Julie Saubot

Projet 7 OpenClassrooms

26 janvier 2024

# Mémoire méthodologique

## Implémentez un modèle de scoring

## Contexte

La note méthodologique suivante concerne le projet 7 « Implémentez un modèle de scoring » de la formation Data Scientist d'Openclassrooms.

Ce projet consiste à mettre en place, pour une société qui proposent des crédits à la consommation « Prêt à dépenser », un outil de scoring crédit pour calculer la probabilité qu'un client rembourse son crédit puis classifie la demande en crédit accordé ou refusé.

Cela passe donc par le développement d'un algorithme de classification binaire en se basant sur des sources de données clients et le développement d'un dashboard interactif pour assurer de la transparence envers leurs clients sur l'octroi ou non de la demande de crédit.

# 1. La méthodologie d'entraînement du modèle

L'entraînement d'un modèle de machine learning suit une méthodologie bien définie. Tout commence par la collecte de données représentatives du problème à résoudre, suivie de l'exploration et du prétraitement de ces données.

Le jeu de données mis à disposition est composé de plusieurs datasets :

- **application_{train|test}.csv** : Il s'agit des données des informations clients où une ligne représente un prêt, divisé en deux fichiers pour Train (avec TARGET) et Test (sans TARGET).
- **bureau.csv** : Tous les crédits antérieurs du client fournis par d'autres institutions financières qui ont été signalés au bureau de crédit
- **bureau_balance.csv** : Soldes mensuels des crédits antérieurs dans le bureau de crédit.
- **POS_CASH_balance.csv** : Instantanés mensuels des soldes des crédits POS (point de vente) et des crédits de trésorerie antérieurs que le demandeur a contractés auprès de Home Credit.
- **credit_card_balance.csv :** Instantanés mensuels des soldes des cartes de crédit antérieures que le demandeur possède auprès de Home Credit.

Nous avons choisi de n'utiliser le jeu de donnée « application_train » auquel ont été ajouté les informations supplémentaires des autres datasets à l'aide de la clé « SK_ID_CURR ». Le jeu de donnée construit est ensuite passé par une étape de feature engineering basé sur un kernel Kaggle. Cette étape passe par la création de nouvelles features en appliquant les fonctions min, max, mean, sum et var aux variables du dataset groupé. Le jeu de donnée comporte des features catégorielles qui sont ensuite encodés en variables numériques à l'aide d'un one hot encoder qui va créer de nouvelles features qui représentent l'absence ou la présence d'une catégorie particulière (1 pour la présence et 0 pour l'absence).:

Les valeurs manquantes ont été éliminées en gardant que les lignes qui ont moins 80% des variables remplies puis en remplaçant les valeurs manquantes restantes par la mediane.

Les données ont ensuite été normalisées à l'aide de MinMaxScaler qui applique une normalisation des données entre 0 et 1.

Une fois les données nettoyés, encodés et normalisées, le jeu de données a été réduit puis séparé en un set d'entrainement et un set de test avec un ratio de 80% pour les données d'entrainement et de 20% pour les données de test. Un modèle DummyClassifier() a été utilisé comme base pour pouvoir comparer les modèles entrainés.

Différents modèles de classification ont ensuite été entrainés sur le set d'entrainement avec un RandomizedSearch avec 3 folds de cross validation en pipeline:

- DecisionTreeClassifier : Un arbre de décision divise l'ensemble des données en sous-ensembles plus petits en fonction des caractéristiques des données. Chaque nœud de l'arbre représente une décision basée sur une caractéristique.
- RandomForestClassifier : ensemble d'arbres de décision
- LGBMClassifier(LightGBM) : algorithme de boosting qui utilise une approche basée sur les histogrammes pour améliorer l'efficacité de l'entraînement.
- GradientBoostingClassifier : algorithme de boosting qui construit plusieurs arbres de décision séquentiels.
- XGBClassifier (XGBoost) : algorithme de boosting similaire à Gradient Boosting mais avec des améliorations, notamment une régularisation plus forte, une gestion plus efficace des valeurs manquantes et un calcul parallèle pour améliorer la vitesse d'entraînement.

Le modèle choisi est ensuite validé sur les données de test, en affichant la courbe ROC et la matrice de confusion. Nous avons ensuite essayé d'optimiser les hyperparamètres du modèle choisi.

## 2. Le traitement du déséquilibre des classes

Notre jeu de données utilisé pour l'entraînement comporte un déséquilibre des classes : la variable cible « target » compte 8% de classe 1 et 92% de classe 0.

Il y a donc beaucoup plus de prêts qui ont été remboursés à temps (classe 0) que de prêts qui n'ont pas été remboursés (classe 1). Cela peut affecter le modèle et entraîner des prédictions biaisées en faveur de la classe des prêts non défaillants. Il y a en effet trop peu d'exemples de la classe minoritaire (classe 1) pour qu'un modèle puisse apprendre efficacement la limite de décision. Il est donc important de choisir la bonne option pour traiter ce problème avant d'exécuter le modèle.

On a pu essayer trois méthodes pour atténuer le déséquilibre des classes :

- la méthode SMOTE (oversampling) : consiste à sur-échantilloner la classe minoritaire en créant de nouveaux exemples qui ressemblent aux autres sans être identiques
- RandomUnderSampler (undersampling) : consiste à réduire le nombre d'exemples de la classe majoritaire
- la méthode ClassWeight : consiste à attribuer des poids différents aux classes lors de l'entraînement d'un modèle, donnant ainsi une importance différente à chaque classe dans la fonction de coût. Cela aide le modèle à mieux s'adapter aux classes minoritaires.

Nous avons d'abord essayé de suréchantillonner la classe 1 avec l'algorithme SMOTE. Nous avons ensuite essayé d'ajouter au suréchantillonage SMOTE le sous-échantillonnage aléatoire de la classe majoritaire (undersampling). Enfin, nous avons essayé la méthode ClassWeight qui améliore significativement notre score métier.

## 3. La fonction coût métier, l'algorithme d'optimisation et la métrique d'évaluation

Le déséquilibre du coût métier entre un faux négatif (FN - mauvais client prédit bon client : donc crédit accordé et perte en capital) et un faux positif (FP - bon client prédit mauvais : donc refus crédit et manque à gagner en marge) doit être pris en compte. On a donc supposé le coût d'un FN dix fois supérieur au coût d'un FP pour créer un score métier (minimisation du coût d'erreur de prédiction des FN et FP) pour comparer les modèles, afin de choisir le meilleur modèle et ses meilleurs hyperparamètres.

Notre fonction cout métier est donc 10FN +FP. Nous avons maintenu pour comparaison et contrôle des mesures plus techniques comme ROCAUC score, l'accuracy score, le recall score, le F1 score et le F2 score. Nous avons également ressorti pour chaque entraînement le temps d'entrainement.

Nous avons entrainé la pipeline d'abord avec comme mesure de performance un AUC score puis un F2 Score et enfin notre score métier.

Nous avons comparé les résultats de la matrice de confusion, plus précisément le nombre de FN et de FP, pour chaque expérimentation :

- Sans le score métier : avec un AUC Score puis un F2 score
- Avec le score métier : SMOTE puis SMOTE et l'undersampling et enfin avec Class weight

Après étude de plusieurs modèles de classification, notre choix s'est porté sur le lightgbm avec la méthode Classweight.

L'optimisation du modèle choisi se fait en ajustant les hyperparamètres du modèle et le seuil de classification pour maximiser cette métrique.

Le seuil de classification, valeur que nous fixons et qui va limiter qu'une valeur appartient à la classe 0 ou a la classe 1, doit être recalculé pour résoudre le problème de classification déséquilibrée (un "predict" suppose un seuil à 0.5 qui n'est pas forcément l'optimum). On a donc testé pour différentes valeurs de seuil le score métier en comparant les vrai labels avec des labels obtenus en appliquant le seuil aux probabilités prédites. Le seuil optimal obtenu, pour lequel le score métier était minimal, est de 0.49.

## 4. Un tableau de synthèse des résultats

Le tableau de synthèse des résultats est composé de 8 lignes qui correspondent à :

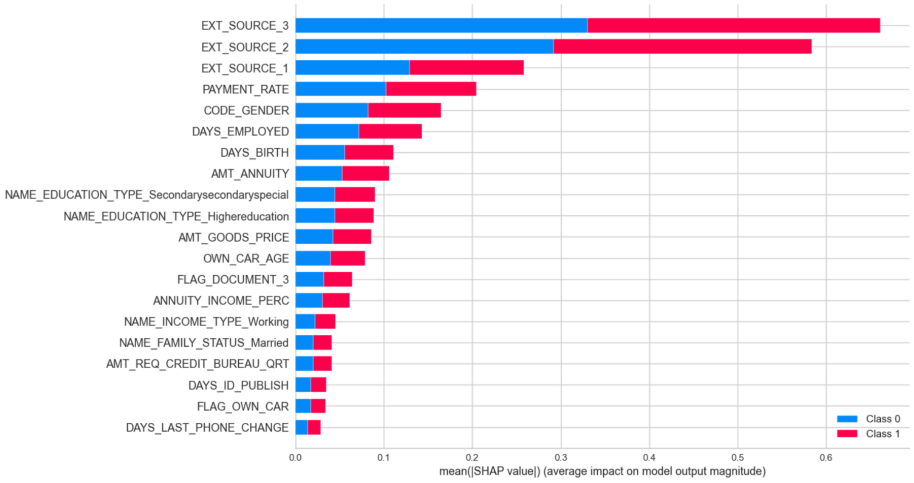
- index 0 : algorithme DummyClassifier comme base pour pouvoir comparer les modèles entrainés.
- Index 1 : modèle ayant obtenu le meilleur AUC Score
- Index 2 : modèle ayant obtenu le meilleur F1 Score
- Index 3 : modèle ayant obtenu le meilleur score métier
- Index 4 : modèle ayant obtenu le meilleur score métier avec la méthode SMOTE
- Index 5 : modèle ayant obtenu le meilleur score métier avec la méthode SMOTE puis undersampling
- Index 6 : modèle ayant obtenu le meilleur score métier avec la méthode ClassWeight : le meilleur modèle choisi
- Index 7 : fine tuning du meilleur modèle (lightgbm)

| | Modèle | Accuracy score | Recall score | Precision score | F1 score | F2 score | Score metier | ROCAUC score | FN | FP | fit time |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | DummyClassifier | 0.850743 | 0.080564 | 0.079777 | 0.080168 | 0.080405 | 50264 | 0.499474 | 4565 | 4614 | 20.000000 |
| 1 | (LGBMClassifier(max_depth=5, min_child_samples=200, num_leaves=60,\n random_state=27, subsample=0.5)) | 0.918750 | 0.006794 | 0.392857 | 0.013358 | 0.008456 | 16097 | 0.757097 | 1608 | 17 | 7.619647 |
| 2 | (XGBClassifier(base_score=None, booster=None, callbacks=None,\n colsample_bylevel=None, colsample_bynode=None,\n colsamp... | 0.916000 | 0.040766 | 0.341969 | 0.072848 | 0.049483 | 15657 | 0.727935 | 1553 | 127 | 180.166359 |
| 3 | (XGBClassifier(base_score=None, booster=None, callbacks=None,\n colsample_bylevel=None, colsample_bynode=None,\n colsamp... | 0.912800 | 0.067326 | 0.317784 | 0.111111 | 0.079924 | 15334 | 0.710319 | 1510 | 234 | 141.460969 |
| 4 | (LGBMClassifier(max_depth=20, min_child_samples=100, num_leaves=60,\n random_state=27, subsample=0.5)) | 0.917250 | 0.022854 | 0.336364 | 0.042799 | 0.028090 | 15893 | 0.741472 | 1582 | 73 | 25.229896 |
| 5 | (LGBMClassifier(max_depth=20, min_child_samples=100, num_leaves=60,\n random_state=27, subsample=0.4)) | 0.840700 | 0.406424 | 0.228234 | 0.292315 | 0.351533 | 11835 | 0.744577 | 961 | 2225 | 2.083536 |
| 6 | (LGBMClassifier(class_weight={0: 1, 1: 10}, max_depth=20, min_child_samples=100,\n num_leaves=40, random_state=27, subsample=0.6)) | 0.771750 | 0.555899 | 0.189633 | 0.282797 | 0.400998 | 11036 | 0.751079 | 719 | 3846 | 5.139081 |
| 7 | RandomizedSearchCV | 0.741750 | 0.611489 | 0.179153 | 0.277117 | 0.412431 | 10826 | 0.749573 | 629 | 4536 | 6.724137 |

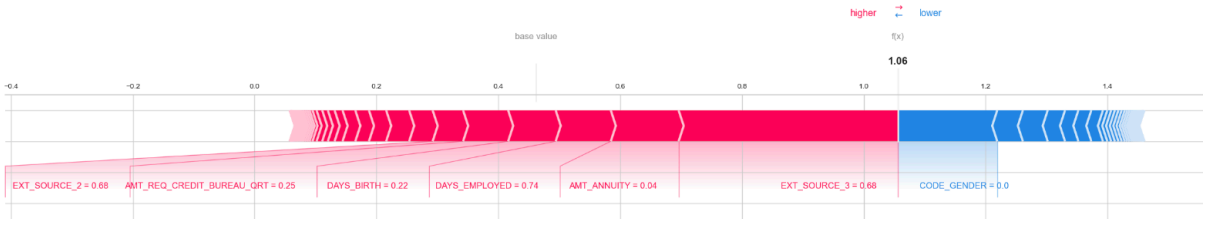
## 5. L’interprétabilité globale et locale du modèle

Nous avons effectué une analyse de la feature importance globale et locale avec la librairie shap. Cette analyse des variables nous permet de visualiser sur quelles variables s’appuie le modèle pour effectuer ses prédictions.

L'interprétabilité globale se réfère à la compréhension générale du comportement du modèle sur l'ensemble des données. Le graphique récapitulatif montre l'importance de chaque feature dans le modèle. Les résultats montrent que "EXT_SOURCE_1", "EXT_SOURCE_2" et "EXT_SOURCE_3" jouent un rôle majeur dans la détermination des résultats.




L'interprétabilité locale se concentre sur la compréhension du modèle pour des instances de données individuelles. Le diagramme de force est une autre façon de voir l'effet de chaque feature sur la prédiction, pour une observation donnée. Dans ce diagramme, les valeurs SHAP positives sont affichées à gauche et les négatives à droite, comme si elles étaient en concurrence les unes avec les autres.




Nous pouvons clairement voir que pour ce client, les variables EXT_SOURCE_3, AMT_ANNUITY et DAYS_EMPLOYED ont contribués au remboursement de son prêt.

## 6. Les limites et les améliorations possibles

Les limites du modèle sont :

- Les données biaisées : Si le jeu de données utilisé pour entraîner le modèle est biaisé, le modèle peut également être biaisé, ce qui peut conduire à des fausses prédiction.

- Le concept changeant : Les habitudes de remboursement des clients peuvent évoluer avec le temps, et si le modèle n'est pas mis à jour régulièrement, il peut devenir moins précis.

- La sensibilité aux données manquantes : Le modèle peut être sensible aux données manquantes. Si certaines informations cruciales sont absentes, cela peut affecter la précision des prédictions.

- La sensibilité aux outliers : La présence d'outliers peut avoir un impact significatif sur la prédiction, surtout si ces outliers ne sont pas représentatifs du comportement général des clients.

Les améliorations possibles sont :

- L'utilisation de techniques avancées de modélisation : L'utilisation de modèles plus avancés tels que les réseaux de neurones ou les méthodes d'apprentissage profond peut permettre au modèle de saisir des relations plus complexes.

- L'interprétabilité améliorée : Les modèles de scoring de crédit nécessitent souvent une interprétabilité élevée. Travailler sur des modèles interprétables, tels que les arbres de décision, peut aider à comprendre comment le modèle prend ses décisions.

- La Mise à jour continue : Les modèles doivent être mis à jour régulièrement pour refléter les changements dans les habitudes de remboursement des clients.

- La validation externe : Effectuer une validation externe du modèle en utilisant des données qui ne font pas partie du jeu d'entraînement initial peut renforcer la confiance dans ses performances.

- La gestion des biais : Identifier et atténuer les biais potentiels dans les données et dans le modèle lui-même est crucial pour assurer une évaluation juste et éthique du risque.

## 7. L'analyse du Data Drift

Une fois le développement et le déploiement du modèle de machine learning, il faut s'assurer qu'il fonctionne comme prévu.

Le modèle peut recevoir des données incorrectes ou erronées ce qui va donc générer une prédiction peu fiable avec une baisse de la performance du modèle. Le modèle peut aussi se dégrader avec le temps même si la qualité des données est bonne. Il peut aussi y avoir de nouvelles données en entrée.

Pour détecter et résoudre ces problèmes à temps, il est nécessaire d'avoir une visibilité sur les performances du modèle. Cela peut en effectuant des contrôles de performance régulier à l'aide de l'agorithme « Data Drift Dectection» de la librairie evidently. Evidently applique plusieurs tests statistiques et méthodes de détection de drift pour détecter si la distribution a changé de manière significative non. Il permet d'avoir accès à un rapport de Drift qui renvoie pour chaque feature la distribution des données mais aussi si du drift a été détecté ou non.

Dans notre projet, on a pu comparer les données d'entrainement (application_train) et les données de test (application_test). On obtient un rapport interactif qui montre comment la distribution des données d'entrée a changé.

On peut voir que les distributions des features sont statistiquement différentes, cela pourrait être un avertissement que les performances de notre modèle pourraient diminuer. On a l'information qu'il y a du drift détecte pour 7.4% des colonnes, c'est à dire 9 colonnes sur 121.

## Dataset Drift

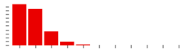
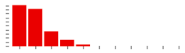
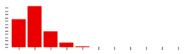
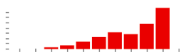
Dataset Drift is NOT detected. Dataset drift detection threshold is 0.5

| 121 | 9 | 0.0744 |
|---|---|---|
| Columns | Drifted Columns | Share of Drifted Columns |

### Data Drift Summary

Drift is detected for 7.438% of columns (9 out of 121).

Search

| Column | Type | Reference Distribution | Current Distribution | Data Drift | Stat Test | Drift Score |
|---|---|---|---|---|---|---|
| AMT_REQ_CREDIT_BUREAU_QRT | num |  |  | Detected | Wasserstein distance (normed) | 0.359052 |
| AMT_REQ_CREDIT_BUREAU_MON | num |  |  | Detected | Wasserstein distance (normed) | 0.281765 |
| AMT_GOODS_PRICE | num |  |  | Detected | Wasserstein distance (normed) | 0.210785 |
| AMT_CREDIT | num |  |  | Detected | Wasserstein distance (normed) | 0.207334 |
| AMT_ANNUITY | num |  |  | Detected | Wasserstein distance (normed) | 0.161102 |
| AMT_REQ_CREDIT_BUREAU_WEEK | num |  |  | Detected | Wasserstein distance (normed) | 0.15426 |
| NAME_CONTRACT_TYPE | cat |  |  | Detected | Jensen-Shannon distance | 0.14755 |
| DAYS_LAST_PHONE_CHANGE | num |  |  | Detected | Wasserstein distance (normed) | 0.138977 |
| FLAG_EMAIL | num |  |  | Detected | Jensen-Shannon distance | 0.122121 |
| FLAG_DOCUMENT_3 | num |  |  | Not Detected | Jensen-Shannon distance | 0.062496 |