**Note méthodologique**

# Méthodologie d’entrainement du model

## Contexte

Avant toute phase de modélisation, il est indispensable de préparer et d’explorer les données afin de garantir la qualité et la fiabilité du jeu d’apprentissage.

Cette étape préliminaire vise à comprendre la structure des données, identifier les anomalies potentielles et assurer une base solide pour l’entraînement des modèles.

Les principales étapes sont les suivantes :

* Découverte des données : compréhension globale du jeu de données, de ses variables et de leur nature (numérique, catégorielle, textuelle, etc.).
* Traitement des valeurs manquantes : gestion raisonnée des observations incomplètes par imputation, suppression ou remplacement selon leur importance statistique.
* Gestion des valeurs aberrantes (outliers) : identification et traitement des valeurs extrêmes susceptibles de biaiser les résultats de modélisation.
* Fusion des jeux de données : intégration et harmonisation des différentes sources d’information pour constituer un ensemble cohérent et complet.

À l’issue de cette phase, un dataframe final de 307 511 lignes (clients) et 312 colonnes (variables) a été obtenu, prêt à être exploité dans le cadre du modèle prédictif.

La variable cible (Target) prend la valeur 0 lorsque le crédit est accordé et 1 lorsqu’il est refusé.

## Approche expérimentale

Le projet vise à **comprendre et anticiper le risque de défaut de crédit** chez certains clients. L’objectif est d’évaluer la probabilité qu’un client ne rembourse pas son prêt et, par conséquent, d’aider à la prise de décision quant à l’octroi du crédit.

La tâche peut être formulée comme une **problématique de classification binaire** :

* **0** → Crédit accordé
* **1** → Crédit refusé ou client en défaut de paiement

Pour établir une première estimation des performances possibles, une **régression logistique** a été utilisée comme modèle de référence (*baseline*).  
Cette approche simple permet de mesurer la performance initiale avant d’introduire des algorithmes plus complexes.

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

Les résultats initiaux révèlent un **déséquilibre marqué** entre les classes :

* Précision (Accuracy) pour la classe 0 : **0.92**
* Précision pour la classe 1 : **0.00**

Cette différence s’explique par la distribution du jeu de données : environ **92 %** des individus appartiennent à la classe 0 (crédit accordé) contre **8 %** à la classe 1 (défaut de paiement).  
Ce déséquilibre entraîne un apprentissage biaisé du modèle, qui tend à privilégier la classe majoritaire.

Afin d’améliorer la capacité du modèle à identifier les clients à risque, il est nécessaire de **corriger ce déséquilibre de classes** par des techniques de rééchantillonnage adaptées.

## Traitement du déséquilibre

Le jeu de données présente une **forte disparité entre les classes**, avec une majorité de clients solvables (classe 0) et une minorité de clients en défaut de paiement (classe 1).  
Ce déséquilibre fausse l’apprentissage du modèle, qui tend à prédire systématiquement la classe majoritaire, réduisant ainsi la capacité de détection des clients réellement à risque.

Pour remédier à ce problème, nous avons recours à une **technique de suréchantillonnage** : **SMOTE** (*Synthetic Minority Over-sampling Technique*).  
Deux variantes sont utilisées selon la nature des variables :

* **SMOTE** pour les variables numériques
* **SMOTENC** pour les variables catégorielles

**3.1. Principe de SMOTE**

SMOTE est une méthode de **suréchantillonnage synthétique** qui vise à créer de nouveaux exemples de la classe minoritaire (ici, les “mauvais payeurs”) au lieu de simplement dupliquer ceux existants.  
Elle génère des points artificiels en **interpolant** entre un individu minoritaire et ses *k* plus proches voisins appartenant à la même classe.

**Exemple illustratif :**  
Si deux clients en défaut se situent en [2,3] et [3,4], SMOTE peut générer un nouveau point intermédiaire tel que [2.6,3.5].  
Ainsi, la classe minoritaire devient plus dense et mieux représentée dans l’espace des variables.

**3.2. Avantages de SMOTE**

* Évite le simple clonage de données → limite le **surapprentissage**.
* Introduit davantage de **variabilité** dans la classe minoritaire.
* Améliore généralement le **rappel (recall)** pour les classes rares.

**3.3. Limites de SMOTE**

* L’**interpolation linéaire** peut générer des points peu réalistes, notamment pour les variables catégorielles.
* Risque d’**overlap** (chevauchement) entre classes si les frontières de décision sont complexes, ce qui peut accroître le taux de faux positifs.
* Ne prend pas en compte le **coût métier** : l’équilibrage statistique n’intègre pas la différence d’impact entre un faux négatif et un faux positif.

**3.4. Variantes de SMOTE**

Plusieurs variantes existent pour améliorer la pertinence de la génération de données synthétiques :

* **Borderline-SMOTE** : crée des échantillons uniquement à proximité des frontières de décision, là où la classification est la plus incertaine.
* **SMOTEENN / SMOTETomek** : combinent suréchantillonnage et nettoyage des données ambiguës (via *Edited Nearest Neighbors* ou *Tomek Links*).
* **ADASYN** (*Adaptive Synthetic Sampling*) : génère davantage d’échantillons dans les zones où la classe minoritaire est particulièrement sous-représentée.

## Test de différents modele

Une fois les données préparées et le déséquilibre traité, plusieurs **modèles de classification supervisée** ont été testés afin d’identifier celui offrant le meilleur compromis entre **performance, robustesse et rapidité d’exécution**.

Les algorithmes retenus sont :

* **XGBoost**
* **LightGBM**
* **CatBoost**

Ces trois modèles reposent sur le même principe fondamental : le **Gradient Boosting**, une approche d’ensemble (*ensemble learning*) qui consiste à construire une succession d’arbres de décision.  
Chaque arbre apprend à corriger les erreurs du précédent, et l’ensemble final produit une prédiction plus précise et plus stable.

**4.1. XGBoost (Extreme Gradient Boosting)**

XGBoost est la version la plus connue et la plus classique du Gradient Boosting.  
Il génère une série d’arbres de décision successifs tout en intégrant une **régularisation** pour éviter le surapprentissage (*overfitting*).  
Très performant sur les données tabulaires, il est largement utilisé dans les compétitions de data science (comme Kaggle).

**À retenir :**

* Modèle fiable, robuste et stable.
* Performant sur des données tabulaires.
* Peut toutefois être plus lent sur de très grands jeux de données.

**4.2. LightGBM (Light Gradient Boosting Machine)**

Développé par Microsoft, **LightGBM** est une version optimisée et plus rapide de XGBoost.  
Il se distingue par sa manière de construire les arbres “**feuille par feuille**” plutôt que “niveau par niveau”, ce qui accélère considérablement le processus tout en réduisant la consommation mémoire.  
Il gère également très bien les jeux de données volumineux et **déséquilibrés** grâce à la pondération automatique des classes.

**À retenir :**

* Le plus rapide des trois modèles.
* Idéal pour de grands volumes de données.
* Offre de bonnes performances même sur des échantillons déséquilibrés.

**4.3. CatBoost (Categorical Boosting)**

Créé par Yandex, **CatBoost** est spécialement conçu pour gérer efficacement les **variables catégorielles** sans nécessiter de transformation manuelle (comme le *one-hot encoding*).  
Il utilise une technique appelée **Ordered Boosting**, qui réduit les risques de surapprentissage tout en assurant une meilleure généralisation du modèle.  
CatBoost offre souvent de bons résultats dès les premières itérations, sans nécessiter un réglage complexe des paramètres.

**À retenir :**

* Excellente gestion des variables catégorielles.
* Simple à utiliser et robuste.
* Donne souvent de bons résultats “out-of-the-box”.

## Optimisation des hyperparamètres

L’**optimisation des hyperparamètres** constitue une étape essentielle pour affiner et maximiser les performances des modèles de machine learning.  
Un bon choix de paramètres permet d’améliorer significativement la précision, la stabilité et la capacité de généralisation du modèle.

Dans le cadre de cette classification binaire, la **métrique d’évaluation principale** utilisée est l’**AUC (Area Under the ROC Curve)**, particulièrement adaptée aux problèmes de classes déséquilibrées.

**5.1. Méthodes d’optimisation**

Plusieurs approches sont envisageables pour rechercher la meilleure combinaison d’hyperparamètres :

* **Grid Search** : exploration exhaustive de toutes les combinaisons possibles d’un ensemble de paramètres prédéfinis.  
  → Avantage : simple à implémenter.  
  → Inconvénient : coûteux en temps de calcul lorsque l’espace des paramètres est large.
* **Random Search** : sélection aléatoire d’un sous-ensemble de combinaisons.  
  → Avantage : plus rapide et souvent aussi efficace que le Grid Search pour identifier de bonnes zones de performance.  
  → Inconvénient : reste limité pour des espaces de grande dimension.
* **Hyperopt** : méthode d’optimisation bayésienne plus avancée, utilisée dans ce projet.  
  Elle explore intelligemment l’espace des hyperparamètres à l’aide d’un modèle probabiliste pour **accélérer la convergence vers les valeurs optimales**.  
  Contrairement à une recherche exhaustive, Hyperopt permet d’obtenir de très bons résultats tout en réduisant le temps de calcul.

**5.2. Intégration d’Hyperopt**

Hyperopt a été privilégié car il s’adapte aussi bien aux **modèles mono-machine** (comme ceux de *scikit-learn* ou *TensorFlow*) qu’aux **architectures distribuées** telles que *Apache Spark MLlib* ou *Horovod*.  
Cette flexibilité permet de tester efficacement un grand nombre de configurations sur un espace paramétrique étendu.

Grâce à cette approche, il devient possible d’analyser finement les modèles de Gradient Boosting (XGBoost, LightGBM, CatBoost) tout en optimisant les hyperparamètres qui influencent le plus leurs performances, tels que :

* le taux d’apprentissage (*learning rate*),
* le nombre d’arbres (*n\_estimators*),
* la profondeur maximale des arbres (*max\_depth*),
* le taux de sous-échantillonnage (*subsample*),
* et les paramètres de régularisation (*lambda*, *alpha*).

## Réduction du nombre de feature et pipeline

Pour améliorer la performance, la robustesse et la rapidité d’entraînement du modèle, une **réduction du nombre de variables (features)** a été effectuée.  
Cette démarche vise à éliminer les variables redondantes ou peu informatives afin de :

* **réduire le bruit** présent dans les données,
* **limiter la complexité du modèle**,
* **prévenir le surapprentissage** (*overfitting*),
* et **accélérer les temps de calcul**.

**6.1. Sélection de variables**

Une première approche a consisté à utiliser la méthode **RFECV (Recursive Feature Elimination with Cross-Validation)**, qui permet de sélectionner automatiquement les variables les plus pertinentes.  
Cette méthode évalue l’impact de chaque feature sur les performances du modèle à travers plusieurs itérations, supprimant progressivement celles qui apportent le moins de valeur prédictive.

En complément, une analyse de l’**importance des variables (Feature Importance)** a été réalisée à partir des modèles d’arbres (XGBoost, LightGBM, CatBoost).  
Cette double approche a permis d’identifier les attributs les plus influents dans la prédiction du risque de crédit.

**6.2. Mise en place d’un pipeline automatisé**

Pour garantir la reproductibilité et l’efficacité du processus, un **pipeline complet** a été mis en place, automatisant l’ensemble des étapes suivantes :

1. **Prétraitement des données** : nettoyage, encodage, normalisation et traitement des valeurs manquantes.
2. **Rééquilibrage des classes** : application de SMOTE / SMOTENC pour corriger le déséquilibre.
3. **Sélection des features** : filtrage des variables pertinentes via Feature Importance.
4. **Entraînement du modèle** : apprentissage du classifieur choisi (LightGBM ).
5. **Évaluation des performances** : calcul des métriques (Accuracy, Recall, F1-score, AUC).

# Synthèse des résultats

Test des modeles avant SMOTE et réduction du nombre de feature :

Une image contenant texte, capture d’écran, Police, ligne

Le contenu généré par l’IA peut être incorrect.

Tous les modèles atteignent une Accuracy élevée (~0.92), ce qui peut sembler bon, mais c’est trompeur : le jeu est très déséquilibré (92 % de clients fiables), donc prédire “0” tout le temps donnerait déjà ~0.92.

Les valeurs de Recall (entre 0.02 et 0.06) et F1-score (très faibles) montrent que les modèles détectent très mal les “mauvais payeurs” (classe minoritaire). Cela confirme que le modèle apprend surtout la classe majoritaire.

Côté performances :

* CatBoost a le meilleur AUC (0.76) mais il est très lent (plus de 100 secondes).
* XGBoost est le plus équilibré entre vitesse et score (AUC = 0.755, F1 = 0.10).
* LightGBM est rapide, mais son Recall très bas (0.02) indique qu’il ne détecte presque aucun cas positif.

👉 Conclusion : sans rééquilibrage, les modèles sont biaisés vers la classe majoritaire et donc peu utiles pour prédire les défauts de paiement.

Test avec SMOTE :

Une image contenant texte, capture d’écran, Police, ligne

Le contenu généré par l’IA peut être incorrect.

L’AUC augmente légèrement pour tous les modèles (≈ +0.02) → preuve que le SMOTE a amélioré la capacité de distinction entre bons et mauvais clients.

Le Recall reste faible, mais un peu meilleur pour XGBoost (0.062 vs 0.056 avant SMOTE). Le F1-score s’améliore globalement, ce qui montre une meilleure détection des clients à risque, même si le compromis précision/rappel reste limité.

Côté comparaison :

* LightGBM obtient le meilleur AUC (0.776) et reste rapide → bon compromis global.
* XGBoost a le meilleur F1-score (0.11) → il détecte un peu mieux la classe minoritaire.
* CatBoost reste performant, mais beaucoup plus lent sans gain significatif.

**Conclusion :**

Le SMOTE améliore légèrement les performances globales, surtout sur le Recall et le F1-score.

Le modèle LightGBM reste le meilleur compromis entre performance et rapidité, tandis que XGBoost est un peu meilleur pour identifier les défauts, au prix d’un peu moins d’AUC.

# Limites et améliorations possibles

Comme tout modèle de machine learning, cette approche présente un certain nombre de **limites** qu’il convient de prendre en compte pour garantir une interprétation correcte des résultats et une utilisation efficace du modèle en production.

**9.1. Limites identifiées**

1. **Qualité et nature des données**  
   Le jeu de données utilisé (issu de Kaggle) est déjà fortement **prétraité** et comporte de nombreuses **features dérivées**.  
   Cela peut limiter la capacité du modèle à **généraliser** sur de nouvelles données issues d’environnements réels, potentiellement plus bruitées ou moins homogènes.
2. **Déséquilibre structurel du jeu de données**  
   Malgré les techniques de suréchantillonnage appliquées (SMOTE, SMOTENC), le déséquilibre entre classes demeure un **facteur de fragilité**.  
   En particulier, une légère variation dans la distribution peut affecter significativement les performances du modèle, notamment sur la classe minoritaire (clients en défaut).
3. **Métrique d’évaluation**  
   L’AUC utilisée pour l’évaluation ne reflète pas toujours parfaitement le **coût métier** réel.  
   Par exemple, un **faux négatif** (client risqué prédit comme solvable) est souvent **bien plus coûteux** qu’un **faux positif** (client solvable prédit comme risqué).  
   Le modèle optimise donc une mesure statistique sans nécessairement correspondre à la logique économique sous-jacente.

**9.2. Pistes d’amélioration**

Plusieurs axes d’amélioration peuvent être envisagés pour renforcer la pertinence et la performance du modèle :

* **Explorer d’autres techniques de rééquilibrage** : par exemple, *Borderline-SMOTE*, *SMOTEENN* ou *ADASYN* pour améliorer la qualité du suréchantillonnage.
* **Intégrer une fonction de coût métier** : pondérer davantage les erreurs selon leur impact économique afin d’adapter le modèle à la réalité opérationnelle.
* **Créer de nouvelles features dérivées** : notamment à partir de ratios financiers, de l’historique des paiements ou de la durée moyenne des crédits.
* **Tester des modèles récents et hybrides** : tels que *TabNet*, *Neural GBM* ou des combinaisons entre *CatBoost* et régression logistique.
* **Affiner le tuning des hyperparamètres** : augmenter le nombre d’itérations et élargir les plages de recherche pour obtenir une optimisation plus fine.

# Data drift

Le **Data Drift** désigne une **évolution des données dans le temps**, autrement dit une modification de la distribution statistique des variables entre la phase d’entraînement et la phase d’exploitation du modèle.  
Ce phénomène peut entraîner une **baisse significative des performances**, car les relations apprises lors de l’entraînement ne reflètent plus fidèlement la réalité des nouvelles données.

**8.1. Détection du drift**

Une première analyse du drift peut être réalisée à l’aide de la bibliothèque **Evidently**, qui permet de **comparer les distributions** des variables entre les jeux d’entraînement et de production.  
Cette comparaison s’appuie sur différentes mesures de distance selon la nature des variables :

* **Distance de Wasserstein** → utilisée pour les variables **numériques**.
* **Distance de Jensen-Shannon** → utilisée pour les variables **catégorielles**.

Ces indicateurs quantifient l’écart entre les distributions d’origine et celles observées en production, et permettent d’identifier les variables les plus sujettes à dérive.

**8.2. Suivi et interprétation du drift**

Le suivi du drift peut être automatisé à l’aide d’un **tableau de bord de monitoring**, qui alerte dès qu’une dérive significative est détectée (par exemple lorsque **plus de 30 % des variables** présentent un changement notable de distribution).  
Ce suivi continu permet d’intervenir rapidement pour recalibrer ou réentraîner le modèle en cas de dégradation des performances.

Il est également recommandé de **pondérer la détection du drift par l’importance des variables** (*feature importance*).  
En d’autres termes, il est plus pertinent de surveiller attentivement les dérives affectant les variables les plus influentes dans les prédictions du modèle, plutôt que celles ayant un impact marginal.

**En résumé :**  
Le Data Drift n’est pas un simple phénomène statistique ; il constitue un **signal d’alerte** sur la fiabilité du modèle en production.  
Une surveillance régulière et une stratégie proactive d’adaptation sont indispensables pour maintenir un niveau de performance constant dans le temps.

**En conclusion**, cette étude a permis de construire une méthodologie complète et reproductible pour la **modélisation du risque de crédit**, intégrant la préparation des données, le traitement du déséquilibre, la sélection des variables, l’optimisation des hyperparamètres et l’évaluation des performances.  
Bien que les résultats soient prometteurs, des ajustements supplémentaires — notamment liés à la prise en compte du coût métier et au suivi du drift — permettraient d’envisager une application en production plus robuste et durable.