Note Méthodologique : Implémentez un modèle de scoring.

Introduction

L'entreprise « Prêt à dépenser » souhaite mettre en œuvre un outil de « scoring crédit » pour calculer la probabilité qu'un client rembourse son crédit.

Pour cela nous allons développer un algorithme de classification qui s'appuie sur les données de ces clients. Ces données sont : comportementales, historiques des crédits et paiements, données provenant de sources externes, etc..

Pour tous ces clients de la base de donnée, nous avons une variable qui indique si oui ou non le client a remboursé son prêt.

L'entraînement des algorithmes (XGBoostClassifier et LGBMClassifier) va donc se dérouler sur ce jeu de donnée de manière supervisée.

I) Méthodologie d'entraînement du modèle

La première étape a été d'analyser, de traiter puis de concaténer les données de sorte à obtenir un dataframe sur lequel entraîner nos modèles.

Les données ont des sources et des formes différentes. Les variables non numériques ont été encodées, les nan et inf ont été remplacés par les valeurs médianes, et de nouvelles variables pertinentes ont été calculées (exemple des ratios).
Pour cette partie, on a utilisé un kernel pré-existant que l'on a customisé selon nos propres besoins.

La deuxième étape fut de choisir un algorithme. Celui-ci doit respecter les contraintes de l'étude, à savoir être un classificateur binaire, qui donne en sortie une probabilité d'appartenance à l'une ou l'autre des 2 classes, et donner le choix de la métrique d'entraînement (plutôt important ici).

Les deux modèles choisis sont XGBoostClassifier et LGBMClassifier. L'examen des résultats des entraînements et prédiction sur les données de test nous donnera le modèle à privilégier pour notre modèle final.

La troisième étape fut de mettre en place le process d'entraînement. Celui-ci consistait en :

- Séparation des données en 1 jeu de train et un jeu de test
- Une recherche des hyper-paramètres optimaux des modèles, sur un gridsearch
- L'entraînement des modèles sur le jeu d'entraînement, en passant par un gridsearchCV (cross validation)

- Utilisation de la fonction SMOTE (sur chaque kfold) pour ré-équilibrer les données.

En effet, les données présentaient un déséquilibre au niveau des classes (92% pour la target 0 et 8% pour la target 1). Ce qui aurait pour conséquence qu'un classifier qui attribuerait un score de 0 à chaque client aurait une précision de 92%...
Un oversampling a donc été mis en place pour « ajouter » des classes 0, en créant des nouveaux clients « synthétiques » (SMOTE).

La métrique choisie pour l'entraînement des modèles est la ROC_AUC de sklearn. Elle est la métrique qui permet de mesurer la performance d'un classificateur binaire. Elle donne la fraction de vrais positifs (dans notre cas on ne détecte pas de défaut de paiement à raison) par rapport aux faux positifs (on a détecté un défaut de paiement à tort).

II) Fonction coût métier et métrique d'évaluation

La problématique métier est de savoir si on accorde un prêt à un client. Notre modèle de classification binaire peut nous donner 2 réponses possible, oui ou non. Or l'entreprise est un organisme financier, qui a plus à perdre à accorder des prêt à des clients qui ne pourront peut-être pas rembourser.

Les clients identifiés par notre classificateur comme solvables alors qu'ils ne le sont pas sont appelés « faux négatifs » (car la classe 0 étant « oui le client va rembourser »).

Notre but est donc de diminuer au maximum le taux de faux négatifs, tout en gardant un taux de faux positifs relativement faible (on ne veut pas non plus exclure trop de clients qui sont en réalité solvables).

Ici c'est une question de compromis, ce point sera à discuter plus en détail avec l'équipe métier.

En effet, le réglage à effectuer pour nous est le paramètre de seuil de probabilité de notre modèle. E dessous d'une certaine probabilité on considère un client comme faisant partie de la classe « 0 » et au-dessus, de la classe « 1 ». Le nombre de faux positifs et faux négatifs dépend bien évidemment de ce seuil.

La métrique d'évaluation qui permet de minimiser le nombre de faux négatifs est le « Recall »

$$recall = \frac{Vrai\ Positif}{Vrai\ Positif + Faux\ Négatif}$$

Le recall permet de calculer le nombre de nombre de clients qui sont identifié à raison par le modèle comme non solvables, divisé par le nombre total de clients identifiés comme non solvables (à raison et à tort). Un recall proche de 1 nous donnera un taux de faux négatifs minimal, notre modèle ratera très peu de clients non solvables.

D'un autre côté, comme nous le disions précédemment, notre modèle doit quand même accorder des prêts à un nombre suffisant de clients ! Autrement dit le taux de faux positifs (clients prédits comme

non solvables qui le sont en réalité) doit être minimal.
Pour cela on doit maximiser la « précision » :

$$precision = \frac{Vrai\ Positif}{Vrai\ Positif + Faux\ Positif}$$

Finalement, la fonction d'optimisation qui prend en compte ces 2 métriques précédentes, avec un facteur de prépondérance réglable « beta » est le Fscore :

$$\text{Fscore} = \frac{precision.\ recall}{\beta^2.\ precision + recall}$$

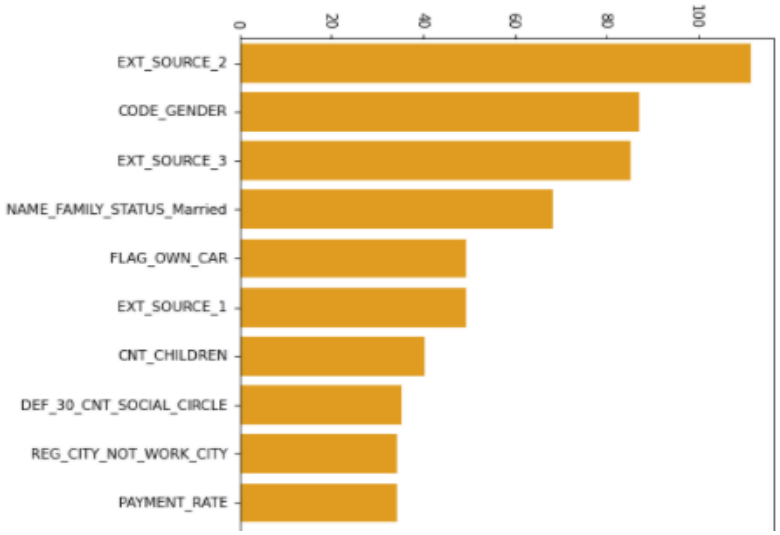
C'est ce paramètre qui doit être discuté avec les interlocuteurs « métier » pour cibler avec précision où se situe le beta optimal en terme de rentabilité.

III) Interprétabilité locale et globale du système

L'objectif est de pouvoir donner des informations pertinentes d'analyse au chargé d'études, pour qu'il comprenne le score donné par l'algorithme a un client donné, et quelles sont ses données/features qui expliquent ce bon ou mauvais score. Nous avons besoin de connaître cela à 2 niveaux :

- D'une manière globale (sur tous les clients) quelles sont les features qui participent à l'élaboration du modèle.
- D'une manière locale (sur un client donné) quelles sont les features qui sont prépondérantes dans son score final.

Pour l'étude globale nous avons utilisé la variable « feature_importances_ » de scikit-learn qui s'applique est récupérable directement en tant que paramètre du modèle entrainé :



Nous pouvons voir ci-dessus les features qui ont le plus de « poids » de manière globale dans le modèle entrainé (ici le LGBMClassifier).

Pour l'étude locale, on a utilisé l'algorithme shap, qui permet de tester l'importance de chaque feature pour un client donné, lesquelles ont eu une influence négative sur le score et lesquelles ont eu une influence positive.

Limites et améliorations possibles

1ère piste d'amélioration : Prendre en compte les résultats de feature importance globaux (et locaux ?) et ré-entraîner le modèle après suppression de variables non pertinentes.

2ème piste d'amélioration : Discussion en interne avec l'équipe métier sur les taux cible de Faux Positifs et Vrai Négatifs, pour réglage fin du seuil de sélection.