbe ReLU: $y = \max(0, z)$ – rampe nulle pour $z < 0$ puis pente 1 pour $z > 0$



Formule et dérivée

$$\text{ReLU}(z) = \max(0, z)$$

Dérivée: 0 si $z < 0$, 1 si $z > 0$ (indéfinie en 0, souvent prise à 0 ou 1)

Atouts

- Calcul simple et rapide (efficace sur GPU)
- Réduit le vanishing gradient vs sigmoïde/tanh

Limite

Dying ReLU: neurones « bloqués » à 0 si poids/biais rendent z négatif de façon persistante.

Couleurs ENCG : Bleu #003366 • Doré #b7950b



Leaky ReLU

Définition : $\max(\alpha \cdot z, z)$ avec $\alpha=0,01$.

Avantage : évite les neurones « morts » (dying ReLU) grâce à une pente faible pour $z<0$.



PReLU (Parametric ReLU)

Principe : le coefficient α est apprenable par backpropagation.

Bénéfice : plus grande flexibilité d'ajustement selon la couche et les données.



ELU (Exponential Linear Unit)

Forme : z si $z>0$; sinon $\alpha \cdot (e^z - 1)$.

Atouts : activations lissées, gradients plus stables, sortie négative contrôlée.



SELU (Scaled ELU)

Propriété : auto-normalisation (maintien approx. de la moyenne/variance).

Usage : réseaux profonds stables (avec initialisation LeCun et AlphaDropout).

Softmax – classification multi-classes



Formules et propriétés

Équations

Softmax : $\hat{y}_i = e^{z_i} / \sum_{j=1..K} e^{z_j}$, $i = 1..K$

Normalisation : $\sum_{i=1..K} \hat{y}_i = 1$ et $0 < \hat{y}_i < 1$

Perte (entropie croisée) : $L = -\sum_{k=1..K} y_k \log(\hat{y}_k)$

Usage et interprétation

- Couche de sortie pour classification multi-classes (K catégories).
- Transforme les logits z_i en probabilités calibrées \hat{y}_i .
- Décision finale : $\text{argmax}_i \hat{y}_i$.
- Stabilité numérique : calculer sur $z - \max(z)$ avant exponentielle.

Exemple de sorties (sommant à 1) :



Bonnes pratiques

Utiliser l'entropie croisée avec Softmax pour entraîner les modèles multi-classes. Pour améliorer la stabilité, soustraire $\max(z)$ des logits avant exponentiation.

MLP multicouches – architecture



Schéma de flux



Paramètres clés

- **Largeur** : neurones par couche (ex. 64 → 32) → **Sortie**
K classes (Softmax) ou 1 (Sigmoid)
- **Profondeur** : nombre de couches cachées L.
- **Activations** : ReLU/Tanh (cachées), Softmax/Sigmoid (sortie).
- **Initialisation** : Xavier/Glorot (Tanh), He (ReLU).
- **Normalisation** : Batch Normalization pour stabilité.

Bonnes pratiques

Standardiser les entrées; ajouter L2/Dropout; early stopping; learning rate adapté.

Résumé

Un MLP apprend des représentations hiérarchiques via des couches denses et des non-linéarités pour résoudre classification/régression.

Propagation avant (forward) – équations



x¹ Formulation couche par couche

Pour chaque couche $l = 1 \dots L$:

$$z^l = W^l \cdot a^{l-1} + b^l$$

$$a^l = f^l(z^l)$$

Sortie du modèle :

$$\hat{y} = f^L(z^L)$$

Flux de la propagation avant

$$a^0 = x \rightarrow \dots \rightarrow a^l \rightarrow a^L = \hat{y}$$

Chaque couche applique une combinaison linéaire suivie d'une non-linéarité f^l .

i Notations

W^l : matrice de poids de la couche l

b^l : vecteur de biais de la couche l

$a^0 = x$: entrée du réseau

a^l : activation (sortie) de la couche l

$a^L = \hat{y}$: prédiction finale

f^l : fonction d'activation de la couche l

Rétropropagation (backprop) – Chaîne de dérivation



1 Gradient en sortie

Calculer $\partial L / \partial a^{(L)}$ (dérivée de la perte par rapport à la sortie du modèle).

Point de départ du calcul des gradients.

Ex.: classification → dérivée de l'entropie croisée.

2 Propagation arrière

Pour chaque couche l :

$$\partial L / \partial z^{(l)} = (\partial L / \partial a^{(l)}) \odot f^{(l)}(z^{(l)})$$

Règle de la chaîne ; \odot = produit d'Hadamard (élément par élément).

On utilise les activations et préactivations du forward pass.

3 Gradients poids/biais

$$\partial L / \partial W^{(l)} = (\partial L / \partial z^{(l)}) \cdot (a^{(l-1)})^T$$

$$\partial L / \partial b^{(l)} = \partial L / \partial z^{(l)}$$

Formules vectorisées pour efficacité et stabilité numérique.

Les dimensions sont compatibles via produits matriciels.

4 Mise à jour des paramètres

$$\theta \leftarrow \theta - \eta \nabla_{\theta} L$$
 (descente de gradient)

η = taux d'apprentissage ; variantes (SGD, Adam, RMSProp...).

Boucler jusqu'à convergence des pertes/valeurs sur validation.

Efficacité – **Calcul vectorisé** (batchs) et **réutilisation du forward pass** (activations/préactivations) pour réduire le coût de calcul et la mémoire.

Extrait de code – MLP Keras minimal

```
# Imports Keras (TensorFlow)
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# Architecture MLP simple
model = Sequential([
    Dense(64, activation='relu', input_shape=(d,)), # d = nombre de features
    Dense(32, activation='relu'),
    Dense(K, activation='softmax') # K = nombre de classes
])

# Compilation
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Entraînement
history = model.fit(
    X_train, y_train,
    epochs=20, batch_size=32,
    validation_data=(X_val, y_val),
    verbose=1
)

# Évaluation
loss, acc = model.evaluate(X_test, y_test)
print(f'Test accuracy: {acc:.2%}' )
```

Astuce: utiliser un EarlyStopping pour arrêter l'entraînement si la validation stagne.

Architecture & hyperparamètres

Entrée: d features • Sortie: K classes.

Couches cachées: 64 → 32 neurones (ReLU).

Optimiseur: Adam(1e-3) • Perte: categorical_crossentropy.

Entraînement: 20 époques • batch 32.

Bonnes pratiques

- Standardiser X; encoder y en one-hot.
- Séparer train/val/test (pas de fuite de données).
- Surveiller l'overfitting (Dropout, régularisation).

Régularisation L1/L2 – pénaliser la complexité



Formulation

L2 (Ridge, Weight Decay)

$$\begin{aligned} L_{\text{tot}} &= L + \lambda \sum_1 ||w^{(1)}||_2^2 \\ &= L + \lambda \sum_{i,j} w_{ij}^2 \end{aligned}$$

Pénalise la somme des carrés des poids (rétrécit vers 0).

L1 (Lasso)

$$\begin{aligned} L_{\text{tot}} &= L + \lambda \sum_1 ||w^{(1)}||_1 \\ &= L + \lambda \sum_{i,j} |w_{ij}| \end{aligned}$$

Encourage la parcimonie (poids exactement nuls).

Effets

- L2 : rétrécit les poids vers 0 → stabilité, continuité.
- L1 : parcimonie et sélection de variables (poids nuls).

Paramètre λ

- Contrôle la force de pénalisation.
- λ grand = forte pénalité (moins d'overfitting, risque de biais).
- Choix via validation croisée.

Usage

- Éviter l'overfitting et stabiliser l'entraînement.
- Combiner avec Dropout / Early stopping / Batch Norm.
- Normaliser les features pour un effet cohérent.



Principe du Dropout

- Désactiver aléatoirement $p\%$ de neurones à l'entraînement (p typ. = 0.2–0.5).
- Évite la co-adaptation et les dépendances fortes entre neurones.
- Agit comme un ensemble implicite de réseaux (effet régularisant).
- En test: tous les neurones actifs, sorties mises à l'échelle par $(1-p)$.
- Objectif: réduire l'overfitting et améliorer la généralisation sans changer l'architecture fondamentale.

Code Keras – Dropout

TensorFlow • Keras

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# d = nb de features, K = nb de classes (softmax)
model = Sequential([
    Dense(128, activation='relu', input_shape=(d,)),
    Dropout(0.5), # 50% dropout
    Dense(64, activation='relu'),
    Dropout(0.3), # 30% dropout
    Dense(K, activation='softmax')
])
```

Bonnes pratiques: appliquer après couches denses/convolutives, ajuster p selon l'overfitting observé.

Batch Normalization – équations et bénéfices



Équations (par mini-batch)

Normaliser

$$\hat{x} = (x - \mu) / \sqrt{\sigma^2 + \epsilon}$$

μ, σ^2 = moyenne et variance du mini-batch; ϵ = petite constante de stabilité numérique

Recenter / rescaler

$$y = \gamma \cdot \hat{x} + \beta$$

γ, β = paramètres apprenables (par couche) restaurent l'échelle et le décalage utiles

Inférence

Utiliser moyennes/variances « running » estimées à l'entraînement (pas les statistiques du batch courant)

Bénéfices et usage

- Gradients plus stables (réduction de l'« internal covariate shift »)
- Entraînement plus rapide, convergence accélérée
- Moins sensible à l'initialisation et au learning rate
- Effet régularisant léger → réduction de l'overfitting
- Permet des learning rates plus élevés
- Usage recommandé

Après couches Dense/Conv, généralement avant l'activation (ex. Conv → BN → ReLU)

CNN – introduction et cas d'usage



- Spécialistes des données spatiales (images 2D, documents, vidéos)
- Partage des poids (filtres de convolution) et invariance locale aux translations
- Avantages : réduction des paramètres vs fully connected ; exploitation de la structure spatiale
- OCR de relevés bancaires automatisé
- Extraction de tableaux (chèques, factures)
- Détection de tampons/signatures sur documents
- Analyse de graphiques financiers (tendances)
- Classification de documents (contrat, facture, RIB)
- Inspection qualité visuelle des produits

En synthèse : les CNN sont idéales pour extraire automatiquement des motifs visuels pertinents et réduire drastiquement le nombre de paramètres grâce au partage des filtres.

Convolution 2D – principe



Formule

$$(X * K)(i, j) = \sum_{m,n} X(i+m, j+n) \cdot K(m, n)$$

où X est l'image (entrée) et K le filtre (kernel).

Le filtre glisse sur l'image et calcule une somme pondérée locale, produisant une carte de caractéristiques.

Concepts clés

- **Filtres** (3×3 , 5×5 , 7×7) détectent des motifs locaux (bords, textures).
- **Stride** = pas de déplacement (1 : glisse pixel par pixel ; 2 : saute 1 pixel).
- **Padding (same)** conserve la taille, (valid) la réduit.
- **Feature maps et canaux** : RGB = 3 canaux en entrée ; couches profondes = 64–512 canaux.
- **Filtres apprenables** par rétropropagation (gradients), adaptés aux données.

Intuition : détection hiérarchique – des motifs simples (bords) vers des motifs complexes (formes, objets).

Pooling – réduction dimensionnelle



Types de pooling

● Max pooling

Garde la valeur maximum locale (fenêtre 2×2) → robuste au bruit, met en avant les activations fortes.

● Average pooling

Prend la moyenne locale → lissage des cartes de caractéristiques.

● Global pooling

1 valeur par feature map (global average/max) → réduit fortement la dimension avant la couche de sortie.

Astuce: combiner conv → ReLU → pooling pour extraire des motifs invariants.

Effets et usage

2x2 Fenêtres (2x2) stride 2

Division par 4 de la taille spatiale: $H \rightarrow H/2, W \rightarrow W/2$.

■ Invariance locale

Moins sensible aux petites translations et déformations.

■ Moins de paramètres et de calcul

Aide à prévenir l'overfitting et accélère l'entraînement.

■ Attention à l'information

Éviter une réduction trop agressive pour préserver les signaux discriminants.

● Pattern classique

Alternance Conv → ReLU → Pool (répétée sur plusieurs blocs).

Architectures CNN célèbres – VGG16 • ResNet50 • Inception v3



VGG16

Empilement de convolutions 3×3 + max-pooling; architecture simple, efficace pour le transfert.

- Beaucoup de paramètres (~138M), profondeur 16–19 couches
- Usage recommandé : baseline de transfer learning



ResNet50

Connexions résiduelles (skip connections) qui facilitent l'entraînement profond.

- Évite le vanishing gradient; 50–152 couches avec shortcuts identité
- Usage : backbone robuste pour tâches de vision



Inception v3

Convolutions multi-échelles parallèles (1×1 , 3×3 , 5×5) avec factorisation.

- Efficacité de calcul améliorée, haute performance en classification
- Usage : modèles précis sous contraintes de latence

Code Keras CNN – Classification MNIST (Conv2D + MaxPooling)



Extrait de code – CNN simple pour MNIST (28×28 niveaux de gris)

```
# Imports Keras / TensorFlow
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Architecture CNN (classification 10 chiffres)
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

# Compilation et entraînement
model.compile(
    optimizer=Adam(1e-3),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
# X_train : (n, 28, 28, 1) – y_train : (n, )
history = model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_val, y_val))

# Évaluation
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
print(f'Test accuracy: {test_acc:.2%}' )
```

Astuce: normaliser les pixels /255.0 et convertir en forme (n, 28, 28, 1).

Architecture

Empilement Conv2D→MaxPool ×2 + Conv2D.

Bloc dense 128 ReLU + Dropout(0.5).

Sortie softmax (10 classes 0–9).

Bonnes pratiques

- Data aug. légère si surapprentissage.
- Surveiller val_accuracy et early stopping.
- Exporter le modèle pour inference.

Applications finance des CNN



OCR relevés bancaires

Extraction automatique IBAN, RIB, montants, dates, noms depuis scans/photos de documents bancaires.

ROI : -80% temps

OCR



Extraction de tableaux

Parsing de factures/rapports PDF, détection lignes/colonnes, extraction de données structurées.

Factures/PDF

Comptabilité automatisée



Tampons / signatures

Vérification d'authenticité, conformité documentaire KYC, détection de fraude documentaire.

KYC & conformité

Anti-fraude



Graphiques financiers

Détection de patterns techniques (têtes-épaules, double bottom), tendances visuelles et signaux trading.

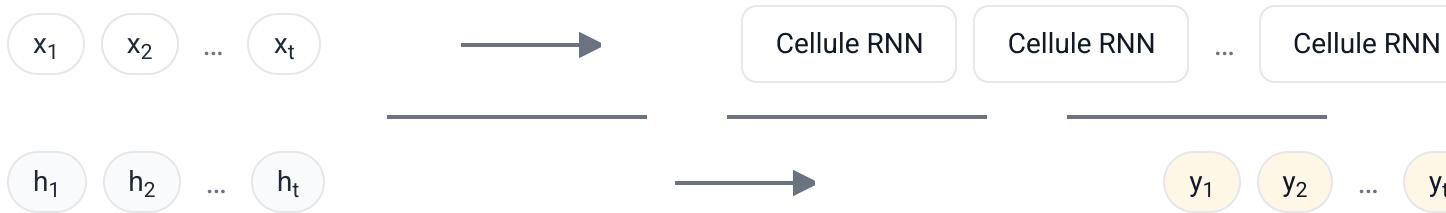
Patterns techniques

Signal trading

RNN – Architecture et dynamique des états



Schéma de flux temporel



Entrées

Séquence x_1, x_2, \dots, x_t

États cachés

h_t transporte la mémoire

Sorties

y_t selon la tâche (clf/régression)

Formules (par pas t)

$$h_t = f(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t + b_h)$$

avec $f = \tanh / \text{ReLU}$

$$y_t = g(W_{hy} \cdot h_t + b_y)$$

avec $g = \text{softmax (clf)} / \text{linéaire (régr.)}$

Flux d'information

Dépendances **courtes** (récentes) vs **longues** (anciennes).

LSTM/GRU aident pour les longues dépendances.

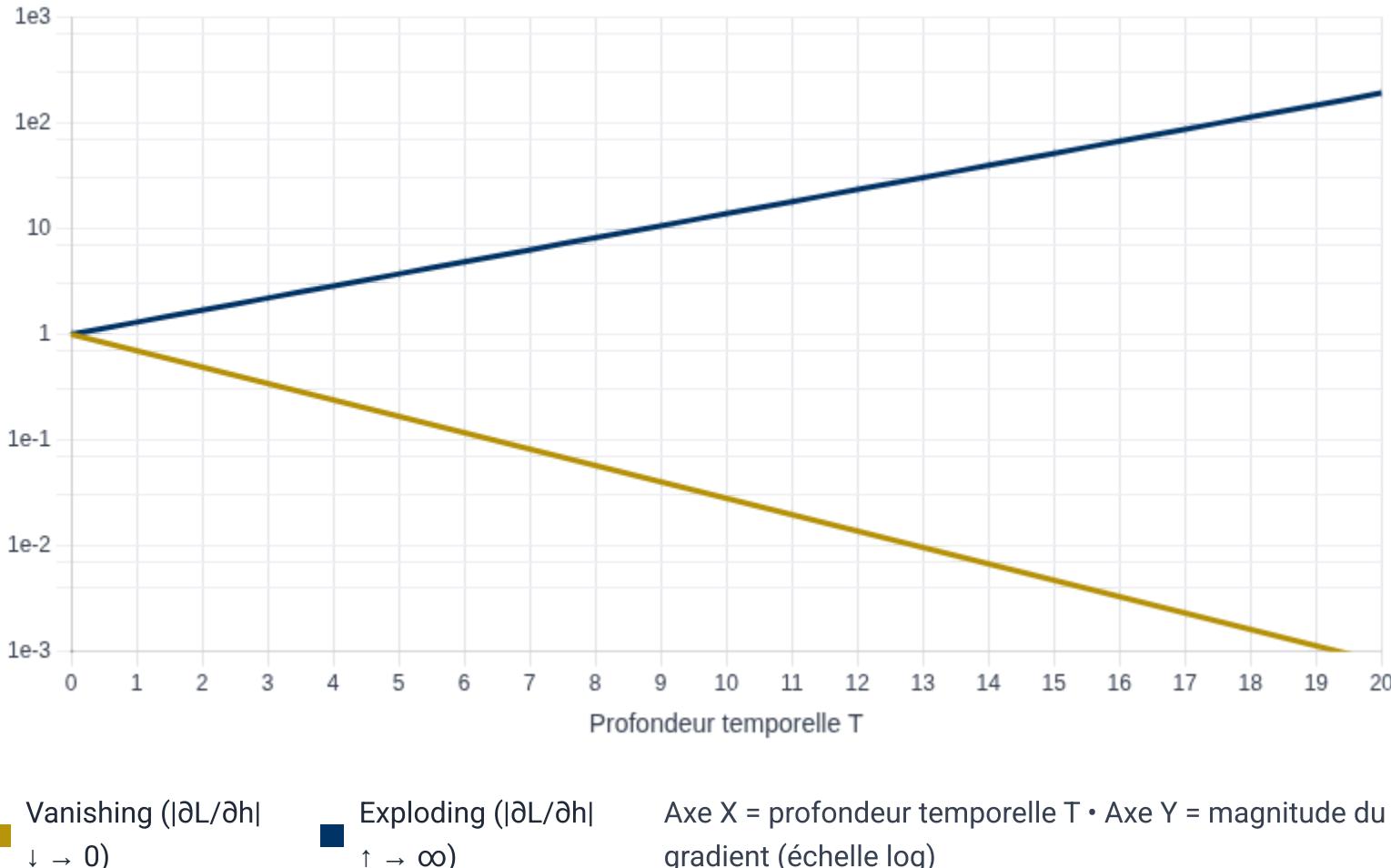
Bonnes pratiques

- Normaliser les features temporelles.
- Limiter la longueur de séquence (fenêtre).
- Utiliser **gradient clipping** si instable.

Vanishing/Exploding gradients – problème et remèdes



Cause (BPTT) : produits répétés de dérivées <1 ou >1 à travers le temps. Vanishing : dérivées <1 \rightarrow produit $\rightarrow 0$ (apprentissage très lent). Exploding : dérivées >1 \rightarrow produit $\rightarrow \infty$ (instabilité).



Remèdes recommandés

- 1) Cellules à portes
LSTM / GRU (gates contrôlent le flux d'information).
- 2) Activations adaptées
ReLU / LeakyReLU pour limiter la saturation.
- 3) Initialisation des poids
He (ReLU) / Xavier (tanh) pour des variances stables.
- 4) Normalisation
Batch Normalization pour stabiliser les gradients.
- 5) Gradient clipping
Borner les gradients dans $[-\theta, \theta]$ pour éviter la divergence.

Couleurs ENCG : Bleu #003366 • Doré #b7950b

LSTM – Équations des portes (gates)



x¹

Formulation complète des portes et états

Portes (gates)

$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$ – oublier infos passées

$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$ – sélectionner nouvelles infos

$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$ – infos potentielles

États (cellule et caché)

$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$ – mémoire long terme

$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$ – filtrer la sortie

$h_t = o_t \odot \tanh(c_t)$ – état caché (sortie)

 σ = sigmoïde $\in (0,1)$

 \odot = produit Hadamard (élément par élément)

 \tanh $\in (-1, 1)$



Intuition – Contrôle du flux d'information

Les « gates » apprennent à **oublier** (f_t) ce qui est inutile, à **ajouter** (i_t, \tilde{c}_t) les informations pertinentes dans la mémoire c_t , puis à **sortir** (o_t) la bonne partie via h_t . Cela permet de capturer des dépendances **long terme** dans les séquences.

Équations GRU

- $z_t = \sigma(W_z[h_{t-1}, x_t])$
Update gate (mélange passé/présent)
- $r_t = \sigma(W_r[h_{t-1}, x_t])$
Reset gate (contrôle l'usage du passé)
- $\tilde{h}_t = \tanh(W_h[r_t \odot h_{t-1}, x_t])$
État candidat (nouvelle information)
- $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$
Interpolation linéaire entre passé et candidat

σ = sigmoïde, \odot = produit de Hadamard (élément par élément)

Interprétation & comparaison

- GRU = simplification du LSTM
2 portes (update, reset) vs 3 portes pour LSTM → moins de paramètres
- Entraînement plus rapide
Bonne performance sur de nombreuses tâches avec moins de calcul
- Choix pratique
GRU si ressources limitées ou données de taille moyenne; LSTM si dépendances longues/complexes
- Usages typiques
Séries temporelles (finance, supply chain), NLP (séquences de mots)

Conseil: commencer par GRU comme baseline, valider puis tester LSTM si nécessaire.

Code Keras LSTM – Prédiction de cours boursiers



Extrait de code – LSTM pour séries temporelles financières (OHLCV)

```
# Imports principaux
from tensorflow.keras import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
import numpy as np

# Préparation des données (fenêtre glissante de 60 jours, 5 features = OHLCV)
window, n_features = 60, 5 # X_train.shape = (samples, 60, 5) ; y_train.shape = (samples,)

# Architecture LSTM pour régression (prédiction du prix/jour suivant)
model = Sequential([
    LSTM(64, return_sequences=True, input_shape=(window, n_features)),
    Dropout(0.2),
    LSTM(32, return_sequences=False),
    Dropout(0.2),
    Dense(16, activation='relu'),
    Dense(1) # prédiction scalaire du prix
])

# Compilation (régression) : MSE + MAE comme métrique
model.compile(optimizer=Adam(1e-3), loss='mse', metrics=['mae'])

# Entraînement
history = model.fit(X_train, y_train,
                      epochs=20, batch_size=64,
                      validation_data=(X_val, y_val), verbose=1)

# Prédiction et évaluation hors échantillon
y_pred = model.predict(X_test)
print(f"MAE: {np.mean(np.abs(y_test - y_pred)):.2f}")
```

Astuce: normaliser/standardiser les features; conserver l'ordre temporel (pas de shuffle pour tests).

Hypothèses & données

Fenêtre: **60 jours**, 5 variables (OHLCV).

Cible: **prix du jour suivant** (régression).

Validation: split chronologique (train → val → test).

Bonnes pratiques

- Éviter la fuite de données (scaler fit sur train seul).
- Évaluer avec **MAE/RMSE** et backtesting.
- Ajouter **dropout** et **early stopping**.

Applications RNN/LSTM en finance



Trading algorithmique

Classification des signaux (achat/vente/neutre) sur séquences de prix, détection de motifs temporels, backtesting de stratégies LSTM.

Sharpe

Rendement annualisé



Prévision de la volatilité

Prédire la volatilité réalisée/implicite (GARCH + LSTM), pilotage du risque (VaR), horizons intraday à mensuel.

VaR

GARCH+LSTM



Analyse de sentiment temporel

Séries d'opinions (news, réseaux, rapports) et corrélation aux mouvements de prix : NLP + LSTM pour suivre l'évolution du sentiment.

NLP+LSTM

Signal marché



Prévision demande / flux

Ventes et flux de trésorerie (supply chain) pour optimiser stocks; KPI : RMSE, MAPE, IC 95%.

RMSE • MAPE

IC 95%



Avantage LSTM : capture des dépendances temporelles longues pour des prévisions plus robustes.



Équations essentielles

Projections linéaires

$$Q = X W_Q, \quad K = X W_K, \quad V = X W_V$$

$X \in \mathbb{R}^{T \times d}$ est la séquence d'entrée ; W_Q, W_K, W_V sont des matrices de paramètres.

Scaled dot-product attention

$$\text{Attention}(Q, K, V) = \text{softmax}(Q K^T / \sqrt{d_k}) V$$

QK^T produit des scores de similarité ; $\sqrt{d_k}$ normalise ; softmax donne des poids $\in [0,1]$ qui somment à 1.



Rôle du mécanisme

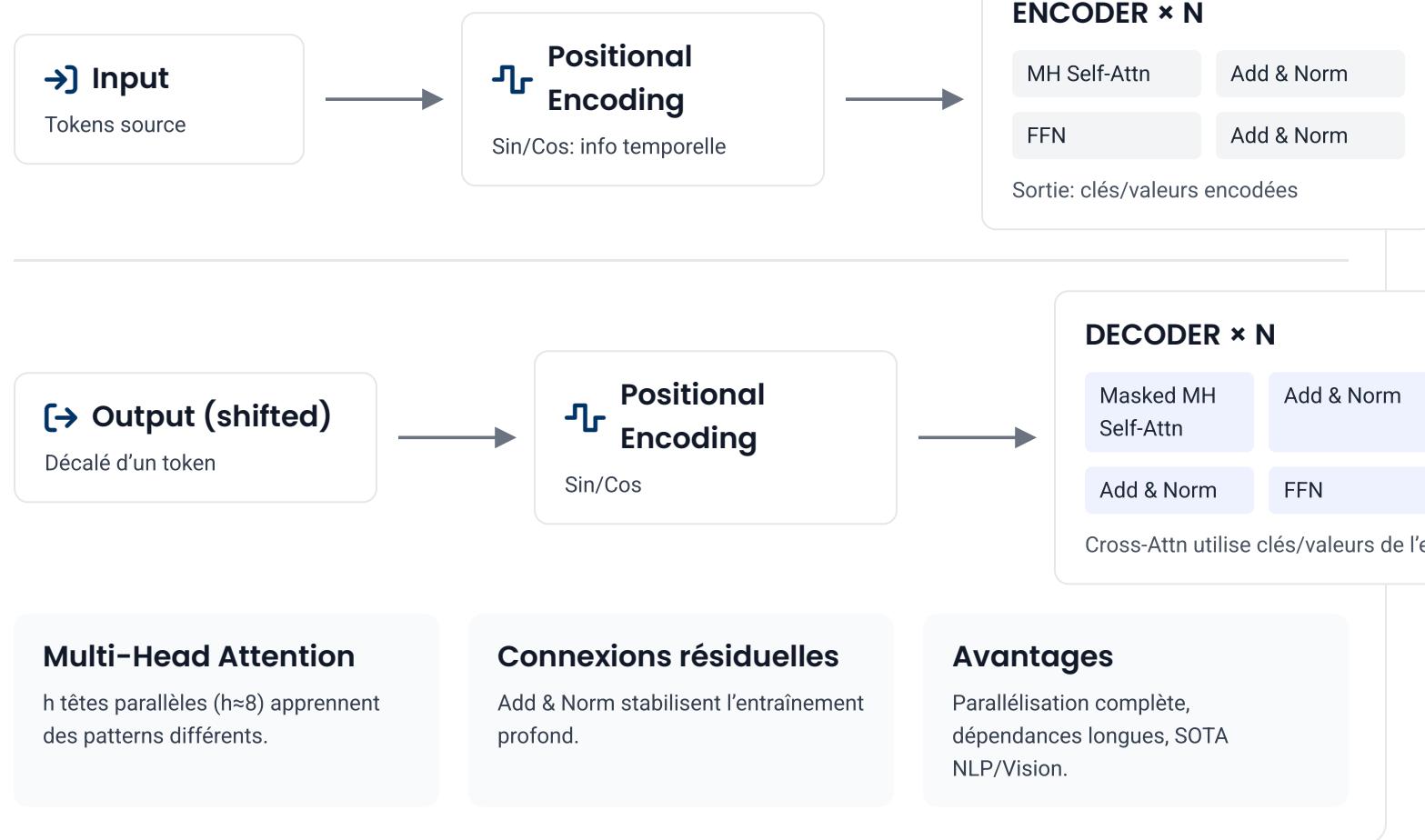
Il pondère automatiquement le contexte pertinent : chaque position *query* « attend » aux *keys* pertinentes, capture des dépendances longues sans récurrence et se parallélise efficacement (contrairement aux RNN). La sortie est une combinaison pondérée des *values*.

Note : d_k désigne la dimension des *keys*.

Transformers – Architecture (Encoder/Decoder, Multi-Head)



Schéma Encoder–Decoder



Composants clés

- **Positional encodings** injectent l'ordre (sin/cos).
- **MH Self-Attn** pondère le contexte pertinent par token.
- **Cross-Attn** (décoder) récupère l'info de l'encoder.
- **FFN** (MLP positionnel) affine la représentation.
- **Add & LayerNorm** stabilisent les sous-couches.

À retenir

Masked self-attention empêche de « voir » les futurs tokens en génération.

Usages

Traduction, résumé, QA/NLU, Time Series Transformer, vision (ViT).

Transformers pour séries temporelles (PyTorch)



Exemple – Transformer Encoder pour séries temporelles

```
# Installation: pip install torch (si nécessaire)
import torch
import torch.nn as nn

# Hyperparamètres du modèle
d_model, nhead, num_layers = 64, 4, 3

# Couche de base: EncoderLayer (batch_first=True → (batch, seq, feat))
encoder_layer = nn.TransformerEncoderLayer(
    d_model=d_model,
    nhead=nhead,
    dim_feedforward=128,
    dropout=0.1,
    activation='relu',
    batch_first=True,
)
encoder = nn.TransformerEncoder(encoder_layer, num_layers=num_layers)

# Couche de sortie: prédiction scalaire
output_layer = nn.Linear(d_model, 1)

# Forward pass – entrée factice: (batch=32, time=50, d_model=64)
x = torch.randn(32, 50, d_model)
z = encoder(x)                      # → (32, 50, 64)
out = output_layer(z[:, -1, :])      # dernière position temporelle
print(out.shape)                    # torch.Size([32, 1])

# Usage: prévision de séries (ventes, prix), classification temporelle, anomalies
```

Astuce: pensez aux encodages positionnels et aux masques d'attention pour les séquences.

Principe & dimensions

Self-attention capture dépendances longues sans récurrence.

Entrée (B, T, d); sortie (B, T, d); on sélectionne $z[:, -1, :]$ pour la prédiction next-step.

Alternative: pooling attention/global sur l'axe temps.

Bonnes pratiques

- Normaliser les features; gérer les valeurs manquantes.
- Ajouter encodage positionnel; utiliser masques causaux si besoin.
- Régularisation: dropout, early stopping; warmup du LR.

BERT vs GPT – Comparaison



| Critère | BERT | GPT | Usages |
|-----------------------------|--|--|--|
| Architecture | Encodeur Transformer bidirectionnel. | Décodeur Transformer auto-régressif unidirectionnel. | Classification/Extraction vs Génération de texte. |
| Pré-entraînement | MLM (Masked LM) + NSP (Next Sentence Prediction). | Prédiction du prochain token (causale, gauche → droite). | Compréhension du contexte vs Génération fluide. |
| Contexte | Bidirectionnel (voit passé + futur simultanément). | Unidirectionnel (ne voit que le passé). | Q/A, NER, classification vs Chat, rédaction, complétion. |
| Fine-tuning | Ajouter une couche sortie spécifique à la tâche. | Few-shot prompting ou fine-tuning. | Tâches supervisées vs Zero/Few-shot. |
| Applications finance | FinBERT pour sentiment des rapports/actualités. | Génération de rapports et résumés automatiques. | Analyse (BERT) vs Synthèse (GPT). |

Transformers – Applications finance



Sentiment financier (FinBERT)

Classification du sentiment de rapports/news/tweets; signal trading corrélé aux prix; fine-tuning BERT sur corpus financier.

F1-score

Corrélation rendements



RAG documents (Q/R analyste)

Recherche sémantique dans rapports/politiques; copilote Q/R; combinaison retrieval + génération (GPT) pour réponses sourcées.

RAG

Qualité des sources



NER KYC/AML (conformité)

Extraction d'entités (personnes, entreprises, montants, dates, risques) dans documents; automatisation screening sanctions; BERT affiné.

KYC/AML

NER



Prévision séries (Transformer)

Alternative à LSTM pour séries financières; attention sur patterns saisonniers; prédition multi-horizon (1j, 1m, 3m).

Multi-horizon

Saisonnalité

Transfer Learning – Principes et bénéfices



- **Feature extraction** : geler le backbone pré-entraîné (*weights frozen*), entraîner seulement la tête de classification (dernières couches) → rapide, peu de données requises.
- **Fine-tuning** : dégeler partiellement les *top-k* couches du backbone avec un *learning rate* faible (1e-5 à 1e-4) pour adapter les features au domaine cible.
- **Stratégie** : commencer par la *feature extraction* → passer à un *fine-tuning* progressif si besoin (dégel par blocs, LR plus faible).
- **Bénéfices** : peu de données nécessaires (centaines vs dizaines de milliers), convergence rapide (moins d'epochs), meilleure généralisation vs *from scratch*, réutilisation de connaissances (ex. ImageNet → documents financiers).
- **Risques à gérer** : *domain shift* (source/target trop différents), surapprentissage sur petit dataset (régularisation, arrêt précoce), choix d'architecture pré-entraînée pertinent (ResNet, VGG, BERT selon la tâche).

Code Keras – Transfer Learning (VGG16)



Extrait de code – Feature extraction puis fine-tuning

```
# Imports
from tensorflow.keras.applications import VGG16
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam

# 1) Charger VGG16 pré-entraîné (sans top) et geler le backbone
base_model = VGG16(weights='imagenet', include_top=False,
                     input_shape=(224, 224, 3))
base_model.trainable = False

# 2) Construire le modèle complet (tête personnalisée)
K = 5 # nombre de classes cibles
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(K, activation='softmax'),
])

# 3) Phase 1 – Feature extraction
model.compile(optimizer=Adam(1e-3),
               loss='categorical_crossentropy',
               metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, validation_data=(x_val, y_val))

# 4) Phase 2 – Fine-tuning (débloquer les derniers blocs)
base_model.trainable = True
for layer in base_model.layers[:15]:
    layer.trainable = False

model.compile(optimizer=Adam(1e-5),
               loss='categorical_crossentropy',
               metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, validation_data=(x_val, y_val))
```

Astuce: utiliser GlobalAveragePooling2D pour réduire les paramètres avant la tête Dense.

Lecture rapide

Backbone VGG16 gelé → GAP → Dense(256) → Dropout(0.5) → Softmax(K).

Deux phases: feature extraction (LR 1e-3) puis fine-tuning (LR 1e-5).

Bonnes pratiques

- Toujours un learning_rate faible en fine-tuning.
- Data augmentation + early stopping.
- Geler les premiers blocs, n'en dégeler que quelques-uns.

Étude de cas – Prévision de défaillance d'entreprises (Deep Learning)



Pipeline en 6 étapes – des données au déploiement

1 Données

Ratios financiers (liquidité, solvabilité, rentabilité)
Délais de paiement fournisseurs/clients
Texte rapports annuels (MD&A)
Séries trimestrielles sur 2 ans

2 Prétraitement

StandardScaler pour ratios
Fenêtre glissante 8 trimestres
Tokenization + embeddings (texte)
SMOTE pour classes déséquilibrées

3 Modèles testés

MLP (baseline ratios)
LSTM (séquences temporelles)
Transformer (attention temporelle)
Ensemble stacking (combinaison)

4 Évaluation

ROC-AUC (discrimination)
PR-AUC (classes rares)
Brier score (calibration)
Courbes ROC/PR et seuils

5 XAI

SHAP values (features clés)
Analyse FP (fausses alertes)
Analyse FN (défaillances manquées)
Explications par instance

6 Déploiement

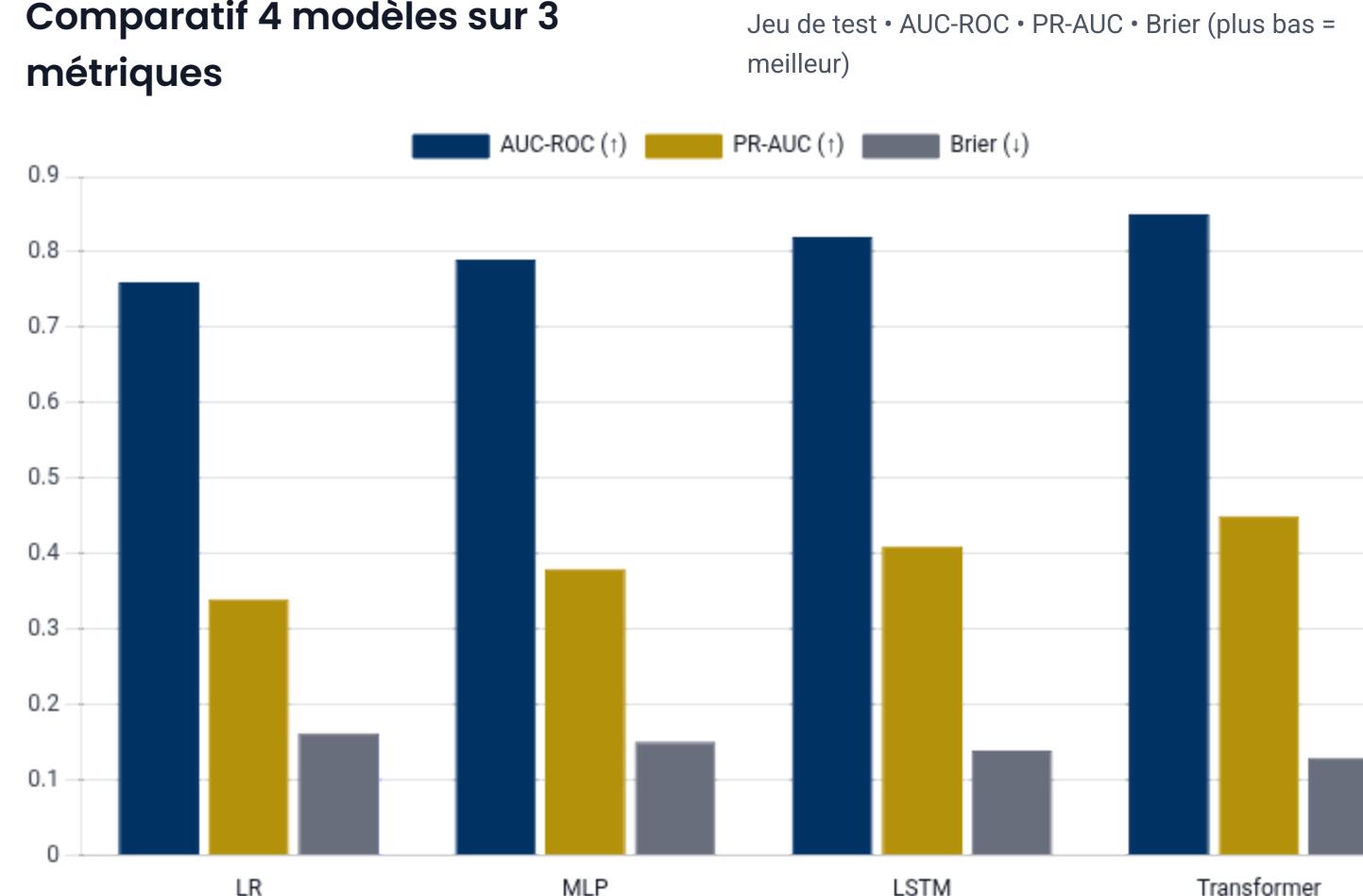
API REST Flask (scoring temps réel)
Monitoring drift trimestriel
Retraining annuel
Dashboard alertes défaillance

Objectif métier : anticiper la probabilité de défaillance (12 mois) pour ajuster octroi, limites et provisions.

Résultats – Métriques et interprétation



Comparatif 4 modèles sur 3 métriques



Lecture : barres groupées par modèle (LR, MLP, LSTM, Transformer). Les métriques AUC-ROC et PR-AUC sont « plus haut = meilleur » tandis que le score de **Brier** est « plus bas = meilleur ».

Note : la **calibration** des probabilités peut être améliorée (ex. Platt scaling) si le Brier est perfectible.

Interprétation clé

- Le **Deep Learning** (LSTM/Transformer) dépasse la baseline LR de **+9 à +11 pts AUC**.
- **Transformer** est le meilleur: capture des dépendances complexes.
- **LSTM** = bon compromis performance/complexité.

Erreurs & seuils

- **FP** (fausses alertes) = entreprises saines classées défaillantes → coût d'opportunité.
- **FN** (défaillances manquées) → pertes potentielles.
- Ajuster le **seuil** selon les coûts métier (Precision/Recall).

Recommandation

- Déployer le **Transformer** avec **monitoring** continu (drift, calibration).
- Vérifier la calibration (Brier) et appliquer *Platt scaling* si besoin.

Quiz interactif – Chapitre 5 (1/2)



1

Connexions résiduelles (skip connections)

Pourquoi aident-elles à entraîner des réseaux profonds ?

(Réponse : contournent des couches, laissent le gradient circuler, évitent le vanishing gradient, facilitent l'apprentissage de l'identité)

2

Attention « scaled dot-product »

Donnez la formule de l'attention.

(Réponse : $\text{Attention}(Q, K, V) = \text{softmax}(QK^T / \sqrt{d_k}) \cdot V$)

3

LSTM vs GRU

Quelle est la différence clé entre ces deux cellules ?

(Réponse : LSTM = 3 portes + état de cellule séparé ; GRU = 2 portes + état caché unique, plus simple, paramètres moindres)

4

Transfer learning

Quand le préférer au training from scratch ?

(Réponse : peu de données cibles, domaines proches, ressources limitées, besoin de convergence rapide)

5

ReLU vs Tanh

Quels avantages et inconvénients pour l'entraînement ?

(Réponse : ReLU – simple, rapide, pas de vanishing côté positif, risque « dying ReLU » ; Tanh – centrée 0, gradient symétrique, mais saturation aux extrêmes)

Quiz interactif – Chapitre 5 (2/2)



1

Applications des CNN en finance

Citez deux applications. (Ex. OCR de relevés bancaires; analyse de graphiques financiers pour détecter des patterns techniques.)

2

Interprétation de h^t dans un RNN

Que représente h^t ? (État caché au temps t encodant l'information de la séquence jusqu'à t .)

3

Pourquoi le vanishing gradient en RNN ?

Expliquez la cause. (BPTT multiplie des dérivées < 1 sur de nombreuses étapes; le produit tend vers 0 de façon exponentielle.)

4

BERT vs GPT – tâches respectives

Quelles tâches cible chaque modèle ? (BERT : compréhension/cls, sentiment, NER, Q/A; GPT : génération de texte, chat, résumés.)

5

KPI pour défaut rare

Quel indicateur privilégier ? (PR-AUC, car plus informatif que ROC-AUC sur données très déséquilibrées; focalisé sur la classe positive rare.)

Introduction au NLP – définitions et contexte



- Définition – NLP (Traitement automatique du langage) : techniques permettant aux machines de comprendre, analyser et générer du texte/langage humain.
- Défis – ambiguïté et contexte, polysémie, idiomes, qualité/bruit des données, multilinguisme et domaine spécifique (finance).
- Enablers – corpus massifs (web/rapports), représentations vectorielles (TF-IDF, embeddings), architectures modernes (Transformers).
- Cas d'usage – classification de textes, analyse de sentiment, extraction d'entités (NER), résumé, Q&A, RAG (retrieval-augmented generation).
- Intérêt business (finance/gestion) – automatiser la veille et la conformité, accélérer l'analyse de rapports, améliorer le service client, appuyer la décision.

Applications du NLP en finance et gestion



Analyse de rapports

Extraction d'insights et de risques depuis rapports annuels, notes d'analystes et PDF.

KPI : F1 / Exactitude

Temps de lecture ↓



Sentiment des marchés

Analyse news, réseaux et communiqués pour capter la polarité et les signaux.

KPI : F1 / PR-AUC

Corrélation rendement



Chatbots bancaires

Assistance 24/7 pour le support client (FAQ, transactions, réclamations).



KPI : Taux de résolution

CSAT / NPS



Extraction d'entités KYC/AML

NER pour noms, adresses, montants et sanctions afin d'automatiser la conformité.

KPI : Précision / Rappel

RGPD & traçabilité

Pipeline NLP – 7 étapes



De la donnée brute au modèle en production (NLP)

1 Collecte

Sources: PDF, CRM, actualités, réseaux. Traçabilité et consentement.

2 Nettoyage

Normaliser casse/ponctuation; gérer accents, HTML, bruit.

3 Tokenisation

Mots/sous-mots; lemmatisation; stop-words adaptés au métier.

4 Représentation

BoW/TF-IDF ou *embeddings* (Word2Vec, BERT) selon la tâche.

5 Modélisation

Baselines (LR/NB) → LSTM/CNN → Transformers (BERT/GPT).

6 Évaluation

F1/PR-AUC sur données déséquilibrées; validation croisée.

7 Déploiement

API/serving, monitoring (drift), logs/traçabilité, mise à jour.

Bonnes pratiques : séparer **train/val/test**, éviter la fuite de données, documenter le pipeline, et prévoir l'**interprétabilité (XAI)** selon le contexte réglementaire.



Principes

- Normalisation: minuscules, gestion des accents (Unicode), ponctuation.
- Tokenisation: mots vs sous-mots (WordPiece/BPE); gérer apostrophes (« c'est », « l'IA »).
- Lemmatisation (recommandée en français) vs stemming (plus agressif).
- Stop-words: liste adaptée au métier; prudence pour mots porteurs (« pas », « non »).
- Chiffres/symboles: conserver s'ils sont informatifs (%), (€), dates, T2).
- Anti-fuite: **après** le split train/val/test, appliquer **fit** des outils sur train uniquement.

Exemples de tokenisation

Exemple 1

Texte: « L'IA, c'est puissant ! »

Mots

```
["IA", "c", "est", "puissant"]
```

Sous-mots (WordPiece)

```
["L", "'", "IA", "'", "c", "'", "est", "puis", "#sant", "!"]
```

Exemple 2

Texte: « Croissance +12,5% au T2 2025 »

Mots

```
["Croissance", "+12,5%", "au", "T2", "2025"]
```

Sous-mots (BPE)

```
["Croissance", "+12,5%", "au", "T2", "2025"]
```

Conseil

Choisir la tokenisation selon le modèle (BoW/LSTM: mots; BERT/GPT: sous-mots).



Concepts clés (gauche)

- Tokenisation : découper le texte en unités (mots, sous-mots).
Ex : WordPiece pour BERT.
- Lemmatisation : ramener un mot à sa forme canonique (ex : « comptables » → « comptable »).
- Stemming vs Lemmatisation : le stemming tronque (plus rapide, moins précis) alors que la lemmatisation utilise l'analyse morpho-syntaxique (plus précise).
- Stop-words : mots fréquents peu informatifs (« le, la, de ») → à filtrer selon le cas d'usage.
- Sensibilité langue & domaine : dictionnaires FR, néologismes métier (finance), chiffres/symboles à conserver si pertinents.
- Objectif : préparer un texte propre et informatif pour la vectorisation (TF-IDF, embeddings).

Code Python – NLTK & spaCy (FR)

Prétraitement

```
# Installer les ressources (hors notebook) :  
# python -m spacy download fr_core_news_sm  
# import nltk; nltk.download('punkt'); nltk.download('stopwords')  
  
from nltk.corpus import stopwords  
from nltk.tokenize import word_tokenize  
import spacy, re  
  
# Exemple de texte financier  
text = "L'intelligence artificielle transforme la finance: \\\nles revenus progressent de 12% YoY, malgré une inflation élevée."  
  
# 1) NLTK : nettoyage simple + stop-words
```

Résultat : une liste de tokens/lemmes exploitable pour TF-IDF, Word2Vec ou Transformers.

Nettoyage et stop-words – bonnes pratiques + code



Bonnes pratiques de nettoyage (fr)

- Normaliser le texte : **lower()**, retirer ponctuation superflue, espaces multiples.
- Chiffres/monnaies : **conserver si utiles** (montants, pourcentages) sinon masquer/retirer.
- Stop-words : base française + **personnalisation métier; garder les négations** ("ne", "pas").
- Lemmatisation > stemming pour le français (meilleure lisibilité sémantique).
- Gérer URL, emails, hashtags, et **tokens courts** (ex. <=2 caractères).
- Encodage/accents et langue : UTF-8, normaliser les accents; détecter la langue si corpus mixte.
- Objectif : nettoyer sans perdre d'informations utiles au modèle.

Code Python – Nettoyage + stop-words (fr)

NLTK
re

```
import re
from nltk.corpus import stopwords

text = "L'IA transforme la finance : +18% de CA en Q2 2025 !
https://ex.me"

def clean(txt: str) -> str:
    t = txt.lower()
    # retirer URLs
    t = re.sub(r"(https?:\/\/[^\s]+)", " ", t)
    # conserver % et symboles monétaires, retirer reste de ponctuation
    t = re.sub(r"[\w\s%\$\u20ac]", " ", t)
    # normaliser espaces
```

Astuce : personnaliser les stop-words métier (ex. "client", "produit") et tester l'impact sur F1/PR-AUC.

Représentation BoW (Sac de mots) – principe et limites



Principe BoW

- Représenter un document par un vecteur de comptages sur un vocabulaire (unigrammes, éventuellement n-grammes).
- Ignorer l'ordre des mots (sac) → focus sur la fréquence des termes.
- Prétraitements utiles : normalisation, stop-words, lemmatisation; pondération possible avec TF-IDF.

Limites et précautions

- Vecteurs très clairsemés (sparse) et vocabulaire potentiellement volumineux → coût mémoire/temps.
- Perte de sémantique et du contexte (synonymie, polysémie non captées).
- Sensibilité au hors-vocabulaire (OOV) et au bruit; préférer embeddings pour capturer la sémantique.

Exemple (mini-corpus) : “profit en hausse” → vecteur [baisse:0, hausse:1, profit:1] ; “baisse du profit” → [baisse:1, hausse:0, profit:1].



Pondération des termes (TF-IDF)

TF – Fréquence de terme

$$tf(t, d) = f / |d|$$

Idée : importance de t dans le document d (fréquence normalisée).

IDF – Rareté globale

$$idf(t) = \log(N / (1 + df(t)))$$

Idée : pénalise les mots fréquents dans le corpus (lissage avec +1).

Score TF-IDF

$$tfidf(t, d) = tf(t, d) \times idf(t)$$

Effet : met en avant les termes caractéristiques d'un document.



Bonnes pratiques d'usage

Filtrer les mots trop fréquents (max_df) et trop rares (min_df), tester les n-grams (ex. (1,2)), normaliser/standardiser les vecteurs si requis et supprimer les stop-words métier.

Vectorisation BoW/TF-IDF (scikit-learn)



Extrait de code – Vectorisation avec CountVectorizer et TfidfVectorizer

```
# 1) Importer les outils scikit-learn
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

# 2) Corpus jouet (fr)
corpus = [
    "Le chiffre d'affaires progresse fortement",
    "La marge opérationnelle se contracte",
    "Le marché anticipate une hausse des revenus"
]

# 3) Sac de mots (BoW) – comptage d'occurrences + bigrammes
bow_vec = CountVectorizer(ngram_range=(1, 2), min_df=1)
X_bow = bow_vec.fit_transform(corpus)
print("BoW shape:", X_bow.shape)
print("Features BoW (extraits):", bow_vec.get_feature_names_out()[:10])

# 4) TF-IDF – pondération par rareté (idf) des termes
tfidf_vec = TfidfVectorizer(ngram_range=(1, 1), min_df=1, norm="l2")
X_tfidf = tfidf_vec.fit_transform(corpus)
print("TF-IDF shape:", X_tfidf.shape)
print("Exemple poids TF-IDF (doc0):", X_tfidf[0].toarray()[0][:8])

# 5) Bonnes pratiques :
# - Nettoyer/normaliser le texte (casse, ponctuation, stop-words)
# - Ajuster ngram_range et min_df selon le corpus et la tâche
# - Utiliser TF-IDF si les mots fréquents dominent le signal
```

Astuce: utilisez ngram_range pour capter des expressions (bigrammes), et min_df pour filtrer le bruit.

Lecture rapide

BoW = vecteurs de comptage;
dimension = vocabulaire.

TF-IDF = pondère les termes rares
mais discriminants.

Sorties creuses (CSR) – efficaces
en mémoire.

Bonnes pratiques

- Personnaliser les stop-words métier (FR/EN).
- Évaluer avec F1/PR-AUC si classes déséquilibrées.
- Conserver un vocab figé pour la production.

Word embeddings – Word2Vec, GloVe : idées clés et avantages



Idées clés

- Vecteurs denses qui représentent le sens des mots (p. ex. 50–300 dimensions)
- Word2Vec (CBOW/Skip-gram) : apprend à partir des contextes, comparaison via similarité cosinus
- GloVe : factorise les cooccurrences globales pour capter structure sémantique

Avantages et usages (finance/gestion)

- Captent similarité/analogies (roi – homme + femme ≈ reine) pour améliorer les modèles
- Usages : similarité de documents, clustering de thèmes, variables d'entrée pour classification/sentiment
- Limites : OOV et polysémie ; alternatives: sous-mots (FastText) ou embeddings contextuels (BERT)

Word2Vec – Étapes et visualisation t-SNE



Pipeline Word2Vec (4 étapes)

1. Préparer le corpus : nettoyage, tokenisation, langue (fr), stop-words, lemmatisation.
 2. Entraîner le modèle **Word2Vec** (gensim) : **vector_size**, **window**, **min_count**, **sg** (CBOW=0, Skip-gram=1).
 3. Extraire les vecteurs et mesurer les similarités (cosinus), analogies sémantiques.
 4. Réduire la dimension (t-SNE/UMAP) et **visualiser** les clusters sémantiques.
- Bonnes pratiques : corpus suffisant, paramètres stables, fixer le **random_state** de t-SNE (perplexity 5–50).

Code Python – gensim + t-SNE

gensim • scikit-learn • matplotlib

```
from gensim.models import Word2Vec
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# 1) Corpus déjà tokenisé (phrases de tokens)
sentences = [
    ["banque", "crédit", "client", "intérêt"],
    ["marché", "action", "rendement", "risque"],
    ["assurance", "prime", "sinistre", "indemnité"]
]

# 2) Entraînement Word2Vec (Skip-gram sg=1)
w2v = Word2Vec(
```

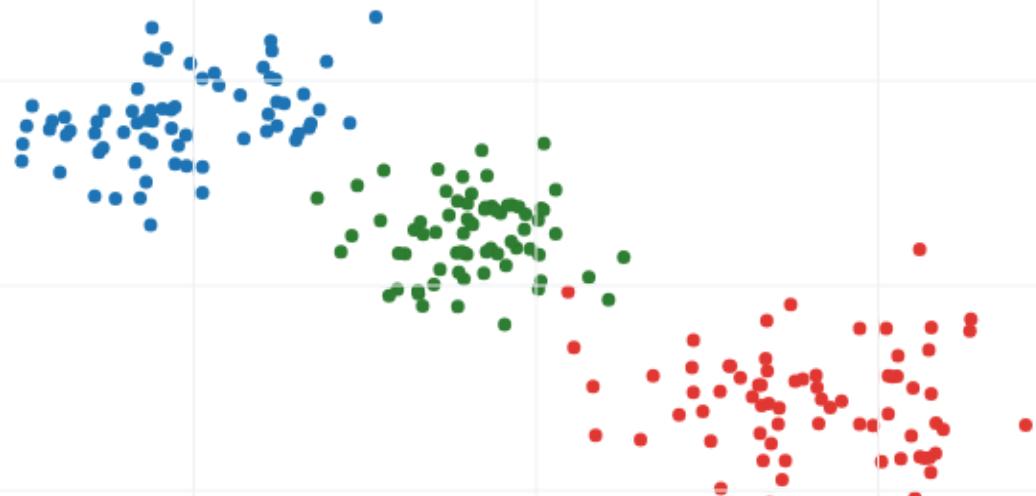
Lecture : des mots proches dans la figure partagent des contextes similaires (proximité sémantique).

t-SNE / UMAP – Visualisation de l'espace sémantique



Nuage 2D – embeddings (ex. Word2Vec/BERT) projetés avec t-SNE/UMAP

Projection (ex.) : t-SNE, perplexité = 30



● Cluster A ● Cluster B ● Cluster C □ Technique : t-SNE (perplexité = 30)

Idée clé

Réduire des vecteurs denses ($d = 100\text{--}768$) en 2D/3D tout en **préservant les voisinages** pour observer des **clusters sémantiques**.

Paramètres clés

- **t-SNE** : perplexité 5–50, itérations ≥ 1000
- **UMAP** : $n_{\text{neighbors}}$ 15–50, min_dist 0.0–0.5
- Échantillonnage si n est grand; standardiser les vecteurs

Comparaison rapide

- **t-SNE** : structure locale excellente; pas de transform out-of-sample direct
- **UMAP** : plus rapide, local + global; transform approx. disponible

NLP – Comparaison RNN/LSTM • CNN-texte • Transformers



| Critère | RNN / LSTM | CNN-texte | Transformers |
|----------------------------|--|---|---|
| Dépendances longues | Bonnes avec LSTM/GRU; RNN simples limités. | Limitées (fenêtres n-gram locales). | Excellent via attention globale. |
| Parallélisation | Faible (séquentiel dans le temps). | Bonne (convolutions parallèles). | Très bonne (entièrement parallélisable). |
| Données requises | Modérées (bonnes sur corpus moyens). | Modérées (baselines efficaces). | Élevées; pré-entraînement recommandé. |
| Vitesse / Latence | Entraînement lent; inférence moyenne. | Rapide (kernels conv optimisés). | Coût élevé; distillation/quantisation utiles. |
| Performance typique | Correcte; solide sur séquences structurées. | Bon baseline textes courts. | SOTA sur nombreuses tâches NLP. |
| Cas d'usage | Séquences temporelles, sentiment séquentiel. | Classification, détection d'intent, n-gram. | Q/A, NER, résumé, RAG, traduction. |
| Limites | Vanishing gradient; peu parallélisables. | Contexte global réduit; fenêtrage fixe. | Coût calcul/mémoire; taille modèle. |

LSTM (Keras) – Classification de textes



Extrait de code – LSTM (Keras) pour classification binaire de textes

```
# 1) Imports
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# 2) Paramètres
MAX_WORDS = 20000 # vocabulaire maximum
MAX_LEN = 200 # longueur de séquence (padding)

# 3) Tokenisation + séquences
tok = Tokenizer(num_words=MAX_WORDS, oov_token="")
tok.fit_on_texts(train_texts)
Xtr = pad_sequences(tok.texts_to_sequences(train_texts), maxlen=MAX_LEN)
Xte = pad_sequences(tok.texts_to_sequences(test_texts), maxlen=MAX_LEN)

# 4) Modèle LSTM
model = Sequential([
    Embedding(input_dim=MAX_WORDS, output_dim=128, input_length=MAX_LEN),
    LSTM(64),
    Dense(1, activation="sigmoid") # binaire
])

# 5) Entraînement
model.compile(optimizer="adam",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.fit(Xtr, y_train,
          epochs=3, batch_size=64, validation_split=0.2, verbose=1)

# 6) Évaluation
print("Accuracy test:", model.evaluate(Xte, y_test, verbose=0)[1])
```

Astuce: utiliser validation_split et early stopping pour limiter le surapprentissage.

À retenir

Embedding apprend des vecteurs de mots adaptés à la tâche.

LSTM(64) capture les dépendances séquentielles du texte.

Sortie **sigmoid** = probabilité pour une **classe positive**.

Bonnes pratiques

- Limiter MAX_WORDS et MAX_LEN pour stabilité/latence.
- Équilibrer les classes; suivre F1/PR-AUC si jeux déséquilibrés.
- Ajouter Dropout et EarlyStopping si nécessaire.



BERT – Compréhension

Encodeur bidirectionnel pour comprendre le contexte (passé+futur).

Usages typiques :

- Classification de textes, Q/A, NER
- Extraction d'informations (rapports)



GPT – Génération

Décodeur auto-régressif pour produire du texte cohérent.

Usages typiques :

- Rédaction, résumé, reformulation
- Assistants et copilotes métiers



FinBERT – Sentiment financier

BERT spécialisé finance pour tonalité des textes financiers.

Usages typiques :

- News/rapports → score de sentiment
- Signal marchés, veille risques

Fine-tuning BERT – pipeline en 6 étapes



Objectif & outils

Adapter BERT à une tâche de classification (ex. sentiment financier) • Transformers (Hugging Face) • KPI: F1, PR-AUC.

1 Préparer le dataset

Texte + labels • split stratifié (train/val/test) • équilibrer si besoin.

Format: CSV/Parquet • colonne texte et cible binaire/multi-classe.

2 Tokenisation WordPiece

Tokenizer BERT • tronquer/padder à **max_length** • casse conforme au modèle.

Sorties: `input_ids`, `attention_mask` (+ `token_type_ids` si besoin).

3 Encodage & DataLoader

Créer jeux **train/val** en lots (batch) • mélanger (shuffle) • gérer longueur.

Batch size typique: 16–32 selon GPU.

4 Entraîner (fine-tuning)

Ajouter une tête de classification • optimiser avec **AdamW**, $lr \approx 2e-5$, 2–4 époques.

Régularisation: dropout • early stopping sur F1 (val).

5 Évaluer & calibrer

F1, PR-AUC (déséquilibre) • matrice de confusion • calibration si nécessaire.

Ajuster seuil selon coûts FP/FN métier.

6 Déployer & moniter

Servir via API REST • suivi dérive (**drift**), ré-entraînement, traçabilité (MLflow).

Conformité: logs, XAI (ex. SHAP) si décision sensible.

Bonnes pratiques – **sélection du modèle**: **bert-base-uncased** (EN) ou **camembert-base** (FR) • vérifier la langue/domaine (FinBERT pour finance).

Transformers (Hugging Face) – Classification de texte



Extrait de code – Pipeline Transformers pour classification

```
# pip install transformers torch --upgrade
from transformers import AutoTokenizer, AutoModelForSequenceClassification, pipeline
import torch

# Modèle pré-entraîné (binaire sentiment) – remplacer par un modèle FR si besoin
model_id = "distilbert-base-uncased-finetuned-sst-2-english"

tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForSequenceClassification.from_pretrained(model_id)

# Utilise le GPU si disponible (device=0), sinon CPU (device=-1)
device = 0 if torch.cuda().is_available() else -1

clf = pipeline(
    "text-classification",
    model=model,
    tokenizer=tokenizer,
    truncation=True,
    padding="longest",
    device=device
)

texts = [
    "Revenue up 18% YoY; guidance raised.",
    "Downgrade issued due to weaker cash flows."
]

preds = clf(texts)
for t, p in zip(texts, preds):
    print(t, "→", p["label"], f"{p['score']:.2%}")
```

Étapes clés

- 1) Charger tokenizer et model.
- 2) Créer le pipeline de classification.
- 3) Inférence sur une liste de textes (batch).

Bonnes pratiques

- Définir truncation/max_length pour des textes longs.
- Utiliser device=0 si GPU disponible.
- Évaluer avec F1 / PR-AUC (données déséquilibrées).

Astuce: pour du français/finance, utiliser un modèle adapté (ex. "tblard/tf-allocine", "yiyangkust/finbert-tone").

Analyse de sentiment financier – Principes & KPI



Principes

- Approches : lexiques vs modèles pré-entraînés (**FinBERT**)
Lexiques = règles simples; FinBERT = contexte financier, meilleure précision.
- Granularité & polarité
Phrase vs document; polarité (positif/négatif/neutre) & intensité.
- Spécificités financières
Vocabulaire métier (guidance, downgrade), chiffres, négations, prudence.
- Prétraitement & langue
Normalisation, lemmatisation; choix FR/EN; adaptation au domaine.
- Production & gouvernance
Latence, explicabilité, contrôle dérive (drift) et ré-entraînement.

KPI & métriques

Données déséquilibrées

- F1-score (classe +)
Bon compromis précision/rappel.
 - PR-AUC
Pertinent si classes rares.
 - Precision@K / Recall@K
Top K alertes/actifs à prioriser.
 - Calibration (Brier)
Probabilités fiables pour décisions.
- Indicateurs business**
- Corrélation sentiment ↔ rendement
 - Stabilité temporelle (drift)
 - Uplift sur stratégie (Sharpe)
 - Latence & couverture sources
- Choix du seuil optimisé par les coûts FP/FN et l'objectif métier.

FinBERT – Inférence rapide sur actualités financières



Extrait de code – Inférence de sentiment financier avec FinBERT (Transformers)

```
# pip install transformers --upgrade
from transformers import AutoTokenizer, AutoModelForSequenceClassification, pipeline

# Modèle FinBERT pré-entraîné pour la tonalité financière
model_id = "yiyangkust/finbert-tone"
clf = pipeline(
    "text-classification",
    model=model_id,
    truncation=True,
    max_length=128,
    return_all_scores=False
)

# Titres d'actualité (EN recommandé pour FinBERT)
news = [
    "Q2 revenue grew 18% YoY while margins expanded.",
    "Company warns of weaker outlook amid rising costs.",
    "Regulator opens investigation into accounting practices."
]

# Inférence
for s in news:
    res = clf(s)[0]
    print(f"{s} → {res['label']} (score={res['score']:.2f})")
```

Astuce: utilisez batch_size et device=0 (GPU) pour accélérer l'inférence en production.

Lecture des résultats

Labels attendus: positive, neutral, negative.

Score: confiance modèle (0–1). Évitez les seuils trop stricts.

Idéal pour **veille marché et alertes** en temps quasi réel.

Bonnes pratiques

- FinBERT est **anglophone** (finances). Traduire les textes FR si besoin.
- Limiter à max_length=128 pour titres/abstracts.
- Sur données locales, prévoir **calibration** et **audit** (biais/domain shift).

Classification de textes – NB • Logistique TF-IDF • Transformers



| Critère | Naïve Bayes (NB) | Régression logistique TF-IDF | Transformers (BERT/FinBERT) |
|--------------------------|---|--|--|
| Représentation | Comptage BoW/TF-IDF, indépendance des termes | Vecteurs TF-IDF avec pondérations par terme | Embeddings contextuels (WordPiece), attention |
| Données nécessaires | Peu à modérées (petits corpus) | Modérées; bonnes bases sur TF-IDF | Plutôt élevées; bénéfice fort du fine-tuning |
| Coût calcul / latence | Très faible; entraînement/inférence rapides CPU | Faible à moyen; déploiement simple | Élevé; souvent GPU pour entraînement/inférence |
| Performance typique (F1) | Baseline correcte; sensible au vocabulaire | Souvent supérieure à NB sur corpus variés | État de l'art sur tâches complexes/contextuelles |
| Interprétabilité | Élevée (poids des termes, probabilités) | Bonne (coefficients, importance des n-grammes) | Faible → recourir à XAI (LIME, SHAP) |
| Cas d'usage recommandés | Baselines rapides, petits jeux de données | Classif. production-ready, compromis perf/coût | Contexte fin (finance, légal), multilingue, RAG |

NER — Reconnaissance d'entités nommées (BIO)



PER — Personnes

Noms de personnes physiques (ex. dirigeants, clients, analystes).



ORG — Organisations

Entreprises, banques, régulateurs, fonds, institutions.



LOC — Lieux

Pays, villes, régions (ex. Maroc, Casablanca, UE).



MISC — Divers

Autres entités (ex. produits, événements, dates, montants).

Code – spaCy NER sur extrait de rapport

</>

Extraction d'entités nommées (NER) sur un texte financier

```
# 1) Installer/télécharger un modèle si nécessaire (à faire une fois) :  
# !python -m spacy download fr_core_news_sm  
  
# 2) Charger spaCy et le modèle français  
import spacy  
nlp = spacy.load("fr_core_news_sm") # ou "en_core_web_sm" pour l'anglais  
  
# 3) Exemple d'extrait de rapport financier  
texte = "La banque Alpha acquiert une fintech à Paris pour 200 M€."  
  
# 4) Traitement et extraction des entités  
doc = nlp(texte)  
for ent in doc.ents:  
    print(ent.text, ent.label_)  
  
# Sortie attendue (peut varier selon le modèle) :  
# "Alpha" ORG | "Paris" LOC | "200 M€" MISC/AMOUNT
```

Conseil: pour l'anglais, utiliser `en_core_web_sm`. Pour des performances supérieures, préférer les modèles `md/lg`.

Sorties attendues

ORG – organisations (banques, entreprises).

LOC – lieux (pays, villes).

MONEY/AMOUNT – montants financiers (ex. 200 M€).

Bonnes pratiques

- Nettoyer/normaliser le texte avant NER.
- Adapter le label mapping aux besoins métier.
- Évaluer via F1 et échantillons annotés.

Cas d'usage NLP en finance – panorama



Extraction KPI depuis PDF

Identifier et extraire CA, EBITDA, dettes dans rapports/Factures.

OCR + NER

Qualité donnée



Veille sentiment marchés

Mesurer la polarité des news/tweets et suivre son impact.

FinBERT

F1 • PR-AUC



Chatbots conformité (KYC/AML)

Répondre aux FAQ, guider les contrôles et tracer les réponses.

RAG + LLM

Traçabilité



Résumé de rapports d'analystes

Produire des synthèses fiables et exploitables rapidement.

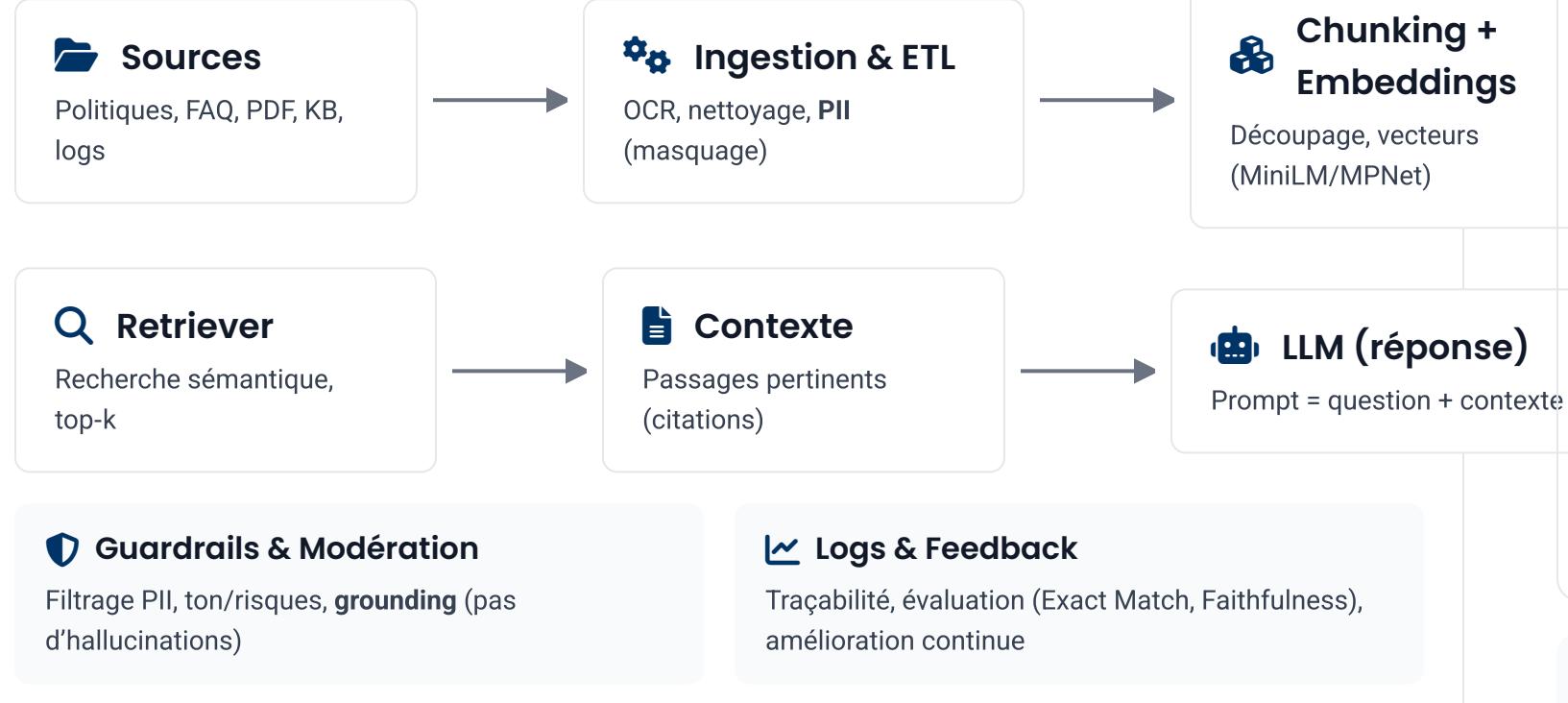
Abstr./Extractif

Gain de temps

Chatbot bancaire (RAG + LLM) – Architecture



Schéma de flux (Retrieval-Augmented Generation)



Composants clés

- Embeddings: **MiniLM/MPNet** (fr/en)
- Vector store: FAISS / Postgres-pgvector
- Retriever: **top-k** + reranking (Cross-Encoder)
- LLM: GPT-like ou **open-source** (LLaMA, Mistral)
- Prompts: system + instructions + contexte



Bonnes pratiques (banque)

- Conformité **RGPD/AI Act**, gestion PII
- Citations de sources & vérifications
- Red teaming & jeux de tests (safety)
- SLO: latence, exactitude, **faithfulness**

Résumé

Le RAG **ancre** la réponse du LLM sur les documents internes → réponses fiables, traçables et conformes.

Quiz interactif – Chapitre 6 (NLP)



1

BoW vs Embeddings

Expliquez la différence entre Sac-de-mots (BoW/TF-IDF) et *word embeddings* (Word2Vec/GloVe/BERT).

2

TF-IDF – formule et intuition

Donnez la formule de TF-IDF et son intuition (termes rares mais discriminants).

3

BERT vs GPT – usages

Quelle est la différence de rôle entre BERT (compréhension) et GPT (génération) en NLP ?

4

NER et tags BIO

Qu'est-ce que la NER et que signifient les tags B-I-O (ex. B-ORG, I-ORG, O) ?

5

Pipeline RAG pour chatbot bancaire

Décrivez brièvement un pipeline RAG : sources → indexation (vecteurs) → retrieval → LLM → contrôle (traçabilité/filtrage).

XAI – Pourquoi expliquer les modèles ?



- Besoin d'interprétation : confiance, adoption métier, diagnostic des erreurs
- Problème de « boîte noire » : modèles complexes (GBM, réseaux profonds) peu transparents
- Bénéfices XAI : conformité, gouvernance des modèles, réduction des risques opérationnels
- Cas d'usage finance : crédit (refus explicables), fraude (raison des alertes), pricing
- Conformité & audit : RGPD art. 22, traçabilité, auditabilité des décisions
- Impact métier : meilleure décision, acceptabilité, alignement avec les équipes risques



Cadre réglementaire (UE/RGPD)

Conformité

- Article 22 RGPD : décisions automatisées et profilage
Droit de ne pas faire l'objet d'une décision fondée exclusivement sur l'IA.
- Transparence & motifs pertinents
Explication des facteurs ayant conduit à la décision (niveau compréhensible).
- Principe de minimisation & finalité
Collecte limitée, usage déterminé, durée de conservation maîtrisée.
- Traçabilité & auditabilité
Journalisation des versions, jeux de données, critères de décision.
- Équité & non-discrimination
Mesure/correction des biais (groupes protégés) et tests d'équité.
- Gouvernance & AI Act
DPO, registre des traitements, DPIA; exigences AI Act selon niveau de risque.

Bénéfices métier de l'explicabilité

Business

- Confiance & adoption
Appui des équipes Risques/Conformité, onboarding des métiers.
- Meilleure décision opérationnelle
Compréhension des « drivers » → ajustement des seuils/règles.
- Réduction des litiges & coûts de conformité
Justification claire des décisions (ex. refus de crédit).
- Qualité & robustesse du modèle
Débogage, détection de drift, priorisation des améliorations.
- Communication client
Motifs standardisés et actionnables pour accompagner les décisions.
- ROI & gouvernance
Accélère l'industrialisation et facilite les comités de validation.

Modèles interprétables vs « boîte noire » — Comparatif



| Critère | Modèles interprétables | Modèles « boîte noire » | Implications |
|----------------------|--|---|---|
| Exemples | Régression logistique, arbres peu profonds, règles, GAM | GBM (XGBoost/LightGBM), Random Forest, DNN/CNN/RNN | Choix modèle selon besoin d'explication vs performance |
| Interprétabilité | Native (coefficients, règles lisibles) | Post-hoc (PDP/ICE, LIME, SHAP, contre-factuels) | Exigences RGPD/risques → traçabilité et justification |
| Performance | Souvent correcte, plafonne sur problèmes complexes | Excellente sur données riches et non linéaires | Trade-off perf. vs explicabilité à arbitrer avec le métier |
| Données & complexité | Peu de données, features limitées, CPU suffisant | Beaucoup de données/features, tuning, souvent GPU | Coûts infra/maintenance plus élevés côté « boîte noire » |
| Cas d'usage | Scoring réglementé, tarification simple, reporting | Fraude, vision/NLP, signaux complexes, recommandation | Imposer XAI et contrôles si modèle « boîte noire » retenu |
| Gouvernance | Explications natives → adoption rapide par métiers/risques | Nécessite documentation XAI, monitoring biais/derives | Facilite audit (LR/arbres) vs cadre XAI formalisé (GBM/DNN) |

Panorama XAI – 4 approches



Importance globale

Mesure la contribution moyenne des variables (Permutation, Gini).

Global

Attention corrélations



PDP / ICE

Effets marginaux d'une variable sur la prédition (moyenne vs individus).

Global & Local

Interactions possibles



LIME

Modèle linéaire local sur données perturbées autour d'un cas.

Local

Post-hoc



SHAP

Attribution fondée sur les valeurs de Shapley (cohérence, additivité).

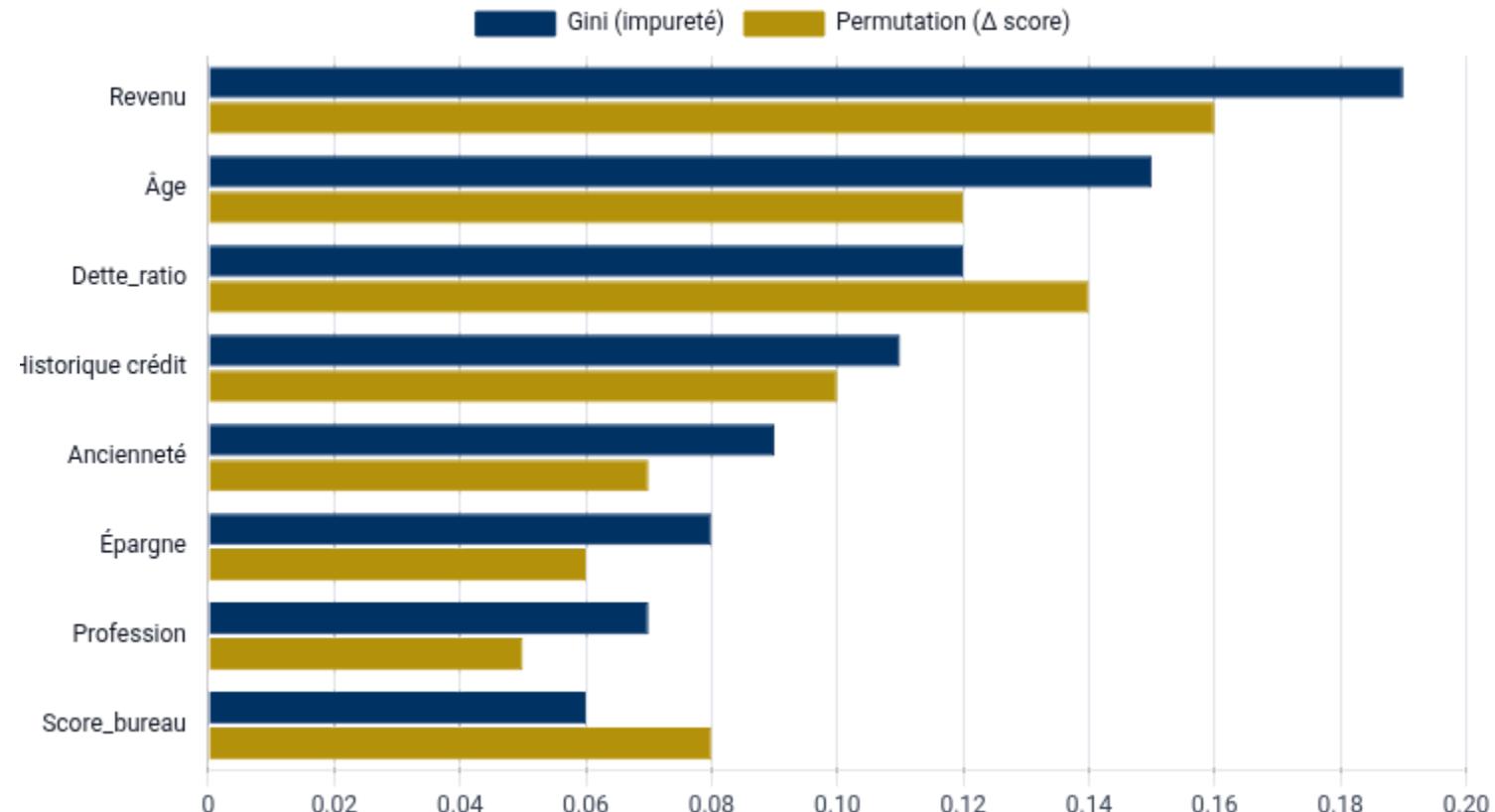
Local & Global

Théorie des jeux

Importance globale des variables — Permutation vs Gini



Top variables — comparaison des importances



Lecture : pour chaque variable, la barre bleue montre l'importance **Gini** (réduction d'impureté), la barre dorée l'importance **par permutation** (baisse de performance en permutant la variable).

Note : Gini et permutation ne sont pas à la même échelle absolue. Comparez surtout le **classement relatif** des variables.

Quand utiliser chaque mesure ?

- **Gini (arbres)** : rapide, disponible nativement pour RF/GBM.
- **Permutation** : plus fidèle en présence de *corrélations*, indépendante du modèle.

Mises en garde

- **Biais** Gini vers variables à forte cardinalité.
- Corrélations : l'importance peut être **partagée** entre features proches.
- Évaluez sur **validation croisée** (AUC/F1/Precision@K).

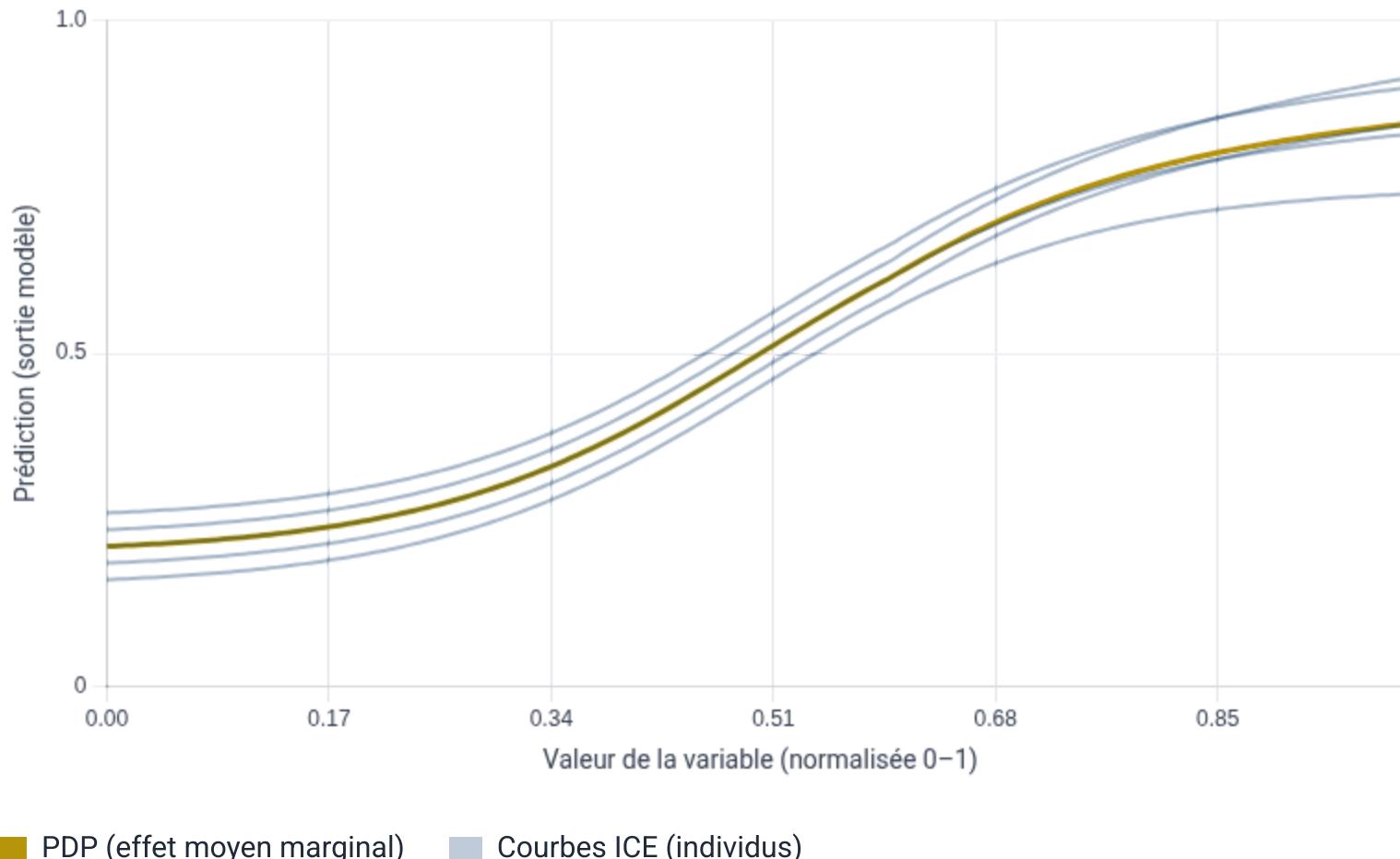
Bonnes pratiques

- Comparer Gini, permutation et compléter par **PDP/SHAP**.
- Vérifier la **stabilité** des importances (ré-échantillonnage).

PDP / ICE – Effets partiels (ex: Ratio_dette)



Variation de la prédiction en fonction d'une variable (PDP = moyenne; ICE = individus)



Interprétation rapide

1) PDP

Effet moyen de la variable en fixant les autres (profil global).

2) ICE

Courbe par individu; révèle interactions et hétérogénéité.

3) Bonnes pratiques

Ne pas extrapoler hors du domaine; standardiser; comparer segments.

Couleurs ENCG : Bleu #003366 • Doré #b7950b

LIME — Principe (explication locale)



Objectif

Approcher localement un modèle complexe f autour d'une instance x_0 par un modèle simple g pour expliquer la prédiction.

1 Perturber x_0

Générer des voisins x' proches de x_0 (échantillonnage dans le voisinage).

Poids de proximité :
 $\pi(x') = \exp(-d(x', x_0)^2 / \sigma^2)$.

2 Prédire avec f

Obtenir les sorties du modèle boîte noire $f(x')$ pour tous les voisins générés.

Conserver probabilités/logits requis par la tâche.

3 Ajuster g local

Ajuster un modèle simple (p.ex. **régression L1**) pondéré par $\pi(x')$.

Minimiser : $\sum \pi(x')[f(x') - g(x')]^2 + \lambda \|w\|_1$.

4 Interpréter

Les poids de g donnent l'**importance locale des features** pour x_0 .

Vérifier la **fidélité locale** (p.ex. R^2 local) et la stabilité.

Résumé : LIME explique **localement** une prédiction en approximant f par un modèle **interprétable** g centré sur x_0 et pondéré par la proximité.

SHAP – Fondements (valeurs de Shapley)

Définition mathématique

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} [|S|! (M - |S| - 1)! / M!] \cdot (f(S \cup \{i\}) - f(S))$$

N = ensemble des features, **M** = |N|

S = sous-ensemble de N sans i

f(·) = prédiction du modèle pour un sous-ensemble

Propriétés des valeurs de Shapley

Efficacité

$\sum_i \phi_i = f(x) - E[f(x)]$ (la somme des contributions reconstitue l'écart à la valeur de référence).

Symétrie

Deux features contribuant de manière identique reçoivent la même attribution.

Nullité

Si une feature n'influence jamais le modèle, alors $\phi_i = 0$.

Additivité

Pour deux modèles additifs, les attributions se somment : $\phi(f_1 + f_2) = \phi(f_1) + \phi(f_2)$.



Interprétation managériale

$\phi_i > 0$ augmente la prédiction; $\phi_i < 0$ la diminue. La somme des ϕ_i plus la valeur de référence (baseline) égale la prédiction du modèle pour l'instance.

SHAP — Explication d'un modèle de crédit (Python)



Extrait de code – SHAP sur modèle Random Forest (scoring crédit)

```
# Installation (si besoin) : pip install shap
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import shap

# 1) Données jouet (profil client) pour scoring crédit
np.random.seed(42)
n = 500
df = pd.DataFrame({
    'age': np.round(np.random.normal(40, 12, n)),
    'revenu': np.round(np.random.normal(3000, 900, n)),
    'ratio_dette': np.round(np.random.uniform(0.1, 0.8, n), 2),
    'historique': np.random.randint(0, 10, n)
})
risque = (-0.02*df['age'] + 0.001*df['revenu'] + 3*df['ratio_dette'] - 0.2*df['historique'])
y = (risque > 0).astype("int")

X_train, X_test, y_train, y_test = train_test_split(df, y, test_size=0.2, random_state=42)

# 2) Modèle et ajustement
rf = RandomForestClassifier(n_estimators=300, random_state=42).fit(X_train, y_train)

# 3) Explicabilité avec SHAP (version unifiée)
explainer = shap.Explainer(rf, X_train)
shap_values = explainer(X_test)

# 4) Visualisations (à exécuter dans Jupyter)
# shap.plots.beeswarm(shap_values, max_display=10)    # importance + sens
# shap.plots.waterfall(shap_values[0])                  # décision locale (1 client)
```

Astuce : utilisez `shap.Explainer` (API unifiée). Pour arbres, `TreeExplainer` fonctionne aussi.

Objectif

Expliquer les prédictions de défaut pour un client et identifier les *features* qui poussent vers « défaut » ou « OK ».

Étapes

- Entrainer le modèle (RF).
- Calculer `shap_values` sur `X_test`.
- Lire **beeswarm** (global) et **waterfall** (local).

Lecture

Beeswarm : importance + signe (couleur). Waterfall : contributions cumulées des variables vers la décision.

Visualisations SHAP – summary, waterfall, force plot



Summary plot (beeswarm)

Vue globale: importance des variables et sens d'impact (couleur = valeur de la feature).

- Classement des variables par contribution moyenne
- Dispersion = hétérogénéité des effets

[Placeholder graphique SHAP summary]



Waterfall (instance)

Décomposition pas à pas de la prédiction d'un individu (base value → output).

- Barres rouges/bleues = hausse/baisse du score
- Explique une décision locale clairement

[Placeholder graphique SHAP waterfall]



Force plot (instance)

Visualisation des forces qui poussent la prédiction vers ↑ ou ↓ par rapport à la base.

- Lecture intuitive des contributions
- Utile en relation client (motifs standardisés)

[Placeholder graphique SHAP force]

Extrait de code — LIME pour expliquer une prédiction (classification)

```
# Installation (si nécessaire) : pip install lime
from lime.lime_tabular import LimeTabularExplainer
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Données tabulaires préparées (placeholders à remplacer)
# X : ndarray (n_samples, n_features) – y : labels (0/1) – features : liste de noms
X, y, features, class_names = ... # {Fournir les variables réelles ici}

# Split + standardisation (recommandé pour LIME avec continuous features)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
scaler = StandardScaler().fit(X_train)
Xtr, Xte = scaler.transform(X_train), scaler.transform(X_test)

# Modèle boîte noire (ex. Random Forest)
rf = RandomForestClassifier(n_estimators=300, random_state=42).fit(Xtr, y_train)

# Explainer LIME (mode classification, discréétisation des variables continues)
explainer = LimeTabularExplainer(
    Xtr, feature_names=features, class_names=class_names,
    discretize_continuous=True, mode='classification', sample_around_instance=True
)

# Choisir une instance test et générer l'explication locale
i = 0
exp = explainer.explain_instance(
    Xte[i], rf.predict_proba, num_features=8
)

# Sorties disponibles : liste des contributions ou visualisations dans notebook
print(exp.as_list()) # [(feature, poids local), ...]
# exp.show_in_notebook() # en environnement Jupyter
```

Astuce: fixer random_state et vérifier l'échelle des variables; utiliser num_features entre 5 et 10 pour une lecture claire.

Lecture rapide

`as_list()` retourne les **contributions locales** (poids) des features pour l'instance.

Les poids positifs tirent la prédiction vers la **classe positive**, négatifs vers la **classe négative**.

Bonnes pratiques

- LIME est **local** et sensible aux paramètres (kernel, discréétisation).
- Standardiser/encoder les features comme pour le modèle.
- Comparer avec **SHAP** pour validation des explications.

Contre-factuels – Explications actionnables



Définitions & principes

Qu'est-ce qu'un contre-factuel ?

Proposition de **modification minimale** de l'entrée x vers x_{cf} qui **inverse** la décision du modèle (refus \rightarrow accord) tout en restant **réaliste et actionnable**.

Objectif

Fournir au décideur/client des **leviers concrets** pour améliorer l'issue (ex. augmenter le revenu déclaré, réduire le ratio d'endettement).

Contraintes & critères de qualité

- **Proximité** : minimiser la distance, ex. $\|x_{cf} - x\|_1$ (ou $\|\cdot\|_2$).
- **Parcimonie** : changer le **moins de variables** possible.
- **Plausibilité** : rester dans des **plages réalistes** de la population.
- **Actionnabilité** : **ne pas** modifier des attributs **non-manipulables** (âge, situation familiale).

Exemple concret – Scoring crédit

Décision locale

Profil actuel

Revenu net mensuel : 2 500 €
Ratio dette/revenu : 0,60
Historique crédit : 3 ans
Âge : 35 ans
(non-manipulable)

Décision : Refus

Proposition contre-factuelle

Contre-factuel proposé

Revenu net mensuel : 2 900 € (+400 €)
Ratio dette/revenu : 0,50 (-0,10)
Historique crédit : 4 ans (+1 an)
Âge : inchangé (non-manipulable)

Décision : Accord

Changements **minimaux** et **actionnables** qui franchissent le seuil d'acceptation tout en respectant les règles métier.

- Variables **non-manipulables** (ex. âge, situation familiale) **exclues** des recommandations. Les valeurs proposées doivent rester **plausibles** (distribution observée).

Cas d'usage – Scoring crédit explicable (4 étapes)



1 Modélisation

Gradient Boosting (ex. XGBoost) entraîné sur données clients (revenus, âge, historique crédit...) Calibration des probabilités (Platt/Isotonic) pour décisions fiables.

2 XAI globale

Feature importance (Permutation) et **SHAP summary** pour comprendre les facteurs clés.
Vérifier corrélations, stabilité des variables.

3 Décisions locales

Explication instance (client) par **SHAP waterfall/force plot**.
Motifs standardisés: top 3 contributeurs ± à la décision.

4 Gouvernance

Seuils d'octroi alignés **risque/ROI**, journalisation des décisions, audits.
Conformité **RGPD** (droit à l'explication), suivi biais & drift.

Résultat attendu: décisions justes et explicables, adoption par les équipes risques, et diminution des litiges grâce à des **justifications claires**.

Quiz interactif – Chapitre 7 (XAI)



1 LIME vs SHAP

Quelle différence principale entre LIME (modèle local approximatif) et SHAP (valeurs de Shapley avec garanties d'additivité/cohérence) ?

2 PDP vs ICE

Quand privilégier les PDP (effet moyen global) et quand utiliser les courbes ICE (variabilité individuelle) ? Donnez un exemple.

3 Valeur SHAP négative

Que signifie une valeur SHAP négative pour une feature donnée dans une prédiction individuelle (par rapport à la valeur attendue) ?

4 Contre-factuels actionnables

Proposez un contre-factuel en scoring crédit (ex. +400€ revenu, -0,1 ratio dette) et précisez une contrainte « non manipulable ».

5 Attention \neq explication fidèle

Pourquoi les poids d'attention d'un Transformer ne constituent-ils pas toujours une « explication » fiable de la décision ?

Chapitre 8 – IA en Finance : vue d'ensemble



- **Domaines clés** : risque de crédit, fraude paiements/cartes, marchés (trading), conformité, gestion de portefeuille
- **Enjeux** : précision, latence temps réel, **explicabilité**, conformité (Bâle, RGPD), robustesse
- **Données** : tabulaires (clients, transactions), séries temporelles (prix/volatilité), textes (news, rapports), logs
- **KPI business** : coût du risque, AUC/KS (crédit), Precision@K (fraude), Sharpe/Drawdown (trading), ROI
- **Contraintes** : classes déséquilibrées, biais, **data/model drift**, gouvernance & traçabilité (audit)

Cartographie des applications IA en finance



Scoring crédit

Estimer la probabilité de défaut pour l'octroi et le pricing.



KPI : AUC / KS

XAI requis (RGPD)



Détection de fraude

Identifier des anomalies et prioriser les alertes actionnables.

KPI : Precision@K

Latence temps réel



Trading algorithmique

Générer des signaux et backtester des stratégies robustes.



KPI : Sharpe

Drawdown



Gestion de portefeuille

Optimiser le couple rendement/risque et la réallocation.

Frontière efficiente

VaR / CVaR

Scoring crédit – Données & Feature engineering



Données (inputs X)

Sources tabulaires & transactions

- Socio-démographie
Âge, situation familiale, ancienneté résidence
- Revenus & dettes
Salaire, charges fixes, encours, garanties
- Historique de crédit
Retards/impayés, ancienneté comptes, incidents
- Comportements transactionnels
Cash-flows, fréquence dépenses, saisonnalité

Cible y Défaut (0/1) sur horizon (ex. 12 mois)

Déséquilibre classe souvent élevé → AUC/KS/PR-AUC

Feature engineering (exemples)

- Ratios & intensités
DTI (Dette/Revenu), utilisation du crédit, charge/actif
- Encodage des catégorielles
One-Hot / Target encoding (K-fold) selon le modèle
- Binning scorecard & WOE
Discrétisation monotone, Weight of Evidence, IV
- Agrégations temporelles
Rolling 3/6/12 mois (moyennes, max, volatilité)
- Qualité & préparation
Imputation (médiane/catégorie « inconnue »), outliers, normalisation

Note: séparer **fit** (train) et **transform** (val/test) pour éviter la fuite de données.

Scoring crédit – Comparaison des modèles



Logistique • Gradient Boosting (XGBoost/LightGBM) • Réseaux (MLP)

| Critère | Logistique | Gradient Boosting | Réseaux (MLP) |
|----------------------------|---|--|--|
| Interprétabilité | Élevée (coefficients, odds-ratio). | Moyenne (SHAP/LIME requis). | Faible (post-hoc XAI nécessaire). |
| Performance (AUC/ks) | Bonne si relation quasi-linéaire. | Très bonne (capture non-linéarités). | Très bonne si données suffisantes. |
| Données / Features | Tabulaire propre; interactions à créer. | Gère interactions automatiquement. | Besoin data volumineuse et normalisée. |
| Prétraitement | Standardisation, encodage one-hot. | Moins sensible; tuning hyperparamètres. | Normalisation + régularisation (Dropout/L2). |
| Temps d'entraînement | Rapide (CPU). | Moyen (tuning: learning_rate, depth...). | Plus long (souvent GPU utile). |
| Explicabilité / Conformité | Forte adéquation RGPD & audit. | OK avec SHAP + documentation. | Nécessite XAI robuste + gouvernance. |

Extrait de code – Pipeline complet (prétraitement + modèle + AUC/Brier)

```
# Imports
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import roc_auc_score, brier_score_loss
from xgboost import XGBClassifier

# Supposez un DataFrame df déjà chargé avec la cible 'defaut'
num_cols = ['age', 'revenu', 'dette_ratio', 'anciennete']
cat_cols = ['statut_marital', 'type_contrat']
X, y = df[num_cols + cat_cols], df['defaut']

# Split avec stratification (données souvent déséquilibrées)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

# Prétraitement : standardisation num., one-hot cat.
num_tf = Pipeline(steps=[('scaler', StandardScaler())])
cat_tf = Pipeline(steps=[('ohe', OneHotEncoder(handle_unknown='ignore'))])
prep = ColumnTransformer(
    transformers=[('num', num_tf, num_cols),
                 ('cat', cat_tf, cat_cols)])
)

# Modèle : Gradient Boosting (XGBoost) – robuste et performant
xgb = XGBClassifier(
    n_estimators=400, learning_rate=0.05, max_depth=4,
    subsample=0.8, colsample_bytree=0.8, random_state=42,
    n_jobs=-1, eval_metric='logloss'
)

# Pipeline complet
pipe = Pipeline(steps=[('prep', prep), ('model', xgb)])

# Entraînement
pipe.fit(X_train, y_train)

# Évaluation : probabilités, AUC ROC et Brier score (calibration)
proba = pipe.predict_proba(X_test)[:, 1]
auc = roc_auc_score(y_test, proba)
brier = brier_score_loss(y_test, proba)
print(f"AUC={auc:.3f} Brier={brier:.3f}")
```

Astuce: stratifier le `train_test_split` et mesurer AUC + Brier (calibration).

Étapes du pipeline

Split stratifié (déséquilibre).
Prétraitement: StandardScaler + OneHot.
Modèle: XGBClassifier.
Évaluer: ROC-AUC, Brier.

KPI à reporter

- AUC / KS (discrimination).
- Brier / calibration plots.
- Recall@FPR, coût du risque.

Fraude – Problématique & données déséquilibrées



Problématique

Temps réel & conformité

- Taux de fraude très faible

Classe minoritaire ($\approx 0,1\text{--}0,5\%$) → apprentissage difficile

- Coût des faux positifs

Fatigue opérationnelle, expérience client dégradée

- Contraintes de latence

Décision en millisecondes sur flux de transactions

- Dérive des données (drift)

Nouveaux schémas de fraude, saisonnalités → recalibrage

- Explicabilité & traçabilité

Justifier les alertes (audit, régulateur, RGPD)

Données & déséquilibre

Imbalanced data

Types de données

Montant, MCC, pays/geo

Device, canal, horaire

Historique séquentiel

Règles/annotations

Déséquilibre (exemple)

Non-fraude $\approx 99,8\%$

Fraude $\approx 0,2\%$

Stratégies

Seuils adaptés, ré-échantillonnage (SMOTE/undersampling), coûts asymétriques

Métriques

Precision@K, Recall@K, ROC-AUC, PR-AUC, coût/alerte

Labellisation

Retards/erreurs de labels → règles de tri, active learning, validation humaine ciblée

Fraude – Techniques & KPI de priorisation



Anomalies non supervisées

IsolationForest, One-Class SVM, LOF pour repérer des patterns atypiques.

Cold-start

Faible supervision



Supervisé + rééquilibrage

XGBoost/LightGBM avec SMOTE, undersampling, class_weights.

AUC/PR-AUC

Seuil optimisé



Temps réel & séquentiel

Features temporelles, LSTM/Autoencoder, règles hybrides + ML.

Latence < 50 ms

Drift monitoring



KPI de priorisation

Classer les alertes pour l'enquête humaine.

Precision@K

Recall@K

Coût/alerte

Fraude – Code Python (Isolation Forest + Precision@K)



Extrait de code – Isolation Forest + Precision@K (fraude)

```
# Hypothèses : X_train, X_test (features), y_test (0=normal, 1=fraude)
import numpy as np
from sklearn.ensemble import IsolationForest

# Modèle non supervisé : anomalies = fraudes potentielles
iso = IsolationForest(
    n_estimators=200, contamination=0.005,
    random_state=42, n_jobs=-1
).fit(X_train)

# Score d'anomalie (plus grand = plus suspect)
scores = -iso.decision_function(X_test) # signe inversé

# Precision@K : qualité du tri des alertes dans un budget d'alertes K
K = max(1, int(0.02 * len(y_test))) # top 2% des transactions
ranks = np.argsort(scores)[::-1] # tri décroissant
topk_idx = ranks[:K]
precision_at_k = (y_test[topk_idx] == 1).mean()

print(f"Precision@K (K={K}) = {precision_at_k:.2%}")

# Option : calculer Recall@K si le nombre total de fraudes est connu
recall_at_k = (y_test[topk_idx] == 1).sum() / (y_test == 1).sum()
print(f"Recall@K (K={K}) = {recall_at_k:.2%}")
```

Astuce: fixer K selon le budget d'alertes (ex. 1–5%). Standardiser les variables si échelles hétérogènes.

Objectif & réglages

Déecter des transactions **anormales** (fraude potentielle) en non supervisé.

contamination ≈ taux de fraude estimé (ex. 0,5%).

Precision@K = fraudes détectées parmi les K plus suspects.

Lecture & bonnes pratiques

- Suivre Precision@K et Recall@K (priorisation des alertes).
- Monitorer le **drift** et réentraîner périodiquement.
- Évaluer le **coût par alerte** et le ROI métier.

Trading algorithmique – Signaux & backtesting



Signaux (features)

Données: prix, volume, news

- Momentum & trend-following
Moyennes mobiles (MA), breakout, force relative
- Mean reversion
Écarts normalisés (z-score), bandes de Bollinger
- Stat-arb & spreads
Pairs trading, co-intégration
- Sentiment NLP
News/tweets → score (FinBERT), agrégation sectorielle
- Microstructure
Order flow, volume/volatilité, indicateurs RSI/MACD
- Fenêtrage & fréquence
Intraday vs journalier; horizon du signal cohérent

Backtesting & évaluation

Rigueur & réalisme

Protocole

Séparation temporelle:
in/out-of-sample
Walk-forward / validation
chronologique

Mesures clés

Sharpe, Sortino, Max drawdown
Courbe d'equity, distribution PnL

Gestion du risque

- Position sizing (ex. volatilité cible)
- Stop-loss / take-profit
- Diversification des signaux

Coûts & frictions

Slippage, commissions, liquidité
Turnover & capacité

Biais à éviter

Look-ahead, survivorship, data
snooping
Overfitting, p-hacking

Trading – Backtest momentum (extrait Python)



Extrait de code – Momentum simple vs moyenne mobile (50 j)

```
# Backtest d'une stratégie momentum simple (didactique)
import pandas as pd
import numpy as np

# 1) Charger les prix (ex: DataFrame avec colonne 'Close')
# df = pd.read_csv('prices.csv', parse_dates=['Date'], index_col='Date')
# Pour l'exemple, simulons un prix;
np.random.seed(42)

ret = np.random.normal(0.0004, 0.01, 600)
price = pd.Series(100 * (ret + 1).cumprod(), name='Close')

# 2) Signal momentum: prix > moyenne mobile (SMA 50 jours)
sma = price.rolling(50).mean()
signal = (price > sma).astype(int) # 1 si tendance haussière, 0 sinon
position = (2*signal - 1).shift(1).fillna(0) # +1 long, -1 short, décalé (pas de look-ahead)

# 3) Rendements et performance stratégie
r = price.pct_change().fillna(0)
strat_ret = position * r
sharpe = np.sqrt(252) * strat_ret.mean() / strat_ret.std(ddof=1)
cum = (1 + strat_ret).cumprod().iloc[-1] - 1

# 4) Coûts de transaction simples (ex: 5 bps par changement de position)
turnover = position.diff().abs().fillna(0) # nombre de flips
cost_bps = 0.0005 * turnover
strat_after_cost = strat_ret - cost_bps
sharpe_net = np.sqrt(252) * strat_after_cost.mean() / strat_after_cost.std(ddof=1)

print(f"Sharpe brut: {sharpe:.2f} | Sharpe net: {sharpe_net:.2f} | Perf cumulée: {cum:.1%}")
```

Lecture rapide

Règle : long si Close > SMA(50), short sinon.

Évaluation : Sharpe annualisé, performance cumulée.

Coûts : pénalisent les flips de position.

Bonnes pratiques

- Éviter le *look-ahead* (utiliser `shift(1)`).
- Inclure frais, slippage et *backward*.
- Comparer à un benchmark (*buy&hold*).

Hypothèses simplificatrices: pas de slippage, pas de contraintes de marché. Démonstration pédagogique.

Gestion de portefeuille – Markowitz (moyenne-variance)



Formulation mathématique

Moments du portefeuille

$$E[r_p] = \mathbf{w}^T \boldsymbol{\mu}$$

$$\sigma_p^2 = \mathbf{w}^T \Sigma \mathbf{w}$$

Avec \mathbf{w} (poids), $\boldsymbol{\mu}$ (rendements espérés), Σ (matrice de covariance).

Optimisation contrainte

$$\min_{\mathbf{w}} \mathbf{w}^T \Sigma \mathbf{w}$$

$$\text{s.c. } \mathbf{w}^T \boldsymbol{\mu} = \mu^*, \quad \sum_i w_i = 1, \quad w_i \geq 0$$

Frontière efficiente pour un niveau de rendement cible μ^* (long-only).

Forme pénalisée (Lagrangien)

$$\max_{\mathbf{w}} \mathbf{w}^T \boldsymbol{\mu} - (\lambda/2) \mathbf{w}^T \Sigma \mathbf{w}$$

$\lambda > 0$ mesure l'aversion au risque (trade-off rendement/risque).



Lecture managériale

La frontière efficiente rassemble les portefeuilles offrant le meilleur couple rendement-risque. Le choix final dépend des objectifs, contraintes réglementaires (Bâle/UCITS) et de l'appétit au risque.

Gestion de portefeuille – Frontière efficiente (code)



Extrait – optimisation moyenne-variance (cvxpy) et frontière efficiente

```
# pip install cvxpy numpy
import numpy as np
import cvxpy as cp

# Hypothèses : 4 actifs (rendements espérés annuels) et matrice de covariance PD
mu = np.array([0.10, 0.12, 0.08, 0.07])
Sigma = np.array([
    [0.040, 0.010, 0.020, 0.000],
    [0.010, 0.090, 0.010, 0.000],
    [0.020, 0.010, 0.060, 0.010],
    [0.000, 0.000, 0.010, 0.030]
])

n = len(mu)
targets = np.linspace(mu.min(), mu.max(), 15)
frontier = []

# Résoudre : min_w w' Σ w  s.c.  1'w=1,  w>=0 (pas de vente à découvert),  mu'w ≥ r*
for r_star in targets:
    w = cp.Variable(n)
    obj = cp.Minimize(cp.quad_form(w, Sigma))
    cons = [cp.sum(w) == 1, w >= 0, (mu @ w) >= r_star]
    cp.Problem(obj, cons).solve(solver="OSQP", verbose=False)
    if w.value is not None:
        ret_p = float(mu @ w.value)
        var_p = float(w.value.T @ Sigma @ w.value)
        sigma_p = np.sqrt(var_p)
        frontier.append((sigma_p, ret_p, w.value))

# Portefeuille tangent (max Sharpe) avec taux sans risque rf
rf = 0.02
if frontier:
    risks, rets, weights = zip(*frontier)
    sharpe = ((np.array(rets) - rf) / np.array(risks))
    best = int(np.argmax(sharpe))
    print("Tangent (Sharpe max) – σ=", risks[best], ", μ=", rets[best])
    print("Poids w* = ", np.round(weights[best], 3))

# Remarque : pour tracer la frontière (σ sur X, μ sur Y), utiliser matplotlib.
```

Lecture & KPI

Frontière efficiente : portefeuilles à risque minimal pour un niveau de rendement visé.

Sharpe max : portefeuille tangent (rendement excédentaire/risque).

Contraintes usuelles : $1'w=1$, $w \geq 0$, budget, limites sectorielles.

Bonnes pratiques

- Vérifier que Σ est définie positive; regulariser si besoin.
- Calibrer μ (rendements attendus) via modèles robustes.
- Intégrer coûts/turnover via pénalités ou contraintes.

Astuce: ajouter la contrainte $w \geq 0$ pour interdire la vente à découvert; enlever cette contrainte pour une frontière « libre ».

Prédiction de prix – Séries temporelles (LSTM)



- Problèmes: non-stationnarité, régimes de marché, bruit, saisonnalité
- Fenêtrage (window) et horizon: $X[t-w \dots t] \rightarrow y[t+h]$
- Caractéristiques: OHLCV, indicateurs techniques, variables exogènes (macro, sentiment)
- Évaluation: MSE/RMSE, MAPE; backtest walk-forward
- Séparation temporelle: train/val/test chronologiques; éviter la fuite d'information
- Bonnes pratiques: scaling, régularisation, early stopping, suivi du drift

Séries temporelles – Code Keras LSTM (extrait)



Extrait – LSTM pour prédire un prix/retour (fenêtre OHLCV)

```
# Imports
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Hypers & forme des données
window = 60          # 60 jours glissants
n_features = 5         # OHLCV = 5 features

# X_train: (n_samples, window, n_features), y_train: (n_samples,)
# (Données préparées en amont : scaling, création de fenêtres)

# Modèle LSTM empilé
model = Sequential([
    LSTM(64, return_sequences=True, input_shape=(window, n_features)),
    Dropout(0.2),
    LSTM(32),
    Dense(1) # régression (prix/retour J+1)
])

model.compile(optimizer="adam", loss="mse", metrics=["mae"])

# Entraînement (validation multi-périodes recommandée)
history = model.fit(
    X_train, y_train,
    epochs=20, batch_size=64,
    validation_data=(X_val, y_val),
    verbose=0
)

# Inférence
y_pred = model.predict(X_test)
```

Astuce: évaluer hors échantillon (walk-forward), puis backtester la stratégie.

Objectif

Prévoir le prix ou le retour à J+1 à partir d'une fenêtre OHLCV.

Entrée / Sortie

Entrée: (window, n_features) = (60, 5).

Sortie: y scalaire (régression).

Bonnes pratiques

- Normaliser par feature; éviter la fuite d'info.
- Comparer à des baselines (ARIMA, naïf).



Pipeline (gauche)

1 Collecte

News, flux RSS, réseaux (tweets), dépêches; horodatage et source.

2 Prétraitement

Nettoyage, langue, déduplication, tokenisation; normalisation entités (tickers).

3 Scoring modèle

FinBERT/Transformers ou lexiques; score de sentiment $\in [-1, 1]$ par document.

4 Agrégation & signaux

Agrégation par titre/secteur/indice et fenêtre (intra-day/journalier); lissage et seuils d'alerte.

KPI & Monitoring (droite)

KPI modèle

- F1 / PR-AUC sur jeux déséquilibrés; Recall@K pour priorisation.
- Latence d'inférence et couverture des sources.

KPI marché

- Corrélation sentiment \leftrightarrow rendements; Information Coefficient (IC).
- Hit ratio des signaux (directionnels) et stabilité temporelle.

Monitoring & drift

- Drift lexical (PSI, distribution tokens); alertes seuils.
- Garde-fous: revue humaine, traçabilité, biais médias.

Stress testing & Monte-Carlo – principes + code



Principes et bonnes pratiques

- Objectif: quantifier les pertes sous **scénarios** (chocs) ou via **simulations Monte-Carlo**.
- Étapes MC: estimer $\mu, \Sigma \rightarrow$ simuler rendements **corrélés** \rightarrow calculer PnL \rightarrow VaR/CVaR.
- Scénarios déterministes: appliquer des **chocs** (ex. -5% actions, +150 bps taux) et mesurer l'impact.
- Bonnes pratiques: $N \geq 10k$ tirages, vérifier positifs definitude de Σ , contrôler le *drift & stationnarité*.
- Gouvernance: traçabilité, validation indépendante, alignement **Bâle/IFRS9**.
- KPI: VaR95, CVaR95, pertes de scénario, probabilité d'excès de perte.

Code Python – Monte-Carlo (VaR & CVaR)

NumPy ·
Cholesky

```
import numpy as np

# Hypothèses: portefeuilles à 3 actifs (rendements journaliers)
mu = np.array([0.0004, 0.0002, 0.0003])          # espérances
Sigma = np.array([[0.0009, 0.0003, 0.0002],        # covariance
                 [0.0003, 0.0007, 0.00025],
                 [0.0002, 0.00025, 0.0008]])
w = np.array([0.5, 0.3, 0.2])                      # pondérations
(somme=1)

N = 10000
L = np.linalg.cholesky(Sigma)                      # corrélations via
Cholesky
```

Interprétation : la **VaR95** borne la perte au 5e centile; la **CVaR95** mesure la perte moyenne dans la queue. Le **stress** applique un choc déterministe aux actifs.

Risque de crédit & Bâle III – notions clés

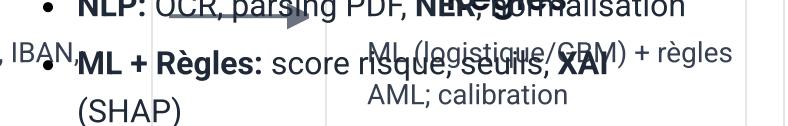
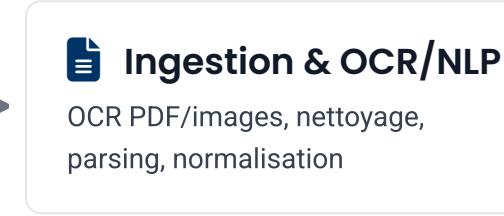


- **RWA (Risk-Weighted Assets)** – Actifs pondérés par le risque; dimensionnent le **capital réglementaire (CET1 ratio)**.
- Paramètres risque **PD / LGD / EAD** – **PD** (Probabilité de défaut), **LGD** (Perte en cas de défaut), **EAD** (Exposition au défaut).
- **IFRS 9** – Perte de crédit attendue (**ECL**) et **staging**: Stage 1 (12 mois), Stage 2 (lifetime), Stage 3 (défaut).
- **Piliers de Bâle III** – Pilier 1: exigences minimales; Pilier 2: ICAAP/SREP; Pilier 3: discipline de marché (transparence).
- Attentes **modèles de crédit** – backtesting/benchmarking, validation indépendante, suivi du **drift**, documentation & **XAI**.
- **KPI & gouvernance** – Taux de défaut, **coût du risque**, appétit au risque; comités modèles & contrôles (audit).

Conformité AML/KYC – Pipeline automatisé (NLP + RPA)



Schéma de flux



Orchestration RPA

Robots RPA déclenchent OCR, enrichissements, dépôts et transferts sécurisés

Audit & Journalisation

Logs détaillés (données, scores, décisions, explications) pour auditabilité

KPI & conformité

Recall@FPR, temps de traitement, taux de faux positifs; RGPD (art.22), traçabilité, revues humaines

Résumé

Pipeline AML/KYC industrialisé: **NLP** pour extraire, **ML+règles** pour scorer, **RPA** pour exécuter, avec **audit** et **XAI** pour la conformité.



RegTech

Technologies pour automatiser la conformité et le reporting réglementaire.

Exemples :

- KYC/AML automatisés (OCR+NLP), screening sanctions/PEP
- Reporting Bâle/IFRS 9 avec traçabilité et contrôles



SupTech

Outils IA des superviseurs pour surveiller les marchés et les risques.

Exemples :

- Détection de manipulations de marché & abus (NLP+anomalies)
- Surveillance risque systémique, stress tests sectoriels



Gouvernance IA

Cadre de maîtrise des modèles : éthique, risques, conformité, monitoring.

Bonnes pratiques :

- Registre des modèles, XAI & équité (biais), contrôle PSI/drift
- Processus MRM (validation indépendante), audits et traçabilité

Quiz interactif – Chapitre 8 (Finance)



1

Fraude – métriques prioritaires

Quelle(s) métrique(s) privilégier pour une fraude très rare et pourquoi ? (p. ex. Precision@K, Recall@K, PR-AUC)

2

Scoring crédit – AUC vs KS

Expliquez la différence entre ROC-AUC et KS et dans quel cas chacun est pertinent pour évaluer un score.

3

Trading – pièges du backtesting

Citez trois sources d'erreur courantes en backtest (p. ex. sur-ajustement, look-ahead bias, coûts de transaction).

4

Portefeuille – Markowitz

Formulez l'objectif d'optimisation moyenne-variance et citez 2 contraintes typiques (somme des poids, non-négativité).

5

Conformité AML/KYC – pipeline

Schématisez un pipeline AML/KYC automatisé (sources → extraction OCR/NLP → scoring risque → alerting/case management → audit).

Chapitre 9 – IA en Gestion : vue d'ensemble



- **Fonctions clés** – CRM/marketing, opérations & supply chain, finance/contrôle, RH
- **Enjeux** – personnalisation, prévisions fiables, optimisation coûts/délais, automatisation
- **Données** – CRM/RFM, ERP/SCM, comptabilité, RH (CV/entretiens), tickets/support
- **KPI** – CA incrémental, churn ↓, taux de service, rotation stock/DSO, time-to-hire, ROI
- **Contraintes** – qualité des données, biais/équité, intégration SI, adoption, conformité (RGPD)

Cartographie des applications IA en gestion



CRM & personnalisation

Recommandation produits, segmentation RFM, next-best-action.

Churn ↓

Conversion ↑



RH & recrutement

CV parsing, matching candidats-postes, prédition turnover.

Temps de tri ↓

Équité/XAI



Supply chain & logistique

Prévision de la demande, optimisation stocks, tournées (VRP).

Ruptures ↓

Taux service ↑



Pricing & BI prédictive

Pricing dynamique (élasticité), dashboards & alertes prédictives.

Marge ↑

A/B testing



Principes & métriques

- Filtrage collaboratif utilisateur–article (user–item) : rapprocher les profils ayant des comportements d'achat/notation similaires.
- Factorisation matricielle (SVD) : approximer la matrice notations $R \approx U\Sigma V^T$ pour apprendre des facteurs latents.
- Métriques : RMSE (qualité prédictions), Precision@K / Recall@K (qualité des Top-K recommandations).
- Cold-start : gérer nouveaux utilisateurs/produits via popularité/CBF (content-based) ou règles métier.
- Cas d'usage : cross-sell, up-sell, next-best-offer, personnalisation CRM.

Code Python – SVD (Surprise)

Surprise • RMSE

```
from surprise import Dataset, SVD, accuracy
from surprise.model_selection import train_test_split

# 1) Charger un dataset user–item (MovieLens 100k inclus)
data = Dataset.load_builtin('ml-100k')
trainset, testset = train_test_split(data, test_size=0.2,
random_state=42)

# 2) Modèle SVD (factorisation matricielle)
model = SVD(n_factors=50, n_epochs=20, biased=True,
random_state=42)
model.fit(trainset)

# 3) Évaluer qualité de prédition (RMSE)
```

Astuce : compléter par Precision@K sur les Top-K pour mesurer la pertinence business des recommandations.

CRM – Segmentation clients (K-Means sur RFM)



Extrait de code – Segmentation RFM avec StandardScaler + KMeans (K=4)

```
# 1) Imports
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# 2) Jeu de données RFM (exemple jouet)
rfm = pd.DataFrame({
    'Recency': [12, 45, 7, 90, 14, 33, 5, 60],
    'Frequency': [5, 2, 12, 1, 8, 4, 15, 2],
    'Monetary': [300, 80, 1200, 40, 520, 150, 1800, 70]
})

# 3) Standardisation (recommandée pour K-Means)
scaler = StandardScaler()
X = scaler.fit_transform(rfm)

# 4) Modèle K-Means (K=4 segments typiques CRM)
km = KMeans(n_clusters=4, n_init=10, random_state=42)
km.fit(X)
rfm['segment'] = km.labels_

# 5) Profilage des segments (moyennes RFM par cluster)
profil = rfm.groupby('segment')[['Recency', 'Frequency', 'Monetary']].mean()
print(profil)

# 6) (Optionnel) Interprétation business rapide :
# seg avec Recency faible & Frequency/Monetary élevés → « Premium/Champions »
```

Lecture rapide

RFM = Recency (fraîcheur), Frequency, Monetary (valeur).

Profilage par moyennes R/F/M → nommer segments (VIP, Actifs, Dormants, Nouveaux).

Action: campagnes ciblées (NBO), offres & priorisation.

Bonnes pratiques

- Retirer outliers extrêmes (winsorisation si besoin).
- Valider la stabilité des segments dans le temps.

Astuce: standardiser R, F, M avant K-Means; tester plusieurs K (méthode du coude/silhouette).

RH – CV parsing et matching candidats–postes



Pipeline (gauche)

1 OCR documents

Extraction du texte depuis CV PDF/scan (mise en page, tableaux).

2 NLP (tokens & entités)

Tokenisation, lemmatisation, NER des compétences, diplômes, expériences.

3 Vectorisation

TF-IDF ou embeddings (Word2Vec/BERT) pour représenter CV & offres.

4 Similarité & ranking

Cosinus/ANN pour scorer les appariements, génération de liste Top-K.

5 Validation & boucle métier

Feedback recruteur, règles RH, journalisation et amélioration continue.

KPI & contraintes (droite)

KPI quantitatifs

■ Précision@K / Rappel@K
Qualité du Top-K appariements proposés.

■ Temps de tri (TTR) ↓
Réduction du temps de présélection des CV.

■ Taux d'acceptation
Part des candidats shortlistés effectivement retenus en entretien.

Équité & conformité

■ Parité démographique / Disparate Impact
Suivi des biais (groupes protégés) et remédiation si nécessaire.

■ RGPD & traçabilité
Consentement, anonymisation des PII, journalisation des décisions.

Bonnes pratiques

Dictionnaire de compétences, taxonomie unifiée, XAI locale (explications par critères).

Extrait de code – Similarité CV ↔ Offre (lemmatisation spaCy + TF-IDF)

```
# 1) Imports
import spacy
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# 2) Modèle de langue (français)
nlp = spacy.load("fr_core_news_sm")

# 3) Fonction de prétraitement (lemmatisation, suppression stop-words)
def preprocess(txt):
    doc = nlp(txt.lower())
    return " ".join([t.lemma_ for t in doc if t.is_alpha and not t.is_stop])

# 4) Corpus : 2 CV candidats + 1 offre
cvs = [
    "Data analyste: SQL, Python, BI, retail, tableaux de bord.",
    "Contrôle de gestion: Excel, comptabilité, prévisions, reporting."
]
offre = "Recherche Data Analyst (retail) maîtrisant SQL, Python et visualisation BI."

# 5) Prétraitement spaCy
docs = [preprocess(x) for x in (cvs + [offre])]

# 6) Vectorisation TF-IDF (n-grammes 1-2)
vec = TfidfVectorizer(ngram_range=(1,2), min_df=1)
X = vec.fit_transform(docs)  # shape: (3, vocab)

# 7) Similarité cosinus CVs vs Offre (dernière ligne)
S = cosine_similarity(X[:2], X[2:])
print("Scores similarité:", S.ravel())

# 8) Meilleure correspondance (ranking)
best_idx = S.ravel().argsort()[:-1][0]
print(f"Top match = CV{best_idx+1}")
```

Astuce: appliquez la lemmatisation avant TF-IDF et testez des n-grammes pour capter les compétences composées (ex. "data analyst").

Étapes & idées clés

Prétraiter texte (minuscule, lemma, stop-words).

Vectoriser avec TfidfVectorizer (n-grammes 1-2).

Comparer CV ↔ Offre via cosine_similarity.

Interprétation & bonnes pratiques

- Score $\in [0,1]$ (1 = adéquation forte).
- Gérer le cold-start (peu d'historique).
- Mesurer biais (parité démographique) & RGPD.

Supply chain – Prévision de la demande (principes)



- **Problèmes** : saisonnalité et effets calendaires, promotions/ruptures, cannibalisation, changements de régime, événements externes (météo, campagnes).
- **Features** : historiques (lag, moyennes mobiles), attributs calendaires (jour/semaine/mois), prix/promo, météo, événements, canaux, variables exogènes.
- **Modèles** : baselines (naïf, moy. mobile), Prophet/SARIMAX, ML tabulaire (Random Forest/XGBoost), Deep Learning (LSTM/Transformer), prévisions hiérarchiques.
- **KPI** : MAPE/WAPE, RMSE, niveau de service, ruptures (OOS), biais (sur/sous-prévision), impact stocks/coûts, ROI opérationnel.

Supply chain – Code Prophet (prévision séries temporelles)



Extrait de code – Prophet (prévision de la demande)

```
# pip install prophet
import pandas as pd
from prophet import Prophet

# 1) Données au format Prophet : colonnes 'ds' (date) et 'y' (quantité demandée)
df = pd.read_csv("demand.csv", parse_dates=["ds"])

# 2) Modèle : saisonnalité multiplicative, désactiver la saisonnalité hebdo si inutile
m = Prophet(seasonality_mode="multiplicative", weekly_seasonality=False)
# Optionnel : ajouter saisonnalité(s) métier (ex. promo mensuelle)
m.add_seasonality(name="mensuelle", period=30.5, fourier_order=5)

# 3) Entraînement
m.fit(df)

# 4) Horizon de prévision (30 jours)
future = m.make_future_dataframe(periods=30, freq="D")
fcst = m.predict(future)

# 5) Sortie utile : point forecast et intervalles
print(fcst[["ds", "yhat", "yhat_lower", "yhat_upper"]].tail(5))

# 6) Évaluation (à réaliser via backtest / cross-validation temporelle)
# from prophet.diagnostics import cross_validation, performance_metrics
# df_cv = cross_validation(m, horizon='14 days', period='7 days')
# print(performance_metrics(df_cv)[['mape', 'rmse']].head())
```

Pré-requis & données

Fréquence homogène (ex. journalière) et gestion des manquants.

Variables exogènes possibles via `add_regressor()`.

Bonnes pratiques

- Comparer avec un baseline (naïf, SMA).
- Utiliser intervalles `yhat_lower/upper` pour la planification.

Astuce: assurez-vous que `ds` est de type date et que `y` est numérique; utilisez une validation glissante pour mesurer MAPE/RMSE.

Logistique – Routage (VRP/TSP) : concepts & pratiques



Concepts (côté modèle)

TSP / VRP

- TSP (Traveling Salesman Problem) : 1 tournée minimale visitant tous les points puis retour dépôt.
- VRP (Vehicle Routing Problem) : plusieurs véhicules, capacités, un ou plusieurs dépôts.

Contraintes

- Fenêtres de temps
- Capacité véhicules
- Retours au dépôt

Objectifs / coûts

- Distance/temps total
- CO₂ / coûts logistiques
- Respect des fenêtres

Heuristiques & mét-heuristiques

2-opt / 3-opt

Clarke-Wright

Recherche tabou

Reclut simulé

Google OR-Tools

Pratique terrain & KPI

Données d'entrée

- Matrice distances/temps
- Fenêtres horaires / SLA

- Demandes & capacités
- Dépôts / véhicules

KPI opérationnels (exemple)

km totaux ↓

Durée tournée ↓

Taux de service ↑

Respect fenêtres ↑

Intégration SI

WMS / TMS / ERP (flux commandes, géocodage, suivi GPS).

Bonnes pratiques

Nettoyer adresses, pénalités de retard, trafic en temps réel, tests A/B de politiques.

Extrait de code – Résolution TSP avec OR-Tools (PATH_CHEAPEST_ARC)

```

# TSP simplifié avec Google OR-Tools (une tournée, 1 véhicule)
from ortools.constraint_solver import pywrapcp, routing_enums_pb2
import numpy as np

# 1) Données : coordonnées (x,y) de 8 villes (exemple jouet)
coords = np.array([
    [0, 0], [2, 3], [5, 4], [1, 6],
    [7, 3], [6, 7], [3, 8], [9, 1]
], dtype=float)
n = len(coords)

# Matrice de distances euclidiennes (entiers requis par OR-Tools)
dist = np.zeros((n, n), dtype=int)
for i in range(n):
    for j in range(n):
        dist[i, j] = int(np.linalg.norm(coords[i] - coords[j]) + 0.5)

# 2) Gestionnaire d'index et modèle de routage
manager = pywrapcp.RoutingIndexManager(n, 1, 0) # 1 véhicule, dépôt=0
routing = pywrapcp.RoutingModel(manager)

# 3) Callback de distance
def distance_callback(from_index, to_index):
    i = manager.IndexToNode(from_index)
    j = manager.IndexToNode(to_index)
    return dist[i, j]

transit_idx = routing.RegisterTransitCallback(distance_callback)
routing.SetArcCostEvaluatorOfAllVehicles(transit_idx)

# 4) Paramètres de recherche
params = pywrapcp.DefaultRoutingSearchParameters()
params.first_solution_strategy = routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC
params.local_search_metaheuristic = routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH
params.time_limit.seconds = 5

# 5) Résolution et extraction de la tournée
solution = routing.SolveWithParameters(params)
if solution:
    index = routing.Start(0)
    route, route_dist = [], 0
    while not routing.IsEnd(index):
        route.append(manager.IndexToNode(index))
        prev = index
        index = solution.Value(routing.NextVar(index))
        if not routing.IsEnd(index):
            route_dist += routing.GetArcCostForVehicle(prev, index, 0)
    route.append(0)
    print("Tournée: ", route)
    print("Distance totale: ", route_dist)
else:
    print("Aucune solution trouvée.")

```

Points clés

Entrées : 8 villes, matrice de distances.

Objectif : minimiser la distance totale.

Heuristique :

PATH_CHEAPEST_ARC +
GUIDED_LOCAL_SEARCH.

Extensions VRP

- Fenêtres de temps, capacités, flottes multiples.
- Coûts réels (km, temps, péages) et contraintes SI.

Astuce: OR-Tools requiert des coûts entiers; arrondir les distances réelles.



■ Économie (côté demande)

ε Élasticité-prix

$(\varepsilon = \Delta Q / \Delta P) \rightarrow$ sensibilité des ventes au prix.

● Segmentation & « price fences »

Tarifs différenciés par segment, canal, moment (yield).

● Contraintes & objectifs

Stock, capacité, prix plancher/plafond, marge cible, image de marque.

● Fonction de demande

Relier $Q(P)$ aux facteurs : saisonnalité, promo, concurrence.

■ Expérimentation (go-to-market)

A/B Tests A/B & bandits

Estimer le « lift » prix; bandits UCB/Thompson pour explorer/exploiter.

● Contraintes opérationnelles

Stocks/capacité, cannibalisation, réactions concurrentielles.

● KPI de pilotage

Marge, CA, conversion, revenu/siège/nuits (RevPAR), élasticité estimée.

● Garde-fous & éthique

Transparence, non-discrimination des segments, limites de variation.

Pricing – Estimation de l'élasticité (régression log-log)



Extrait de code – OLS sur $\log(Q) = a + b \cdot \log(P) + e$ (élasticité b)

```
# Estimation simple de l'élasticité-prix (régression log-log)
import numpy as np
import statsmodels.api as sm

# Données jouet (P=prix, Q=quantités vendues)
P = np.array([10, 12, 11, 13, 9, 8], dtype=float)
Q = np.array([120, 95, 105, 90, 140, 160], dtype=float)

# Transformation log-log
y = np.log(Q)
x = np.log(P)
X = sm.add_constant(x) # ajoute l'intercept a

# Régression OLS
model = sm.OLS(y, X).fit()
a, b = model.params # a = constante ; b = élasticité-prix

print(f"Élasticité (b) = {b:.3f}")
print(f"IC95% de b : [{model.conf_int().loc[1,0]:.3f}, {model.conf_int().loc[1,1]:.3f}]")
print(f"R² = {model.rsquared:.3f}")
```

Astuce: élasticité $b < -1 \Rightarrow$ demande **élastique** $\bullet -1 < b < 0 \Rightarrow$ **inélastique**.

Interprétation

$b = \frac{\% \Delta Q}{\% \Delta P}$ (log-log). Par ex. $b = -1.2 \Rightarrow +1\% \text{ prix} \rightarrow -1.2\% \text{ quantités.}$

Consulter IC95% et R^2 pour la fiabilité de l'estimation.

Mises en garde

- Utiliser données hors promo/ruptures (effets confondants).
- Vérifier stationnarité et linéarité en logs.

Analyse financière & prévisions



● Budgets glissants & scénarios

Rolling forecast (mensuel), scénarios base/optimiste/pessimiste, budget driver-based (volumes, prix, mix).

● Cash forecasting (trésorerie)

Horizons J+7 / J+30 / 13 semaines; entrées (ventes, encaissements DSO), sorties (achats, salaires, CAPEX). Méthodes: ARIMA/ML.

● Données & préparation

Sources ERP/banque, calendriers de paiement, saisonnalité & événements; nettoyage des anomalies (spikes, ruptures).

● KPI de pilotage

DSO, DPO, DIO; CCC = DSO + DIO - DPO; MAPE (prévision vs réalisé), variance budgétaire, position de cash minimale.

● Gouvernance & contrôle

Piste d'audit, séparation des tâches, validations, versioning des modèles, explicabilité (XAI), seuils d'alerte & plans d'action.

● Reporting & outils

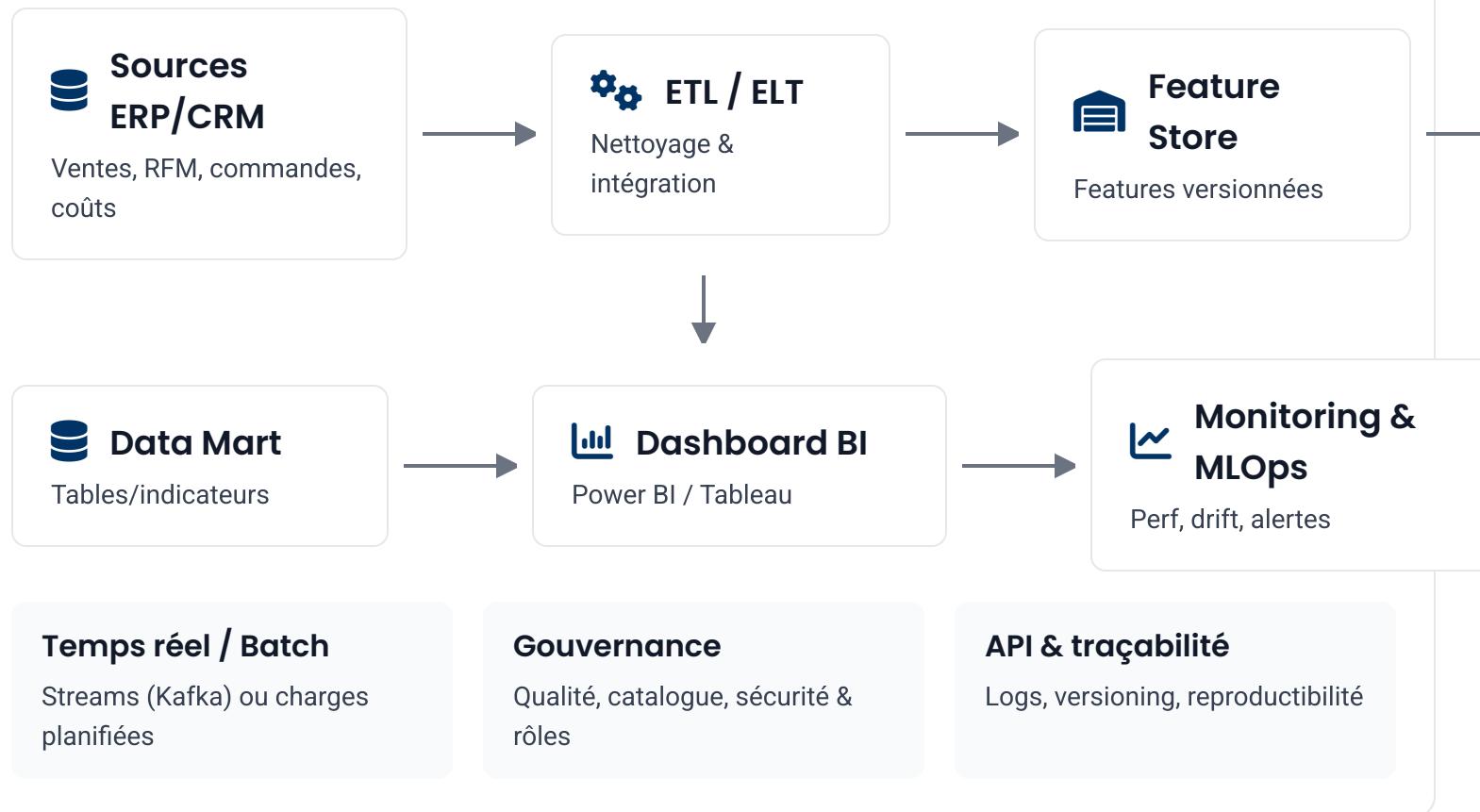
Tableaux de bord (Power BI/Tableau), alertes de dérive (drift), suivi *forecast-to-actual*, partage avec finance/opérations.

Rappel CCC (Cash Conversion Cycle) : **CCC = DSO + DIO - DPO** → viser un CCC plus court pour améliorer la trésorerie.

BI & dashboards prédictifs – Architecture



Schéma d'architecture



Points clés de mise en œuvre

- **ETL/ELT** (Airflow/DBT) et **Feature Store** partagés
- **Modèles** (API) Serving modèles via API (latence contrôlée), Scoring/Prévision
- Data mart orienté **métriques** métiers.
- Dashboards BI avec **indicateurs prédictifs**.

KPI & gouvernance

Accuracy/MAE des modèles, fraîcheur des données, adoption BI, conformité RGPD.

Résumé

Chaîne ERP → ETL/ELT → Features → Modèles → Data mart → BI avec monitoring continu.

Approches PERT/CPM (classiques)

- Réseau d'activités: nœuds/arcs, dépendances, dates au plus tôt/au plus tard
- Chemin critique (CPM): marges nulles, pilotage délai/coût/ressources

Estimation PERT (durée attendue et variance)

Durée: $E = (a + 4m + b) / 6$ • Variance: $Var = ((b - a)^2) / 36$

a = optimiste • m = le plus probable • b = pessimiste

- Ressources: nivellation/affectation, contraintes de capacité et fenêtres
- KPI: respect délais, dérive coûts, chemin critique, risque planning

Apports de l'IA/ML au pilotage

Explicabilité requise

- Estimation des durées et coûts par apprentissage supervisé (Régression, GBM/XGBoost)

Features: complexité tâche, compétences, charge, historique, saisonnalité

- Prédiction des risques de retard/derrapage (classification, SHAP pour causes)

Alertes précoce, priorisation des tâches critiques

- Simulation Monte-Carlo pilotée par ML

Distributions issues d'historiques → probabilités d'achèvement par date

- Ordonnancement intelligent

Heuristiques CPM + optimisation (OR-Tools) / RL pour arbitrages ressources

Impacts (observés typiques):

Dérive délais -10 à -20%

Respect planning +5 à +10 pts

Gouvernance: traçabilité des modèles, revue périodique, intégration PMO/BI.

Audit & contrôle interne – Détection d'anomalies



Usages (exemples)

- Écritures comptables atypiques (montants, dates, comptes miroir)
- Doublons fournisseurs / paiements suspects
- Incohérences TVA / factures (numérotation, taux, pièces manquantes)
- Séparation des tâches et contournement de contrôles

Approche technique

Précision & explicabilité

Données & features

Journal comptable
(montants, comptes, dates)

Agrégats par tiers / libellé /
période

Algorithmes

IsolationForest, LOF, One-Class SVM; règles + ML; autoencoder

Pipeline & KPI

Prétraitement (normalisation, encodage), seuils d'alerte

Métriques: Precision@K, Recall@K, coût faux positifs

Gouvernance & XAI

Revues humaines, journalisation, explications (SHAP), traçabilité

Audit – Exemple Python (IsolationForest sur écritures)



Extrait de code – Détection d'anomalies en audit (IsolationForest)

```
# 1) Import des bibliothèques
import pandas as pd
from sklearn.ensemble import IsolationForest

# 2) Charger les écritures comptables
j = pd.read_csv('journal.csv') # colonnes : date, compte, tiers, montant, ...

# 3) Sélection des features numériques et nettoyage simple
X = j.select_dtypes('number').fillna(0)

# 4) Entraînement du modèle (anomalies attendues ~2%)
iso = IsolationForest(n_estimators=200, contamination=0.02, random_state=42)
iso.fit(X)

# 5) Prédiction : -1 = anomalie, 1 = normal
j['anomaly'] = (iso.predict(X) == -1).astype(int)

# 6) Score d'anomalie (plus négatif = plus suspect)
j['score'] = iso.decision_function(X)

# 7) Taux d'anomalies et top 5 écritures suspectes
rate = j['anomaly'].mean()
print(f'Taux anomalies : {rate:.2%}')
top5 = j.nsmallest(5, 'score')[['anomaly', 'score']]
print(top5)
```

Astuce : réglez contamination selon le taux d'anomalies anticipé et validez par revue humaine.

Lecture des résultats

anomaly = 1 signale une écriture atypique à investiguer.

score < 0 indique des points plus suspects (priorisation).

Exporter les top5 pour revue par l'audit.

Bonnes pratiques

- Inclure variables catégorielles (one-hot) et dates (saisonnalité).
- Comparer avec LOF/Autoencoder; documenter seuils d'alerte.
- Tracer un log d'audit pour conformité (RGPD, contrôle interne).

Chatbots RH – Architecture (FAQ → RAG → LLM)



Schéma de flux



Automatisation des processus (RPA + IA) – 4 composantes



OCR & Capture intelligente

Numériser documents (factures, KYC), extraire champs clés (IBAN, montants, dates) et valider la qualité avant traitement.



Règles métiers & ML

Appliquer règles déterministes et modèles IA (classification, NER, scoring) pour décider, prioriser et détecter anomalies.



Orchestration RPA

Exécuter les workflows bout-en-bout (bots, files d'attente, gestion exceptions), intégrés aux SI (ERP/CRM/Email/API).



Monitoring & Audit

Suivre KPI (temps de traitement, taux d'automatisation), tracer les décisions, alerter dérives et assurer conformité.

Cas d'usage – ROI CRM & Supply chain



CRM – Recommandation & personnalisation

Impacts commerciaux

+4–8% Chiffre d'affaires incrémental

Cross/upsell via recommandations « next-best-offer »

+6–12% Taux de conversion

Personnalisation des offres & canaux

-10–20% Churn (attrition client)

Actions de rétention ciblées par scoring de churn

Hypothèses & horizon

Données RFM/CRM propres

Horizon 6–12 mois

ROI piloté par A/B tests; coûts: intégration, licences, formation

Supply chain – Prévision & optimisation

Impacts opérationnels

-10–20% Stocks moyens

Meilleure précision de prévision de la demande

-8–15% Ruptures

Recalage des seuils de réapprovisionnement

+3–6 pts Taux de service

Meilleure disponibilité produit & pilotage des priorités

Hypothèses & horizon

Données historiques propres

Horizon 3–9 mois

Coûts: intégration ERP/SCM, optimisation paramètres, conduite du changement

Quiz interactif – Chapitre 9 (Gestion)



1

CRM – Recommandation

Quelle(s) métrique(s) prioriser pour évaluer un système de recommandation (ex. Precision@K, Recall@K, taux de conversion) et pourquoi ?

2

Segmentation clients

Comment valider la qualité d'une segmentation RFM par K-Means (indices silhouette, profilage business, stabilité) ?

3

Prévision de la demande

Citez 2 KPI pour évaluer une prévision de demande (ex. MAPE, WAPE) et 2 facteurs à prendre en compte (saisonnalité, promotions).

4

Pricing dynamique

Qu'est-ce que l'élasticité-prix et comment l'estimer simplement (régression log-log) pour piloter des tests A/B ?

5

Supply chain – Routage (VRP)

Citez 2 contraintes clés d'un VRP (ex. capacités véhicule, fenêtres de temps) et 2 KPI d'évaluation (km, temps, taux de service).

Chapitre 10 – Éthique et IA responsable : enjeux



- **Risques clés** : décisions injustes, opacité, dérives d'usage, atteintes à la vie privée
- **Pourquoi maintenant ?** Diffusion massive de l'IA, impacts sociaux, nouvelles régulations (RGPD, AI Act)
- **Objectifs** : équité, transparence, sûreté/robustesse, respect de la vie privée
- **Secteurs sensibles** : crédit/assurance, santé, RH/recrutement, services publics
- **Rôle du manager** : gouvernance et contrôles, arbitrage coûts/risques, adoption responsable, pilotage de la conformité

Biais : sources fréquentes (4 familles)



Données

Échantillonnage non représentatif, historique discriminant, variables manquantes; collecte déséquilibrée ou non pertinente.



Mesure

Étiquetage erroné, capteurs/champs incomplets, proxy mal choisis, dérives de définition entre systèmes.



Modèle

Sur/Sous-apprentissage, objectifs mal pondérés (coûts FP/FN), variables sensibles non contrôlées, explicabilité insuffisante.



Opérationnel

Data/Concept drift, boucles de rétroaction, adoption inégale par les équipes, contraintes réglementaires ignorées.



Définitions formelles

Démographic Parity

$$P(\hat{y} = 1 | A = a) \approx P(\hat{y} = 1 | A = b)$$

Taux de positifs égalisés entre groupes.

Disparate Impact (règle 80%)

$$DI = P(\hat{y} = 1 | A = a) / P(\hat{y} = 1 | A = b) \geq 0.8$$

Rapport des taux positifs entre groupes.

Equal Opportunity

$$TPR_a = P(\hat{y} = 1 | Y = 1, A = a) \approx TPR_b$$

Égalité des vrais positifs (sensibilité).

Equalized Odds

$$TPR_a \approx TPR_b \quad \& \quad FPR_a \approx FPR_b$$

Égalité TPR et FPR entre groupes.

Calibration intra-groupe

$$P(Y = 1 | \hat{p} = p, A = a) = p$$

Probabilités prédites fiables par groupe.



Remarque – Choix de la métrique selon le contexte

Les critères peuvent être incompatibles simultanément. Sélectionner la métrique (parité, opportunité, odds, calibration) en fonction des coûts d'erreur et des exigences réglementaires.

Fairness metrics – Comparaison et usages



| Métrique | Définition / objectif | Quand l'utiliser |
|---------------------------------|--|---|
| Demographic Parity | Taux de positifs similaire entre groupes: $P(\hat{y}=1 A=a) \approx P(\hat{y}=1 A=b)$. Variante: Disparate Impact $\geq 0,8$. | Recrutement/marketing à volume cible, allocation de ressources où la parité de sélection est recherchée. |
| Equal Opportunity | Égalité du rappel par groupe pour la classe positive: TPR identique entre groupes. | Accès au crédit / dépistage où l'on veut éviter de manquer des positifs dans un groupe (réduire faux négatifs). |
| Equalized Odds | Égalité simultanée de TPR et FPR entre groupes (erreurs symétriques alignées). | Décisions sensibles avec coûts d'erreurs similaires (ex. contrôle/filtrage) et exigence d'équité stricte. |
| Calibration intra-groupe | Probabilités cohérentes par groupe: $P(Y=1 \hat{y}=p, A=a)=p$. Même niveau de confiance pour un score donné. | Tarification / scoring probabiliste (assurance, crédit) où la fiabilité des scores influe sur le pricing et la gestion du risque. |

Transparence & explicabilité – leviers



Post-hoc

Expliquer des modèles opaques sur données réelles.

LIME / SHAP

PDP / ICE



Documentation

Rendre explicites objectifs, données et limites.

Model Cards

Datasheets



Traçabilité

Suivre versions données, features et modèles.

Feature store

Model registry • Logs



Communication

Informer clairement utilisateurs et décideurs.

Motifs standardisés

Droit à l'explication (RGPD)

Principes RGPD

Conformité

- Base légale & finalité déterminée
Traitements fondés et limités à des objectifs explicites.
- Minimisation & exactitude
Données adéquates, pertinentes, tenues à jour.
- Limitation de conservation
Durées définies, archivage sécurisé, purge planifiée.
- Droits des personnes
Accès, rectification, effacement, opposition, portabilité.
- DPIA & risque
Étude d'impact pour traitements à risque élevé.
- Article 22 (décision automatisée)
Supervision humaine & droit à l'explication.

Techniques de mise en œuvre

- Anonymisation / pseudonymisation
Réduire le risque de ré-identification.
- Masquage / tokenisation
Cacher champs sensibles (PII) en non-prod.
- Chiffrement
Au repos & en transit (KMS, TLS).
- Agrégation & échantillonnage
Limiter la granularité et l'exposition.
- Contrôle d'accès (RBAC)
Moindre privilège, revue périodique.
- Journalisation & traçabilité
Logs d'accès, data lineage, alertes.

Privacy by design / by default : intégrer la protection des données dès la conception.



Evasion (exemples adversariaux)

Perturbations à l'inférence pour tromper le modèle.
Contremesures : robustesse, détection d'anomalies, défenses adversariales.



Poisoning (empoisonnement)

Altération des données d'entraînement pour biaiser/dégrader le modèle. Contremesures : hygiène des données, validation, filtrage.



Model extraction (exfiltration)

Reconstitution du modèle via des requêtes API.
Contremesures : rate-limiting, réponses tronquées, watermarking, surveillance.



Membership inference

Inférer si un individu appartenait au jeu d'entraînement.
Contremesures : régularisation, confidentialité différentielle, calibration.

Impacts sociaux & environnement



- Emploi & compétences: substitution/complémentarité, re/up-skilling, nouveaux métiers (data/IA)
- Inégalités & inclusion: biais algorithmiques, fracture numérique, accessibilité & équité d'accès
- Environnement: empreinte énergétique (CO₂e), sobriété/Green AI (datasets, modèles & hardware efficents)
- Acceptabilité sociale & gouvernance: transparence, contrôle humain, participation des parties prenantes

Responsabilité & accountability — Processus en 5 étapes



1 Cartographier les risques

Identifier cas d'usage, parties prenantes, impacts potentiels (équité, vie privée, sécurité, conformité).

Livrable : registre des risques IA.

2 Définir les rôles

Attribuer responsabilités : propriétaire de modèle, risk owner, DPO, équipe validation indépendante.

Livrable : RACI & gouvernance.

3 Mettre des contrôles

Contrôles ex-ante et ex-post : revues, validation, seuils d'alerte, tests d'équité/robustesse, documentation.

Livrable : plan de contrôle & checklists.

4 Suivre les incidents

Mettre en place un registre d'incidents, SLA de remédiation, analyses de causes, actions correctives/préventives.

Livrable : registre & plans CAPA.

5 Rapporter

Reporting au comité éthique/risques, audit périodique, communication régulateurs, indicateurs de pilotage.

Livrable : tableaux de bord & rapports.

Objectif : assurer une **gouvernance IA** continue (équité, conformité, robustesse) tout au long du cycle de vie du modèle.

Principes FAIRS – IA responsable



Fairness

Équité et non-discrimination entre groupes; mesurer et atténuer les biais.

DP / EO / EOdds

Politiques d'équité



Accountability

Gouvernance claire, rôles et contrôles; traçabilité et audits périodiques.

Model Registry

Comité éthique



Interpretability

Explications compréhensibles des décisions (LIME, SHAP, PDP/ICE).

Transparence

RGPD art. 22



Reliability

Robustesse, sécurité, tests et monitoring (performance, drift, attaques).

Robustesse

Monitoring



Sustainability

Réduire l'empreinte carbone, optimiser les ressources, respecter l'environnement et la société.

Green AI



Classification des risques (EU AI Act)

Risque inacceptable – Interdit

Manipulation subliminale, notation sociale, exploitation vulnérabilités.

Risque élevé – Fortes obligations

Crédit, recrutement, éducation, santé, infrastructures critiques.

Risque limité – Transparence

Obligation d'informer (ex. systèmes conversationnels, deepfakes).

Risque minimal – Libre usage

Outils courants (filtre spam, assistants simples) – bonnes pratiques.

La classification conditionne les obligations applicables et les contrôles.

Obligations pour systèmes à risque élevé

Conformité
CE

Gestion des risques

Analyse, mitigation, tests avant mise en service.

Documentation technique

Dossiers techniques, instructions d'usage.

Transparence & supervision humaine

Information utilisateur, contrôle humain effectif.

Gouvernance des données

Qualité, pertinence, représentativité, traçabilité.

Journalisation / traçabilité

Logs suffisants pour audit et analyse d'incident.

Robustesse & précision

Sécurité, cybersécurité, performance mesurée.

Évaluation de conformité & marquage CE

Surveillance post-marché & reporting incidents

Cas d'éthique – finance/gestion

...

Apple Card • Amazon Hiring • COMPAS



Apple Card (2019) – Limites de crédit

Allégations d'écart de limites entre co-emprunteurs (genre) → enquête régulatrice.

Points clés :

- Risque de disparate impact non intentionnel
- Leçons: co-applicants, seuils, audits XAI requis



Amazon Hiring (2018) – Recrutement

Modèle pénalisant des CV féminins (biais appris des historiques masculins).

Points clés :

- Biais de données: labels et représentativité
- Leçons: filtrer features, fairness tests, supervision RH



COMPAS – Scores de risque (justice)

Différences TPR/FPR entre groupes; débats sur la « bonne » métrique d'équité.

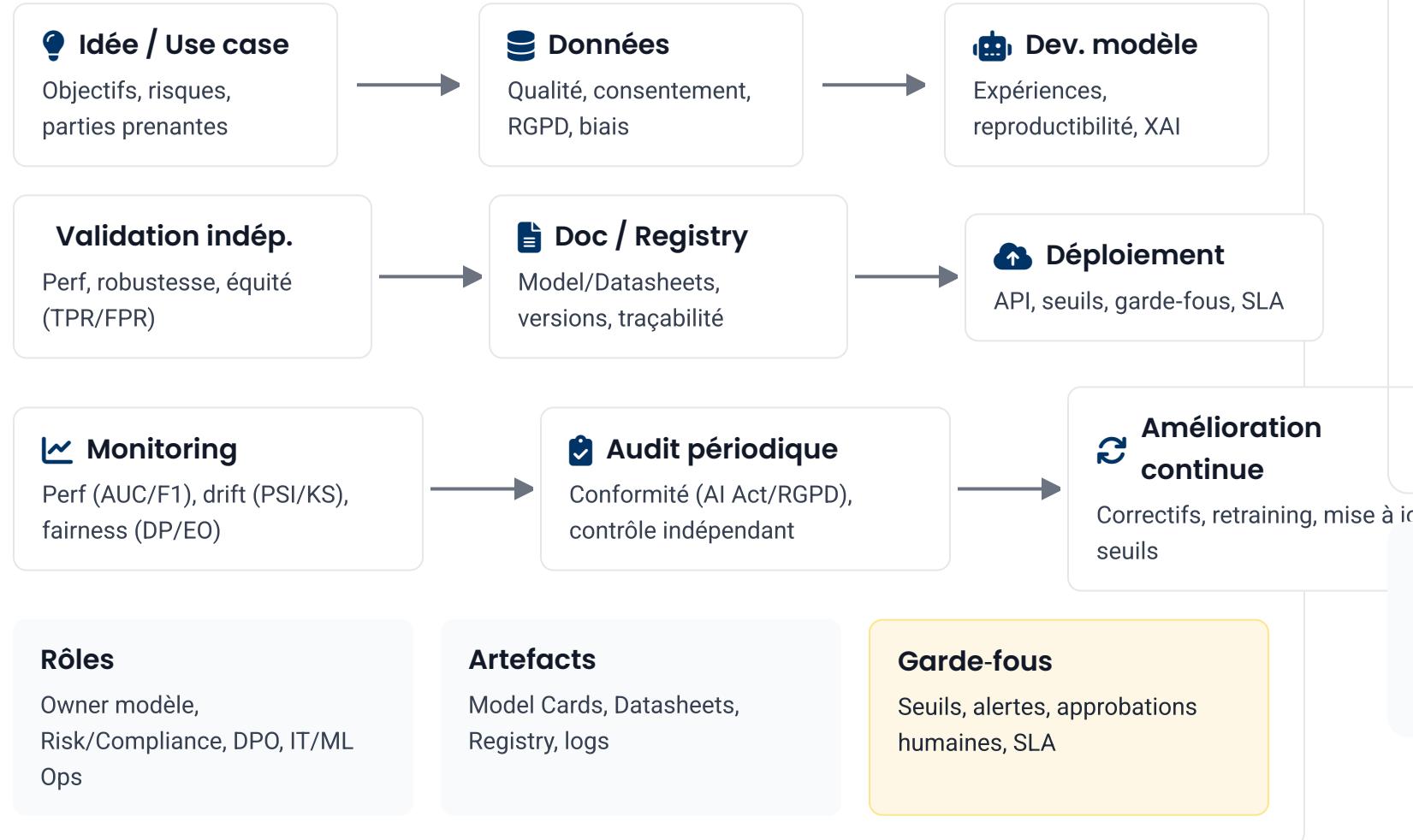
Points clés :

- Trade-offs: Equal Opportunity vs Calibration
- Leçons: choix de métriques, transparence, contrôle humain

Audits algorithmiques & gouvernance IA – Schéma



Parcours de l'audit algorithmique (de l'idée à l'amélioration continue)



Points clés de gouvernance

- **Politiques IA** alignées RGPD & AI Act.
- **Contrôles** avant/après déploiement (indépendants).
- **Indicateurs** suivis: AUC/F1, FPR, PSI, DP/EO.
- **Fréquence** d'audit: mensuelle/trimestrielle.

Décisions

Comité éthique / Model Risk, plans de remédiation, communication aux métiers

Résumé

Un cycle **documenté et contrôlé** assure conformité, transparence et amélioration continue des modèles d'IA.

Code Python – Mesurer des écarts d'équité (DPD & EOD)



Extrait – Démographic Parity Diff (DPD) & Equal Opportunity Diff (EOD)

```
# Mesure simple des écarts d'équité : DPD et EOD
import numpy as np

# y_true : étiquette réelle (0/1) ; y_pred : prédiction binaire (0/1)
# A : attribut sensible (0/1) – ex. groupe de référence=0, groupe protégé=1

def rate(y):
    return y.mean() if y.size > 0 else np.nan

def dpd(y_pred, A):
    # Demographic Parity Difference : P(ŷ=1|A=1) - P(ŷ=1|A=0)
    p1 = rate(y_pred[A == 1])
    p0 = rate(y_pred[A == 0])
    return p1 - p0

def tpr(y_true, y_pred):
    pos = (y_true == 1)
    return ((y_pred[pos] == 1).mean()) if pos.any() else np.nan

def eod(y_true, y_pred, A):
    # Equal Opportunity Difference : TPR(A=1) - TPR(A=0)
    tpr1 = tpr(y_true[A == 1], y_pred[A == 1])
    tpr0 = tpr(y_true[A == 0], y_pred[A == 0])
    return tpr1 - tpr0

# Exemple minimal (pour illustration) :
y_true = np.array([1,0,1,0,1,0])
y_pred = np.array([1,0,1,0,0,0])
A      = np.array([1,1,0,0,1,0])

dp_diff  = dpd(y_pred, A)  # > 0 : + de positifs pour A=1 ; < 0 : l'inverse
eo_diff  = eod(y_true, y_pred, A) # > 0 : TPR plus élevé pour A=1

print(f"DPD = {dp_diff:.3f}, EOD = {eo_diff:.3f}")
```

Interprétation

DPD = $P(\hat{y}=1|A=1) - P(\hat{y}=1|A=0)$.

EOD = $TPR(A=1) - TPR(A=0)$.

Valeurs proches de 0 suggèrent plus d'équité (selon la métrique).

Bonnes pratiques

- Comparer plusieurs métriques (DPD, EOD, calibration).
- Analyser l'impact métier (coûts FP/FN).
- Documenter A, le contexte légal (RGPD/AI Act) et les remédiations.

Astuce: normaliser les labels/attributs (0/1) et vérifier la taille des sous-populations pour éviter des métriques instables.

Quiz interactif – Chapitre 10 (Éthique)



1

Sources de biais & atténuation

Citez 2 sources de biais (ex. données, mesure, modèle, opérationnel) et un moyen d'atténuation (ex. re-sampling, reweighing, audit).

2

Equal Opportunity

Définissez Equal Opportunity (TPR égal entre groupes) et donnez un contexte d'usage (ex. dépistage, accès au crédit).

3

AI Act – Risque élevé

Quelles obligations clés pour un système « risque élevé » (gestion des risques, qualité données, documentation, traçabilité, surveillance humaine, robustesse) ?

4

Anonymisation vs pseudonymisation

Expliquez la différence et l'impact RGPD (données personnelles encore présentes en cas de pseudonymisation).

5

Audit algorithmique efficace

Décrivez un processus d'audit: cartographier risques → rôles & responsabilités → tests (perf, drift, fairness) → documentation/registre → plans de remédiation.

Chapitre 11 – MLOps : pourquoi et pour qui ?

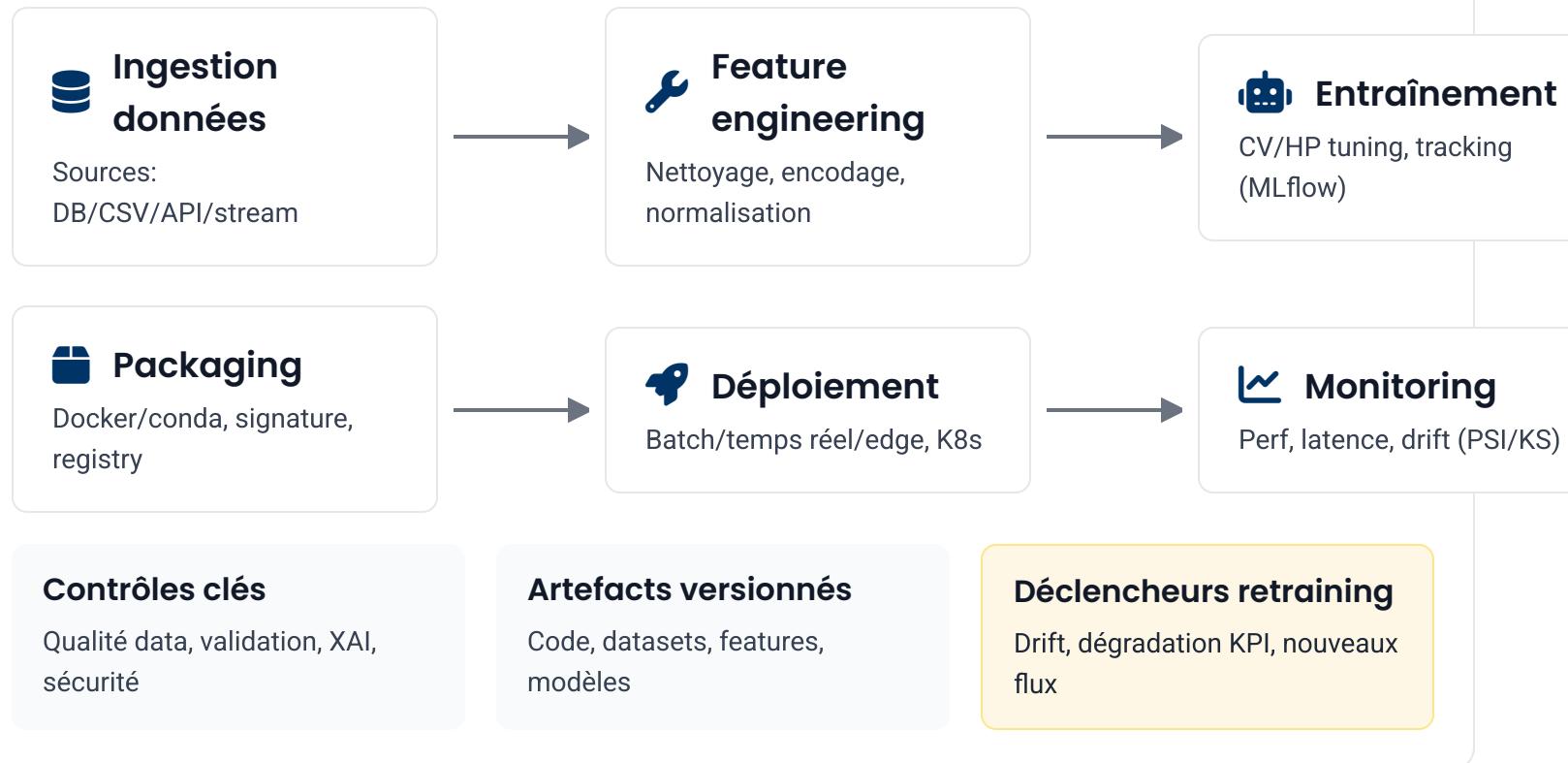


- Objectifs MLOps : fiabilité des modèles, reproductibilité, time-to-production réduit
- Bénéfices clés : traçabilité données/modèles, dette technique ↓, conformité (audit, RGPD/AI Act)
- Bonnes pratiques : versioning (code/data/models), environnements Docker, pipelines automatisés
- Parties prenantes : Data/ML engineers, DevOps/IT Ops, métiers, risques & conformité
- Contexte ENCG – cas d'usage : scoring crédit, détection de fraude, prévision de demande/ventes
- Attendus du cours : comprendre la chaîne MLOps, choisir des outils (MLflow/DVC/K8s), piloter des déploiements

Cycle de vie ML – de la data au monitoring & retraining



Schéma de flux



Points de contrôle

- **Data quality** (schémas, valeurs, NA, ranges).
- **Performance** (AUC/F1/RMSE) & **latence**.
- **Drift entrée/sortie** (PSI, KS) + alertes. Jeu test, XAI, fairness, sécurité.
- **Conformité/XAI** (explications, logs, traçabilité).

Pratiques MLOps

CI/CD, registry, tests data, canary/A-B, rollback.

Periodique / déclencheur (drift)

Contrôles clés

Qualité data, validation, XAI, sécurité

Artefacts versionnés

Code, datasets, features, modèles

Déclencheurs retraining

Drift, dégradation KPI, nouveaux flux

Résumé

Le cycle de vie ML est **itératif** et **monitoré** pour garantir fiabilité, conformité et valeur métier.

Versioning des données et des modèles



Concepts de versioning

Traçabilité & Reproductibilité

Objets à versionner

Code, **datasets** (ou snapshots), features, **hyperparamètres**, métriques, artefacts (modèles, scaler, schémas).

Traçabilité de bout en bout

Chaque *run* relie commit → data snapshot → config → modèle → métriques → environnement.

Reproductibilité

Environnement figé (Docker), seed fixé, dépendances connues, données immuables (hash/checksum), pipeline déclaré.

Bonnes pratiques

Nommage clair des versions, tags sémantiques (v1.2.0), review indépendante et Model Card minimale.

Outils & pratiques (Git / DVC / MLflow)

Prod-ready

Git/LFS

Versionne le **code**, notebooks et petits artefacts; Git LFS pour fichiers volumineux; branches feature/, release/ et tags.

DVC

Gère **datasets** et pipelines (dvc.yaml); stockage distant (S3/GDrive/Azure); dvc.lock fige versions; dvc repro rejoue la chaîne.

MLflow

Trace **expériences** (params, métriques, artefacts) et gère le **Model Registry** (Staging → Production) avec signature d'entrée.

Checklist versioning

- Commit + tag du run
- Data snapshot DVC
- MLflow run complet

Traçabilité minimale

- hash data + modèle
- requirements/Docker
- Model Card courte

Astuce: relier MLflow run à un tag Git et au dvc.lock pour une traçabilité end-to-end.

Environnements reproductibles – Docker & virtualenv



Concepts & bonnes pratiques

Docker

virtualenv/conda

- Reproductibilité: figez OS, dépendances et versions pour supprimer les écarts « fonctionne chez moi ».
- Isolation: conteneurs (Docker) pour la prod; virtualenv/conda pratique pour le dev local.
- Fichier **requirements.txt** (ou **environment.yml**) et **Dockerfile** comme artefacts versionnés (Git).
- Sécurité & taille: base image minimale (**python:***-slim), **.dockerignore**, pins de versions.
- CI/CD: build image → tests → scan vulnérabilités → push registry → déploiement.
- Règle d'or: « même code, mêmes données, même environnement » du notebook au serveur.

Exemple – Dockerfile (service de prédiction)

Docker

```
# Image de base légère
FROM python:3.10-slim

# Répertoire de travail
WORKDIR /app

# Dépendances (pinnées)
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Code de l'appli
COPY . .
```

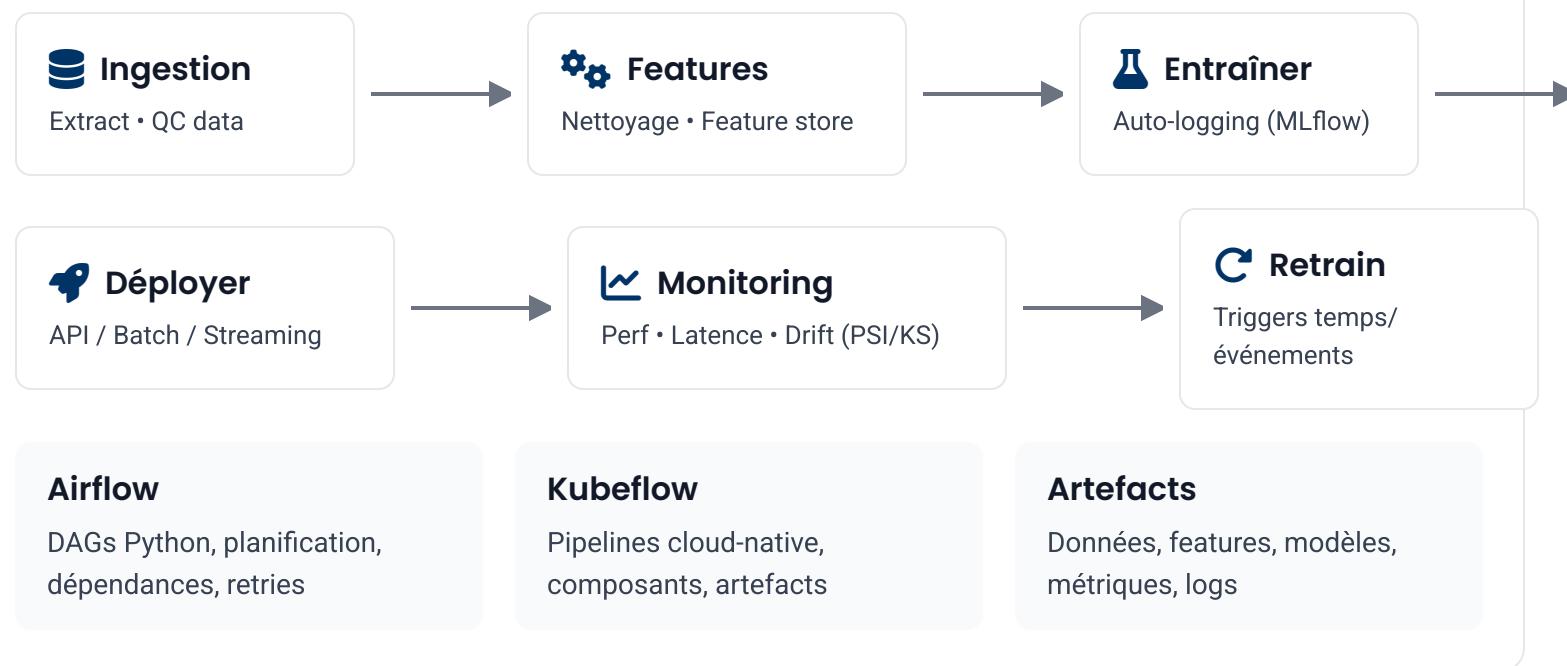
Build & run: docker build -t ia-service . puis docker run -p 8080:8080 ia-service.

Pipelines ML automatisés – DAG Airflow/Kubeflow



Orchestration par DAG (Airflow / Kubeflow Pipelines)

Orchestrateur (planification, dépendances, retries) • Journalisation • Métadonnées d'exécution



Points clés (orchestration)

- **Idempotence et réexécutions** par tâche.
- **Séparation** data/compute; stockage d'artefacts.

Évaluation régulière des données à chaque étape. Métrique et model registry

- Traçabilité: métadonnées, **runs**, versions.

Bonnes pratiques

Déclencher retraining par drift/temps, contrôler coûts et SLAs, intégrer XAI.

Résumé

Un **DAG** définit l'ordre et la fiabilité des tâches ML de bout en bout – de l'ingestion au monitoring – pour un passage en production **robuste**.

Tracking d'expériences – MLflow / Weights & Biases



Quoi logger (expériences)

Traçabilité

- Paramètres (hyperparamètres)
lr, max_depth, batch_size, seed...
- Métriques
loss, AUC/F1/PR-AUC, RMSE, latence P95
- Artefacts
model.pkl, scaler, figures (ROC, PR), rapports
- Données & schéma
version dataset, features, signature d'entrée/sortie
- Environnement
requirements.txt/conda.yaml, versions lib
- Lien au code
commit Git, tag, message
- Métadonnées
run id, nom d'expérience, tags (expérience, data_version)

Bonnes pratiques de tracking

- Nommage clair des runs & projets
convention: usecase_modèle_date_version
- Activer l'autologging
mlflow.sklearn.autolog() ou W&B callbacks
- Lier run → commit Git
logger le SHA, lien repository
- Versionner les données
DVC ou hash dataset; éviter la dérive silencieuse
- Fixer les seeds
reproductibilité (numpy/pytorch/sklearn)
- Comparer & promouvoir via registre
Model Registry (Staging → Prod), critères objectifs
- Dashboards & alertes
suivre métriques clés, alerter régression perf
- Conformité & privacy
éviter PII dans logs; contrôler l'accès aux artefacts

Feature Store – standardiser et réutiliser les features



- **Définition** – Référentiel centralisé de variables (features) *versionnées*, documentées, disponibles en **hors-ligne** (batch) et **en ligne** (temps réel) pour entraînement et service.
- **Bénéfices clés** – Cohérence *train/serve*, réutilisation des features, gouvernance (qualité, lignée), réduction du *time-to-market*.
- **Qualités attendues** – Fraîcheur/latence maîtrisées, historisation, contrôles de qualité, accès sécurisé, calcul batch/stream.
- **Exemples d'outils** – **Feast** (open-source), **Tecton** (SaaS/Cloud). Intégration possible avec MLflow, Spark, Kafka.
- **Cas d'usage** – Scoring crédit temps réel (ancienneté compte, ratio dette), **churn** marketing, **détection de fraude** (agrégats transactionnels), **prévision de la demande** (features calendaires).
- **Bonnes pratiques** – Définir des *feature views* (schema + source + transformation), tests de dérive/qualité, gestion des droits et du cycle de vie.

Model Registry — Processus en 5 étapes



Objectif

Gérer les versions de modèles, leur qualité et leur cycle de vie de manière gouvernée et traçable.

1 Enregistrement

Créer une entrée avec **version, signature, métriques et artefacts** (ex. MLflow).

Etat: **None** → **Staging**

2 Revue / Validation

Contrôles qualité, **tests, XAI, fairness**, conformité et approbation du risk/IT.

Gate: critères d'acceptation définis

3 Promotion

Changer de **stade** (Staging → **Production**) avec **approbations** et plan de rollback.

Déploiement: batch/temps réel

4 Dépréciation

Retirer la version active (remplacement/rollback), **archivage** des artefacts et documentation.

Motifs: drift, perf, risques

5 Traçabilité

Lien **données, features, params, commits, métriques et décisions** pour audit.

Preuve d'audit & conformité

Bonnes pratiques — **Model Registry**: versions explicites, règles de promotion, journaux de décisions, et liens vers **XAI** et tests de robustesse.

Stratégies de déploiement (MLOps)



Batch (traitement par lots)

Exécutions planifiées (ETL nocturnes, scoring hebdo).

Idéal quand la latence n'est pas critique.

Contraintes : fenêtres d'exécution, gestion des dépendances, coûts calcul/storage.



Temps réel (API faible latence)

Inférences synchrones via API (<100 ms) pour décision instantanée (ex. scoring transaction).

Contraintes : SLA/haute dispo, autoscaling, validation/observabilité en ligne.



Streaming (évenementiel)

Traitement flux (Kafka/Kinesis) : détection fraude, alertes, personnalisation continue.

Contraintes : ordering/lag, exactly-once, backpressure, monitoring du throughput.



Edge (sur device)

Inférence embarquée (mobile/IoT) pour latence minimale et fonctionnement offline.

Contraintes : ressources limitées, sécurité, mises à jour modèles & compatibilité.



Concepts clés (côté service)

- Endpoints REST clairs: POST /predict, POST /predict_proba, GET /health.
- Contrats d'entrée/sortie (JSON) + validation de schéma (pydantic/Marshmallow).
- Prétraitement cohérent train/serve (pipeline/feature store).
- Journalisation & traçabilité: requête, version modèle, latence.
- Sécurité: authentification (token), CORS, TLS, contrôle des quotas.
- Robustesse: timeouts, gestion d'erreurs (codes HTTP), limite de taille de payload.
- Tests avec curl/Postman; surveiller la latence P95/P99.

Code Flask – endpoint /predict

Flask · scikit-learn

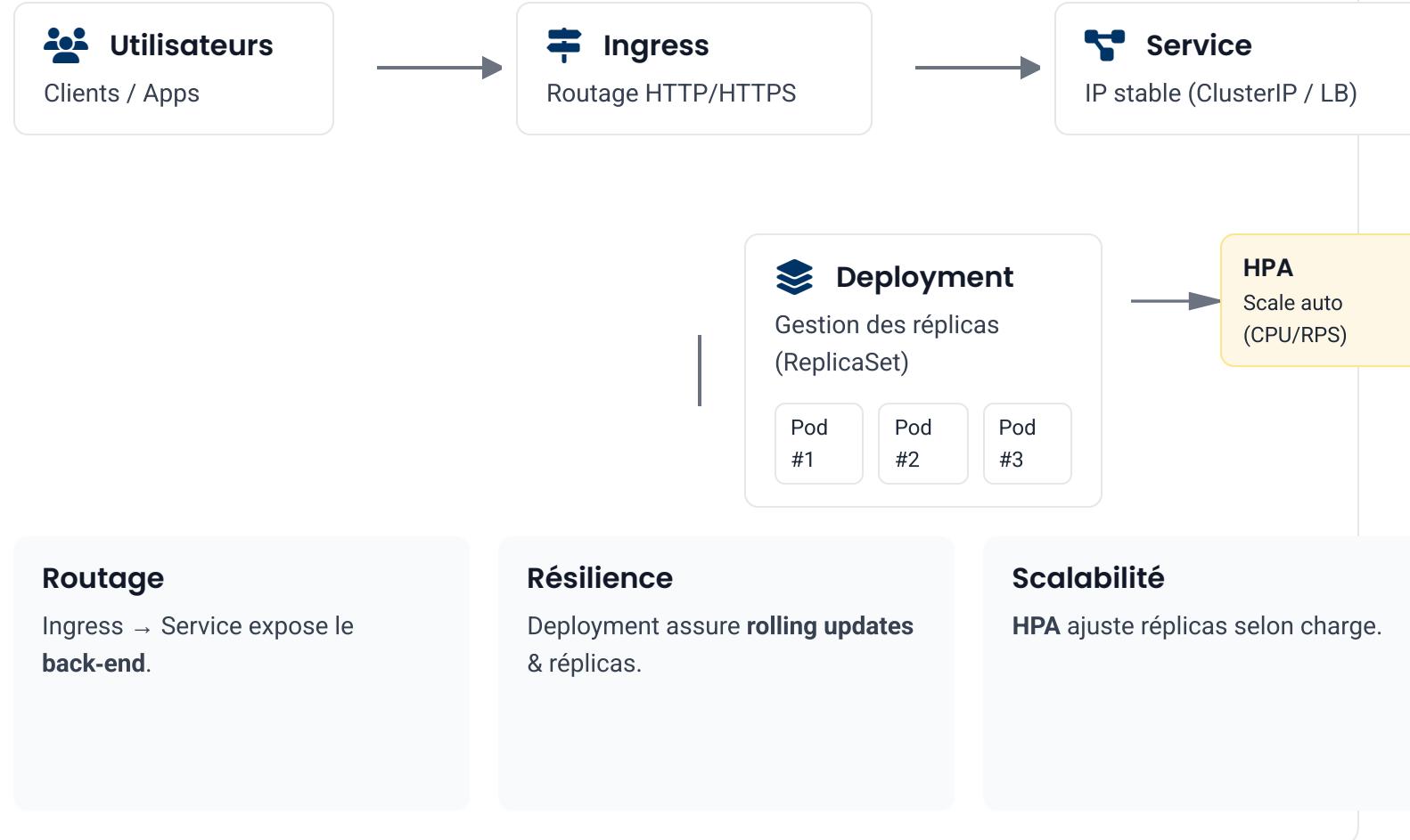
```
from flask import Flask, request, jsonify
import joblib
import numpy as np

app = Flask(__name__)
model = joblib.load("model.joblib")    # modèle sklearn
scaler = joblib.load("scaler.joblib")   # pipeline/standardisation (si
besoin)
FEATURES = ["age","revenu","dette_ratio","historique_credit"]

def preprocess(payload: dict):
    # Validation minimale du schéma d'entrée
    if "features" not in payload: raise ValueError("Champ 'features'")

Contrat JSON attendu: {"features": {"age":..., "revenu":..., "dette_ratio":..., "historique_credit":...}}
```

Schéma d'orchestration



Points clés

- **Ingress** : point d'entrée HTTP/HTTPS, routage par hôtes/chemins.
- **Service** : découverte & IP stable (ClusterIP/NodePort/LB).
- **Deployment** : gère Pods & mises à jour progressives.
- **HPA** : scale horizontal sur CPU, RPS, latence.

Bonnes pratiques

- Probes **readiness/liveness**, ressources **requests/limits**.
- Observabilité : Prometheus/Grafana, logs corrélés.

Résumé

Ingress → Service → Deployment/Pods ; **HPA** ajuste dynamiquement le nombre de Pods pour assurer la QoS.

CI/CD pour ML — Pipeline en 6 étapes



Objectif: automatiser et fiabiliser la mise en production des modèles ML avec contrôles de qualité, sécurité et validation progressive.

1 Tests

Unitaires (code), vérifications **data** (schémas, NA, plages), tests entraînement rapide.

Bloque les régressions dès le début.

2 Build Docker

Création de l'image avec **dépendances** figées (Dockerfile) pour reproductibilité.

Taggage par commit/version.

3 Scan sécurité

Analyse image (vulnérabilités), SAST/Secrets, politiques de base **sécurité**.

Fail du pipeline si critique.

4 Staging

Déploiement en environnement **pré-prod** (données anonymisées, petite charge).

Model registry: état "Staging".

5 Tests d'intégration

Contrôles **bout-en-bout**: API, latency, compatibilité features, XAI, journalisation.

Validation fonctionnelle/métier.

6 Production

Promotion "Production", déploiement contrôlé (**canary/A-B**), monitoring perf & drift.

Rollback si seuils dépassés.

Chaînage CI/CD: Tests → Build Docker → Scan → Staging → Tests d'intégration → Prod. Chaque étape génère des artefacts traçables.

Monitoring production – performance, drift et latence



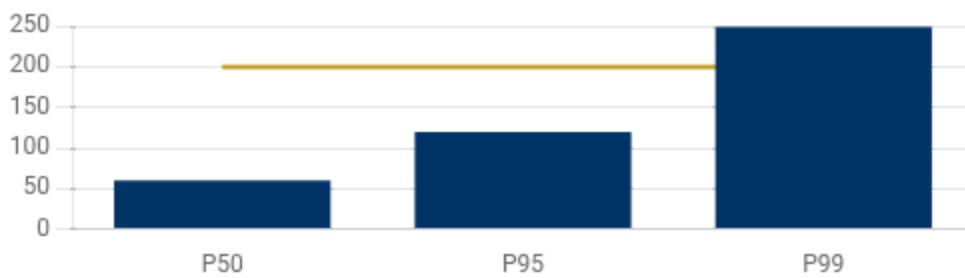
Métriques de production

Suivi continu

- Performance modèle
AUC / F1, Recall@K (fraude),
Brier (calibration)
- Latence
P95 / P99 sous objectif (ex.
200 ms)
- Fiabilité
Taux d'erreurs (4xx/5xx),
disponibilité
- KPI métier
Coût du risque, ROI, alertes
actionnables

Latence (ms) – P50 / P95 / P99 vs cible

Cible 200 ms



Drift & Alerting

Données

Data drift

PSI > 0.2 (alerte), KS $p < 0.05$ (drift feature)

Modèle

Performance drift

AUC $\downarrow > 5\%$ sur 3 jours, F1 en-dessous du seuil

Qualité

Data quality

NA, formats, ranges, valeurs rares/aberrantes

Règles d'alerte

- Seuils + hystérésis
- SLO/SLA documentés
- Escalade (N1 \rightarrow N2)

Actions

- Canary/rollback
- Retraining déclenché
- Gel du scoring si risque

Alertes

Email/Slack + dashboard (Perf, Drift, Latence)

Détection de drift – PSI et test KS (extrait Python)



Extrait de code – calcul du PSI et test KS entre deux distributions

```
# Détection de drift : PSI (Population Stability Index) et test KS
import numpy as np
from scipy import stats as st

# PSI: compare la distribution de a (référence) et b (production)
def psi(a, b, bins=10):
    # mêmes bornes de classes (quantiles) pour les deux échantillons
    edges = np.quantile(a, np.linspace(0, 1, bins+1))
    ca, _ = np.histogram(a, edges)
    cb, _ = np.histogram(b, edges)
    pa = ca/len(a) + 1e-6
    pb = cb/len(b) + 1e-6
    return np.sum((pa - pb) * np.log(pa/pb))

# Exemple d'utilisation (ex: scores/probas modèle)
rng = np.random.default_rng(42)
ref = rng.normal(0.5, 0.12, 5000)      # référence (train)
prd = rng.normal(0.54, 0.13, 5000)      # production (mois courant)

psi_val = psi(ref, prd, bins=10)
ks = st.ks_2samp(ref, prd) # Kolmogorov-Smirnov

print(f"PSI = {psi_val:.3f}")
print(f"KS stat = {ks.statistic:.3f} | p-val = {ks.pvalue:.3e}")

# Seuils usuels (indicatifs) : PSI < 0.10 pas de drift | 0.10-0.25 modéré | >0.25 important
```

Astuce: utiliser les mêmes bornes (edges) pour les deux distributions afin d'obtenir un PSI cohérent.

Interprétation

PSI compare les parts par classe (réf. vs prod.).

Seuils (indicatifs): <0,10 OK • 0,10–0,25 alerte • >0,25 drift fort.

KS teste l'égalité des distributions (p-val < 0,05 → différence significative).

Bonnes pratiques

- Surveiller features clés et scores du modèle.
- Échantillons suffisants (≥ 1000) et fenêtres temporelles stables.
- Définir des seuils d'alerte et un plan de retraining.



Concepts clés (méthodo)

- Test A/B : répartition du trafic (ex. 50/50 ou 70/30) entre **version A** (référence) et **version B** (nouveau modèle).
- Définir **métrique primaire** (ex. AUC/F1, CSAT, marge) et métriques de **garde-fou** (latence P95, taux d'erreur).
- Durée & taille d'échantillon suffisantes pour **significativité** (risque α , puissance $1-\beta$).
- **Canary release** : montée progressive (1% → 5% → 25% → 100%) avec **rollback** si dérive.
- Randomisation par **identifiant stable** (client, session) pour éviter des fuites et assurer la cohérence.

• Décision : promouvoir B si la **métrique cible** s'améliore sans dégrader les garde-fous.

Code Python – Router A/B / canary (Flask)

Flask · Logging

```
from flask import Flask, request, jsonify
import os, random, time

app = Flask(__name__)

# Pour canary, régler TRAFFIC_V2 = 0.05 (5%), puis augmenter si OK
TRAFFIC_V2 = float(os.getenv("TRAFFIC_V2", "0.10"))

# TODO: charger modèles et pipeline
# model_v1, model_v2 = load_v1(), load_v2()

def preprocess(payload):
    # TODO: valider/transformer l'entrée (schéma)
```

Astuce : exporter **TRAFFIC_V2** à 0.05 pour un canary 5% et augmenter par paliers si les **métriques** restent dans les garde-fous.

Retraining automatique – stratégies, prérequis, risques



- **Stratégie périodique** (hebdo/mensuel) – fenêtres fixes, coût prévisible, simplicité opérationnelle.
- **Event-based** (déclenché) – relance sur *drift* (PSI/KS) ou dégradation KPI métier/techniques.
- **Online/stream** – mises à jour incrémentales; nécessite forte observabilité et garde-fous.
- **Prérequis** – pipelines/feature store fiables; **validation** (tests data & modèles, backtesting); **registry + CI/CD** avec plans de **rollback**.
- **Risques** – dérive de l'objectif, dette technique, indisponibilité/instabilité; coûts infra & latency.
- **Mitigation** – seuils & garde-fous, canary/A-B testing, monitoring drift (PSI/KS), XAI pour contrôle, **SLA** et alerting.

MLOps – Cas d'usage finance/gestion



Scoring crédit

API temps réel avec XAI et monitoring continu.

AUC / KS

SHAP

SLA latence



Détection de fraude

Streaming + canary deployment pour limiter le risque.

Precision@K

Faible latence

Canary



Churn marketing

Batch hebdo, ciblage rétention et interprétabilité.

Recall@K

SHAP

ROI



Prévision de la demande

Pipeline quotidien avec retraining contrôlé.

MAPE

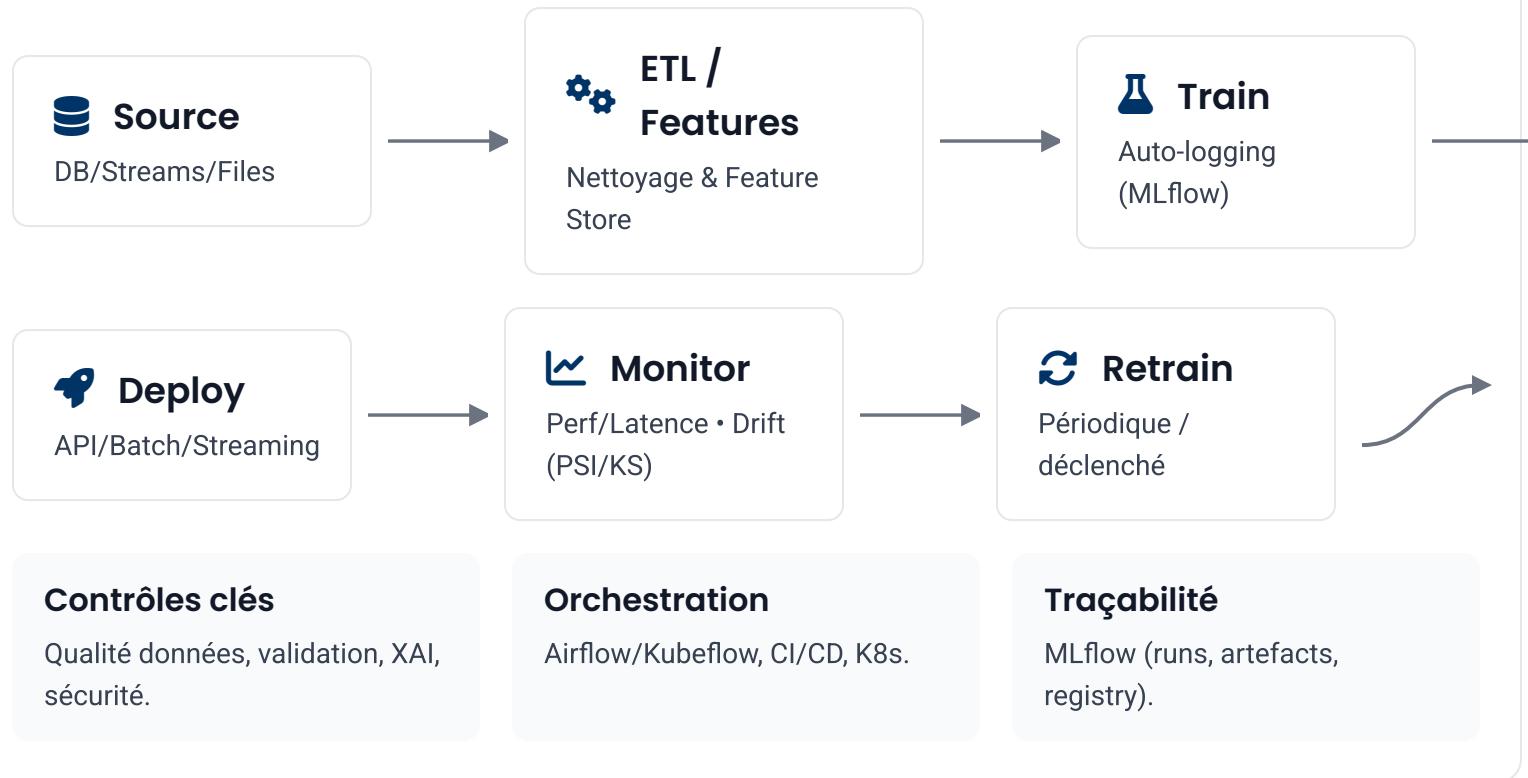
Monitoring drift

Retraining

Pipeline MLOps complet – vue d'ensemble



Schéma de flux



Contrôles clés

Qualité données, validation, XAI, sécurité.

Orchestration

Airflow/Kubeflow, CI/CD, K8s.

Traçabilité

MLflow (runs, artefacts, registry).

Points clés (bonnes pratiques)

- **Séparer** dev/staging/prod, versionner code/données/modèles.
- **Register** perf, dérive (PSI/KS), latence et Model Registry
 - Mettre en place **canary/A/B** avant promotion.
 - Automatiser le **retrain** avec garde-fous et rollback.

Rappel

Le pipeline doit être **reproductible, auditables et scalable**.

Résumé

Source → ETL → Train → Register → Deploy → Monitor → Retrain : boucle **MLOps** continue pour des modèles robustes en production.

Code exemple – mini pipeline entraînement + logging MLflow



Extrait – entraînement + suivi d'expériences avec MLflow

```
# Mini pipeline : entraînement + logging MLflow
import mlflow, mlflow.sklearn
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression

# 1) Données jouets
X, y = make_classification(n_samples=2000, n_features=10,
                           n_informative=5, random_state=42)
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# 2) Définir l'expérience
mlflow.set_experiment("encg_mlops_scoring")

# 3) Entrainer & logger
with mlflow.start_run():
    C = 1.0
    model = LogisticRegression(max_iter=1000, C=C, solver="lbfgs")
    model.fit(Xtr, ytr)
    proba = model.predict_proba(Xte)[:, 1]
    auc = roc_auc_score(yte, proba)

    mlflow.log_param("C", C)
    mlflow.log_metric("auc", float(auc))
    mlflow.sklearn.log_model(model, "model")

    print(f"AUC={auc:.3f}")
```

Astuce : lancez `mlflow ui` pour visualiser runs/paramètres/métriques.

Étapes clés

- Créer l'expérience (`set_experiment`).
- Démarrer un run (`start_run()`).
- Logger `params`, `metrics` et `model`.

Bonnes pratiques

- Versionner code/données (Git, DVC).
- Nommer clairement les runs & seeds.
- Utiliser un Model Registry.

Quiz interactif – Chapitre 11 (MLOps)



1

CI/CD logiciel vs CI/CD ML

Quelles différences clés (tests de données, dérive, modèles/artefacts) entre CI/CD logiciel et CI/CD pour le ML ?

2

Rôle d'un Feature Store

À quoi sert un feature store (cohérence train/serve, réutilisation, gouvernance) et citez un cas d'usage concret ?

3

Détection de drift

Citez 2 métriques de drift (ex. PSI, KS) et indiquez quand les utiliser en production.

4

Canary/A-B : avantages & risques

Quels bénéfices et risques d'un canary deployment ou A/B testing pour un modèle en prod ?

5

Reproductibilité : artefacts à versionner

Listez 3 artefacts à versionner (ex. données/features, code & dépendances, modèle & paramètres) pour assurer la reproductibilité.

Chapitre 12 – Projets pratiques : objectifs et organisation



- **Objectifs** : mettre en pratique IA/ML sur cas réels (finance & gestion), avec rigueur et impact business
- **Modalités** : binôme/équipe (3–4), durée 4–6 semaines, jalons hebdo et feedbacks encadrant
- **Livrables** : notebooks propres, features documentées, slides (10–12), démo (7–10 min), README
- **Outils** : Python, scikit-learn, TensorFlow/Transformers, MLflow, Git (gestion de versions)
- **Évaluation** : Technique 35%, Data/FE 20%, XAI/Risques 15%, Impact KPI 20%, Présentation 10%
- **Conformité & XAI** : métriques expliquées (SHAP/LIME), traçabilité, respect RGPD

Méthodologie CRISP-DM – 6 étapes



1 Business understanding

Clarifier objectifs métier, périmètre et KPI (succès mesurable).

Question: quel impact business attendu ?

2 Data understanding

Explorer sources, qualité, biais; définir dictionnaire des données.

Vérifier distributions, NA, fuites potentielles.

3 Data preparation

Nettoyer, transformer, sélectionner features; splits train/val/test.

Pipelines reproductibles (sklearn, feature store).

4 Modeling

Baselines → modèles avancés (GBM, DL); tuning; XAI.

Tracer paramètres & métriques (MLflow/W&B).

5 Evaluation

Valider vs KPI métier (AUC, F1, ROI), robustesse, équité, risques.

Revues & validation indépendante.

6 Deployment

Déployer (batch/temps réel), moniturer perf/drift, plan de retraining.

Traçabilité & gouvernance (model registry).

CRISP-DM = Cross-Industry Standard Process for Data Mining – cadre itératif reliant objectifs métier, données, modèles et déploiement.

Calendrier projet (S1 → S6) – 6 semaines



Projet 1 – Scoring crédit (brief & données)



Objectif & KPI

Probabilité de défaut (PD)

But du projet

- Construire un modèle qui estime la PD pour soutenir l'octroi et le pricing.

KPI de performance

ROC-AUC, KS

Recall@FPR (ex. 5%)

Brier score, calibration

Stabilité (drift)

Contraintes & conformité

- Classes déséquilibrées, seuils métier
- Explicabilité (RGPD, XAI: SHAP, motifs standardisés)

Données & Feature engineering

Sources & variables clés

Revenus, dettes, DTI

Durée du prêt, montant

Âge, statut/ancienneté emploi

Historique crédit, incidents

Features dérivées

- Ratios & agrégats (DTI, charges, durées normalisées)
- Encodage catégoriel (One-Hot / WoE), binning scorecards
- Gestion des NA, standardisation, détection outliers

Splits & validation

- Train/Validation/Test stratifiés; retenue temporelle si historique.

Projet 1 – Pipeline & starter code (Scoring crédit)



Pipeline (de bout en bout)

- EDA & cadrage business: distribution, déséquilibre de classes, KPI (ROC-AUC, KS, Recall@FPR).
- Prétraitement: **ColumnTransformer** (num=StandardScaler, cat=OneHotEncoder).
- Modélisation: baseline (Régression logistique) puis **XGBoost**.
- Calibration & seuil: Platt/Isotonic, choix du seuil vs FPR cible.
- XAI: **SHAP** (motifs de décision) + motifs standardisés RGPD.
- Export & déploiement: **Pipeline sklearn** (joblib) → Model Registry → API.
- Contraintes: données déséquilibrées, explicabilité, conformité (RGPD, auditabilité).

Starter code – sklearn + XGBoost

sklearn • xgboost

```
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.metrics import roc_auc_score
from xgboost import XGBClassifier

# Colonnes à adapter à votre dataset
num_cols = ["revenu", "dettes", "duree_credit"]
cat_cols = ["statut_emploi", "logement"]

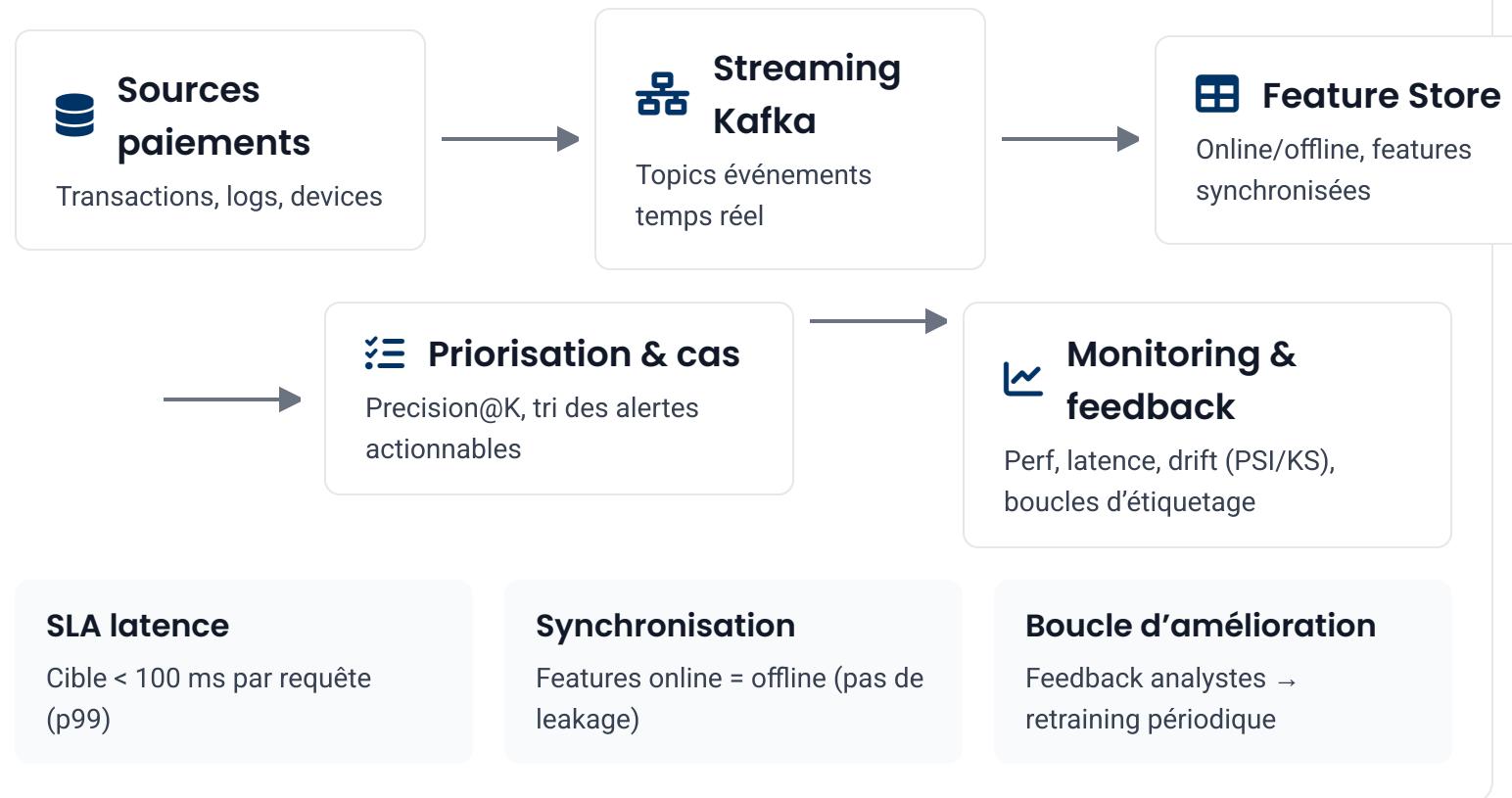
pre = ColumnTransformer([
    ("num", StandardScaler(), num_cols),
```

Étapes suivantes: calibration probabilités, choix du seuil (FPR cible), SHAP, export joblib.

Projet 2 – Détection de fraude temps réel (architecture)



Schéma de flux



Points clés

- **KPI alerte** : Precision@K, coût faux positifs, temps de traitement.
- **Latence** : p95/p99 sous seuils (ex. 50–100 ms). Modèle (IF/GBM) < 100 ms
- **Drift** : PSI/KS, dérive conceptuelle, ré-entraînement.

Stack de référence

Kafka • Feast/Redis • FastAPI/Flask • Prometheus/Grafana • MLflow

Résumé

Pipeline temps réel : **Kafka** → **Feature Store** → **Scoring API** → priorisation → **monitoring & feedback** pour améliorer en continu.

Projet 2 – Isolation Forest et Precision@K (Fraude)



Starter – Détection de fraude: IsolationForest + ranking Precision@K

```
# Hypothèses: X_train, X_test (numpy/pandas); y_test ∈ {0,1} où 1 = fraude
import numpy as np
from sklearn.ensemble import IsolationForest

# 1) Modèle anomalies (non supervisé) – robuste à l'extrême déséquilibre
iso = IsolationForest(n_estimators=200, contamination=0.01, random_state=42)
iso.fit(X_train)

# 2) Scores d'anomalie (plus grand = plus suspect)
scores = -iso.decision_function(X_test) # négation: valeurs élevées = anomalies

# 3) Fonction Precision@K (prioriser les meilleures alertes)
def precision_at_k(y_true, scores, k):
    idx = np.argsort(scores)[::-1][:k]
    return (np.array(y_true)[idx] == 1).mean()

# 4) Exemple: top 1% des transactions les plus risquées
K = max(1, int(0.01 * len(scores)))
p_at_k = precision_at_k(y_test, scores, K)
print(f"Precision@{K} = {p_at_k:.2%}")

# Option: récupérer le ranking pour l'envoi d'alertes
ranking_idx = np.argsort(scores)[::-1] # du plus suspect au moins suspect
```

Astuce: pour sklearn, decision_function plus grand = plus normal; on inverse le signe pour un ranking « plus grand = plus suspect ».

Pourquoi Precision@K ?

Rang priorisé des alertes quand le volume de fraude est très faible.

Optimise l'usage des équipes d'enquête (capacité limitée).

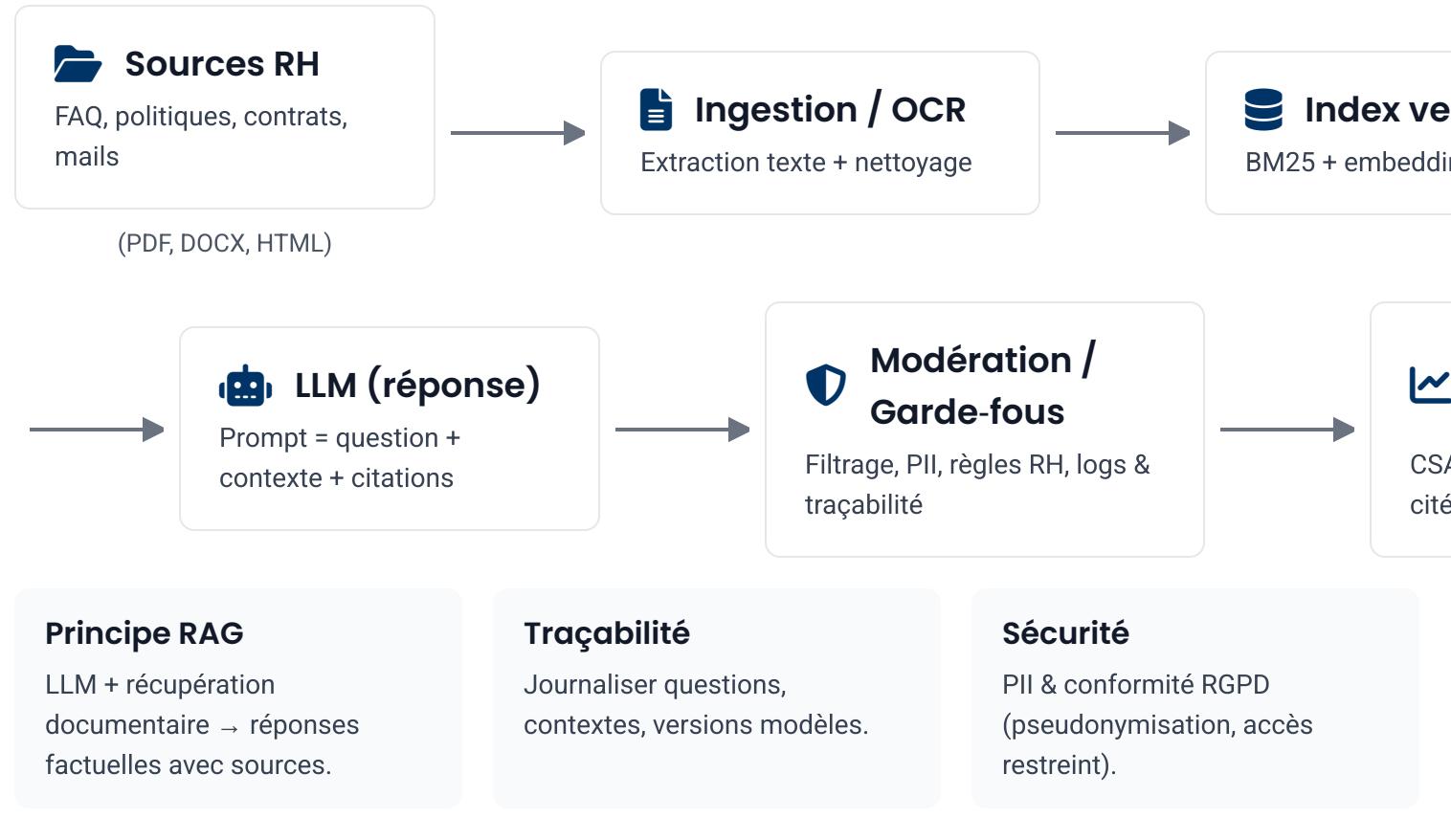
Bonnes pratiques

- Évaluer aussi Recall@K et le coût par alerte.
- Mettre à jour contamination et monitorer le drift.
- Boucle de feedback analyste → réapprentissage.

Projet 3 – Chatbot RH (RAG) : architecture cible



Schéma de flux (RAG)



Points clés (mise en œuvre)

- **Index hybride** : BM25 + embeddings (top-k fusionnés).
- **Prompt structuré** : rôle, contexte, citations, format réponse. Top-k passages pertinents
- **Garde-fous** : filtres PII, liste de refus, seuil confiance.
- **KPI** : CSAT, TTR, exactitude avec sources, taux escalade.

Retriever

Top-k passages pertinents

Dashboard qualité

Bonnes pratiques

Échantillons d'or RH, revue humaine, mises à jour index planifiées.

Projet 3 – Starter: Retrieval + LLM (RAG)



Concepts RAG (Retrieval-Augmented Generation)

- Embeddings sémantiques (p. ex. SBERT) pour projeter documents et requêtes dans le même espace vectoriel.
 - Top-k retrieval par similarité (cosine) pour récupérer les passages les plus pertinents.
 - Prompt structuré: *Question + Contexte + Consignes* (répondre en citant les sources).
 - Traçabilité: renvoyer extraits/URLs cités; logs pour qualité (CSAT, TTR).
 - Garde-fous: filtrage contenu sensible, longueur contexte, politiques sécurité.
- Cas RH: FAQ, politiques internes, processus d'onboarding (documents → base de connaissances).

Code starter – Retrieval + Embeddings

Sentence-Transformers

```
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity

# 1) Corpus interne (extraits de FAQ/politiques RH)
corpus = [
    "La période d'essai est de 3 mois renouvelable.",
    "Les congés payés s'acquièrent à raison de 2,5 jours par mois.",
    "La procédure d'onboarding inclut la remise du matériel et des accès."
]

# 2) Embeddings de documents et de la question
model = SentenceTransformer('all-MiniLM-L6-v2')
E_docs = model.encode(corpus, normalize_embeddings=True)
```

Étapes suivantes: index vectoriel persistant, citations formatées, évaluation qualité (exactitude, couverture), et logs.

Projet 4 – Prévision de la demande (brief & KPI)



Objectif & KPI

Objectif

- Prévoir la demande par produit/magasin à **horizons courts** (J+7) et **moyens** (J+30) pour piloter stocks et approvisionnements.

KPI d'évaluation

- MAPE / WAPE (erreur relative)
- Ruptures & surstocks (coût associé)
- Service level (taux de service)
- Backtest multi-périodes (rolling)

Contraintes & périmètre

- Saisonnalité, promotions, effets prix, jours fériés, campagnes marketing.
- Granularité: journalier hebdomadaire; agrégation par SKU/site si nécessaire.

Features & Modèles

Features (exemples)

- Historique ventes, lags (t-1, t-7, t-28)
- Promotions/prix, campagnes marketing
- Calendrier (jour semaine, fériés)
- Météo & événements locaux (si pertinent)

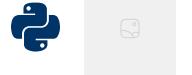
Modèles candidats

- Prophet / SARIMAX pour tendance + saisonnalités explicites.
- XGBoost/Random Forest avec lags & variables calendaires.
- LSTM (séquences) pour dépendances temporelles longues.

Bonnes pratiques

- Validation temporelle (walk-forward) & backtest glissant.
- Prévoir par segment (SKU/site) puis agréger si besoin.
- Relier les KPI prévision aux décisions stocks (coûts).

Projet 4 – Starter: Prophet (prévision séries temporelles)



Extrait de code – Prévision avec Prophet (série ds/y)

```
# Installer si besoin : pip install prophet
import pandas as pd
from prophet import Prophet

# 1) Charger des données avec colonnes 'ds' (date) et 'y' (valeur)
df = pd.read_csv('demand.csv') # ex.: ds=YYYY-MM-DD, y=volume

# 2) Créer et entraîner le modèle (saisonalité multiplicative)
m = Prophet(seasonality_mode='multiplicative')
m.fit(df)

# 3) Générer l'horizon de prévision (p. ex. 30 jours)
fut = m.make_future_dataframe(periods=30)
fc = m.predict(fut)

# 4) Inspecter les dernières prévisions (intervalle de confiance)
print(fc[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail())
```

Astuce: assurez-vous que df['ds'] est au format date (pd.to_datetime) et que df['y'] est numérique.

Pré-requis données

Colonnes: ds (datetime), y (valeur).

Fréquence: homogène (jour/semaine/mois).

Nettoyage: NA, ruptures, valeurs extrêmes.

Bonnes pratiques

- Séparer train/test, évaluer MAPE/WAPE.
- Ajouter jours fériés, promotions si disponible.
- Tracer prévision pour détection d'écart.

Projet 5 – Sentiment marchés (objectif & pipeline)



Objectif & KPI

Finance de marché

Objectif

- Produire un score de sentiment (-1...+1) par titre/secteur/indice à partir des actualités.
- Transformer ces scores en signaux exploitables (achat/vente/neutre).

KPI à suivre

Modèle F1 / PR-AUC (données déséquilibrées)

Marché Corrélation sentiment → rendements; couverture des titres

Backtest Sharpe / Drawdown des signaux; Precision@K alertes

Robustesse Stabilité temporelle; dérive lexicale (drift)

Pipeline (NLP → signaux de marché)

Collecte & préparation

Flux news/tweets, déduplication, langue (FR/EN), nettoyage.

Scoring de sentiment

Modèle FinBERT (Transformers) → score -1...+1 par texte.

Agrégation temporelle

Regroupement par titre/secteur, fenêtre glissante, pondération par source.

Génération de signaux

Seuils/quantiles → achat/vente; combinaison avec signaux prix.

Backtest & monitoring

Backtesting (Sharpe/Drawdown); suivi drift lexical & recalibrage.

Astuce: débuter par un lexique sentiment baseline puis comparer à FinBERT; consigner les runs (MLflow) et les corrélations avec les rendements.



Extrait de code – Pipeline Transformers (FinBERT)

```
# Inférence de sentiment financier avec FinBERT
from transformers import pipeline

model_id = "yiyanghkust/finbert-tone"
clf = pipeline("text-classification", model=model_id)

texts = [
    "Q2 revenue grew 18% YoY while margins expanded.",
    "Guidance cut due to weaker demand.",
    "Management expects stable outlook for next quarter."
]

preds = clf(texts)
for t, p in zip(texts, preds):
    print(f"{p['label']}: {p['score']:.2f} - {t}") # POSITIVE / NEGATIVE / NEUTRAL

# Astuce : pour accélérer, passer device=0 (GPU) si disponible.
```

Astuce: Agréger les scores par titre/secteur puis corréler aux rendements (backtest) pour évaluer la valeur ajoutée.

Lecture des résultats

Labels : POSITIVE, NEGATIVE, NEUTRAL avec score de confiance.

Nettoyer le texte (langue, encodage) pour limiter le bruit.

Évaluer sur données labellisées (F1 / PR-AUC).

Utiliser batch_size et cache pour des volumes élevés.

Tracer et expliquer (XAI) selon les exigences conformité.

Bonnes pratiques & livrables / évaluation



Bonnes pratiques (projets IA)

Conseils

- Commencer par une **baseline** simple, puis itérer
- Traçabilité** des expériences (params, métriques, artefacts)
- XAI & éthique** (SHAP, biais, RGPD)
- Storytelling business** (problème → solution → impact)
- Versionner** données/modèles (Git, MLflow)
- Séparer **train/val/test**, éviter la fuite de données
- Coder proprement (notebooks + scripts, **PEP8**)
- Suivre des **KPI** métiers pertinents (AUC/PR-AUC, ROI)

Livrables & critères d'évaluation

Barème

Livrables attendus

- Code propre (notebooks + scripts)
- Slides 10–12

Rapport 8–12 pages

Démo 7–10 min

Critères d'évaluation (poids)

- | | |
|-----|---|
| 35% | Qualité technique du modèle (méthode, validation) |
| 20% | Données & feature engineering |
| 15% | Explicabilité & risques (biais, RGPD) |
| 20% | Impact business (KPI, ROI, recommandations) |
| 10% | Présentation & storytelling |

Noter la **reproductibilité** (env., seeds), la clarté des hypothèses et la qualité des visualisations.

Ressources & datasets publics – Liens utiles



Finance

Crédit, fraude, prêts, marchés.

[German Credit \(Kaggle\)](#)

[Credit Card Fraud \(Kaggle\)](#)

[Lending Club Loans \(Kaggle\)](#)



Marketing / CRM

Segmentation, churn, campagnes.

[Bank Marketing \(UCI\)](#)

[Online Retail \(UCI\)](#)

[Telco Churn \(Kaggle\)](#)



Ressources Humaines (RH)

Attrition, matching profils.

[IBM HR Attrition \(Kaggle\)](#)

[HR Job Change DS \(Kaggle\)](#)



NLP

Avis, actualités, entités.

[Amazon Fine Food Reviews \(Kaggle\)](#)

[Financial PhraseBank \(Kaggle\)](#)

[Hugging Face Datasets](#)

Conclusion – Récapitulatif du parcours (12 chapitres)



- 1 Introduction à l'IA
- 2 Fondements mathématiques
- 3 Apprentissage supervisé
- 4 Apprentissage non supervisé
- 5 Réseaux de neurones & Deep Learning
- 6 Traitement du langage (NLP)
- 7 IA explicable (XAI)
- 8 IA en Finance
- 9 IA en Gestion
- 10 Éthique & Réglementation
- 11 MLOps & Déploiement
- 12 Projets pratiques

Compétences acquises – Ce que vous savez faire



- Maîtriser les principaux algorithmes ML/DL et leurs métriques
- Concevoir des pipelines de données et de modélisation en Python
- Appliquer l'IA à la finance et à la gestion (scoring, fraude, séries; CRM, supply, pricing)
- Interpréter les modèles (XAI) et intégrer les contraintes RGPD/AI Act
- Préparer un déploiement (API, Docker, monitoring du drift)
- Conduire un projet IA de bout en bout (CRISP-DM, livrables, ROI)

Perspectives IA 2026 – 4 tendances clés



IA générative multimodale

Texte-image-audio-vidéo pour assistants et production de contenus fiables.

Multimodal

RAG



Agents métiers

Copilotes autonomes orchestrant des workflows et tâches répétitives.

Automatisation

ROI



XAI + AI Act

Explicabilité et conformité deviennent des exigences de déploiement.

XAI

AI Act



Green AI

Efficience énergétique, optimisation des coûts cloud, sobriété des modèles.

CO₂

Efficiency

Carrières & débouchés – 3 profils IA



Data / ML Analyst

Analyses, tableaux de bord et baselines ML pour suivre les KPI.

Compétences & missions :

- SQL, Python/pandas, BI (Power BI/Tableau)
- Statistiques, A/B testing, suivi des KPI



Data Scientist / ML Engineer

Construction de modèles et industrialisation (MLOps).

Compétences & missions :

- scikit-learn, TensorFlow/PyTorch, feature engineering
- Docker, APIs, CI/CD, monitoring (MLflow)



Product / Project Manager IA

Cadrage, priorisation et pilotage de la valeur (ROI).

Compétences & missions :

- Business & data literacy, gestion produit, ROI
- XAI, RGPD/AI Act, gouvernance & risques

Ressources d'apprentissage – 4 catégories



Livres

Géron – Hands-On ML • Goodfellow – Deep Learning •
Russell & Norvig – AI: A Modern Approach

Théorie

Pratique



MOOCs

Coursera (Andrew Ng) • fast.ai • DeepLearning.AI • edX

Auto-formation

Certificats



Plateformes

Kaggle • Papers With Code • Hugging Face • UCI Datasets

Datasets

Modèles/pré-entraînés



Blogs / News

Distill • Google AI Blog • OpenAI • Towards Data Science

Veille

Tendances

Communautés & événements – Panorama



Communautés

Apprendre, partager, trouver des projets et du feedback.

Kaggle

Hugging Face

r/MachineLearning



Événements

Conférences et meetups pour la veille et le réseau.

NeurIPS • ICML

AI4Finance

Meetups locaux



GitHub

Contribuer à l'open-source et bâtir un portfolio.

Repos ML/DL

Awesome lists

Notebooks



Réseaux pros

Echanger avec des praticiens et recruteurs.

LinkedIn (IA/Finance)

Slack/Discord data

Groupes locaux

Certifications professionnelles recommandées



Google Professional ML Engineer

Concevoir, déployer et moniterer des modèles sur GCP (MLOps, XAI, data pipelines).

Google Cloud

MLOps • XAI



AWS Certified ML – Specialty

Data engineering, entraînement & déploiement sur AWS (SageMaker, sécurité, coûts).

AWS

Data • Déploiement



Microsoft Azure AI Engineer

Cognitive Services, Responsible AI, MLOps sur Azure (pipelines, sécurité, gouvernance).

Azure

Responsible AI



TensorFlow Developer Certificate

Deep learning appliquéd (CNN, NLP, séries) avec TensorFlow/Keras et bonnes pratiques.

TensorFlow

DL • Keras

Outils & environnements de développement



Outils (librairies & pratiques)

Productivité

Python scientifique

numpy, pandas, scikit-learn

Deep Learning & NLP

TensorFlow / PyTorch, transformers, spaCy

Expérimentation & suivi

MLflow, Weights & Biases (params, métriques, artefacts)

Éditeurs & notebooks

VS Code, Jupyter Notebook/Lab

Contrôle de version

Git • GitHub/GitLab (branches, PR, issues)

Environnements

Gestion des dépendances

- Conda / venv
requirements.txt, reproductibilité

Notebooks cloud

- Colab / JupyterHub
GPU/TPU selon besoins

Orchestration & serving

- Kubernetes • API REST
scalabilité, latence maîtrisée

Conteneurs

- Docker
images, Dockerfile, isolation

Cloud & CI/CD

- GCP • AWS • Azure
stockage, compute, MLOps
- GitHub Actions / GitLab CI
tests, build, déploiement

Bonnes pratiques

- Seeds • README •
requirements
traçabilité & reproductibilité

Roadmap d'apprentissage post-cours (5 étapes)



Objectif

Structurer votre progression après le cours pour gagner en profondeur technique, en pratique MLOps et en impact métier.

1 Consolider

Réviser statistiques/ML et refaire les TP clés.

Renforcer Python (numpy, pandas, scikit-learn).

2 Approfondir

Étudier Transformers, séries temporelles, optimisation.

Lire des articles et répliquer des notebooks.

3 Industrialiser

Monter des environnements: Docker, CI/CD, monitoring.

Pratiquer MLflow, Feature Store, bases Kubernetes.

4 Spécialiser

Choisir un domaine (Finance, NLP, Supply, Marketing).

Construire 2 projets orientés métier avec XAI/KPI.

5 Valoriser

Créer un portfolio GitHub (code, README, démos).

Contribuer (Kaggle, open-source) et réseau professionnel.

Conseil – **1 heure par jour** pendant 12 semaines vaut mieux que des sessions irrégulières : régularité, mesure des progrès, feedback.

Conseils pour vos premiers projets IA



- **Commencer simple (baseline)**

Établir une baseline claire (logistique/mean) avant de complexifier.

- **Documenter les choix**

Tracer données, features, métriques, limites et risques (biais, drift).

- **Mesurer l'impact business**

Relier les métriques ML (AUC, F1) aux KPI métiers (ROI, coût du risque).

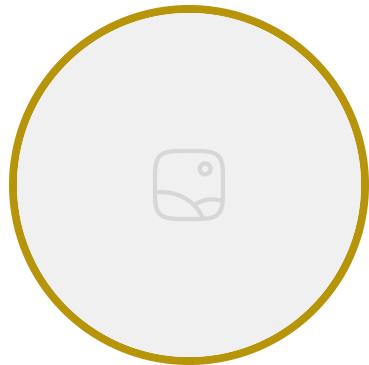
- **Assurer la reproductibilité**

Utiliser Git, requirements.txt/conda, seeds; consigner versions modèles/data.

- **Soigner la présentation**

Storytelling clair: problème → méthode → résultats → recommandations.

Remerciements & contacts



[Nom Prénom]

[Prof./Maître de conf.] – ENCG Settat

Spécialités

- IA appliquée à la finance
- NLP (traitement du langage)
- IA explicable (XAI)
- MLOps & déploiement



Remerciements

Merci pour votre engagement tout au long du cours. Continuez à expérimenter, mesurer, expliquer et déployer avec exigence et éthique.



Permanences & bureau

Créneaux [\[créneaux ici\]](#)

Bureau [\[N°/Bâtiment\]](#)



Contacts

[@ \[email@encg.ma\]](#)

 [\[Teams/Slack\]](#)



Sujets de mémoire / projet

- ✓ Scoring crédit & calibration (XAI)
- ✓ Pricing dynamique & optimisation

- ✓ Détection de fraude en temps réel
- ✓ RAG & chatbots (NLP) en entreprise



Merci & bon courage !

Devenez acteurs de la transformation par l'IA

Expérimitez • Mesurez • Expliquez • Déployez — avec éthique

Visez l'impact: un cas d'usage ciblé, des KPI clairs, une adoption responsable (XAI, RGPD, AI Act).