

Projet 7 : Implémentez un modèle de scoring

Note méthodologique

Sommaire

- 1) Méthodologie d'entraînement du modèle
- 2) Traitement du déséquilibre des classes
- 3) Fonction coût métier, algorithme d'optimisation et métrique d'évaluation
- 4) Synthèse des résultats
- 5) Interprétabilité globale et locale du modèle
- 6) Limites et améliorations possibles
- 7) Data drift

1) Méthodologie d'entraînement du modèle

L'objectif est de mettre en œuvre un outil de scoring pour calculer la probabilité qu'un client rembourse son crédit.

Les données sont disponibles à cette adresse : <https://www.kaggle.com/c/home-credit-default-risk/data>

Nous disposons de 10 fichiers CSV contenant les informations des clients.

a) Préparation du jeu de données

Le fichier principal 'application_train' donne les résultats de l'accord ou non du prêt (la colonne 'TARGET'). 0 pour un client sans risque, 1 pour un client à risque.

Après avoir analysé les valeurs manquantes et aberrantes, la première étape est de détacher la 'TARGET' des données et de séparer en un jeu d'entraînement (80%) et un de test (20%) en utilisant train_test_split avec l'option 'stratify' qui permet de garder les proportions d'accords/refus.

b) Encodage

Pour utiliser un modèle de machine learning, il faut des variables numériques, l'utilisation de 'get_dummies' permet de transformer les variables catégorielles en numériques. A noter que pour les variables catégorielles qui n'ont que 2 valeurs possibles, l'encodage est 0 ou 1.

c) Feature engineering

La troisième étape va être de créer des nouvelles variables. Un kernel était à disposition : <https://www.kaggle.com/code/jsaguiar/lightgbm-with-simple-features/script>

Ce code va appliquer plusieurs fonctions d'agrégation (somme, moyenne, maximum ...) sur des variables du jeu de données. Il va aussi remplacer des valeurs aberrantes par une valeur manquante. Certaines valeurs indiquaient par exemple qu'un client était employé depuis plus de 1000 ans !!!

d) Tracking

Différents modèles ont été testés :

- DummyClassifier pour avoir une base-line
- LogisticRegression
- HistGradientBoostingClassifier
- KNeighborsClassifier
- CatBoostClassifier
- XGBClassifier
- LGBMClassifier

La librairie MLFlow a permis d'enregistrer les performances d'un modèle selon ses hyper-paramètres et les métriques choisies. L'utilisation de RandomizedSearchCV permet de tester différents hyper-paramètres, puis GridSearchCV pour affiner le modèle retenu.

e) Entraînement et modèle retenu

Les modèles ont été entraînés sur un échantillon du jeu de données (25%) afin de gagner en temps.

Pour les modèles qui n'acceptent pas les valeurs manquantes, celles-ci ont été remplacées par la médiane de la colonne.

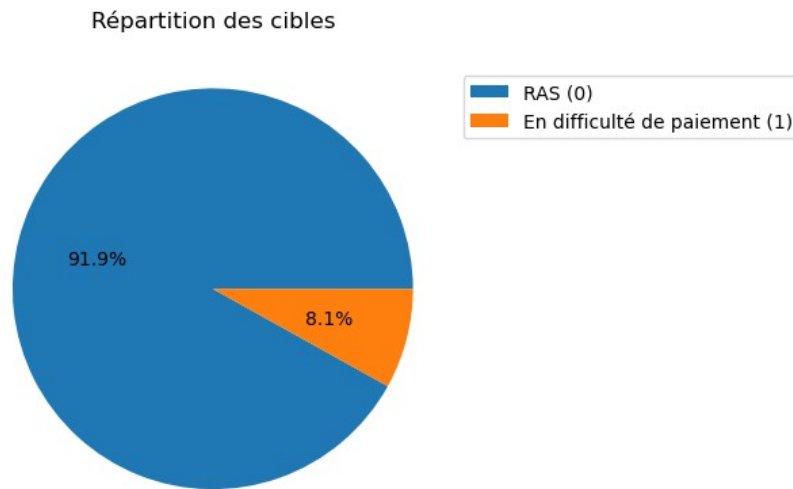
Choix du modèle :

XGBClassifier avec les hyper-paramètres :

- 'scale_pos_weight':10,
- 'max_depth' : 5,
- 'learning_rate' : 0.1,
- 'n_estimators' : 200,
- 'subsample' : 1.0,
- 'colsample_bytree' : 1.0,
- 'gamma' : 0.1,
- 'reg_alpha' : 0,
- 'reg_lambda' : 0.15,
- 'min_child_weight' : 5

2) Traitement du déséquilibre des classes

Les classes (0 ou 1) ne sont pas équilibrées :



La classe 0 est très majoritaire mais il faut aussi prendre en compte le déséquilibre entre les mauvaises prédictions, un 'mauvais' client prédit 'bon' et un 'bon' client prédit 'mauvais' n'ont pas les mêmes conséquences.

L'utilisation de SMOTE permet un oversampling qui augmente les exemples de la classe minoritaire afin d'obtenir autant d'exemples positifs que négatifs. Pour le modèle final, les résultats étaient meilleurs sans l'oversampling.

L'utilisation de l'hyper-paramètre 'class_weights' donne un poids aux classes. Exemple : `class_weights = {0:1, 1:10}` pour `LogisticRegression` donne 10 fois plus d'importance à la classe 1 qu'à la classe 0.

3) Fonction coût métier, algorithme d'optimisation et métrique d'évaluation

Le 'mauvais' client prédit 'bon' aura des conséquences plus importantes qu'un 'bon' client prédit 'mauvais'. Il faut donc prendre en compte de manière différente les faux positifs et les faux négatifs.

Pour entraîner les modèles, l'optimiseur RandomizedSearchCV a besoin d'un score pour comparer les hyper-paramètres. L'utilisation de fbeta_score et de la fonction make_scorer permet de donner plus ou moins d'importance aux faux négatifs ou aux faux positifs.

Formule de fbeta_score :

$$(\beta^2 + 1) \frac{TP}{\beta^2(FN + FP) + (1 - \beta^2)TP}$$

- TP représente les vrais positifs
- FP représente les faux positifs
- TN représente les vrais négatifs
- FN représente les faux négatifs

Si la valeur β est comprise entre 0 ou 1, les faux positifs auront plus d'importance, si elle est supérieure à 1, on privilège les faux négatifs.

Après plusieurs tests, une valeur de β égale à 3 a été retenue.

Pour prendre une décision d'accord ou de refus, le modèle calcule la probabilité que le client soit dans l'une ou l'autre des catégories. Si la probabilité que le client soit sans risque est supérieure à 50%, alors il est prédit sans risque mais cette frontière des 50% est modifiable. Dans cette situation, pour diminuer les faux négatifs, la frontière de décision est passée à 60%.

Pour comparer les modèles, plusieurs métriques d'évaluation ont été prises en compte :

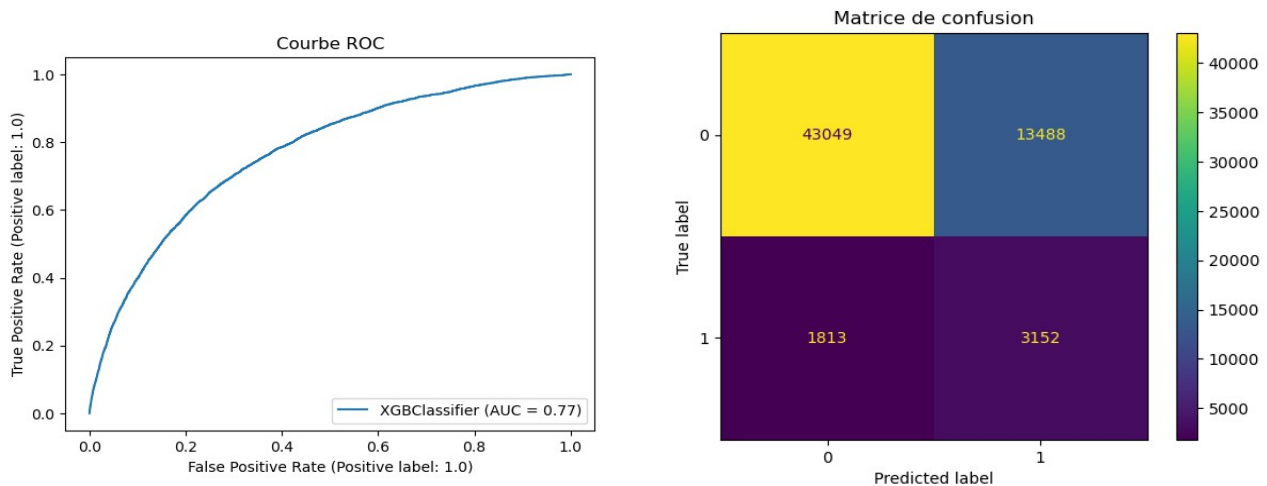
- l'accuracy qui donne le pourcentage de bonnes prédictions
- la précision : $\frac{TP}{TP + FP}$
- le recall : $\frac{TP}{TP + FN}$
- le f1_score : $\frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$
- l'AUC évalue le taux de vrais positifs en fonction du taux de faux positifs
- un score métier : $10 \times \text{fn} + \text{fp}$

4) Synthèse des résultats

Tous les tests ont été enregistrés avec MLFlow afin de choisir le meilleur modèle :

Run Name	Created	accuracy	auc	beta_score \Downarrow	f1_score	precision	recall	score_metier
XGBClassifier_sans_smote	22 hours ago	0.64	0.695	0.546	0.25	0.15	0.77	32826
XGBoost_avec_smote	2 hours ago	0.74	0.692	0.507	0.28	0.18	0.63	32287
CatBoost	2 hours ago	0.73	0.69	0.507	0.28	0.18	0.64	32578
LightGBM_avec_smote	2 hours ago	0.74	0.689	0.504	0.28	0.18	0.63	32547
HistGradientBoosting	3 hours ago	0.74	0.688	0.503	0.28	0.18	0.63	32744
KNeighborsClassifier	2 hours ago	0.86	0.51	0.098	0.1	0.1	0.1	49225
DummyClassifier	22 hours ago	0.92	0.5	0	0	0	0	49650

Le modèle retenu est le XGBClassifier. Après optimisation, voici ses résultats sur le data test :



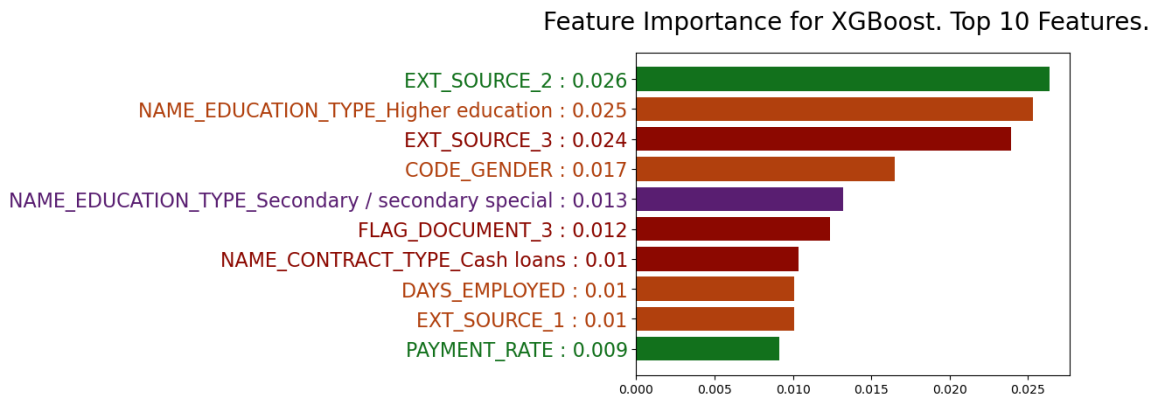
L'aire sous la courbe ROC est de 0,77. Cette valeur suggère que le modèle a une bonne capacité à distinguer les clients à risque.

Ceci est confirmé par la matrice de confusion qui indique un nombre de faux négatifs de 1 813 clients sur 61 502, soit environ 2,9%, pour un taux de vrais positifs de 5,1%.

Le nombre de faux positifs est lui de 13 488, soit environ 22%. Le modèle prédit trop souvent à tort que certains clients ne rembourseront pas leur crédit.

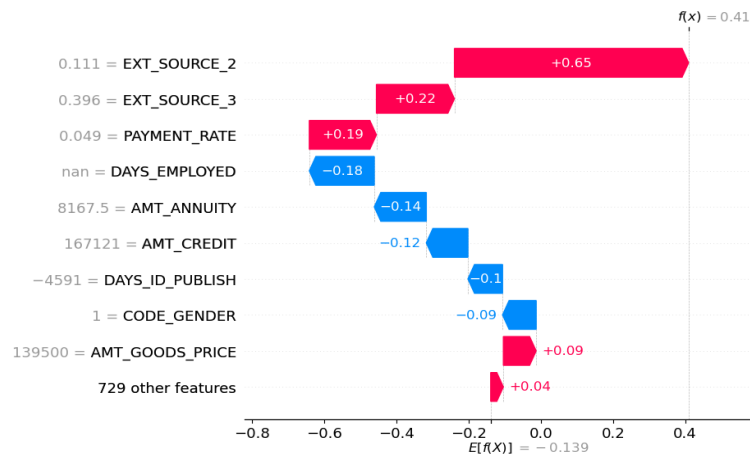
5) Interprétabilité globale et locale du modèle

Il est important de comprendre comment le modèle prend en compte les données, lesquelles ont le plus d'influence. Les valeurs de Shap permettent de quantifier cette influence. Elles permettent de classer les variables de façon globale, quelle donnée agit le plus sur la prédiction. Voici les 10 variables les plus importantes pour le modèle :



On peut se poser la question du refus d'un prêt.
Pourquoi ce client a-t-il été prédit à risque ?
Que faudrait-il changer pour avoir un avis favorable ?

Les valeurs de Shap permettent aussi de donner des informations locales sur les prédictions :



Cette représentation explique comment les valeurs des caractéristiques d'un client ont contribué positivement ou négativement à la prédiction du modèle.

6) Limites et améliorations possibles

La modélisation s'est effectuée sur des métriques personnelles et le taux de faux positifs est de 22%. Il serait intéressant de travailler sur cet aspect en ajoutant d'autres scores métiers.

L'optimisation de l'algorithme pourrait être améliorée en effectuant un fine tuning plus poussé. La recherche d'hyper-paramètres est coûteuse en temps mais peut améliorer sensiblement les performances.










Le dashboard essaye de répondre aux exigences du métier mais il faudrait un retour utilisateur pour répondre au mieux à la demande.

7) Data drift

L'entraînement et les prédictions du modèle ne se font évidemment pas sur les mêmes jeux de données. Mais avec le temps, les caractéristiques des clients risquent de changer (évolution du comportement des clients, changement de l'environnement, erreurs de collecte ...), et si cet écart devient trop important avec les valeurs d'entraînement, le modèle va perdre en performance.

La librairie Evidently permet de comparer deux data sets et de détecter une dérive de données. L'analyse a été faite sur un échantillon de 5 000 individus par jeu :

Drift is detected for 7.5% of columns (9 out of 120).

Q Search X						
Column	Type	Reference Distribution	Current Distribution	Data Drift	Stat Test	Drift Score
> DAYS_EMPLOYED	num			Detected	Wasserstein distance (normed)	30.680109
> AMT_REQ_CREDIT_BUREAU_QRT	num			Detected	Wasserstein distance (normed)	0.431269
> AMT_REQ_CREDIT_BUREAU_MON	num			Detected	Wasserstein distance (normed)	0.275444
> AMT_GOODS_PRICE	num			Detected	Wasserstein distance (normed)	0.230258
> AMT_CREDIT	num			Detected	Wasserstein distance (normed)	0.226241
> NAME_CONTRACT_TYPE	cat			Detected	Jensen-Shannon distance	0.152407
> AMT_ANNUITY	num			Detected	Wasserstein distance (normed)	0.131807
> FLAG_EMAIL	num			Detected	Jensen-Shannon distance	0.116003

Le rapport montre que le data drift n'a été détecté que pour 9 colonnes sur 120, soit 7,5%.