Implémentez un modèle de Scoring

Note méthodologique

Oorvasi Sooprayen

Introduction

L’entreprise "Prêt à dépenser" propose des crédits à la consommation pour des personnes ayant peu ou pas du tout d'historique de prêts. Elle souhaite développer un modèle de scoring afin d'établir la probabilité de défaut de paiement d'un client.

Ce modèle s'appuiera sur des données variées (données comportementales, données provenant d'autres institutions financières etc...).

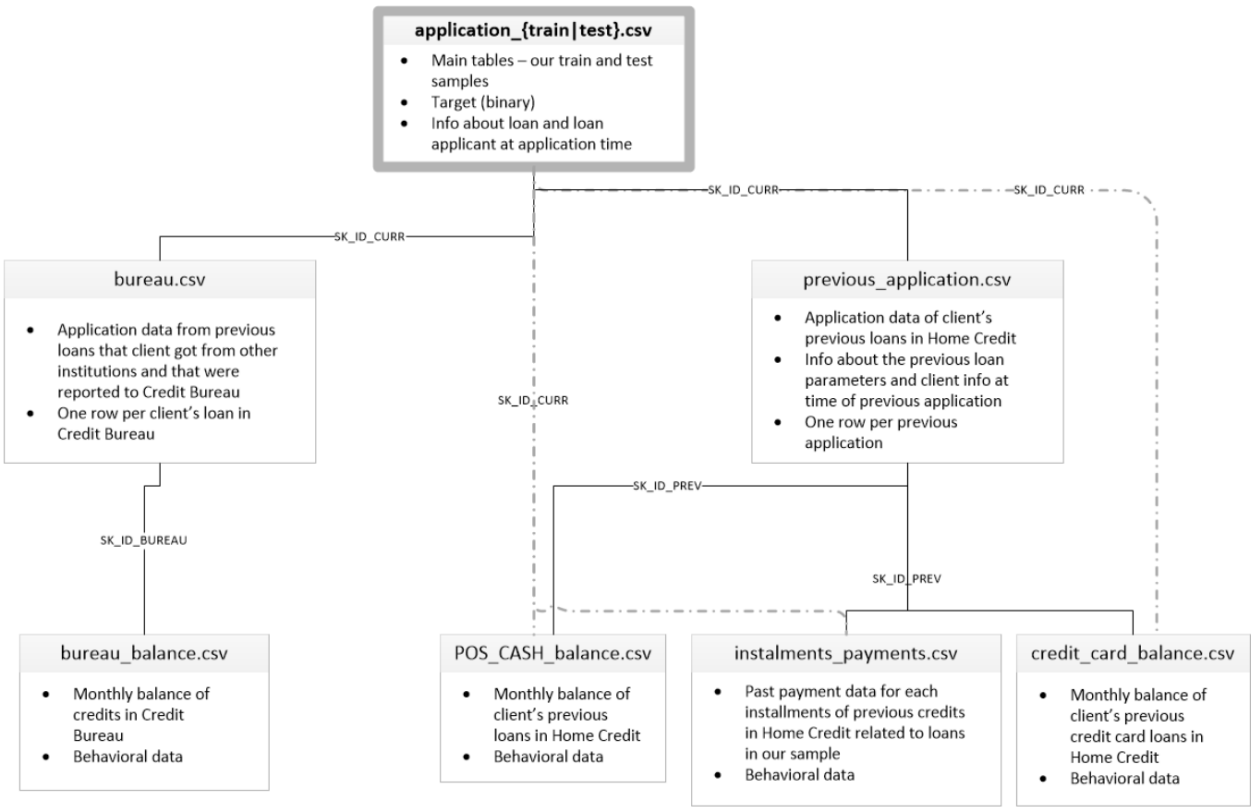
L'entreprise souhaite également disposer d'un tableau de bord qui permet d'expliciter les raisons de la décision d'accorder ou non un prêt.

Il s’agit ici d’un problème de classification binaire supervisée dans lequel la variable cible (target) à prédire est égale à 0 si un prêt est remboursable et 1 s’il ne l’est pas.

Ce document d’écrit de façon détaillée la méthodologie d’entrainement du modèle, la fonction coût métier, l’algorithme d’optimisation et la métrique d’évaluation, l’interprétabilité du modèle et les améliorations.

Méthologie d'entraînement

Nous disposons en entrée du modèle de 7 fichiers organisés de la manière suivante :



On fusionne les tables avec un JOIN sur la clé du client (SK_ID_CURR), en utilisant un script récupéré depuis kaggle (https://www.kaggle.com/jsaguiar/lightgbm-with-simple-features/) qui prend en compte l'élimination des valeurs aberrantes, l'agrégations sur les tables secondaires (MIN, MAX, MEAN, VAR, SUM), la transformation des variables catégorielles en vecteurs one-hot.

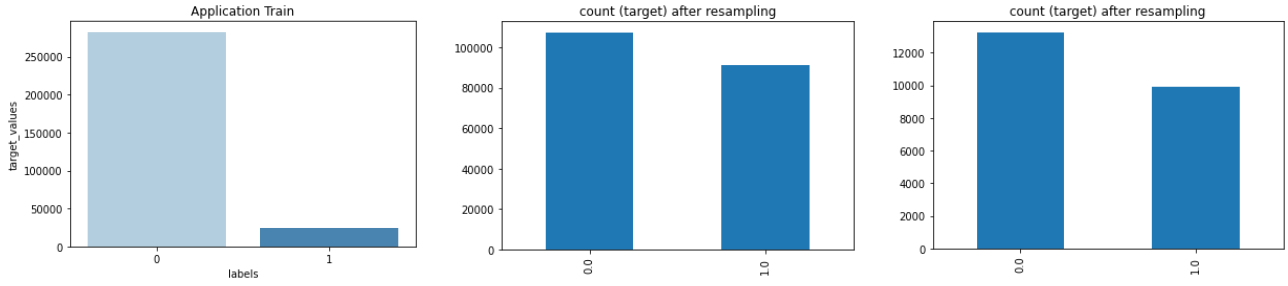
Le jeu de données nettoyés contient 206455 lignes et 729 colonnes

Le jeu de données nettoyés est ensuite divisé entre les jeux de données train (70%) et test (30%). Les valeurs manquantes ont été remplacées par la valeur médiane avec le module de scikit-learn SimpleImputer. Les données ont ensuite été mise à l'échelle avec MinMaxScaler.

Afin d'avoir une première idée des performances possibles, la modélisation par DummyClassifier est utilisée d'obtenir une baseline. Nous nous basons sur les résultats trouvés comme base de comparaison.

Vu du déséquilibre de données, plusieurs stratégies de la bibliothèque imbalanced-learn ont été testés aussi pour rééquilibrer les classes cibles pour assurer un équilibre dans les classes.

Avant Resampling	**SMOTE (Synthetic Minority Oversampling Technique) sampling_strategy=0.85**	**Random undersampling sampling_strategy=0.75**



Les algorithmes de classification calculent la probabilité d'appartenir à une classe donnée, ici le fait d'avoir un défaut de paiement. L'attribution d'une classe est ensuite réalisée à l'aide d'un seuil : selon que la probabilité est au-dessus ou en dessous du seuil, la personne est jugée comme risquant d'avoir un incident de remboursement ou non. 6 différents modèles ont été testés comme : LogisticRegression, LGBMClassifier, XGBClassifier, KneighborsClassifier, RandomForestClassifier, AdaBoostClassifier.

Ces algorithmes ont été comparés sur les résultats obtenus pour la métrique ROC AUC, ainsi que sur le temps de calcul. Il est apparu que LightGBM était plus rapide et le plus performants, un travail d'optimisation est ensuite fait sur le LightGBM.

Fonction de coût

La fonction coût est une estimation du gain financier ou de perte financière pour l'entreprise. Ici, un faux positif (bon client considéré comme mauvais = crédit non accordé à tort, donc manque à gagner de la marge pour la banque) n'a pas le même coût qu'un faux négatif (mauvais client à qui on accorde un prêt, donc perte sur le capital non remboursé) et donc l'objectif est de minimiser :

- Le nombre de clients ayant des situations sans risque de défaut de paiement dont les demandes de crédits sont refusées par erreur (l'erreur de type I ou les faux positifs) ;
- Le nombre de crédits accordés par erreur aux clients ayant des risques élevés de défaut de paiement (l'erreur de type II ou les faux négatifs).

Pour atteindre cet objectif, il faut :

- Maximiser le rappel et la précision ;
- Minimiser le taux de faux positifs (FPR) et le taux de faux négatifs (FNR).

On met ici comme hypothèse qu'un faux négatif est environ 10 fois plus coûteux qu'un faux positif. Les mesures techniques tels que le f1 score ne le prennent pas en compte. Nous proposons donc cette fonction pour calculer le nouveau fscore à minimiser :

fscore = poids*fn +fp

fp: faux positif

fn: faux négatif

poids : importance de fn par rapport à fp

Métrique

La métrique mesure la qualité globale du modèle. La métrique n'influence pas le calcul en lui-même mais permet de choisir à quel moment l'algorithme est optimal.

Ici, nous calculons un score qui doit permettre de décider si oui ou non un prêt sera accordé.

Nous retenons la courbe ROC, la courbe précision-recall, le Fbeta-score ainsi qu'une fonction personnalisée.

L'efficacité du modèle est déterminée en observant l'aire sous la courbe AUC (ou Area Under the Curve). Ainsi, le modèle le plus efficace a une AUC égale à 1, et le modèle le moins efficace a une AUC égale à 0,5.

La courbe ROC (Receiver Operation Characteristique) représente le taux de vrais positifs (True Positive Rate : tpr) en fonction du taux de faux positifs (False Positive Rate : fpr)

La courbe Precision – Recall represente naturellement la précision en fonction du recall. Là aussi il est possible de calculer l'aire sous la courbe qui peut être interprétée comme la moyenne de la précision pour chaque incrément du recall.

Les métriques sont calculées en comparant y_pred avec les vrais valeurs (y_true). On récupère le taux des faux positives et négatives de la matrice de confusion :

```
(TN, FP, FN, TP) = metrics.confusion_matrix(y_test, y_pred).ravel()
```

Optimisation

Une fois le modèle choisi, on effectue une nouvelle recherche des hyper-paramètres via HyperOpt se basant sur la fonction métier proposée, de cette façon, ils seront choisis de sorte à minimiser la perte pour l'entreprise.

HyperOpt nécessite 4 paramètres pour une implémentation de base à savoir : la fonction objectif (à minimiser loss = 1-score), l'espace de recherche (plages pour les hyperparamètres), l'algorithme d'optimisation et le nombre d'itérations.

Pour choisir les paramètres, nous les avons fait varier successivement. Nous obtenons finalement le résultat suivant :

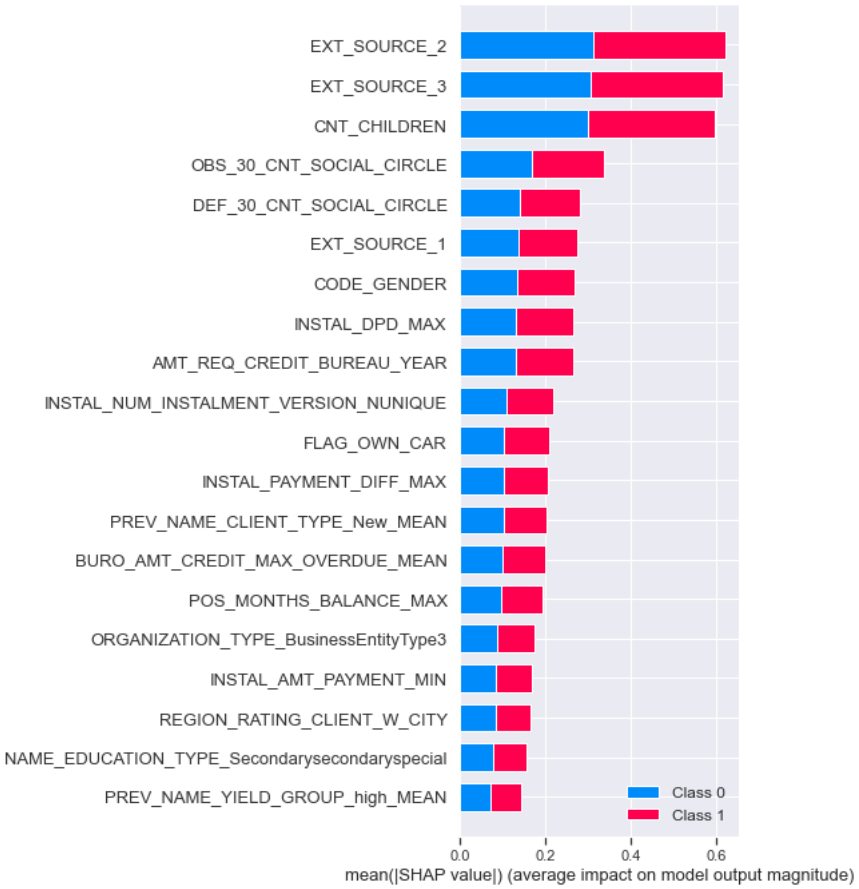
La bibliothèque hyperopt a un but similaire à gridsearch, mais au lieu d'effectuer une recherche exhaustive de l'espace de paramètre, il évalue quelques points de données bien choisis, puis extrapole la solution optimale basée sur la modélisation. En pratique, cela signifie qu'il faut souvent beaucoup moins d'itérations pour trouver une bonne solution.

L'algorithme « hyperopt » utilise une approche Bayésienne pour déterminer les meilleurs paramètres d'une fonction. A chaque étape, il essaye de construire un model probabiliste de la fonction et choisi les paramètres les plus promettant pour la prochaine étape.

Interprétabilité

L'objectif métier est de communiquer des informations pertinentes d'analyse au chargé d'étude, afin qu'il comprenne pourquoi un client donné est considéré comme bon ou mauvais prospect et quelles sont ses données/features qui expliquent son évaluation.

Il est donc nécessaire à la fois, de connaître d'une manière générale les principales features qui contribuent à l'élaboration du modèle, et de manière spécifique pour le client qu'elle est l'influence de chaque feature dans le calcul de son propre score (feature importance locale). Pour cela nous avons utilisé une librairie spécialisée de type Shap afin de fournir un calcul de la feature importance globale indépendante de l'algorithme.



Limites et améliorations

- Revoir la stratégie de traiter les valeurs manquantes
- C’est possible d’affiner les prédictions du modèle en essayant plusieurs poids de façon plus adéquate par des experts métier concernant le calcul pour la fonction de coût.
- Élargir le nombre d’hyperparamètres pourrait peut-être permettre une amélioration des performances des différents modèles.