Projet 7 - Implémentez un modèle de scoring

Données du projet: https://www.kaggle.com/c/home-credit-default-risk/data

RISQUE DE DÉFAILLANCE DU CRÉDIT IMMOBILIER

- La concaténation des données des différents csv, le clearing et le feature engineering (calcul de nouvelles variables à partir de certaines variables, min-max-mean-sum-var de certaines variables, encodage de variables catégorielles) a été extrait d'un kernel d'un concours kaggle. Cette personne avait réalisé un super travail en étant bien classée avec un très bon score pour son modèle.

Toutes les tables sont jointes en utilisant la clé SK_ID_CURR (ID du client)

Variable cible:

- 1 - client ayant des difficultés de paiement : il/elle a eu un retard de paiement de plus de X jours sur au moins une des Y premières échéances du prêt dans notre échantillon
- 0 - tous les autres cas

Lien du kernel: https://www.kaggle.com/jsaguiar/lightgbm-with-simple-features/script

- Le but ici est de récupérer juste sa partie de travail avant le preprocessing (imputing nan et scaling) et de s'occuper des 4 prochaines parties:
    - Modélisation et Optimisation
    - Feature importance globale et locale
    - Mise en production d'une api (mise en place avec Flask, déployée sur Heroku) prédisant l'accord d'un prêt ou non avec score
    - Mise en production d'un dashboard (mis en place et déployé sur streamlit) qui appelle l'api

Structure du projet:

- ☐ Le code de la modélisation (du prétraitement à la prédiction):
    - notebooks/Feature_engineering_hcdr.ipynb
    - notebooks/Modeling_and_Optimization_hcdr.ipynb
    - notebooks/Preparation_of_predictions_and_Feature_Importance.ipynb
    - GitHub: https://github.com/Ludo13009/projet-7-home-credit-OC
- ☐ Le code permettant de déployer le modèle sous forme d'API
    - app.py
    - GitHub: https://github.com/Ludo13009/projet_7_home_credit_api
- ☐ Le code du Dashboard:
    - app.py
    - GitHub: https://github.com/Ludo13009/projet-7-home-credit-OC

Notes méthodologique

- **Démarche d'entraînement du modèle**
- **Algorithme d'optimisation et métriques d'évaluation**
- **Interprétabilité globale et locale du modèle LightGBM**
- **Limites et Améliorations**

Démarche d'entraînement du modèle

La première étape est le cleaning et feature engineering des données comme noté dans l'introduction du projet.

Le dataset, après feature engineering, contient 307 507 lignes (les clients) et 341 colonnes (les critères dont l'ID du client et sa target).

Séparation train and test data: Séparation train(70 %) et test(30 %) des données

On sauvegarde en csv les données test qui seront nos clients d'exemples pour API et dashboard.

Deuxième étape: Preprocessing

- ☐ On sépare les colonnes avec des entiers - variables continues dont des catégorielles (devenues continues par encodage durant le feature engineering) des colonnes avec des float - variables discrètes.
- ☐ Application de Simple Imputer (de scikit learn) pour remplacer les Nan selon 2 stratégies:
    - Si la variable est continue, on remplace Nan par la valeur la plus fréquente pour cette variable.
    - Si la variable est discrète, on remplace Nan par la moyenne des valeurs pour cette variable.
        - Bien sûr, on fit le modèle par les données train, et on transforme ensuite les données train et test.
- ☐ Application de RobustScaler pour scaler les données contenant de nombreux outliers. Fit par les données train et transformation des données train et test.

On sauvegarde ce pipeline de preprocessing ('preprocessor.pkl', dans le dossiers 'models')

Troisième étape: Resampling

Les données sont largement déséquilibrées:

- environ 92 % des données sont des clients à qui on accorde le prêt (classe 0 majoritaire)
- environ 8 % des données sont des clients à qui on refuse le prêt (classe 1 minoritaire)

La bibliothèque imblearn détient des fonctions pouvant résoudre le problème:

- Le sous-échantillonnage aléatoire (random undersampling) des observations majoritaires: on retire aléatoirement des observations majoritaires
- Le sous-échantillonnage synthétique (TomekLinks) des observations majoritaires: on retire des observations majoritaires, ressemblantes à des minoritaires (des majoritaires ambiguës)
- Le sur-échantillonnage aléatoire (random oversampling) des observations minoritaires : on tire au hasard des individus minoritaires (on les duplique) que l'on ajoute aux données.

- Le sur-échantillonnage synthétique (SMOTE pour Synthetic Minority Oversampling Technique) produit des observations minoritaires ressemblantes mais distinctes de celles déjà existantes.

Nous allons combiner SMOTE avec RandomUnderSampler pour rééquilibrer les classes. J'utilise pas Tomek car cela prend beaucoup trop de temps par rapport à la valeur ajoutée. On applique cette méthode que sur les données train.

- Avec une stratégie de SMOTE (sampling à 0.4) et de RUS (sampling à 0.6), Nous passons de 197 857 classe 0 à 131 903 et de 17397 classe 1 à 79 142.
- On s'approche des 60% / 40% pour la répartition des classes contre 92/8 au début.
- Si j'ai pas voulu avoir une répartition 50/50 c'est par soucis de qualité. Effectivement, les classes sont plus équilibrées mais on a ajouté des clients virtuels qui ressemblaient statistiquement parlant à de vrais clients minoritaires, et on a retiré des clients majoritaires. Equilibré encore plus c'est risqué d'overfit avec ces données là par rapport aux données test sans resampling.

Quatrième étape: Baseline model

On entraîne avec nos données train un Dummy Classifier classique avec comme stratégie "uniform". On obtient un score pour les données test faible de 0.50.

Cinquième étape: Choix du modèle et entraînement

J'ai fait le choix de 2 modèles (Random Forest Classifier et Light GBM Classifier) que j'ai comparé après entraînement. Le meilleur est le LightGBM. On se basera sur ce modèle pour la suite.

Pour voir l'intégralité de la comparaison: cf notebooks/Modeling_and_Optimization_hcdr.ipynb

LightGBM est un cadre de boosting de gradient rapide, distribué et haute performance basé sur des algorithmes d'arbre de décision, utilisé pour le classement, la classification et de nombreuses autres tâches d'apprentissage automatique.

→Entraînement du modèle LightGBM avec les données train et avec les paramètres optimisés (voir plus tard)

Sixième étape: Evaluation des performances

- Cross validation des données train (cv = 5) pour vérifier la stabilité du score et la généralisation du modèle selon 2 métriques (Voir plus tard)
- Scores des données test pour contrôler la cohérence (interprétés plus tard)

Étape supplémentaire:

- Light GBM propose un autre paramètre scale_pos_weight pour les classifications binaire comme dans notre cas
- On divise le nombre d'échantillons majoritaires(0) par ceux minoritaires(1) (on obtient 12)
- On assigne ce poids résultant pour valoriser la classe minoritaire(1)
- Plus l'échantillon minoritaire est bas, plus le poids sera grand

Algorithme d’optimisation et métriques d’évaluation

GridSearchCV de scikitlearn

- Les paramètres que j'ai choisi d'optimiser sont:
    - n_estimators: Nombre d'arbres dans la forêt (100, 500, 1000)
    - max_depth: La profondeur maximale de l'arbre (2 à 5)
    - num_leaves: Feuilles d'arbre maximales (20, 25, 30)
    - learning_rate: Augmentation du taux d'apprentissage(0.1, 0.3, 0.5)
    - pos_scale_weight = 12
- Optimisation selon 2 métriques:
    - Score ROC AUC
    - Métrique métier (F-beta score)

Métrique Métier: La fonction F-beta, un dilemme entre précision et rappel

Prédiction:

- 0 - Négatif (Accord du prêt)
- 1 - Positif (Refus du prêt)

Score beta:

- **F-1 score:** "Dans l' analyse statistique de la classification binaire , le score F1 ou la mesure F1 est une mesure de la précision d'un test . Il est calculé à partir de la précision et du rappel du test, où la précision est le nombre de vrais résultats positifs divisé par le nombre de tous les résultats positifs, y compris ceux qui ne sont pas identifiés correctement, et le rappel est le nombre de vrais résultats positifs divisé par le nombre de tous les échantillons qui auraient dû être identifiés comme positifs."
- **F-beta score:** "Le F-beta score applique des pondérations supplémentaires, valorisant l'une de la précision ou du rappel plus que l'autre. Si beta=1, nous retrouvons le F-1 score où les 2 critères sont équitablement valorisés"

Attribuer fallacieusement ‘Positif’ à un client, c'est perdre un bénéfice potentiel. Cependant, attribuer fallacieusement ‘Négatif’ à un client, c'est perdre de l'argent. Nous voulons donc éviter à tout prix les faux négatifs. Maximiser la précision, c’est minimiser les faux positifs. Nous voulons, nous, maximiser le rappel, pour minimiser les faux négatifs. Plus le Bêta est petit(0.5), plus on valorise la précision, plus le bêta est grand(2), plus on valorise le rappel. Nous allons prendre un Bêta fixé à 2. **VOIR FIGURE 1 ANNEXE**

Résultats performance:

- 0.97 pour le score auc et 0.96 pour le score f beta (GridSearchCV sur les données train)
- *Stabilisation* à 0.99 pour le score auc et 0.90 pour le score f beta (Cross validation sur les données train)
- *Cohérence*: accuracy 0.71, roc auc 0.74, précision 0.17, rappel 0.64, f1 0.26, f-beta 0.41

Interprétabilité globale et locale du modèle LightGBM

Méthode shape (SHapley Additive exPlanation)
But: “Calculer la valeur de Shapley pour toutes les variables à chaque client du dataset. Cette approche explique la sortie du modèle par la somme des effets de chaque variable. Ils se basent sur la valeur de Shapley qui provient de la théorie des jeux. L’idée est de moyenner l’impact qu’une variable a pour toutes les combinaisons de variables possibles.”
Approche locale:
Grâce à la valeur de Shap, on peut déterminer l’effet des différentes variables d’une prédiction pour un modèle qui explique l’écart de cette prédiction par rapport à la valeur de base.
Approche globale:
En moyennant les valeurs absolues des valeurs de Shap pour chaque variable, nous pouvons remonter à l’importance globale des variables.

Annexe - Figure 2 - Importance globale
Sur l’image du haut, l’ importance des variables est calculée en moyennant la valeur absolue des valeurs de Shap. Les valeurs de Shap sont représentées pour chaque variable dans leur ordre d’importance.
Sur l’image du bas, chaque point représente une valeur de Shap pour un exemple, les points rouges représentant des valeurs élevées de la variable et les points bleus des valeurs basses de la variable.
Exemple:
On a une information supplémentaire sur l’impact de la variable en fonction de sa valeur. Par exemple AMT_GOOD_PRICES qui est la variable la plus importante, a un impact négatif quand la valeur de cette variable est élevée.

Annexe - Figure 3 - Importance local
Les critères en rouge ont un impact positif (contribue à ce que la prédiction soit plus élevée que la valeur de base) et ceux en bleus ont un impact négatif (contribue à ce que la prédiction soit plus basse que la valeur de base)
Exemple:
La valeur de base du dataset est à 3.057, la prédiction est de 14.532. Cet exemple a une prédiction de valeur bien plus élevée que la moyenne.

- Contribution positive de AMT_GOOD_PRICES (+1.67) et INSTAL_DBD_MAX (+1.56)
- Contribution négative de OCCUPATION_TYPE_High_skill_tech_staff (-1.16)

Ici, il faut comprendre que plus la valeur d’AMT_GOOD_PRICES est haute, plus la Shap value est basse. Autrement dit, l’effet de cette variable sera d’autant plus grand (positivement) quand sa valeur est petite. Ce critère étant le prix du bien immobilier que le client souhaite, ce raisonnement est logique.

Plus de précision avec 2 exemples (un prêt accordé et un prêt refusé) et des graphes sur le notebook suivant:
https://github.com/Ludo13009/projet-7-home-credit-OC/blob/main/notebooks/Preparation_of_predictions_and_Feature_Importance..ipynb

Limites et Améliorations

Les limites

- Assez bons résultats en termes d'accuracy (0.71) et de score auc (0.74).
- Le rappel atteint un résultat bien meilleur avec le travail réalisé (Resampling des données, optimisation des paramètres selon le score f-beta, Valorisation de la classe minoritaire avec plus de poids accordé dans le modèle LightGBM). Cependant, il pourrait être encore meilleur si on avait de base un jeu bien plus équilibré.
- Améliorer le rappel a bien sûr fait baisser la précision qui atteint un score faible de 0.17. Les scores de 0.26 pour le f-1 et de 0.41 pour le f-beta suivent la même logique.

Ces limites sont dues à un jeu largement déséquilibré (92 % de clients dont le prêt est accordé, 8 % dont le prêt est refusé). Le modèle apprend beaucoup mieux les caractéristiques d'un "bon client" que d'un "mauvais client". Le rééquilibrage des données a permis d'atténuer ce côté, mais pas d'enlever tout le problème ! Ce rééquilibrage crée des échantillons synthétiques non réelles, qui ne sont pas présentes dans les données test, entraînant un largement meilleur score sur les données train que sur les données test.

Les améliorations

- Augmenter le bêta (prendre 3 et +)
- Augmenter le poids donné à la classe minoritaire pendant l'entraînement (plus de 12)

Ces 2 axes d'amélioration peuvent améliorer le f-beta score (le rappel plus précisément). Mais il diminuerait le f-1 score (la précision donc), et potentiellement aussi les autres scores (ROC AUC, Accuracy …)

Notre travail aboutirait à de meilleurs résultats pour tous les scores si le jeu de données était rééquilibré par de vrais clients négatifs dans l'avenir. Effectivement, le modèle apprendrait aussi bien les caractéristiques d'une classe comme de l'autre.

Annexes

Figure 1 - Précision et Rappel - Wikipédia

- Précision = Vrai positifs / Vrai positifs + Faux positifs
- Rappel = Vrai positifs / Vrais positifs + Faux négatifs

On veut le moins de faux négatifs possibles, donc le plus grand rappel possible.

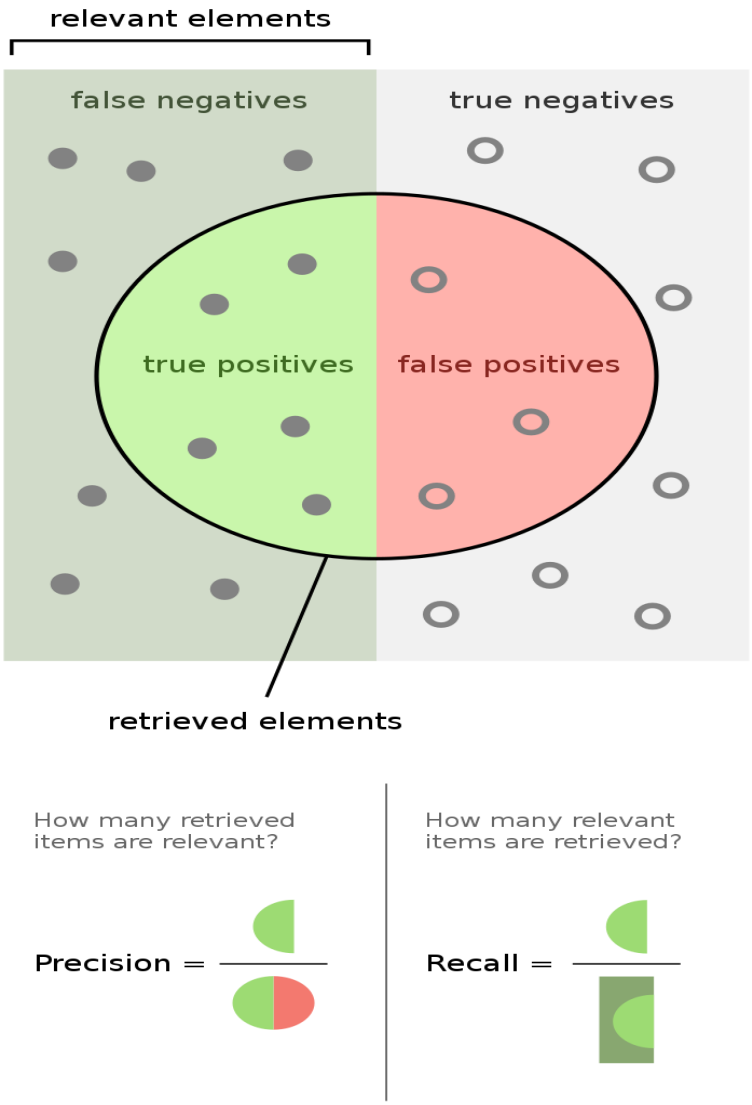

Figure 2 - Importance globale

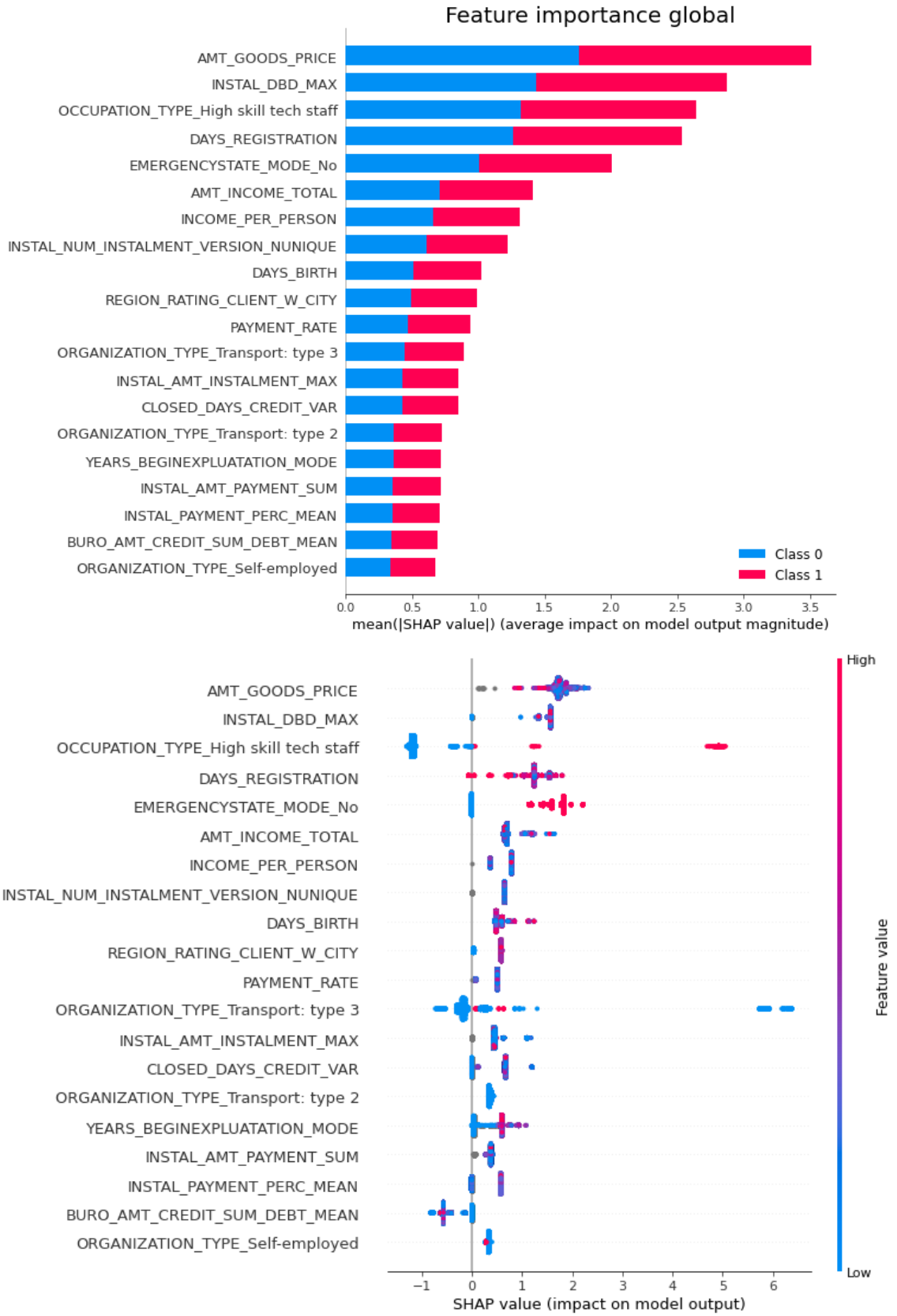

Figure 3 - Importance local