Projet 7: Implémenter un modèle de scoring

Note Méthodologique

Asmae
RAJI KARROUCHI
06/12/2022

Introduction

La société “Prêt à dépenser” propose des crédits à la consommation pour des personnes ayant peu ou pas d’historique de prêt. Elle nous a recruté en tant que Data Scientist pour mettre en œuvre un outil de “scoring crédit" pour calculer la probabilité qu’un client rembourse son crédit, puis classifie la demande en crédit accordé ou refusé. Il s’agit donc d’un algorithme de classification dont il est question, qui s’appuiera sur des sources de données variées telles que des données comportementales, données provenant d’institutions financières et autres.

Transparence

Les chargés de relation client de “Prêt à dépenser” ont fait remonter que les clients sont de plus en plus demandeurs de transparence vis-à-vis des décisions d’octroi de crédit. Cette demande de transparence va dans le sens des valeurs que l’entreprise veut incarner. C’est la raison pour laquelle elle souhaite développer avec notre aide un dashboard interactif pour que les chargés de relation clients puissent à la fois expliquer de la façon la plus transparente possible les décisions d’octroi de crédit, mais également permettre à leur clients de disposer de leurs informations personnelles et de les explorer facilement.

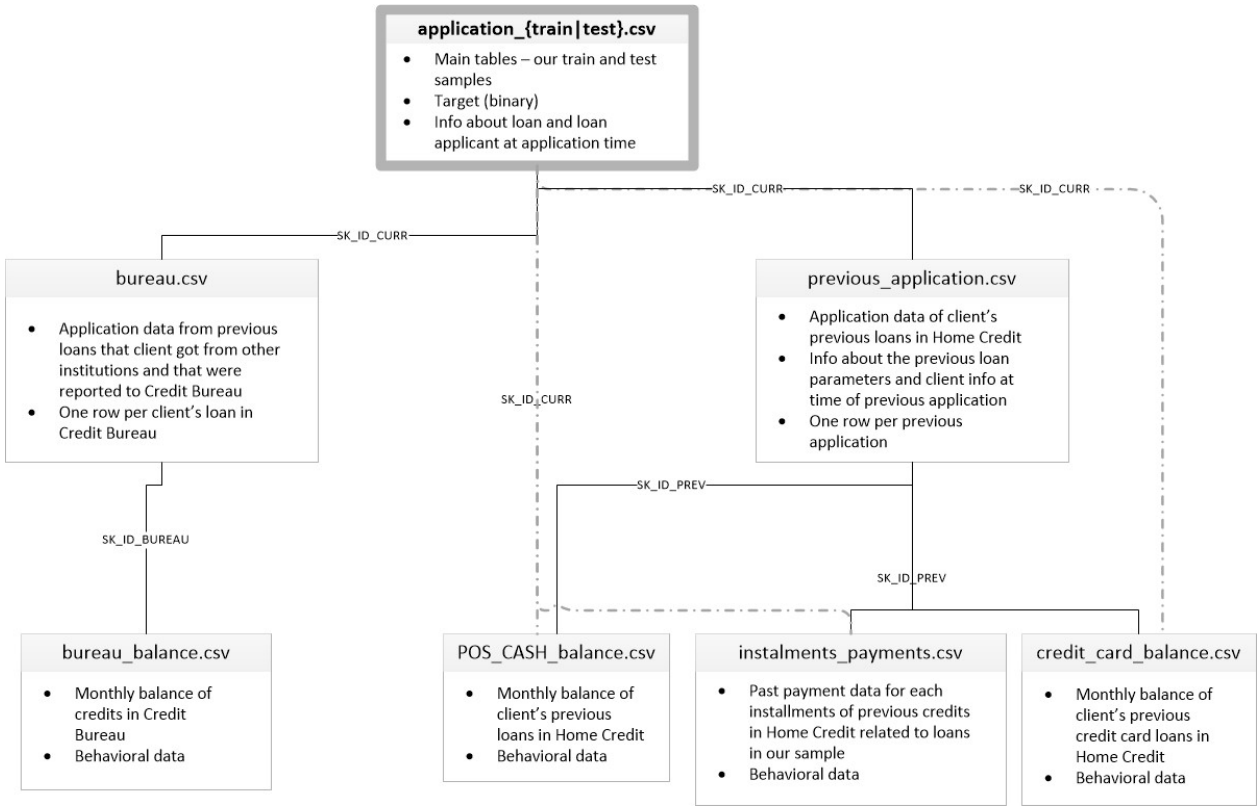
Dashboard

Michaël, notre manager, nous a donnée des spécifications pour le dashboard interactif :

- Permettre de visualiser le score et l'interprétation de ce score pour chaque client de façon intelligible pour une personne non experte en data science.
- Permettre de visualiser des informations descriptives relatives à un client (via un système de filtre)
- Permettre de comparer les informations descriptives relatives à un client à l’ensemble des clients ou à un groupe de clients similaires

Données

Les données nous sont fournies dans une base de données dont on nous donne un graphique explicatif ci-dessous:



Les informations concernant les clients se trouvent dans les tables application_train et application_test. Les autres tables sont des tables nous offrent des informations supplémentaires, notamment du “Credit Bureau” qui donne des informations sur les prêts précédents.

Entraînement du modèle

Environnement de travail

Pour manier et explorer les données et pour choisir et entraîner les modèles, nous allons utiliser l'environnement Python Anaconda. Il contient des notebooks Jupiter dans lequel nous allons exécuter le code concernant ces tâches. Il possède également de nombreuses librairies spécialisées dans l'analyse de données qui nous seront très utiles.

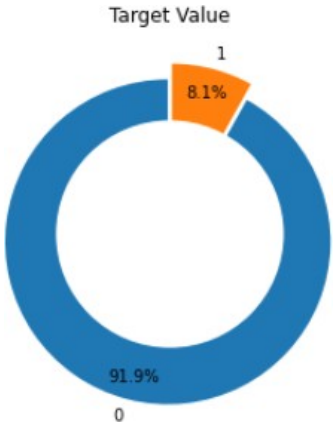
Traitement des données

Le traitement des données est une phase capitale pour la Data Science et ce fut l'objet principal de nombreux projets de cette formation. Cependant, pour celui-ci, nous avons choisi d'utiliser un travail déjà fait. En effet, nous avons utilisé un **kernel** existant sur le site kaggle.com.

https://www.kaggle.com/jsaguiar/lightgbm-with-simple-features

Ce kernel nous offre un traitement des données ainsi que du feature-engineering et a été utilisé avec succès par son créateur pour obtenir des prédictions très précises sur le même dataset. Nous allons utiliser ce kernel et le modifier en quelques aspects afin de pouvoir l'utiliser. En effet, nous ne pouvons pas utiliser les données du dataset "application_test" car les TARGET (les résultats) de ce dataset ne sont pas connus et sont utilisés pour donner un score aux modèles des utilisateurs Kaggle.

En plus de ce kernel, nous pouvons détecter que la variable cible (est-ce que le client rembourse son prêt) est **déséquilibrée.** 92% des clients remboursent leurs prêts contre8% qui ne les remboursent pas. Cela peut entraîner plusieurs solutions pour créer un
bon modèle. Nous avons décidé d'utiliser le "weight" où l'on indique à nos algorithmes qu'un résultat vaut "plus" qu'un autre. Ici, on souhaite que les clients qui ne remboursent pas valent 92/8 fois plus (grossièrement 10 fois plus) car on veut vraiment les limiter et éviter de deviner tout le temps 0 pour avoir 92% de précision.



Choix du modèle

Nous avons essayé plusieurs modèles pour ce problème. Le but étant de deviner une classe binaire. (0 ou 1 se traduisant par “Remboursera son prêt” ou “Ne remboursera pas son prêt”). Nous avons d’abord choisi quatre modèles :

- Cost-Sensitive Logistic Regression
- Cost-Sensitive Decision Trees
- Cost-Sensitive XGBoost
- Cost-Sensitive LightGBM

Le cost-sensitive indiquant la méthode dont on a parlé pour le déséquilibre du dataset.

Nos tests nous ont montré que le LightGBM était le plus efficace et beaucoup moins lent que le deuxième meilleur. Ce modèle avait aussi été utilisé par le créateur du kernel que nous avons utilisé, ce qui nous a confortés dans notre choix.

Modèle	Temps moy. Training (s)	Temps moy. Fitting (s)	Temps moy. Scoring (s)	Score de Test
Logistic Regression	0,58	7,98	0,05	**0,53**
Decision Tree	0,61	1,29	0,05	**0,58**
XGBoost	0,71	45,31	0,17	**0,64**
Light GBM	0,71	4,98	0,32	**0,64**

Optimisation du modèle

L'algorithme d'optimisation

Pour optimiser notre modèle, nous avons utilisé une GridSearchCV (étape de cross-validation) qui consiste à faire tourner l'algorithme plusieurs fois en changeant les paramètres pour trouver leurs meilleures valeurs. Ces hyperparamètres conditionnent le score du modèle. Score qu'on choisit selon les résultats que l'on souhaite obtenir.

La métrique d'évaluation

Notre méthode d'évaluation, ou score, dicte comment notre optimisation va se dérouler. Ici, on souhaite avoir le moins possible de cas dans lequel on prédit qu'un client va rembourser son prêt alors qu'il ne va pas le faire. Pour ceci, on ne peut pas deviner normalement les résultats car le prix d'un mauvais client pas découvert et beaucoup plus grand que celui d'un bon client dont on refuse le prêt. Pour ceci, on peut utiliser les métriques suivantes :

AUC ROC (Area Under the Curve, Receiver Operating Curve)

Montre la distinction entre les deux classes (0 et 1). On crée une courbe en mettant le False Positive Rate en abscisse et le True Positive Rate en ordonnée. On calcule ensuitel'aire sous cette courbe. C'est un chiffre entre 0 et 1.

Recall

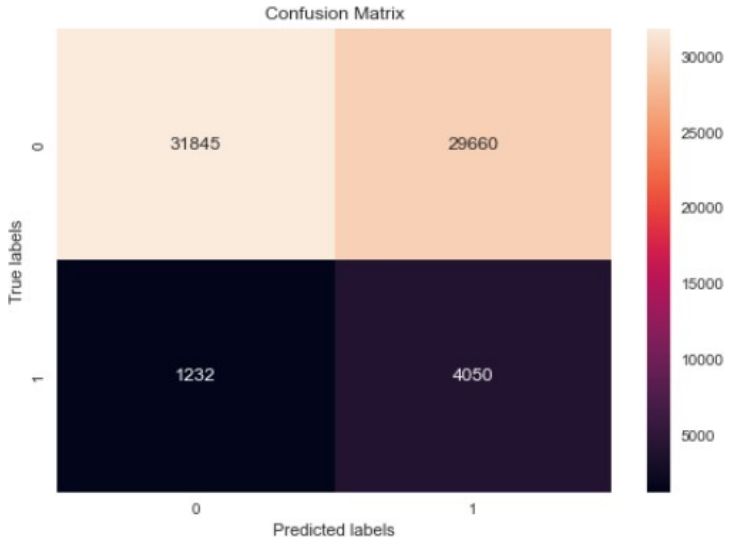
Donne le taux de vrais positifs (classe = 1) qu'on a réussi à deviner.

FBeta Score

Utilise la précision et le recall avec un paramètre Beta qui jauge l'importance de l'un ou de l'autre. On va utiliser ce score car il nous permet de mettre en avant le Recall par rapport à la Précision.

La fonction coût métier

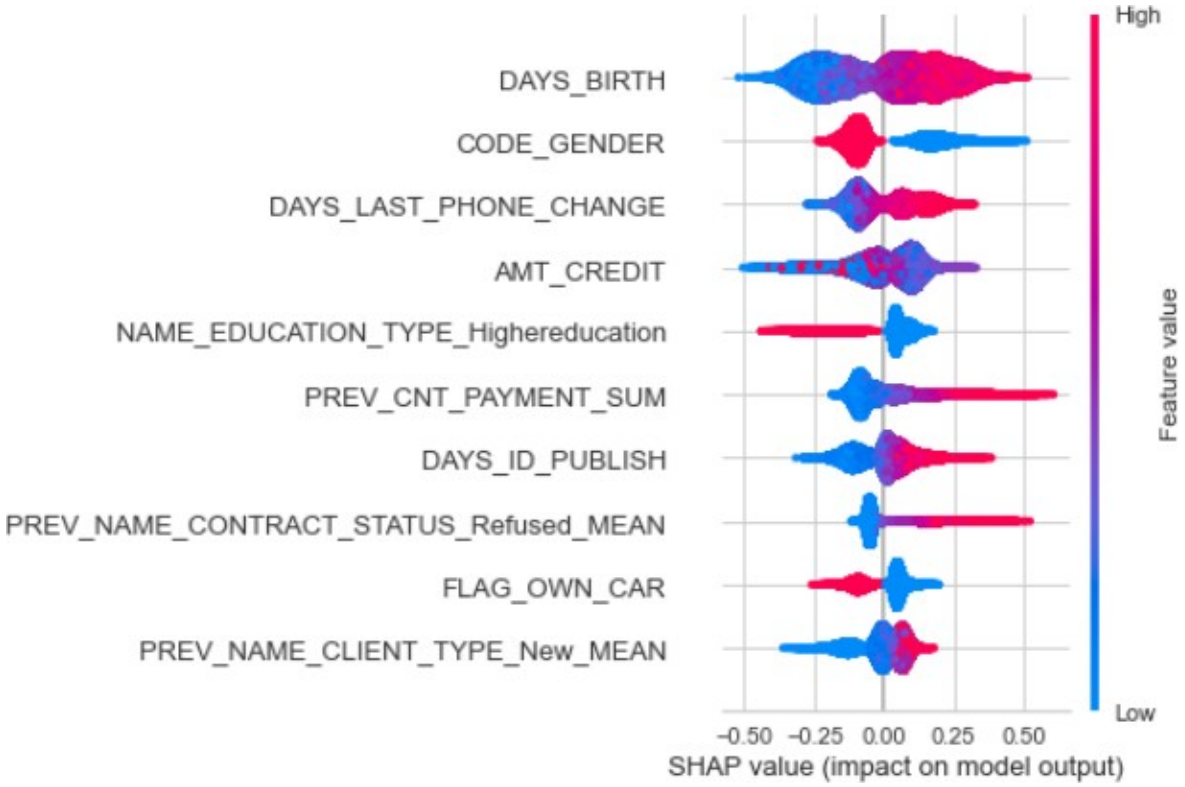
On peut utiliser une fonction coût métier pour deviner quand notre algorithme doit deviner 1 et quand il doit deviner 0. En utilisant le fBeta-score à 10, on obtient qu'une prédiction positive doit être supérieure à 0.014 et en ramenant le Beta à 2.5, on obtient un threshold de 0.062. Cela nous donne la matrice de confusion ci-dessous:



Interprétabilité du modèle

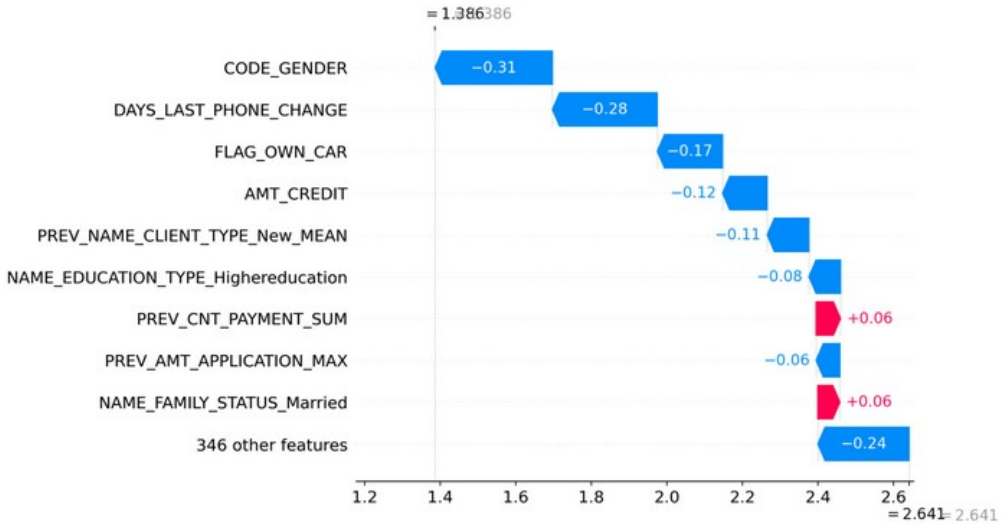
Interprétabilité globale

Grâce à la librairie Shap (https://shap.readthedocs.io/), on peut créer des graphs qui expliquent notre modèle (car le LightGBM est pris en compte par celle-ci). Le graph ci-dessous nous suggère que les variables les plus importantes de notre dataset sont donc notamment l'âge du client, son genre, le montant du prêt, etc.



Interprétabilité locale

Cette même librairie nous permet, client par client, de déterminer grâce au graphique ci-dessous, quelles sont les variables qui ont jouées pour l'acceptation ou non de ce client. Ici, on voit que le genre, la dernière fois que le client a changé de téléphone et lefait de posséder ou non une voiture ont joué en défaveur de ce client.



Limites et Améliorations

Limites

Limites matérielles des outils utilisés

les temps de chargement de l'API comme du Dashboard sur Heroku sont plus ou moins longs etne seraient pas du tout acceptables dans un environnement professionnel.

Limites matérielles du matériel

Lors de l'optimisation des modèles, de nombreux hyperparamètres n'ont pas été testé en profondeur car le temps d'exécuter tous ces GridSearch était déjà extrêmement conséquent. L'exécution de certains GridSearch dépassait les 2h.

En utilisant des services en ligne avec plusieurs machines ou d'autres solutions in-house, nous aurions pu faire tourner ces fonctions beaucoup plus longtemps et explorer beaucoup mieux le potentiel de tous les modèles testés, ainsi qu'améliorer celui choisi.

Limites de l'utilisation de ce modèle

Le problème de l'obtention de prêt est un problème très complexe qui se joue sur beaucoup de variables. De plus, notre modèle ne possède pas des scores qui reflètent la réalité sur nos datas d'entraînement.

Même s'il est modélisé pour limiter les pertes dela banque, il peut empêcher des clients tout à fait solvable de toucher un prêt qu'ils auraient remboursé. On peut enfin pointer du doigt que certaines raisons principales
pour lesquelles le modèle discrimine les clients (comme le genre) sont des biais qu'il est potentiellement illégal d'avoir sinon immoral. Aux Etats-Unis par exemple, l'"Equal Credit Opportunity Act" de 1974 rends illégal de ne pas donner de prêt à cause de certaines caractéristiques dont le sexe.

Améliorations possibles

Utilisation d'un meilleur service d'hébergement

En payant pour une version plus puissante ou en utilisant un autre service d'hébergement dans le cloud avec un calcul distribué, la vitesse d'exécution serait aussibien plus rapide

Personnaliser le pré-traitement

On pourrait utiliser un pré-traitement personnalisé et faire notre propre feature engineering au lieu d'utiliser un kernel déjà fait. Peut-être que nos spécificités et nos objectifs seraient mieux servis par une autre façon de faire.

Plus d'optimisation des modèles

Avec plus de puissance ou de temps disponible, il est certain que nous aurions un modèle plus performant. (et peut-être aurions nous utilisé un autre type d'algorithme)

On aurait par exemple pu retester tous les modèles en utilisant un fBeta-score personnalisé et beaucoup de paramètres différents.

Utiliser la technique SMOTE

Pour gérer notre problème de déséquilibre du dataset, nous avons utilisé les "weight" dans les algorithmes de façon à rééquilibrer au moment de l'entraînement du modèle. Il existe d'autres méthodes pour gérer ce problème, notamment utiliser SMOTE ou Synthetic Minority Oversampling TEchnique.

Cette technique consiste à prendre deux points qui sont proches dans l'espace des variables (un espace à plus de 100 dimensions ici), tracer une ligne entre les deux et créer un nouveau point sur cette ligne. Au final, on va créer des faux clients qui ne remboursent pas leur prêt en s'inspirant des (8%) clients qui ne remboursent pas existants.