Note méthodologique

Contexte

La mission a été définie dans le cadre du projet 7 du parcours Data Science d'Openclassrooms « Implémentez un modèle de scoring » qui focalise sur le contrôle de versions, le déploiement sur cloud avec pipeline d'intégration continue et le dashboard de suivi de l'algorithme.

Dans le cadre de ce projet, nous sommes Data Scientist pour l'entreprise « Prêt à dépenser ». Cette entreprise propose des emprunts bancaires à des personnes ayant peu ou pas d'historique de prêt. Le but est de créer un modèle de classification binaire prédisant si le client remboursera ou non son crédit en cas d'octroi.

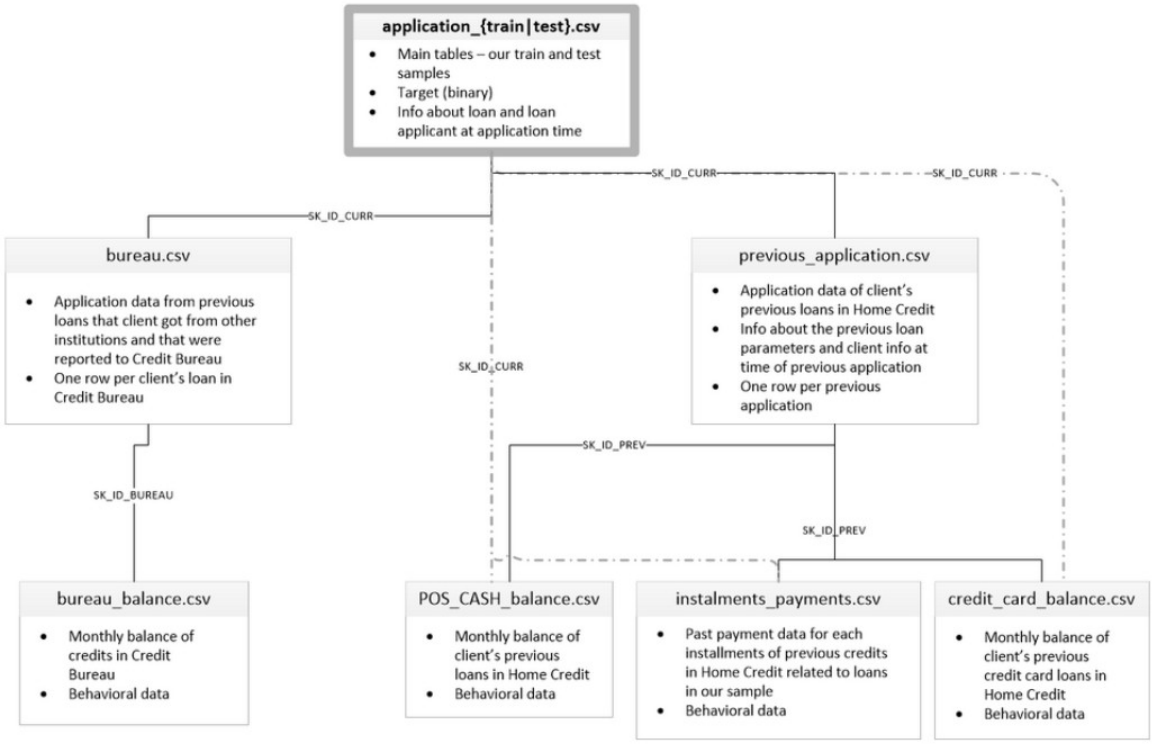
Les clients étant de plus en plus demandeurs de transparence, un tableau de bord doit être mis à disposition des conseillers clientèle afin qu'ils soient en mesure d'expliquer le résultat de l'algorithme.

Un client qui ne rembourse pas son crédit coûte beaucoup plus cher qu'un client qui se verrait refuser son emprunt alors qu'il l'aurait remboursé si celui-ci lui avait été octroyé. Cela doit être pris en compte dans l'algorithme.

Pré-traitement des données

Les données

Les données sont disponibles sur ce lien : https://www.kaggle.com/c/home-credit-default-risk/data
Voici le schéma relationnel des données :



La fusion des datasets

On constate que ces tables n'ont pas toutes les mêmes clés primaires sur lesquelles nous pourrions opérer des jointures.

Pour préparer notre dataset, il faut donc définir un dataset cible, ici application_train. Sur ce dataset seront jointes les données des autres tables.

Les autres tables vont donc être transformées pour que ces jointures puissent se faire. Des aggrégations selon les clés de jointure voulues sont faites et ces tables agrégées sont ainsi jointes au dataset cible pour former un dataset unique.

Nettoyage et imputation

Les données sont nettoyées lorsque l'on constate des incohérences et les colonnes contenant trop de NaN sont supprimées.

Nous utilisons One Hot Encoding sur les variables catégorielles puis IterativeImputer pour combler les NaN. Rappelons qu'à ce stade, les features contiennent moins de 10% de NaN.

Vérification d'un possible Data Leakage

Nous vérifions qu'il n'y ait pas de fuite de données, c'est-à-dire qu'une variable soit fortement corrélée à la variable cible. Cela n'est pas le cas. La variable ayant la plus forte corrélation avec la variable « TARGET » accuse une corrélation de 8,3%.

Equilibrage des classes

La target a 2 classes possibles : 0 et 1. 8% du dataset concerne la classe 1 (défaut de remboursement) et 92% la classe 0 (pas de défaut de remboursement).

Ce déséquilibre gênerait certainement l'algorithme. Nous allons donc utiliser SMOTE pour équilibrer le dataset. SMOTE va créer des lignes de la classe 1 ressemblant plus ou moins à celles existantes jusqu'à ce qu'il y ait 50% de chaque classe dans le dataset. Cette solution n'est pas parfaite mais elle permet de ne pas perdre de données contrairement à un sous-échantillonage de la classe majoritaire.

Afin de vérifier les résultats de l'algorithme sur des données « pures ». SMOTE ne sera appliqué que sur le jeu d'entraînement et non sur le jeu de test.

Modélisation

Les métriques d'évaluation

Nous utiliserons différentes métriques afin d'entraîner et d'évaluer les modèles.

La première et la plus importante est une métrique personnalisée que nous créons et qui s'appelle « score métier ». Pour chaque Faux positif sur le jeu de test, elle compte 1 et pour chaque Faux négatif, elle compte 10. Pour une bonne classification, elle compte 0.

Cette métrique est donc une fonction coût qu'il faut minimiser et qui suppose qu'un faux positif coûte 10 fois moins cher qu'un faux négatif.

Nous utiliserons également les accuracy train et test nous permettant de constater un éventuel overfitting. L'AUC avec visualisation de la courbe ROC ainsi que la matrice de confusion nous aideront à évaluer la performance du modèle.

La méthode

Les algorithmes utilisés sont un DummyCLassifier, une régression logistique, un LightGBM et un XGboost.

Une première version de l'algorithme non paramétrée (utilisant l'accuracy test comme métrique) sera exécutée et permettra de visualiser les feature importances.

Manuellement, un filtrage des features selon différents seuils de feature importances sera essayé. Le but, à la manière d'un RFECV, est de s'arrêter lorsque la performance de l'algorithme sur l'accuracy test se dégrade tout en consommant moins de ressources qu'un RFECV.

Le feature engineering effectué, il convient de définir les hyperparamètres. Nous utiliserons randomizedsearchcv pour cela car il est moins consommateur de ressources. La métrique à optimiser dans ce RandomizedSearchCV sera le score métier.

A ce stade, les résultats, quoique s'améliorant légèrement, ne seront pas mirobolants. C'est dans l'étape suivante que se joue l'amélioration du modèle : l'optimisation du seuil de predict_proba.

Le seuil optimal de probabilités est défini en cherchant à minimiser la métrique score métier. Le modèle sera ensuite enregistré avec la valeur du seuil retenu.

Les résultats

	Model	Temps exécution	Accuracy train	Accuracy test	AUC	Score métier test
0	Dummy Classifier	0.019	0.50	0.92	0.50	496.50
1	Régression logistique	79.070	0.72	0.71	0.70	318.12
2	LightGBM	73.600	0.96	0.93	0.55	441.91
3	LightGBM optimisé	47.360	0.93	0.84	0.78	227.70
4	XGBoost	155.650	0.97	0.93	0.60	392.91
5	XGBoost optimisé	185.720	0.93	0.85	0.77	235.46

Interprétabilité du modèle

Possibles améliorations

1) Métrique d'évaluation

Le score métier sur lequel a été optimisé le modèle a été établi sur une hypothèse qui doit être vérifié par le métier. Il pourrait également prendre en compte le montant du crédit demandé pour établir son score avec une valeur non en points selon que la prédiction soit faux négatif ou faux positif mais en euros perdus, soit par non-remboursement du client soit par manque à gagner.

Il pourrait également être possible d'ajouter de la complexité en préconisant des assurances emprunteurs différentes selon les risques, cela permettrait d'accepter beaucoup plus de personnes, malgré des taux plus élevés pour ceux qui auraient été refusés.

2) Optimisation des hyperparamètres

Les hyperparamètres ont été définis avec RandomizedSearchCV. Avec plus de puissance de calcul, nous pourrions ajouter beaucoup plus de paramètres, être plus fin et utiliser GridSearchCV qui pourrait être plus intéressant.

3) Ajouter des données externes

Les données relatives à la conjoncture économique par secteur ou localisation pourrait aider à affiner le modèle et éviter de mauvaises surprises sur des emprunts de longue durée.

4) Meilleure compréhension du fonctionnement emprunteur

En allant à la source du fonctionnement de l'emprunteur (pourquoi il rembourse ou pourquoi il ne rembourse pas), de nouvelles variables pertinentes pourraient être définies et prises en compte, ce qui améliorerait considérablement le score.

5) Stratégie d'entreprise

Les banques reçoivent souvent la critique de « ne prêter qu'aux riches » et de ne pas jouer leur rôle dans la création monétaire. Beaucoup de concurrents essaient de s'en détacher en finançant des projets plus écologiques ou sociaux quitte à ce que les projets soient ne soient pas forcément les plus sécuritaires.

La prise en compte de ces stratégies marketing pourrait modifier l'algorithme.