Note Méthodologique

Le projet 7 vise à implémenter un modèle de scoring pour l'entreprise Prêt à dépenser. Cet outil permettra de calculer la probabilité qu'un client rembourse son crédit et de classer la demande en "crédit accordé" ou "crédit refusé".

La note technique, livrable du projet, a pour but de présenter en détail la démarche d'élaboration du modèle.

Elle décrira les étapes de construction, les données et les méthodes utilisées, les résultats obtenus et les perspectives d'amélioration.

L'objectif est de fournir un outil performant et fiable pour l'entreprise, basé sur l'analyse de données historiques de crédit et permettant d'identifier les clients à risque.

# Méthododologie d’entrainement du modèle :

## Traitement des données

Les données mises à disposition sont issues d’une base de données Kaggle de la compétition Home Credit Default Risk disponible [ici](https://www.kaggle.com/competitions/home-credit-default-risk/data).

Elle comporte 10 fichiers, soit 346 colonnes. Chacune des tables est reliée par une clé comme indiqué ci-dessous :



Afin de faciliter l’analyse exploratoire, la préparation des données et le feature engineering nécessaires à l’élaboration du modèle de scoring, nous nous basons sur ce kernel Kaggle.

À la suite de ce traitement, nous avons 307 507 observations et 797 variables pour le dataset d’entrainement et 48 744 observations et 796 variables pour le dataset de test (qui ne comporte pas de colonne « TARGET »).

Nous traitons désormais les données manquantes qui sont présente à environ 25% dans nos dataset. Pour commencer, nous supprimons les variables ayant plus de 50% de données manquantes. Ensuite, nous traitons les valeurs aberrantes avant de procéder à l’imputation.

Nous appliquons différentes méthodes d’imputation simples (0, médiane et mode).

Nous avons appliqué un preprocessing à nos données à savoir un one hot encoder pour les variables catégorielles et un Standard scaler pour les variables quantitatives.

Le dataset de test ne contient pas de colonne « TARGET » et sera utilisé lors du déploiement de notre modèle. Pour la modélisation, nous n’utilisons que le dataset d’entrainement qui sera séparé en plusieurs parties afin de disposer d’un jeu d’entrainement (246 005 observations), d’un jeu de validation (49 201 observations) et d’un jeu de test (12 301 observations) pour entraîner et tester nos différents modèles.

* 1. Modélisation :

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithme | Fonctionnement | Avantages | Inconvénients |
| Dummy Classifier | Prédit la classe la plus fréquente | Simple à implémenter | Ne prend pas en compte les caractéristiques |
| Logistic Regression | Modélise la probabilité d'une classe | Interprétable | Sensible aux outliers |
| SVC | Trouve un hyperplan séparant les classes | Robuste aux outliers | Peut-être complexe pour des données non linéaires |
| Decision Tree | Crée un arbre de décision pour prédire la classe | Facile à comprendre | Sujet au surajustement |
| Random Forest | Combine plusieurs arbres de décision pour améliorer la précision | Réduit le surajustement | Plus complexe à interpréter |
| XG Boost | Utilise un ensemble d'arbres de décision pour minimiser une fonction de perte | Offre de bonnes performances | Plus complexe à implémenter |
| Light GBM | Gradient boosting avec échantillonnage basé sur le gradient | Rapide et performant | Nécessite un réglage fin des hyperparamètres |

Par la suite, chacun de ces algorithmes sera testé avec recherche d’hyperparamètres et cross validation (via une GridSearchCV).

Un pipeline a également été créée afin de rassembler les étapes de traitement du déséquilibre des données, preprocessing et entraînement du modèle.

# Traitement du déséquilibre des classes

Le problème est un problème de classification binaire avec une classe sous représentée (8.07 % de clients en défaut contre 91.93 % de clients sans défaut).

A screenshot of a graph

Description automatically generated

Ce déséquilibre des classes doit être pris en compte dans l’entraînement des modèles puisqu’un modèle « naif » prédisant systématiquement que les clients sont sans défaut aurait une accuracy de 0.92 et pourrait être considéré à tort comme un modèle performant alors qu’il ne permettrait pas de détecter les clients à risque.

Parmi les différentes méthodes testées, nous optons finalement pour la méthode d'undersampling avec RandomUnderSampler pour le rééquilibrage de nos données. En effet les méthodes class\_weight et RandomUnderSampler offrent les meilleurs résultats de score métier. De plus, le temps d'exécution de RandomUnderSampler est bien inférieur à ceux des autres méthodes.

# La fonction coût métier, l'algorithme d'optimisation et la métrique d'évaluation

## Fonction coût métier

Une fonction de coût métier sur mesure a été créée pour la société Prêt à dépenser. Nous créons donc un nouveau score métier qui a pour objectif de pénaliser plus sévèrement les Faux Négatifs qui sont donc des clients ayant fait défaut mais qui ont été prédit comme ne faisant pas défaut. En effet, ce type d’erreur engendre une perte en capital pour l’entreprise.

A screenshot of a computer

Description automatically generated

Notre metric à maximiser ici sera de la forme suivante dans notre code :



## Algorithme d’optimisation

L’algorithme jugé le plus performant est Light GBM.

L’optimisation a consisté à tester différents hyperparamètres afin d’améliorer ses performances.

Les hyperparamètres testés sont les suivants :

* n\_estimators : nombre d’arbres à ajuster -> valeurs : [100, 300, 500, 800]
* max\_depth : définit une limite sur la profondeur de l'arbre -> valeurs : [-1, 2, 5, 7]
* num\_leaves : spécifie le nombre de feuilles dans un arbre -> valeurs : [7, 15, 31, 63, 127]
* learning\_rate : ajuste le taux d’apprentissage du modèle -> valeurs : [0.05, 0.1, 0.2, 0.4]

De plus, différents seuils de probabilité de défaut ont été testés afin de maximiser notre score métier.

Notre modèle le plus performant est donc Light GBM avec les paramètres {'learning\_rate':0.05, 'max\_depth': -1, 'n\_estimators': 500, 'num\_leaves': 31}

Après optimisation des hyperparamètres de Light GBM nous obtenons les résultats suivants pour le jeu de validation et le jeu de test.

A screenshot of a computer

Description automatically generated

## Interprétabilité du modèle - SHAP

Commençons par une analyse d'importance globale, en utilisant **summary\_plot** pour visualiser les features les plus importantes pour un modèle basé sur un ensemble de validation. Ce type de tracé agrège les valeurs SHAP pour toutes les features et tous les échantillons de l'ensemble sélectionné, puis les trie de sorte que la première affichée soit la feature la plus importante.

A screen shot of a graph

Description automatically generated

# Les limites et les améliorations possibles

**Métrique d’évaluation :** La modélisation actuelle s'appuie sur une métrique d'évaluation propre au métier, conçue pour pénaliser plus fortement les Faux Négatifs (FN), c'est-à-dire les cas où un mauvais client est prédit comme bon, ce qui entraîne l'octroi d'un crédit et une perte en capital. Cependant, il pourrait être plus pertinent de collaborer avec les équipes métier afin de créer et d'utiliser une métrique plus adaptée aux besoins spécifiques du domaine.

**Feature engineering :** De même, les équipes métier pourraient apporter une contribution précieuse au processus de feature engineering en collaborant à la création de variables plus pertinentes d'un point de vue métier. Cela permettrait d'améliorer la précision et l'efficacité du modèle.

**Algorithmes et hyperparamètres :** L'exploration d'autres hyperparamètres et d'algorithmes de classification constitue un autre axe d'amélioration potentiel. En testant différentes options et en comparant leurs performances, il est possible d'optimiser le modèle et d'obtenir des résultats encore plus précis.

En résumé, l'amélioration du modèle peut se faire par le biais de collaborations avec les équipes métier pour affiner la métrique d'évaluation et le feature engineering, ainsi que par l'exploration d'autres options algorithmiques et de hyperparamètres.

# Data Drift

On constate qu'il n'y a pas de data drift entre nos données de référence (données d'entrainement) et nos données actuelles (données de test), bien qu'un léger drift ait été détecté dans 10 de nos colonnes.

A screenshot of a computer

Description automatically generated