Projet 7: Implémentez un sytème de scoring

Note méthodologique

Stagiaire: DESOKI Hany **Mentor:** MUSCAT Tristan

24 Janvier 2022

Lien github du projet: `https://github.com/sourisaboule/Projet_7`

Introduction

Une société financière nommé "Prêt à dépenser" qui propose des crédits à la consommation souhaite un système de scoring qui permet de voir la probabilité d'un remboursement afin de les épauler pour une décision de si on accepte ou refuse le prêt. Nous avons à disposition pas mal de données. Avec ces données nous pouvons utiliser un modèle de machine learning pour obtenir ce score. Ce modèle a été déployer en ligne sous forme d'API et analysé à l'aide d'un dashboard. Ces liens sont dans le fichier notice_utilisation.txt.

Processus de modélisation

Données à dispositions

Nous avons à notre disposition les données issus de Kaggle qui montre les comportement financier et sociaux relatifs à des milliers de clients et sont séparés en plusieurs tables. La target est binaire et se trouve dans la table principale ('application') avec 0 pour crédit accepté et 1 pour refusé.

Preprocessing

Nos données sont séparées en plusieurs tables et contiennent des données numériques et catégoriel ainsi que des données manquantes, il y a donc beaucoup de preprocessing à faire. Heureusement pour nous, nous avons à notre disposition un kernel Kaggle qui fait la majorité de ce preprocessing. Ce kernel fais de nombreuses choses: Du feature engineering sur les revenus de clients par rapport au montant du crédit, du one hot encoding, des sommes et des moyennes sur les précédents crédits pour permettre de joindre toutes les tables en une seule table afin de pouvoir être passé dans un modèle de machine learning.

Cependant la manière dont ce kernel est fait n'est pas parfaite car le preprocessing se fait sur les données en entiers avant d'être séparés en un train et test set, ce qui augmente le risque de data leakage. Pour ce faire il faut adapter ce kernel en une classe que j'ai crée pour l'occasion nommé **CustomPreprocessing** dans le fichier custom_preprocessing.py. Cette classe assure que toutes nos données entrées soient traîtées de manière similaire et prend en compte les données du train set. Ainsi la table X en sortie gardera exactement le même format et sans données manquantes.

Classes déséquilibrées

En jetant un oeil à nos données, on constate rapidement que notre target est déséquilibré. En effet seulement 10% des prêts ont été refusés, ce qui veut dire que si l'on a un modèle qui prédit systématiquement que le prêt est accepté (dummy model), on obtiendrait une accuracy de 90%, ce qui est trompeur. Pour ce faire, nous allons utiliser SMOTE de la bibliothèque imblearn.
SMOTE permet d'artificiellement ajouter des points de la classes minoritaires afin de rendre les classes égales et ainsi aider le modèle à s'entraîner. Il faut noter que ces points ajoutés s'appliquent uniquement lors de l'entraînement. La deuxième chose à faire est de choisir une métrique plus pertinente:

Dans notre cas on veut minimiser le plus possible les faux négatifs, c'est à dire les crédits acceptés pour des personnes qui n'auront potentiellement pas les moyens de rembourser, car j'estime qu'ils sont plus coûteux que les faux positifs.
Pour ce faire, on va utiliser le fbeta score:

$$precision = \frac{vrai\_positif}{vrai\_positif + faux\_positif} \tag{1}$$

$$recall = \frac{vrai\_positif}{vrai\_positif + faux\_negatif} \tag{2}$$

$$fbeta = (1 + beta)\frac{precision * recall}{beta * precision + recall} \tag{3}$$

Avec un beta élevé (ici 9) le recall aura beaucoup plus de poids que la precision. Ainsi on pourra minimiser la fonction coût lors de l'apprentissage du modèle pour diminuer le plus possible les faux négatifs.

Entraînement du modèle

Pour ce modèle, j'ai choisi d'utiliser XgboostClassifier, il s'agit d'un modèle ayant un bon compromis, en effet ce modèle est souvent réputé pour son efficacité même sur des données complexes, de plus le modèle prend très peu de place en mémoire, c'est une donnée importante car on veut éviter d'envoyer un modèle de plusieurs centaines de Mo dans le cloud, et enfin, étant donnée qu'il s'agit d'un modèle basée sur des arbres, le modèle garde en mémoire l'importance des features. Il s'agit d'un bon outil pour l'interprétabilité.

Ici nous allons faire une recherche de paramètres avec gridsearchcv. Les paramètres que l'on va optimiser sont:

- reg_lambda
- gamma
- eta
- max_depth

Après l'entrainement, on obtient comme paramètre: eta: 0.05, gamma: 0, max_depth: 3, reg_lambda: 2.

Une fois ce modèle entrainé on pourra prédire le score de toutes les prêts grace à la méthode prédict_proba. Ce score va de 0 à 1. Plus le score est proche de 1, plus le prêt est susceptible d'être accepté. Cependant, par défaut le seuil de prédiction est à 0.5. Il est donc intéressant d'analyser les scores globaux du modèle en faisant varier le seuil de prédiction:

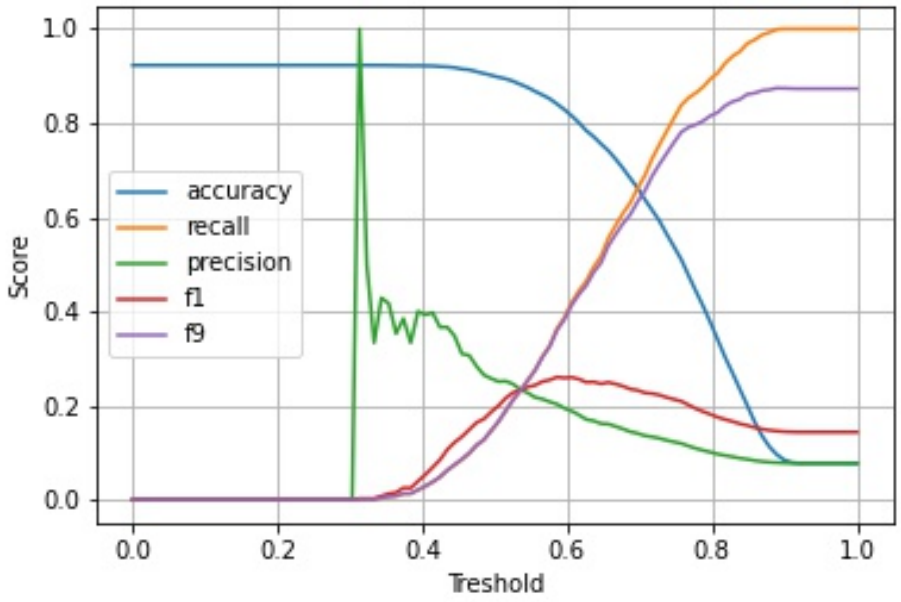

Figure 1: Evolution des scores en fonction du seuil de précision

En regardant ce graphique on constate que le seuil de précision le plus intéressant est de 0.7, en effet, c'est ici que l'on a le meilleur compromis entre le score f9 et l'accuracy.

Interprétabilité du modèle

Comme dit précédemment, xgboost comprend l'attribut feature_importance, ce qui va nous donner les features qui ont été le plus pertinentes pour l'apprentissage du modèle.
Malheureusement, les valeurs des features importances les plus grandes ne sont pas assez élevés pour faire une interprétation claire de ce que le score favorise. Un autre moyen d'interpreter est avec le dashboard.
On peut remarquer certaines différences entre les clients qui ont été refusé et les autres, par exemple les revenus sont en général plus bas pour les clients refusés.

Amélioration et limites

Nous avons vu le procédé de modélisation. Cependant il y a pas mal d'améliorations possibles. La limite principale est la connaissance du sujet. Le machine learning possède évidemment pas mal d'avantage, mais un gros inconvénient est le fait qu'il faut très souvent s'appuyer sur une expertise du domaine.
Ce qui fait que l'on aurai pu avoir un meilleur feature engineering pour obtenir des variables plus pertinentes, ou alors avoir une interprétation du modèle plus précise par exemple. On pourrait également choisir une meilleure fonction coût comme par exemple accorder plus d'importance à la minimisation des faux positifs.