Note méthodologique

Pour le projet 7 du parcours DataScientist - Implémentez un modèle de scoring

Omankhir Charafeddine

OPENCLASSROOMS

SOMMAIRE

I. Contexte

L'entreprise "Prêt à dépenser" propose des crédits à la consommation à des personnes ayant peu ou pas d'historique de prêt cherche à évaluer la probabilité de remboursement de chaque client et ainsi classer les demandes en crédits accordés ou refusés. L'entreprise souhaite développer un algorithme de classification. Pour cela, elle se basera sur des sources de données variées, telles que les données comportementales et celles provenant d'autres institutions financières.
Cependant, les clients sont de plus en plus demandeurs de transparence en ce qui concerne les décisions d'octroi de crédit. Pour répondre à cette demande, l'entreprise a décidé de développer un dashboard interactif. Ce tableau de bord offrira une transparence maximale aux clients et permettra aux chargés de relation client d'expliquer facilement les décisions d'octroi de crédit.
Le jeu de données fourni comprend 300 000 clients avec plus de 300 variables, mais il présente un déséquilibre important de classes, avec 90 % de clients qui remboursent leur prêt et 10 % de clients qui ne le remboursent pas.

II. Méthodologie d'entrainement du modèle

1. Feature engineering

Le feature engineering que nous allons mettre en place doit répondre à trois problématiques principales :

- L'historique des clients
- La gestion des valeurs manquantes
- La gestion des variables qualitatives et quantitatives

Pour la première problématique, nous avons utilisé un kernel disponible sur Kaggle (disponible ici)

Pour la deuxième problématique, nous avons procédé de la manière suivante :

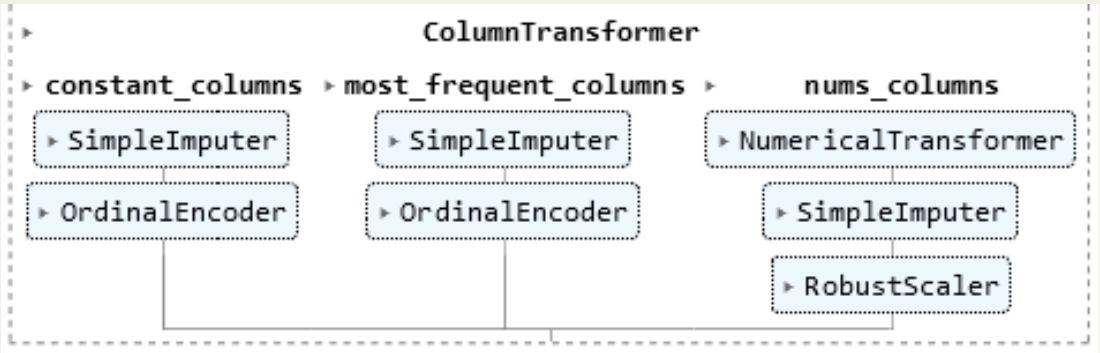
- Nous avons supprimé les variables avec un taux de valeurs manquantes supérieur ou égal à 50%.
- Compte tenu de la taille du jeu de données, des méthodes telles que IterativeImputer ou KNNimputer ne sont pas adaptées en termes de temps d'exécution ou de calcul. Ainsi, pour les variables quantitatives, les valeurs manquantes ont été remplacées par la médiane. Pour les variables qualitatives, les valeurs manquantes ont été remplacées par une constante "XNA" pour les variables avec un pourcentage de valeurs manquantes élevé, et par la valeur la plus fréquente pour les autres variables.

II. Méthodologie d'entrainement du modèle

1. Feature engineering

Pour la gestion des variables qualitatives et quantitatives, deux pipelines sklearn ont été créés : un pour les modèles basés sur les arbres de décision et un autre pour les modèles linéaires.

- Pour les modèles basés sur les arbres de décision, les variables qualitatives ont été encodées avec un OrdinalEncoder. Quant aux variables quantitatives, elles ont subi un premier prétraitement pour remplacer les valeurs +/- infinies par une valeur manquante. Ensuite, une normalisation avec un RobustScaler a été appliquée pour limiter l'impact des valeurs aberrantes.

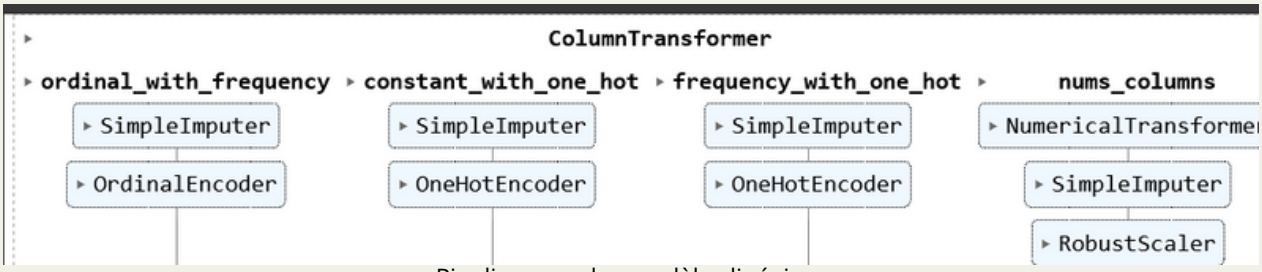


Pipeline pour les modèles basé sur les arbres de décision

- Pour les autres modèles, les variables ordinales ont été encodées avec OrdinalEncoder et les variables nominales avec OneHotEncoder.

II. Méthodologie d'entrainement du modèle

1. Feature engineering



Pipeline pour les modèles linéaires

2. Séparation jeu d'entrainement / test

Avant de séparer notre jeu de données on va prendre un échantillon de 20% du jeu de données initial afin d'accélérer les temps d'entraînement. Pour entraîner et évaluer nos modèles de manière adéquate, nous allons séparer notre jeu de données en deux parties : les données d'entraînement (46 125 individus) et les données de test (15 376 individus). Nous pourrons déterminer le score généralisé à l'aide d'une validation croisée à 3 folds en utilisant StratifiedKFold pour conserver la même distribution de classe. Le score obtenu sur les données de test représentera la capacité de notre modèle à généraliser.

II. Méthodologie d’entrainement du modèle

3. Sélection des features

Pour la sélection des caractéristiques, nous utiliserons Boruta, une méthode robuste pour sélectionner les variables les plus importantes en les comparant à des variables "fantômes". Étant donné le temps de calcul nécessaire, cette étape sera exécutée en dehors des pipelines, uniquement sur l'ensemble de formation.

4. Traitement du déséquilibre des classes

Comme le jeu de données est déséquilibré, avec 90% des clients remboursant leur prêt et 10% ne le remboursant pas, il est important d'utiliser des méthodes pour gérer ce déséquilibre. Ici, pour la gestion du déséquilibre des classes on utilisera seulement les paramètres des différents modèles comme "class_weight", "scale_pos_weight" qui vont permettre d’ajusté la fonction de coût du modèle de sorte à ce que le fait de mal classer une observation de la classe minoritaire soit plus fortement pénalisé que le fait de mal classer une observation de la classe majoritaire.

II. Méthodologie d'entrainement du modèle

5. Choix de la métrique à optimiser

	Pred Negative class	Pred Positive class
Negative class (Bon payeur)	TN (Bon payeur bien prédit)	FP (Bon payeur prédit mauvais payeur)
Positive class (Mauvais payeur)	FN (Mauvais payeur prédit bon payeur)	TP (Mauvais payeur bien prédit)

Notre principal objectif ici est de limiter les pertes importantes en minimisant les faux négatifs et d'éviter de perdre des clients potentiels en minimisant les faux positifs. Dans ce contexte, nous pouvons utiliser deux métriques techniques que nous chercherons à maximiser:

- **L'aire sous la courbe (AUC) :** qui mesure la capacité du modèle à distinguer les vrais mauvais payeurs (TP) des faux mauvais payeurs (FP).
- **Le fbeta score :** qui combine la précision et le rappel tout en accordant un poids à l'une ou l'autre de ces métriques en modifiant le paramètre bêta. La précision représente le pourcentage de mauvais payeurs correctement prédits, tandis que le rappel mesure la capacité du modèle à prédire correctement les mauvais payeurs. Avec un bêta égal à 2 donnant plus de poids au rappel.

II. Méthodologie d'entrainement du modèle

5. Choix de la métrique à optimiser

	Pred Negative class	Pred Positive class
Negative class (Bon payeur)	TN (Bon payeur bien prédit)	FP (Bon payeur prédit mauvais payeur)
Positive class (Mauvais payeur)	FN (Mauvais payeur prédit bon payeur)	TP (Mauvais payeur bien prédit)

On peut également créer une métrique métier à minimiser en supposant que le coût d'un faux négatif (FN) est K fois supérieur à celui d'un faux positive (FP). Ainsi, on peut essayer de minimiser la fonction de coût suivante :

(FP + k * FN) / (TN + FP + k* FN + TP)

6. Choix du modèle final

Pour le choix du modèle de classification final, nous avons utilisé la méthodologie d'apprentissage établie dans la partie précédente. Les différents modèles ont été optimisés en fonction des métriques citées précédemment. Ensuite, un modèle final a été sélectionné pour chacune des métriques optimisées en fonction de ses performances sur la métrique en question, mais également sur sa performance globale. Pour le modèle final, celui-ci a été choisi en fonction de sa performance globale sur le jeu de test.

II. Méthodologie d'entrainement du modèle

6. Choix du modèle finale

Le modèle final est le XGBoost avec un score généralisé de 0.768 sur l'aire sous la courbe (AUC) et un score de 0.766 sur le jeu de test, démontrant une bonne performance globale.

III. Interprétation globale et locale

Pour l'interprétation globale et locale du modèle, nous utiliserons la bibliothèque SHAP, qui fournira une interprétation globale en prenant la moyenne des valeurs absolues des valeurs de Shapley, en classant par ordre décroissant les variables qui contribuent le plus à l'explication de la différence entre la valeur de référence et la prédiction. Pour l'interprétation locale, nous utiliserons directement les valeurs de Shapley pour l'échantillon choisi afin de comprendre la prédiction.

III. Interprétation global et locale

Pour l'interprétation globale on peut constater sur la figure 1 que des variables telles que le genre, les scores provenant de sources externes, et la possession ou non d'un véhicule ont le plus grand impact sur l'obtention du prêt ou non. Pour une compréhension plus approfondie de notre modèle on peut observer la figure 2. Nous pouvons voir, par exemple, que des scores élevés provenant de sources externes ont un impact positif sur l'obtention du prêt, tandis que le fait d'être une femme (encodée comme 1 dans nos données) a un impact négatif.

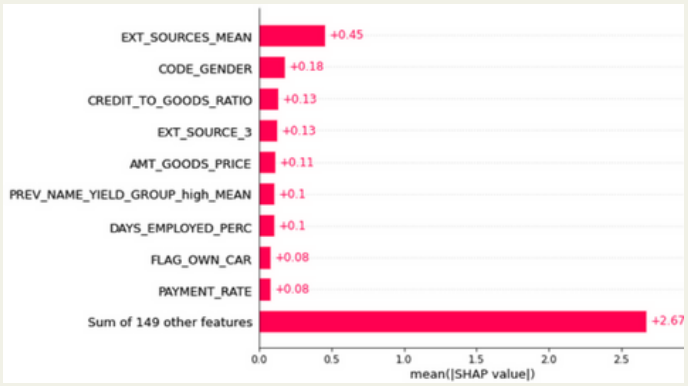


Figure 1

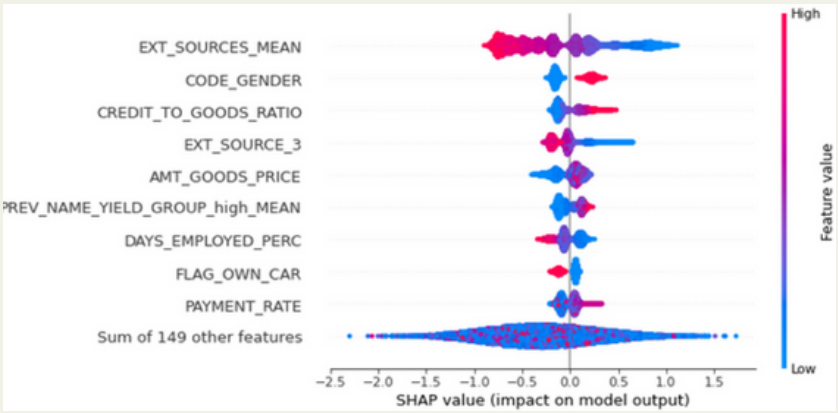
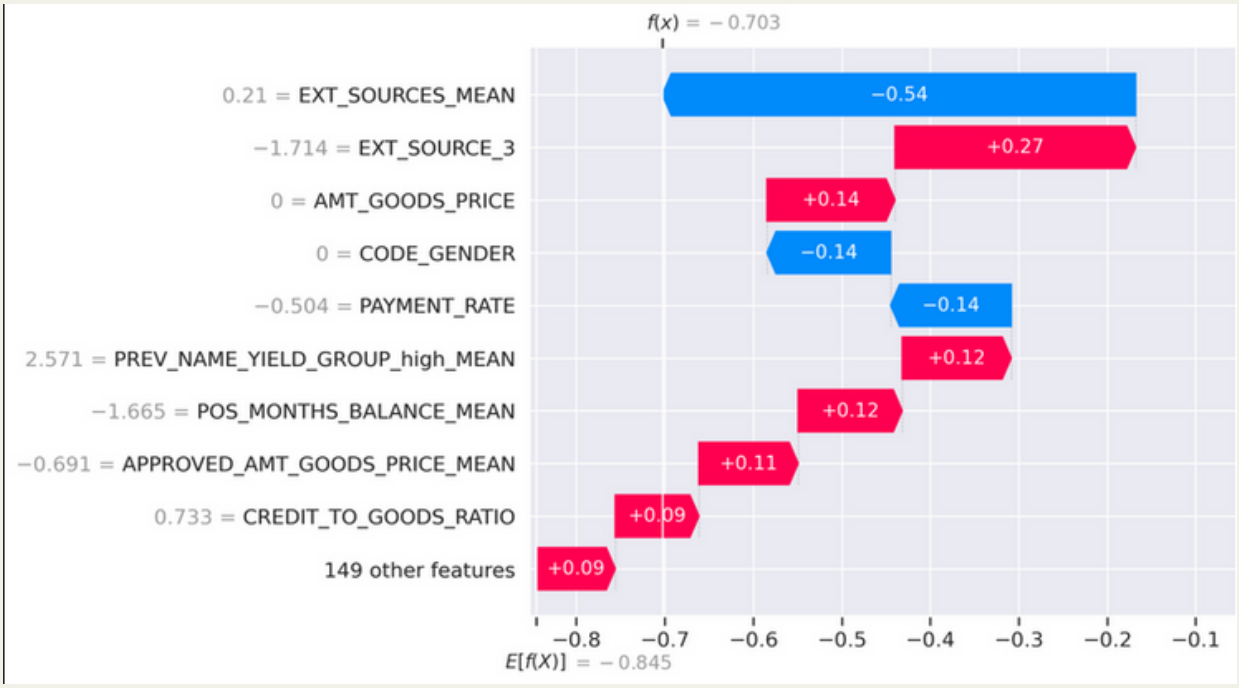


Figure 2

Pour l'interprétation locale, on va prendre l'exemple d'un client dans notre jeu de données. On peut voir que la contribution des 149 autres variables ont une contribution finale négative sur la prédiction mais au vu d'une moyenne des scores de source externe élevée, cette variable aura un impact très significatif dans l'obtention du prêt.

III. Interprétation global et locale



IV. Analyse du data drift

Pour notre analyse du Data Drift, nous avons utilisé Evidently avec le DataDriftPreset. Nous avons utilisé le jeu de données d'entraînement comme référence et le jeu de données de production comme données actuelles. Après l'analyse, nous avons constaté une différence dans 12 colonnes sur 158, soit 0,07% des colonnes. Les colonnes concernées sont PAYMENT_RATE, AMT_GOODS_PRICE, NAME_CONTRACT_TYPE,PREV_NAME_PORTFOLIO_XNA_MEAN, PREV_NAME_YIELD_GROUP_XNA_MEAN,

IV. Analyse du data drift

FLAG_EMAIL, PHONE_TO_BIRTH_RATIO,
PREV_NAME_CONTRACT_STATUS_Approved_MEAN,
CLOSED_DAYS_CREDIT_UPDATE_MEAN,
PREV_NAME_PAYMENT_TYPE_XNA_MEAN,
PREV_NAME_YIELD_GROUP_high_MEAN et
PREV_NAME_PAYMENT_TYPE_Cash through the bank_MEAN.

Pour rappel, le data drift survient lorsque les données sur lesquelles le modèle s'exécute diffère trop des données d'entraînement. Notre rapport sur le Data Drift montre quelques différences dans la distribution entre le jeu de données d'entraînement et de production. Avant de ré-entraîner notre modèle en ajoutant les nouvelles données, nous pourrions utiliser le rapport TargetDriftPreset d'Evidently pour vérifier la distribution des données en fonction des classes et voir s'il est pertinent d'ajouter de nouvelles données. Nous pourrions également planifier un ré-entraînement du modèle tous les x temps ou tous les x clients pour maintenir une bonne performance du modèle.

V. Conclusion

Pour résumer, nous avons entraîné un modèle de classification sur un jeu de données déséquilibré pour déterminer la probabilité de remboursement d'un client en utilisant leurs informations comportementales et leur historique. Pour résoudre le déséquilibre, nous avons utilisé des paramètres tels que class_weight et scale_pos_weight pour pénaliser le modèle. Nous avons obtenu un AUC de 0,766 et de bonnes performances globales du modèle.

Cependant, pour améliorer encore les performances, il serait utile de mieux comprendre le jeu de données et d'utiliser cette compréhension pour améliorer le feature engineering ou compléter le kernel utilisé. Pour gérer le déséquilibre des classes, nous pourrions également essayer des méthodes comme l'oversampling ou l'undersampling, les comparer et choisir la plus adéquate. Enfin, nous pourrions affiner notre recherche d'hyperparamètres et travailler avec l'équipe métier afin d'affiner la métrique d'évaluation.