Note Méthodologique

Projet 7 de la formation Data Scientist

Le projet consiste à 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 problème de classification binaire supervisée dans lequel la variable la cible (TARGET) à prévoir vaudra 0 si le client est solvable et 1 s'il ne l'est pas.

1. La méthodologie d'entraînement du modèle

1.1 Les étapes à suivre avant l’entraînement lu modèle

• Feature engineering :

Cette partie a été entièrement inspirée du notebook Kaggle suivant: https://www.kaggle.com/code/willkoehrsen/start-here-a-gentle-introduction/notebook.

Pour toute variable catégorielle (dtype == objet) avec 2 catégories uniques, nous utiliserons Label Encoding, et pour toute variable catégorielle avec plus de 2 catégories uniques, nous utiliserons One-Hot Encoding.

Pour Label Encodin, nous utilisons Scikit-Learn LabelEncoder et pour One-Hot Encoding, la fonction pandas get_dummies(df).

• Séparation des données :

Les données ont été divisées en train set (75 %) et test set (25 %) en utilisant la function train_test_split de la bibliothèque Skicit-learn.

• Pré-traitement des données :

On a faite l'imputation des valeurs manquantes (en apliquant SimpleImputer (strategy=median]) de la librairie Scikit-Learn) et on a normalisé les valeurs (en utilisant Min-MaxScaler de la librairie Scikit-Learn qui ramène l’ensemble des valeurs entre 0 et 1).

1.2 Modeles testés

• Régression Logistique :

Cet algorithme appartient à la famille des algorithmes linéaires. On a testé le classifier LogisticRegression de la librairie Scikit-Learn.

• Random Forest :

Cet algorithme (composé de nombreux arbres de décision) appartient à la famille des algorithmes ensemblistes. On a testé RandomForestClassifier de la librairie Scikit-Learn.

• Light Gradient Boosting Machine :

Cet algorithme appartient à la famille des gradient boosting algorithmes (il repose sur l'intuition que le meilleur modèle suivant possible, lorsqu'il est combiné avec des modèles précédents, minimise l'erreur de prédiction globale.). On a testé RandomForestClassifier de la librairie Scikit-Learn.

1.3 Gestion des données déséquilibrées

• Unbalanced data :

On a testé les 3 algorithmes sur les données déséquilibrées.

• Undersampling :

On a supprimé des observations de la classe majoritaire afin de rééquilibrer le jeu de données (avec la classe RandomUnderSampler).

• Oversampling :

On a répété des observations de la classe minoritaire afin de rééquilibrer le jeu de données (SMOTE - Synthetic Minority Oversampling Technique)).

1.4 Optimisation des hyperparamètres

On a utilisé la méthode GridSearch afin de optimise les valeurs des paramètres en choisissant différent valeur dans l'intervalle proposée.

1.5 Évaluation des modeles

Afin d'estimer la précision des modèles, on a utilisé roc_auc_score, ce score (qui se mesure de 0 à 1) est défini comme la zone sous la courbe ROC, qui est la courbe ayant le taux de faux positifs sur l'axe des x et le taux de vrais positifs sur l'axe des y à tous les seuils de classification. Plus roc_auc_score est élevée, meilleures sont les performances du modèle.

1.6 Choix du meilleur modèle

Modele/Stratégie de gestion des données déséquilibrées	roc_auc_score
Régression Logistique/ Imbalanced data	0.6881
Régression Logistique/ Undersampling	0.7221
Régression Logistique/ Oversampling	0.7352
RandomForestClassifier/ Imbalanced data	0.7150
RandomForestClassifier/ Undersampling	0.7437
RandomForestClassifier/ Oversampling	0.7121
LGBM/ Imbalanced data	0.7581
LGBM/ Undersampling	0.7560
LGBM/ Oversampling	0.7534

Le modèle LGBM appliqué aux données déséquilibrées donne les meilleurs résultats.

2. Optimisation du modèle sélectionné d'un point de vue métier

2.1 Les abréviations specifiques

Il y a les abréviations suivantes que on a utilisée dans cette étude que l'on va définir ci-dessous.

• **True Positive (TP):** Les Targets qui sont réellement vrais et que nous avons prédits vrais (Le prêt est (correctement) refusé : la banque ne gagne ni ne perd d'argent).

• **True Negative (TN)**: Les Targets qui sont en réalité faux et que nous avons prédits faux (Le prêt est remboursé : la banque gagne de l'argent).

• **False Positive (FP)**: Les Targets qui sont en fait faux mais que nous avons prédits vrais (Le prêt est refusé par erreur : la banque perd de l'argent qu'elle aurait pu gagner, mais ne perd en fait pas d'argent (Type-I Error)).

• **False Negative (FN)**: Les Targets qui sont réellement vrais mais que nous avons prédit faux (Le prêt est accordé mais le client fait défaut : la banque perd de l'argent (Type-II Error)).

2.2 En quoi consiste la problématique 'métier' ?

On doit prendre en compte qu'un FP (bon client considéré comme mauvais => crédit non accordé, donc manque à gagner du capital pour la banque) n'a pas le même coût qu'un FN (mauvais client à qui on accorde un prêt, donc perte sur le capital non remboursé). Un faux négatif est en effet 10 fois plus coûteux qu'un faux positif.

Donc on doit adapter le modelé afin de minimiser FN. Pour y parvenir on peut appliquer 2 stratégies.

2.3 Les strategies des minimization de perts

• Optimisation des hyperparamètres via HyperOpt :

On va définir une fonction métier adaptée au projet qui permet d'attribuer plus de poids à la minimisation des FN.

Apres, on a à nouveau effectué une nouvelle optimisation d'hyperparamètres via HyperOpt basée sur la fonction métier proposée (Hyperopt est une bibliothèque Python pour l'optimisation série et parallèle sur des espaces de recherche qui peuvent inclure des dimensions réelles, discrètes et conditionnelles). Les hyperparamètres seront choisis de manière à minimiser la perte d'argent pour la banque. On a remarqué que le score auc est amélioré (grâce à l'utilisation des hyperparamètres définis par la fonction).

• Adaptation de seuil (le meilleur f1_score):

La métrique ROC AUC nous donne une probabilité qu'un échantillon appartienne à une classe. Par défaut le seuil de décision est fixé à 0.5.

Par exemple, si le model prédit qu'un client à une probabilité de 0.6 d'appartenir à la classe des clients qui honore le remboursement de leur prêt (classe 0), alors en fonction du seuil fixé à 0.5, le client sera classé dans la classe 0. Si le seuil avait été fixé à 0.8 alors le client aurait

été classé dans la classe 1 et le prêt lui aurait été refusé. Donc le choix du seuil est très important. C'est lui qui détermine la réussite de l'objectif fixé. Pour réinitialiser le seuil on a utilisé les fonctions spéciales (que l'on peut trouver dans le notebook) pour trouver le meilleur seuil compte tenu du meilleur f1_score et de la règle selon laquelle un faux négatif est en effet 10 fois plus coûteux qu'un faux positif.

La métrique f1_score est la moyenne harmonique de la précision et du rappel.

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

Donc, le classificateur n'obtiendra un bon f1_score que si son rappel et sa précision sont élevés.

La précision est l'exactitude des prédictions positives :

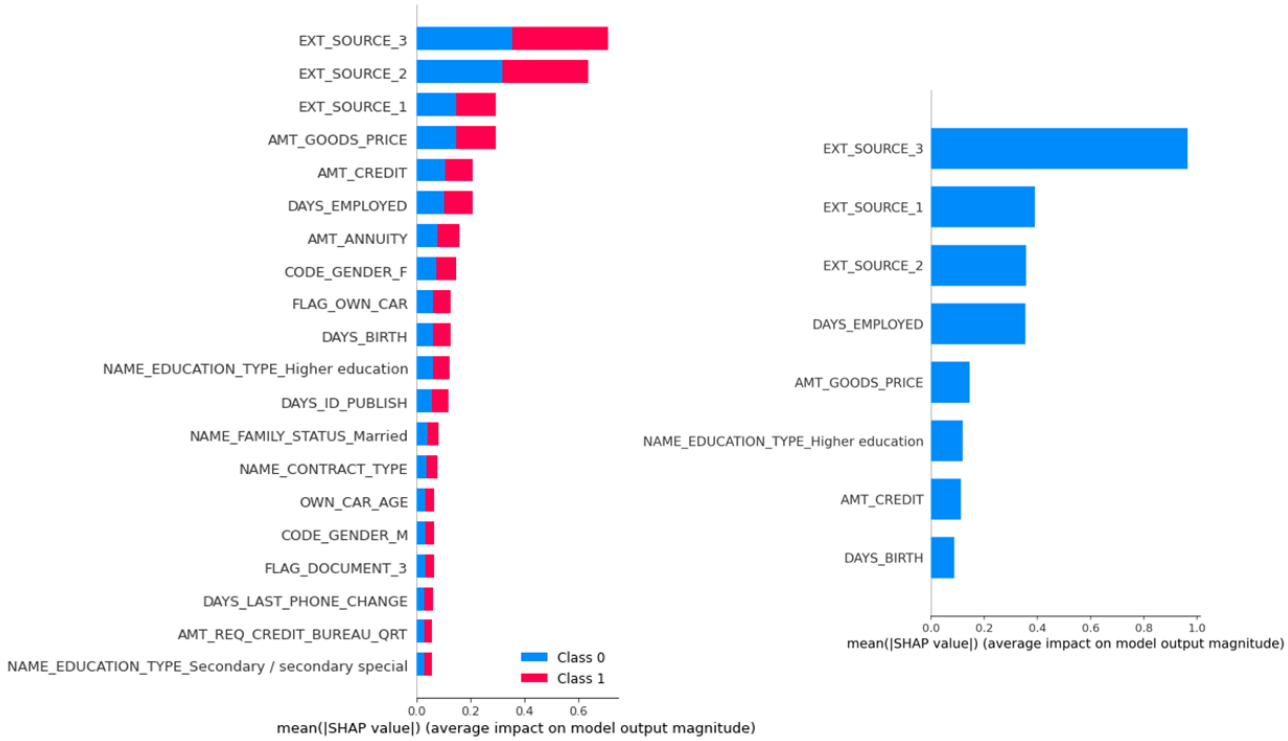
précision = TP / (TP + FP)

TP (abréviation de l'anglais True Positive) est le nombre de vrais positifs et FP est le nombre de faux positifs. Quant au rappel, c'est le taux d'observations positives ayant été correctement détectées par le classficateur :

rappel = TP / (TP + FN)

FN est le nombre de faux positifs.

3. Feature importance



On a expliqué les prédictions (les importances des features globales et locales (exemple - Client ID 100001)) du modèle choisi à l'aide de la bibliothèque SHAP. 3 variables EXT_SOURCE se sont trouvés les plus importantes lorsqu'elles prédisent la Target dans les deux cas.

4. Les limites et les améliorations possibles

• Améliorer la modélisation en affinant le travail de features engineering en collaboration avec les équipes métier.

• Utilisez la recherche d'expérience utilisateur pour adapter le dashboard aux besoins des utilisateurs.

• Chacun des modèles entrainés a été optimisé en testant plusieurs valeurs des 2-4 principaux hyperparamètres (car l'exécution de GridSearch prend beaucoup de temps). Elargir le nombre d'hyperparamètres pourrait peut-être permettre une amélioration des performances des différents modèles.