Note méthodologique : Implémentez un modèle de scoring

Omar Jamadi
Dossier GitHub

24 avril 2023

Introduction

La société financière "Prêt à dépenser" propose des crédits à la consommation pour des personnes ayant peu ou pas du tout d'historique de prêt. L'entreprise souhaite mettre en œuvre un outil de *scoring crédit* pour calculer la probabilité qu'un client rembourse son crédit et classifier la demande en crédit accordé ou refusé. L'objectif est de développer un algorithme de classification en s'appuyant sur des sources de données variées.

Nous utiliserons pour concevoir le modèle les données issues de la compétition Kaggle *Home Credit Default Risk* composées de 8 jeux de données regroupant des informations bancaires et personnelles anonymisées de 356255 clients.

Ce document présente la méthodologie employée pour développer le modèle de classification. La 1re section est dédiée au pré-traitement des données et détaille le processus de feature engineering ainsi que la sélection des variables. La 2e section présente les différentes étapes de modélisations, du choix des métriques d'évaluation à l'optimisation du meilleur modèle. L'interprétabilité globale et locale du modèle sera l'objet de la 3e section tandis que la 4e section se concentrera sur une analyse du data drift. Enfin, les limites et améliorations possibles du modèle seront discutées.

1 Pré-traitement des données

Le fichier principal *application* contenant les informations personnelles des clients et le détail des crédits est scindé en deux jeux de données :

- un jeu d'entrainement ***application_train*** où la variable cible indiquant si le prêt a été remboursé ou non est disponible (307511 clients) ;
- un jeu de test ***application_test*** où la variable cible est absente (48744 clients).

La distribution de la variable cible dans le jeu d'entrainement est fortement déséquilibrée : 92 % de prêts remboursés et 8 % de prêt non remboursés. Le reste des jeux de données concernent l'historique bancaire de chaque client (crédits passés, crédits actuels, soldes mensuels, etc).

1.1 Nettoyage

Avant de fusionner l'ensemble des différents jeux de données, nous avons effectué pour chacun des jeux :

1. une étude des valeurs manquantes pour chaque variable et la suppression des variables avec un faible taux de remplissage ;
2. une vérification des doublons ;
3. une comparaison des ID clients avec le jeu de données principal *application_train* ;
4. une étude des modalités de chaque variable catégorielle afin de détecter d'éventuelles erreurs ;
5. un calcul du pourcentage occupé par le mode de chaque variable catégorielle et la suppression des variables dont le mode occupe plus de 94 % des valeurs ;
6. une analyse des centiles pour détecter et filtrer d'éventuels outliers.

Afin d'éviter toute fuite de données nous avons concentré l'analyse exploratoire sur le jeu d'entrainement et avons limité l'étude du jeu de test aux valeurs manquantes et à la comparaison des modalités des variables catégorielles.

1.2 Feature engineering basique

Pour améliorer les performances de notre futur modèle de machine learning, nous avons créé pour chaque jeu, à partir des features existantes, de nouvelles features en nous basant sur la compréhension "métier". Par exemple, il nous a semblé intéressant de calculer le ratio entre les revenus mensuels d'un client et les annuités de son crédit ou encore de calculer la durée totale du crédit en divisant la somme empruntée par le montant des annuités.

Lors de l'analyse exploratoire du jeu de données principal, nous avons pu établir que les variables *EXT_SOURCE* (scores clients établis par des organismes bancaires) étaient parmi les plus corrélées à la variable cible. Nous avons donc créé des features faisant intervenir ces variables. Pour les construire, nous avons utilisé les variables les plus corrélées à ces dernières et avons effectué des opérations mathématiques basiques comme le ratio ou la multiplication. Par exemple, nous avons calculé le produit des trois variables *EXT_SOURCE* ou le ratio entre *EXT_SOURCE_1* et l'âge du client.

Nous avons également créé des features lors de l'agrégation des différents jeux en utilisant des fonctions telles que le minimum, le maximum, la moyenne et la somme pour les variables numériques et la moyenne et la somme pour les variables catégorielles. Par exemple, nous avons compté le nombre de crédits antérieurs de chaque client.
Les kernels Kaggle suivants ont été une source d'inspiration pour la réalisation des différents scripts d'agrégations :

1. https://www.kaggle.com/code/jsaguiar/lightgbm-with-simple-features/script
2. https://www.kaggle.com/code/hikmetsezen/base-model-with-0-804-auc-on-home-credit
3. https://www.kaggle.com/code/ogrellier/lighgbm-with-selected-features

Les variables binaires ont été encodées à l'aide d'un LabelEncoder (0 ou 1) tandis que le reste des variables catégorielles ont été traitées avec un encodage one-hot (création d'une variable binaire pour chaque modalité).

1.3 Agrégation et fusion des jeux de données

Après avoir nettoyé et créé des features, nous avons réalisé des agrégations pour chaque jeu de données en utilisant les ID clients afin que chaque ligne corresponde à un client unique. Nous avons ensuite fusionné tous les jeux de données agrégés en un fichier unique, regroupant ainsi toutes les informations disponibles pour chaque client/prêt. À ce stade, le jeu de données unique possède 24 % de valeurs manquantes et 773 variables.

Avant de procéder à l'imputation des valeurs manquantes et à la sélection des variables, nous allons extraire du jeu d'entrainement (80 %) un jeu de validation (20 %) qui servira à évaluer le modèle. Il est très important d'effectuer la séparation des données entrainement/validation avant toute imputation ou sélection des variables pour éviter la fuite de données du jeu de validation vers le jeu d'entrainement. On utilise l'hyperparamètre *stratify* afin de respecter la distribution de la variable cible.

Nous obtenons un jeu d'entrainement de 246003 clients, un jeu de validation de 61501 clients, et un jeu de test de 48744 clients.

1.4 Imputation des valeurs manquantes

Nous avons vu lors de l'analyse exploratoire que les variables *EXT_SOURCE* étaient corrélées à la variable cible et avaient un fort pouvoir informatif (*Information Value*). Nous avons donc employé une technique d'imputation sophistiquée (algorithme *XGBOOST*) pour imputer les valeurs manquantes de ces variables. On utilise les variables initialement présentes dans le jeu de données principal. On impute en premier la variable qui a le taux de remplissage le plus élevé et on termine l'imputation avec celle qui a le taux de remplissage le plus faible. Chaque variable imputée est utilisée pour l'imputation de la variable suivante. Pour éviter toute fuite de données, le modèle *XGBOOST* n'est entrainé que sur le jeu d'entrainement.

Pour le reste des variables, nous avons utilisé l'imputation par la médiane en se basant sur le jeu d'entrainement.

1.5 Feature engineering avancé

Pour finir sur la création de variable, nous avons également calculé pour chaque client la moyenne de la variable cible des 500 clients les plus proches. Pour ce faire, on utilise un modèle KNN entrainé sur les variables *EXT_SOURCE*, et *AMT_MONTHS*. Pour éviter toute fuite de données, le modèle KNN est entrainé uniquement sur le jeu d'entrainement.

1.6 Sélection des variables

Le jeu de données contenant un nombre élevé de variables (773), on effectue une sélection des features afin d'améliorer la qualité, la performance et l'interprétabilité de notre futur modèle de machine learning. L'objectif est de trouver un sous-ensemble de variables pertinentes tout en minimisant la perte d'information. Trois techniques sont employées :

1. la suppression des features corrélées ;
2. la sélection des features avec LightGBM, qui permet de calculer l'importance de chaque variable. Cette méthode permet d'éliminer de façon récursive les variables d'importance nulle mais également de sélectionner les variables qui permettent d'obtenir x % d'importance cumulée ;
3. la sélection des features avec la librairie BorutaPy (modèle *Random Forest*).

Pour éviter le surapprentissage et prévenir toute fuite de données, on utilise ces techniques sur le jeu d'entrainement uniquement. L'impact de la sélection des variables est estimée avec la métrique *ROC AUC* sur le jeu de validation.

On crée trois jeux de données que l'on utilisera pour la sélection du modèle : un jeu réduit de 609 variables où l'on a supprimé les features d'importance nulle (LightGBM) ; un jeu de 294 features où l'on a supprimé les variables corrélées (seuil de 0.9) et les variables d'importance nulle (LightGBM et Boruta) ; un jeu de 151 features qui permet d'expliquer 60 % d'importance cumulée (LightGBM).

2 Modélisation

2.1 Métriques d'évaluation des performances

Pour évaluer et comparer les performances des différents modèles de classification, nous utilisons les métriques suivantes qui permettent de comparer les classes réelles aux classes prédites par chaque modèle :

- **Précision** : La précision mesure la proportion de vrais positifs parmi toutes les instances classées comme positives. Cette métrique est importante lorsque les coûts des faux positifs sont élevés.
- **Rappel** : Le rappel mesure la proportion de vrais positifs parmi toutes les instances réellement positives. Cette métrique est importante lorsque les coûts des faux négatifs sont élevés, comme dans notre cas.
- **Score F1** : Le score F1 est la moyenne harmonique de la précision et du rappel. Il permet de trouver un compromis entre les deux métriques pour évaluer la performance globale du modèle.
- **Score F2** : Le score F2 est une variante du score F qui met davantage l'accent sur le rappel que sur la précision. Il est utile lorsque les faux négatifs sont beaucoup plus coûteux que les faux positifs.
- **AUC ROC** : L'aire sous la courbe (AUC) de la courbe ROC (Receiver Operating Characteristic) mesure la capacité du modèle à discriminer entre les classes positives et négatives, indépendamment du seuil de décision. Une AUC proche de 1 indique un modèle performant, tandis qu'une AUC proche de 0.5 indique un modèle aléatoire.
- **AUC Precision-Recall** : L'AUC de la courbe Precision-Recall mesure la performance du modèle en fonction du compromis entre la précision et le rappel. Cette métrique est particulièrement utile pour les problèmes avec un déséquilibre de classe important.

En plus de ces métriques standards, nous utilisons également un score personnalisé qui prend en compte les coûts spécifiques de notre problème. En particulier, nous mettons l'accent sur le fait qu'un mauvais client prédit bon client (faux négatif) coûte dix fois plus qu'un bon client prédit mauvais (faux positif).

Nous associons les poids suivants aux quatre valeurs possibles de la matrice de confusion (vrai négatif TN, vrai positif TP, faux positif FP, faux négatif FN) :

- p_{TN} = 1 pour TN
- p_{TP} = 1 pour TP
- p_{FP} = -1 pour FP
- p_{FN} = -10 pour FN

Nous définissons la formule suivante pour calculer le score personnalisé :

$$\text{score} = p_{\text{TN}} \cdot \text{TN} + p_{\text{TP}} \cdot \text{TP} + p_{\text{FP}} \cdot \text{FP} + p_{\text{FN}} \cdot \text{FN} \tag{1}$$

Ce score est ensuite normalisé entre 0 et 1. Il est à noter que les scores F1, F2 et le score personnalisé dépendent du seuil de décision (fixé par défaut à 0.5). Pour chacun de ses scores, une optimisation du seuil est donc nécessaire afin de calculer le meilleur score possible.

2.2 Sélection du modèle

Afin de sélectionner le meilleur modèle pour notre problème de classification, nous avons testé plusieurs modèles candidats. Voici la liste des modèles évalués :

- Modèles naïfs :
  - **Baseline_Random** : un modèle qui classe les clients de façon aléatoire.
  - **Baseline_All_Pay_Back** : un modèle qui classe tous les clients comme bons clients.
  - **Baseline_None_Pay_Back** : un modèle qui classe tous les clients comme mauvais clients.
- **Régression logistique** : un modèle linéaire qui estime la probabilité d'appartenance à une classe (les données sont normalisées au préalable).
- **KNN** : un modèle basé sur les k plus proches voisins pour déterminer la classe d'un client (les données sont normalisées au préalable).
- Modèles ensemblistes :
  - **Random Forest**
  - **Extra Trees**
  - **LightGBM**
  - **XGBoost**
  - **CatBoost**

Dans un premier temps, nous n'utilisons aucune technique de rééchantillonnage, mais seulement les méthodes de pondération des classes inhérentes à chaque modèle (méthode indisponible pour KNN). Ces méthodes attribuent des poids différents aux classes lors de l'apprentissage du modèle, de manière à accorder plus d'importance aux instances de la classe minoritaire et à atténuer l'effet du déséquilibre des classes.

La métrique AUC Precision-Recall a été retenue pour choisir le meilleur modèle parmi ceux testés, car elle est particulièrement adaptée aux problèmes de classification avec un déséquilibre de classe important. En cas d'égalité entre plusieurs modèles, le rappel est utilisé comme critère de sélection afin de limiter au maximum le nombre de faux négatifs.

Pour évaluer la performance de chaque modèle, nous avons mis en place une validation croisée à 3 splits. Cette méthode consiste à diviser les données en 3 sous-ensembles distincts et à entraîner et évaluer le modèle sur chaque combinaison de sous-ensembles, ce qui permet d'obtenir une estimation plus fiable de la performance du modèle.

De plus, nous avons testé les trois jeux de données issus du processus de sélection des variables pour déterminer quelle combinaison de variables offre les meilleures performances.

modèle	précision	rappel	score F1	score F2	AUC ROC	AUC Prec-Rec	score métier	Durée (s)
LightGBM	0.187356	0.698459	0.334426	0.451899	0.788966	0.275682	0.748478	4.562774
CatBoost	0.213622	0.596737	0.331205	0.448950	0.783744	0.274768	0.746179	35.543428
Logistic_Regression	0.182130	0.694481	0.326639	0.444427	0.780385	0.259270	0.743449	7.164344
XGBoost	0.200992	0.592758	0.317367	0.431456	0.769228	0.253914	0.738653	28.905607
Extra_Trees	0.571963	0.008561	0.309286	0.424417	0.757507	0.234000	0.734347	12.733225
Random_Forest	0.470003	0.013798	0.302461	0.423829	0.757537	0.224338	0.730907	24.166584
K-Nearest_Neighbors	0.289266	0.053732	0.215064	0.328580	0.624496	0.128605	0.694866	34.699192
Baseline_Random	0.080548	0.499547	0.149386	0.305099	0.500000	0.080723	0.674327	1.750990
Baseline_All_Pay_Back	1.000000	0.000000	0.149386	0.305099	0.500000	0.080723	0.674327	1.731026
Baseline_None_Pay_Back	0.080723	1.000000	0.149386	0.305099	0.500000	0.080723	0.325673	1.747302

FIGURE 1 – Performances moyennes des modèles sur le jeu de données réduit à 151 variables à l'issue de la validation croisée.

Synthèse des résultats

Parmi les trois jeux de données testés lors de la validation croisée, le jeu contenant le nombre de variables le plus faible (151 variables) a permis d'obtenir des scores supérieurs et un temps de calcul réduit par rapport aux autres jeux de données, y compris celui avec 673 variables.

L'amélioration des performances observée en réduisant le nombre de variables peut, à première vue, sembler paradoxale. Cependant, cette amélioration peut s'expliquer par les deux facteurs suivants : premièrement, en réduisant la complexité du modèle, on diminue les risques de surapprentissage et on favorise la capacité du modèle à se généraliser efficacement sur de nouvelles données ; deuxièmement, en éliminant les variables qui sont soit non pertinentes, soit redondantes, le modèle est en mesure de se focaliser sur les caractéristiques ayant un véritable impact sur la prédiction.

La figure 1 présente les résultats obtenus lors de la validation croisée effectuée sur le jeu réduit à 151 variables. Parmi tous les modèles testés, le modèle ensembliste LightGBM a les scores les plus élevés pour toutes les métriques à l'exception de la précision. Il a également le temps de calcul le plus court si l'on ignore les modèles naïfs. Avec ce modèle, 69.8 % des mauvais clients sont détectés et 18.7 % des clients prédits comme mauvais sont réellement mauvais. Outre LightGBM, le modèle ensembliste CatBoost et la régression logistique montrent aussi des scores satisfaisants.

Il est important de mentionner que le score métier que nous avons élaboré ne semble pas être particulièrement pertinent ici. En effet, les différences entre les scores des modèles naïfs et celui du meilleur modèle sont relativement faibles comparées aux écarts observés pour les autres métriques standards. Le score métier en l'état ne permet pas de différencier significativement les performances des modèles. Cela provient probablement du fort déséquilibre de classe.

2.3 Déséquilibre de classe

Nous avons vu précédemment que la distribution de la variable cible était fortement déséquilibrée. En effet, le nombre de clients qui ont remboursé leurs prêts est nettement supérieur au nombre de clients qui ne l'ont pas remboursé. Nous avons utilisé dans la section précédente les méthodes de pondération des classes intégrées à chaque modèle. Nous allons maintenant comparer plusieurs techniques de rééquilibrage des classes pour les trois meilleurs modèles (LightGBM, CatBoost et la régression logistique) :

- **Random OverSampling** : Cette méthode consiste à créer de nouvelles instances de la classe minoritaire en tirant au hasard des exemples existants avec remise. Cela permet d'augmenter la fréquence des exemples de la classe minoritaire dans l'ensemble d'apprentissage ;
- **Random UnderSampling** : Cette méthode consiste à supprimer aléatoirement des instances de la classe majoritaire pour réduire sa fréquence dans l'ensemble d'apprentissage ;
- **SMOTE (Synthetic Minority Over-sampling Technique)** : Cette méthode génère de nouvelles instances de la classe minoritaire en interpolant les caractéristiques des exemples existants. Elle permet d'augmenter la diversité des exemples de la classe minoritaire ;

modèle	Rééchantillonnage	précision	rappel	score F1	score F2	AUC ROC	AUC Prec-Rec	Durée (s)
CatBoost	Random_UnderSampling	0.326995	0.325864	0.341589	0.454823	0.793004	0.280563	19.247930
LightGBM	Random_OverSampling	0.337725	0.296253	0.336114	0.451105	0.789481	0.276943	4.826250
LightGBM	Class_weight	0.187356	0.698459	0.334426	0.451899	0.788966	0.275682	4.275702
LightGBM	Random_UnderSampling	0.319444	0.322238	0.335628	0.451102	0.788782	0.275079	2.961775
CatBoost	Class_weight	0.213622	0.596737	0.331205	0.448950	0.783744	0.274768	30.878731
CatBoost	Random_OverSampling	0.354217	0.240508	0.330526	0.448762	0.785023	0.269805	33.505190
CatBoost	SMOTE_Random_UnderSampling	0.404454	0.156662	0.330538	0.449071	0.786125	0.267579	28.633102
CatBoost	SMOTE	0.472177	0.065415	0.328731	0.447546	0.785436	0.266515	34.096157
Logistic_Regression	SMOTE	0.313211	0.310656	0.327018	0.442235	0.779154	0.261771	7.505686
Logistic_Regression	Random_OverSampling	0.326418	0.286686	0.326965	0.443796	0.779536	0.261001	7.518346
Logistic_Regression	SMOTE_Random_UnderSampling	0.251209	0.467570	0.327213	0.443842	0.779758	0.260976	6.020458
Logistic_Regression	Random_UnderSampling	0.322265	0.291419	0.326901	0.444754	0.780048	0.259976	4.044783
Logistic_Regression	Class_weight	0.182130	0.694481	0.326639	0.444427	0.780385	0.259270	6.893398
LightGBM	SMOTE	0.462675	0.047890	0.327356	0.447925	0.784871	0.258406	5.012723
LightGBM	SMOTE_Random_UnderSampling	0.389731	0.141455	0.327674	0.447120	0.784756	0.257638	4.110957

FIGURE 2 – Performances moyennes des modèles sur le jeu de données réduit à 151 variables pour différentes méthodes de rééchantillonnage. L'entrée *Class weight* correspond à la pondération des classes intégrée à chaque modèle.

- **SMOTE combiné avec Random UnderSampling** : Cette approche utilise l'algorithme SMOTE pour augmenter la fréquence de la classe minoritaire et le Random UnderSampling pour réduire la fréquence de la classe majoritaire.

Pour empêcher toute fuite de données, ces techniques de rééchantillonnage ne sont appliquées qu'au jeu d'entrainement lors de la validation croisée.

Synthèse des résultats

La figure 2 présente les résultats obtenus pour différentes méthodes de rééchantillonnage lors de la validation croisée effectuée sur le jeu réduit à 151 variables. Pour chacun des trois modèles, les méthodes de rééchantillonnage ont permis d'obtenir des scores légèrement plus élevés que les méthodes de pondération des classes et ce pour la plupart des métriques d'évaluation à l'exception du rappel. Par exemple, l'utilisation du Random OverSampling pour le modèle LightGBM a permis d'augmenter l'AUC precision-recall de 0.27567 à 0.2769 mais a diminué le rappel de 0.698 à 0.296. Notre objectif premier étant de détecter le plus de mauvais clients possible, l'utilisation de ces méthodes de rééchantillonnage n'est pas une bonne idée ici. En conséquence, LightGBM couplé à la pondération des classes apparait indiscutablement comme le meilleur choix de modèle pour notre problème de classification car il allie rapidité et performance.

2.4 Optimisation du modèle

Afin d'optimiser les hyperparamètres de notre modèle, nous avons employé la librairie Optuna. Optuna est une bibliothèque spécialisée dans l'optimisation des hyperparamètres, qui automatise la recherche des meilleures combinaisons en ayant recours à des techniques d'optimisation bayésiennes. Ces techniques permettant de guider la recherche vers les zones de l'espace des hyperparamètres qui sont susceptibles d'améliorer les performances du modèle.

Dans le cadre de notre étude, nous avons utilisé Optuna pour optimiser notre modèle LightGBM, en nous basant sur la métrique AUC Precision-Recall. Pour évaluer la performance du modèle avec différentes combinaisons d'hyperparamètres, nous avons réalisé une validation croisée à 3 splits. Nous avons mené deux études successives, comprenant chacune 50 essais. Chaque essai examine une combinaison unique d'hyperparamètres, ces derniers étant ajustés en fonction des résultats des essais précédents.

L'optimisation des hyperparamètres a permis d'augmenter l'AUC Precision-Recall de 0.2756 à 0.2871. Le modèle étant désormais optimisé nous pouvons maintenant évaluer ses performances sur le jeu de validation.

2.5 Évaluation du meilleur modèle

La figure 3 présente les performances du modèle optimisé sur le jeu de validation. Celui-ci parvient à détecter 70 % des mauvais clients, et 19,8 % des clients prédits comme mauvais s'avèrent effectivement l'être.

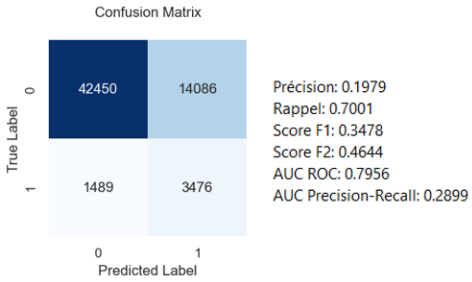

FIGURE 3 – Matrice de confusion et métriques de performance du meilleur modèle évalué sur le jeu de validation.

L'AUC Precision-Recall atteint une valeur de 0,2899. Ces performances surpassent celles observées lors de la validation croisée sur le jeu d'entraînement, mettant en évidence la bonne capacité de généralisation de notre modèle.

Avant d'effectuer les prédictions sur le jeu de test, nous avons ré-entrainé notre modèle en combinant les jeux d'entraînement et de validation. Cette étape permet d'exploiter la totalité des données disponibles pour améliorer la robustesse du modèle et assurer une meilleure généralisation. Après soumission sur la plateforme Kaggle, les prédictions sur le jeu de test ont permis d'obtenir un score (AUC ROC) public de 0.7898 et un score privé de 0.79118.

3 Interprétabilité du modèle

L'interprétation globale et locale de notre modèle a été réalisée à l'aide de la bibliothèque SHAP (SHapley Additive exPlanations) qui permet d'attribuer des valeurs d'importance à chaque variable en se basant sur la théorie des valeurs de Shapley.

La figure 4 (gauche) liste les 20 variables ayant le plus d'importance pour le modèle. On retrouve à la 1re place la variable *TARGET_NEIGHBORS_500_MEAN* que nous avons créé lors du feature engineering avancé, à la 2e place le montant des annuités et à la 3e place la moyenne des scores clients calculés lors du feature engineering basique. De façon générale, on remarque la présence de nombreuses variables liées à ces scores clients, mais également des informations bancaires comme le nombre d'échéances restantes de crédits antérieurs, ce qui semble logique au vue de la problématique. Les informations personnelles du client comme son statut marital ou le fait qu'il ait fait des études supérieures sont également importantes pour le modèle.

La figure 4 (droite) illustre la prédiction de défaillance d'un client et montre la contribution de chaque variable à la prédiction. Le nom de la variable est précédé par la valeur de la variable en question. Sous l'axe des abscisses, la valeur de base (E[f(X)]) est affichée, indiquant la valeur attendue du modèle évalué sur l'ensemble de données de fond. La fonction logit est utilisée pour l'axe des abscisses. Les valeurs SHAP de chaque variable sont additionnées pour correspondre à la sortie du modèle avec toutes les variables incluses. Les valeurs SHAP positives poussent le modèle à prédire un défaut de paiement, tandis que les valeurs SHAP négatives poussent le modèle à prédire un remboursement du crédit.

4 Analyse du Data Drift

L'analyse du data drift a été effectuée avec la librairie Evidently en prenant comme référence le jeu d'entrainement et en considérant que le jeu de test contient les données de nouveaux clients une fois le modèle en production. Le data drift fait référence à un changement dans la distribution des données d'entrée d'un modèle de machine learning par rapport aux données sur lesquelles il a été entrainées. Ce phénomène peut mener à une dégradation des performances et de la précision du modèle. Evidently utilise comme test statistique la distance de Wasserstein lorsque les jeux de données font plus de 10000 observations. Le seuil de drift est fixé à 0.1.

L'analyse du drift sur nos données a montré que 14 des 151 variables présentaient un drift soit 9.2 % des variables. Il est à noter que certaines variables très importantes pour le modèle présentent un drift. C'est le cas, par exemple, de la variable indiquant le montant des annuités. Une investigation approfondie est donc

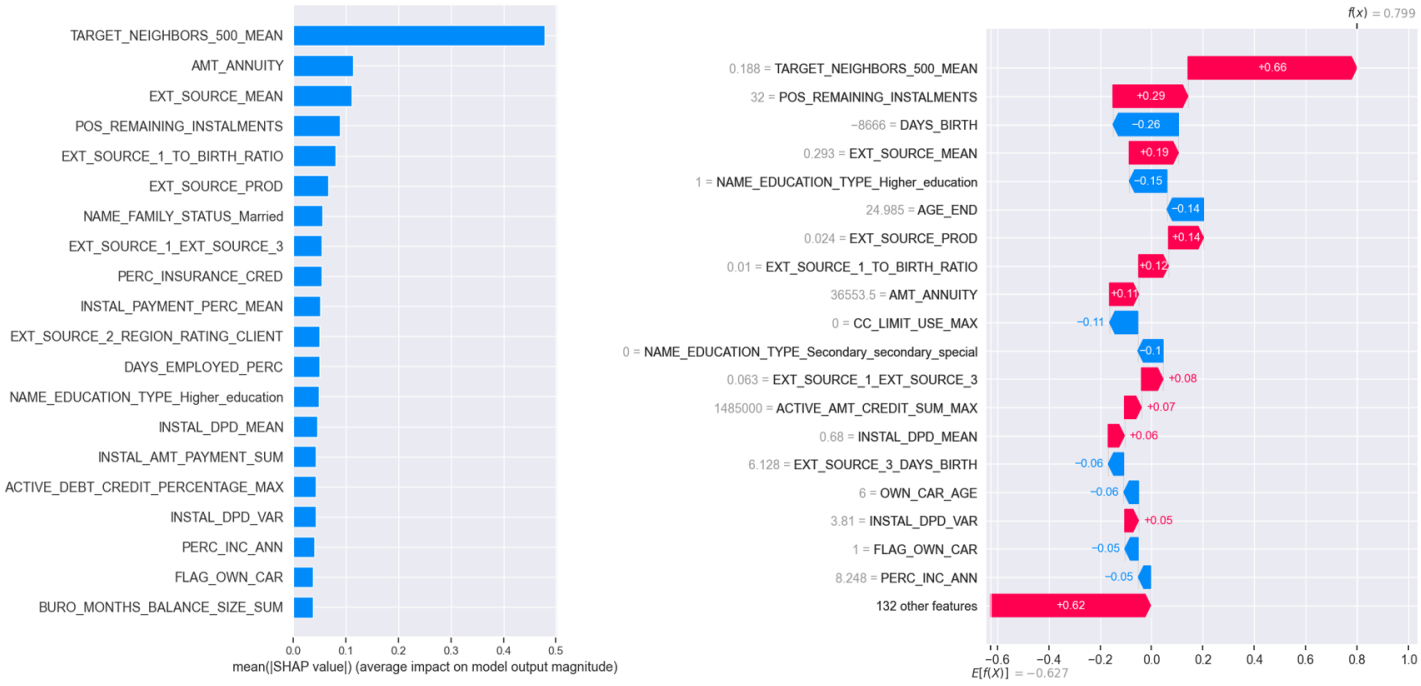

FIGURE 4 – **Gauche : Interprétabilité globale** - Liste des 20 variables ayant le plus d'importance pour le modèle. L'importance des variables est calculée en moyennant la valeur absolue des valeurs de SHAP. **Droite : Interprétabilité locale** - graphique en cascade illustrant la prédiction de défaillance d'un client et montrant la contribution de chaque variable à la prédiction.

nécessaire pour comprendre la nature des changements (temporaires, saisonniers ou permanents) et identifier les causes potentielles de ces dérives.

5 Limites et améliorations possibles

Dans cette étude, certaines limites et améliorations possibles ont été identifiées. Tout d'abord, notre approche s'est concentrée sur la détection du plus grand nombre possible de mauvais clients, dans le but de minimiser les risques et les pertes pour l'entreprise. Cependant, cette stratégie a pour conséquence une diminution de la précision du modèle, ce qui signifie qu'un nombre non négligeable de clients sont classés à tort comme étant à risque alors qu'ils ne le sont pas réellement. Ce point pourrait être discuté avec l'organisme financier.

Une discussion avec des experts métiers permettrait également d'améliorer le score métier pour qu'il tienne compte du déséquilibre des classes. L'intérêt de créer de nouvelles variables pour améliorer le modèle pourrait aussi être discuté.

Finalement, l'interprétabilité du modèle pourrait être grandement améliorée avec une meilleure documentation et communication concernant les variables EXT_SOURCE qui sont très importantes pour le modèle mais dont l'origine reste obscure.