Note Méthodologique

projet 7 “Implémentez un modèle de scoring”

Parcours Data scientist

Islem HABIBI

Janvier 2024

1. Introduction

Ce rapport détaille le septième projet du parcours de Data Scientist intitulé "Implémentation d'un modèle de scoring". L'objectif principal de ce projet est de développer un outil prédictif capable d'évaluer le risque de défaillance d'un client quant au remboursement de sa dette. Cette démarche est essentielle pour l'entreprise "Prêt à dépenser", qui cherche à optimiser sa gestion des risques financiers en mettant en place un modèle de scoring fiable et efficace.

2. Traitement des données

2.1. Préparation des données

Pour l'entraînement du modèle on a utilisé les deux bases de données “application_train et “application_test”. Ces deux bases ont été fusionnées dans un premier lieu pour faciliter leur nettoyage. une variable “TARGET” dont les valeurs sont des NaaNs pour avoir un équilibre entre les deux bases lors de la fusion. La base de données “HomeCredit_columns_description” a été utilisée pour comprendre la signification de chaque variable, leur importance pour le modèle, évaluer les possibilités de feature engineering et finalement éliminer les variables non nécessaires pour la création de modèle.

Sélection des features

Les variables dont la pertinence est discutable ont été qui ont été supprimer de notre jeux de données sont:

- REGION_POPULATION_RELATIVE
- OWN_CAR_AGE
- FLAG_EMP_PHONE
- FLAG_WORK_PHONE, FLAG_PHONE
- REGION_RATING_CLIENT, REGION_RATING_CLIENT_W_CITY
- HOUR_APPR_PROCESS_START
- REG_REGION_NOT_WORK_REGION, LIVE_REGION_NOT_WORK_REGION, REG_CITY_NOT_WORK_CITY, LIVE_CITY_NOT_WORK_CITY
- REG_REGION_NOT_LIVE_REGION, REG_CITY_NOT_LIVE_CITY
- EXT_SOURCE_1, EXT_SOURCE_2, EXT_SOURCE_3
- DAYS_LAST_PHONE_CHANGE
- WEEKDAY_APPR_PROCESS_START

et aussi certaines variables AVG et toutes les variables mode et médiane des caractéristiques des bâtiments occupés par l'emprunteur.
Les deux variables FLAG_DOCUMENT (2-21) et AMT_REQ_CREDIT_BUREAU_(HOUR, MON, ...) ont été remplacées par deux variables plus généralistes : "provided_flag_documents", représentant ainsi la somme des documents renseignés, et "AMT_REQ_CREDIT_BUREAU_SUM", qui représente la somme des demandes de renseignements avant la demande de crédit, respectivement.

Nettoyage des données

- Les variables (DAYS_BIRTH, DAYS_EMPLOYED, DAYS_REGISTRATION et DAYS_ID_PUBLISH) présentaient exclusivement des valeurs négatives. Cependant, ces variables sont nécessaires pour l'entraînement de notre modèle. Ainsi, toutes ces valeurs ont été transformées en valeurs absolues pour la suite du projet.
- Les outliers pour les variables CODE_GENDER, NAME_FAMILY_STATUS et NAME_INCOME_TYPE ont été éliminés de notre jeu de données. L'analyse de la corrélation des variables entre elles a également permis l'élimination d'une de ces variables dans chaque cas.
- L'analyse de la corrélation des variables avec la target n'a pas abouti à l'élimination d'aucune variable.

Après toutes ces opérations, notre jeu de données global va être divisé en deux ensembles comme au début : un ensemble d'entraînement (correspondant au application_train initialement défini) et un ensemble de test (correspondant au application_test initialement défini). Le jeu de test sera uniquement utilisé ultérieurement pour tester notre modèle en dashboard. Pour le reste du projet, nous n'utiliserons que le jeu d'entraînement.

Encodage et Normalisation

L'encodage et la normalisation des variables sont pris en charge par la fonction data_processing. Dans un premier lieu, la fonction encode les variables ayant 2 types de catégories avec la librairie LabelEncoder. Ensuite, elle encode le reste des variables catégorielles avec OneHotEncoder, et enfin, toutes les variables sont normalisées avec MinMaxScaler. Cette fonction traite également les données de test et sera utilisée avec le dashboard.

2.2. Séparation des données

Maintien de la proportion des classes dans chaque sous-ensemble.

Afin de réduire le temps de compilation, j'ai extrait un échantillon de 1500 individus de chaque classe cible. Ainsi, dans cet échantillon, un équilibre a été maintenu entre les deux classes. Cette équité nous permettra d'éviter tout déséquilibre de classification, tout en assurant une distribution équilibrée des classes dans chaque sous-ensemble. Par conséquent, l'évaluation des performances du modèle sera plus fiable, garantissant qu'aucun biais excessif ne favorisera une classe par rapport à une autre lors de l'ajustement du modèle. et finalement notre échantillon sera divisé en ensembles d'entraînement, de validation.

3. Entraînement et validation

3.1. Cross-validation et Optimisation

Dans notre projet, nous sommes confrontés à un problème de classification. Pour le résoudre, nous envisageons d'utiliser différents algorithmes de classification et de sélectionner le modèle qui offre les meilleures performances. Les modèles que nous avons pris en compte pour cette tâche sont la régression logistique (`LogisticRegression`), XGBoost (`XGBClassifier`), les K plus proches voisins (`KNeighborsClassifier`) et l'arbre de décision (`DecisionTreeClassifier`). Pour chacun de ces modèles, nous avons préparé un dictionnaire `param_grids` qui contient les hyperparamètres que nous souhaitons tester.

Afin de trouver la meilleure combinaison d'hyperparamètres pour ces modèles de classification, nous avons utilisé la fonction `GridSearchCV` de la bibliothèque scikit-learn pour effectuer une recherche exhaustive. Cette fonction nous permet de tester différentes combinaisons d'hyperparamètres pour chaque modèle et d'évaluer leur performance en utilisant diverses métriques de scoring.

Pour chaque modèle et chaque métrique de scoring, une recherche par grille (grid search) est exécutée avec une validation croisée. Après l'entraînement, les meilleurs hyperparamètres, le meilleur estimateur et le meilleur score sont enregistrés dans un DataFrame pour chaque méthode de scoring.

Tableau 1: Résultat de la gridsearch et cross validation. ce tableau contient le meilleur hyperparamètre pour chaque méthode de scoring pour chaque algorithme de classification

	Model	Scoring Method	Best Parameters	Best Estimator	Best Score
0	Logistic Regression	Accuracy	{'C': 10, 'max_iter': 200, 'penalty': 'l2'}	LogisticRegression(C=10, max_iter=200)	0.589583
1	Logistic Regression	Precision	{'C': 10, 'max_iter': 200, 'penalty': 'l2'}	LogisticRegression(C=10, max_iter=200)	0.592035
2	Logistic Regression	Recall	{'C': 175, 'max_iter': 200, 'penalty': 'l2'}	LogisticRegression(C=175, max_iter=200)	0.613359
3	Logistic Regression	F1	{'C': 175, 'max_iter': 200, 'penalty': 'l2'}	LogisticRegression(C=175, max_iter=200)	0.600706
4	Logistic Regression	ROC AUC	{'C': 10, 'max_iter': 200, 'penalty': 'l2'}	LogisticRegression(C=10, max_iter=200)	0.589402
5	XGBoost	Accuracy	{'eta': 0.2, 'learning_rate': 0.1, 'max_depth'...	XGBClassifier(base_score=None, booster=None, c...	0.613333
6	XGBoost	Precision	{'eta': 0.2, 'learning_rate': 0.01, 'max_depth...	XGBClassifier(base_score=None, booster=None, c...	0.792308
7	XGBoost	Recall	{'eta': 0.2, 'learning_rate': 0.2, 'max_depth'...	XGBClassifier(base_score=None, booster=None, c...	0.645529
8	XGBoost	F1	{'eta': 0.2, 'learning_rate': 0.2, 'max_depth'...	XGBClassifier(base_score=None, booster=None, c...	0.627743
9	XGBoost	ROC AUC	{'eta': 0.2, 'learning_rate': 0.1, 'max_depth'...	XGBClassifier(base_score=None, booster=None, c...	0.613075
10	K-Neighbors	Accuracy	{'algorithm': 'auto', 'n_neighbors': 8, 'weigh...	KNeighborsClassifier(n_neighbors=8, weights='d...	0.558750
11	K-Neighbors	Precision	{'algorithm': 'auto', 'n_neighbors': 2, 'weigh...	KNeighborsClassifier(n_neighbors=2)	0.592859
12	K-Neighbors	Recall	{'algorithm': 'auto', 'n_neighbors': 9, 'weigh...	KNeighborsClassifier(n_neighbors=9)	0.608482
13	K-Neighbors	F1	{'algorithm': 'auto', 'n_neighbors': 8, 'weigh...	KNeighborsClassifier(n_neighbors=8, weights='d...	0.581713
14	K-Neighbors	ROC AUC	{'algorithm': 'auto', 'n_neighbors': 8, 'weigh...	KNeighborsClassifier(n_neighbors=8, weights='d...	0.558253
15	Decision Tree	Accuracy	{'criterion': 'gini', 'max_depth': 10, 'max_fe...	DecisionTreeClassifier(max_depth=10, max_featu...	0.585833
16	Decision Tree	Precision	{'criterion': 'gini', 'max_depth': 3, 'max_fea...	DecisionTreeClassifier(max_depth=3, max_featur...	0.608544
17	Decision Tree	Recall	{'criterion': 'gini', 'max_depth': 3, 'max_fea...	DecisionTreeClassifier(max_depth=3, max_featur...	0.867184
18	Decision Tree	F1	{'criterion': 'gini', 'max_depth': 3, 'max_fea...	DecisionTreeClassifier(max_depth=3, max_featur...	0.649018
19	Decision Tree	ROC AUC	{'criterion': 'gini', 'max_depth': 5, 'max_fea...	DecisionTreeClassifier(max_depth=5, max_featur...	0.582096

3.2. Testing des modèles

Après avoir effectué la recherche des hyperparamètres et généré un DataFrame, j'ai créé une fonction `Model_testing` qui, pour chaque algorithme de machine learning, va tester tous les estimateurs pour chaque méthode de scoring et générer les métriques suivantes : recall (sensibilité), spécificité, précision, score F1, exactitude (accuracy), score AUC de la courbe ROC, score métier, vrais négatifs (TN), faux positifs (FP), faux négatifs (FN) et vrais positifs (TP) afin de pouvoir les comparer entre eux avec la fonction `best_param`. Finalement, cela permet de définir le meilleur hyperparamètre pour chaque modèle de classification.

Le meilleur hyperparamètre sera choisi selon :

1. Le score métier
2. Le recall (sensibilité) (True Positive / (True Positive + False Negative))
3. La spécificité (True Negative / (True Negative + False Positive))
4. Le ROC AUC Score
5. L'accuracy (exactitude) ((True Positive + True Negative) / (Total samples))
6. Le score de précision

La métrique "score métier" est une métrique que j'ai créée et qui pénalise davantage les faux négatifs par rapport aux faux positifs. Cette métrique est calculée avec la fonction `score_metier`.

3.3. Optimisation des modèles de classification

Pour optimiser le seuil de classification, nous avons testé différents seuils pour le meilleur hyperparamètre de chaque modèle de classification. À chaque itération, nous avons calculé les métriques suivantes : recall (sensibilité), spécificité, précision, score F1, exactitude (accuracy), score AUC de la courbe ROC, score métier, vrais négatifs (TN), faux positifs (FP), faux négatifs (FN) et vrais positifs (TP). Le meilleur seuil sera celui qui présente le meilleur score AUC de la courbe ROC et sera identifié à l'aide de la fonction `best_threshold`.

3.4. Sélection du modèle final

Une fois le meilleur hyperparamètre de chaque modèle identifié, ainsi que le seuil optimal, la fonction de scoring va évaluer les performances des modèles et calculer les métriques suivantes : rappel (sensibilité), précision, score F1, exactitude (accuracy), score AUC de la courbe ROC, ainsi que le score métier. Elle va également générer la courbe ROC AUC et la matrice de confusion.

Le meilleur modèle final de classification est celui qui présente la meilleure exactitude (accuracy)

Tableau 2: Performances des différents modèles testés avec les métriques d'évaluation, les paramètres optimaux et les scores métier.

	Model	Precision	Recall	F1	Accuracy	ROC AUC Score	Score Metier
0	Logistic Regression	0.56	0.58	0.57	0.58	0.58	1330
1	XGBoost	0.61	0.65	0.63	0.63	0.63	1122
2	K-Neighbors	0.54	0.34	0.42	0.55	0.54	1981
3	Decision Tree	0.48	1.00	0.65	0.48	0.50	323

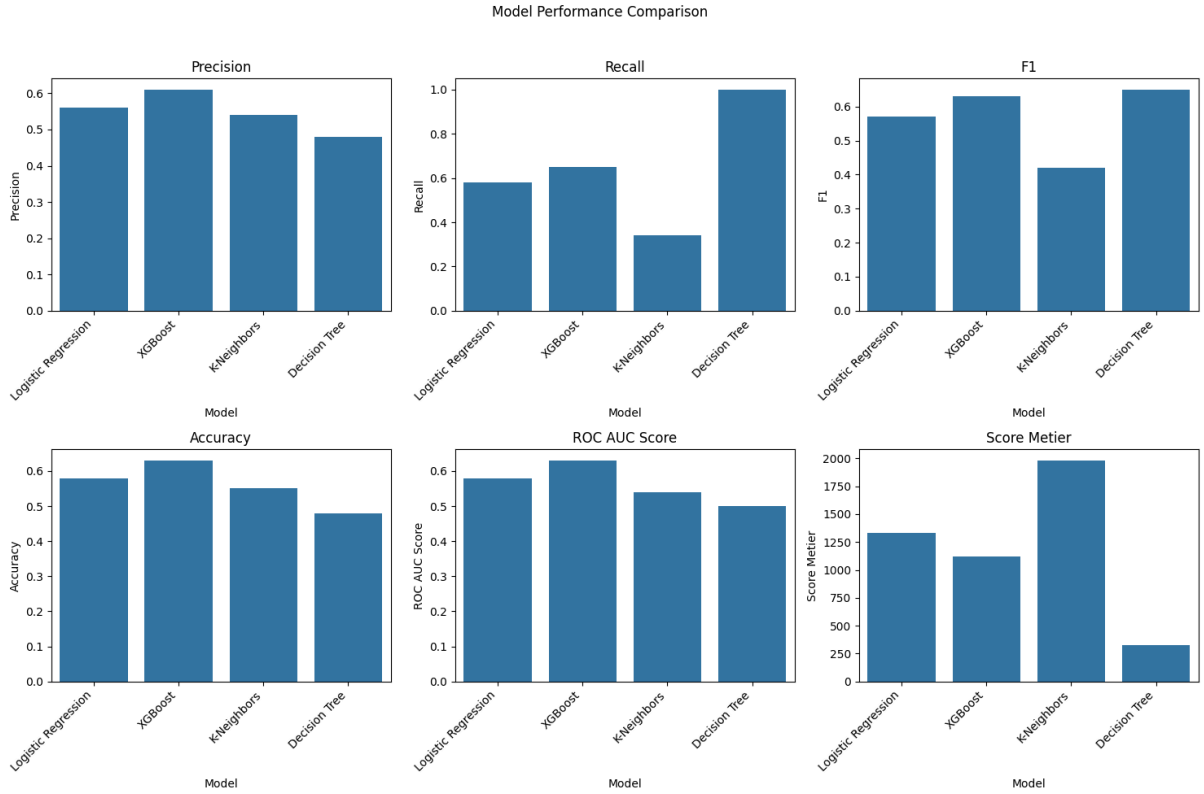

Figure 1: Performances des différents modèles testés avec les métriques d'évaluation, les paramètres optimaux et les scores métier.

Tous les algorithmes de classification testés avec notre jeu de données présentaient des performances comparables et proches pour la métrique de précision. Le meilleur algorithme de classification pour notre jeu de données est XGBoost. Les performances de ce modèle sont présentées ci-dessous. Ce modèle avait la meilleure accuracy et le meilleur score ROC AUC parmi tous les algorithmes testés. Ce modèle sera considéré pour la suite du projet et pour la création de l'API.

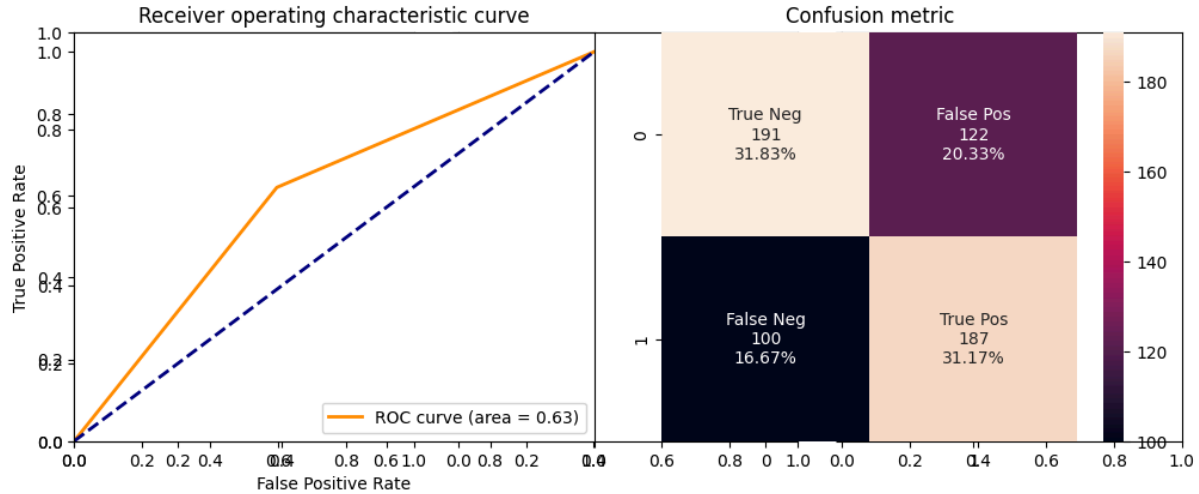

Figure 2 : Courbe ROC et matrice de confusion du meilleur modèle de classification

Tableau 3 : Les métriques de performance du meilleur modèle de classification

Precision	Recall	F1	Accuracy	ROC AUC Score	Score Metier
0.61	0.65	0.63	0.63	0.63	1122

4. Analyse du Data Drift

Le data drift de notre dataset a été analysé par la bibliothèque Evidently pour comparer les distributions des caractéristiques entre les données d'entraînement et de production. Cette analyse a détecté un drift dans 28,571 % des colonnes, c'est-à-dire 8 colonnes parmi 28.

5. Déploiement

Le modèle final a été enregistré à l'aide de MLflow et au format Pickle. Par la suite, ce format Pickle a été déployé sur le cloud via Heroku sous la forme d'une API Flask (Fig. 3). En plus de la fonction de prédiction, cette API fournit également la valeur de Shapley, ainsi qu'un graphique montrant les variables à améliorer pour affiner la prédiction, en cas de forte probabilité que le client rencontre des difficultés de paiement. Afin de faciliter l'interaction utilisateur avec l'API, deux fichiers, "index.html" et "home.html", ont été créés. Ces fichiers permettent aux utilisateurs d'interagir avec l'API, facilitant ainsi l'accès aux prédictions et aux informations visuelles sur les facteurs influents.

L'API est disponible à l'adresse suivante : https://p7-api-40a72eb22034.herokuapp.com/

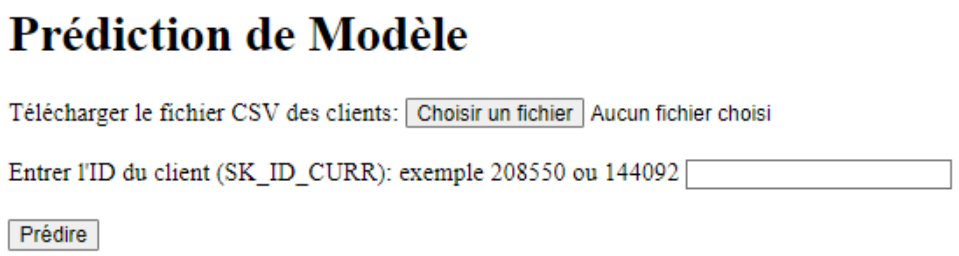

Figure 3 : Interface de L'API

6. Spécifications du Dashboard Interactif

Le dashboard (Fig. 4) dédié à l'utilisation de l'API de prédiction de crédit a été élaboré en utilisant la bibliothèque Streamlit. Cette plateforme offre aux utilisateurs la possibilité d'effectuer des prédictions sur des données de test préalablement traitées, encodées, et normalisées, en se basant uniquement sur leur identifiant (ID_SK_CURR). Le dashboard

permet non seulement d'observer les revenus du client parmi l'ensemble des revenus de notre clientèle, mais également de visualiser les résultats de la prédiction sous la forme d'une jauge, créée avec Plotly. Si la prédiction suggère un risque élevé de difficultés de paiement pour le client, indiquant que l'octroi du prêt ne serait pas recommandé, un graphique SHAP s'affiche. Ce graphique met en lumière les variables ayant un impact positif ou négatif sur la décision, offrant ainsi une explication détaillée des facteurs influençant la prédiction. Cette approche transparente vise à fournir aux utilisateurs une compréhension approfondie des raisons derrière chaque prédiction, facilitant ainsi des décisions éclairées en matière d'octroi de crédit. Le dashboard a été déployé sur le cloud via Heroku.
Le dashboard est disponible à l'adresse suivante : https://p7-dashboard-aef5610371fe.herokuapp.com/

Figure 3 : Le dashboard interactif de l'API créé avec Streamlit.

7. Conclusion

Le projet "Implémentation d'un modèle de scoring" a atteint son objectif principal en développant un outil prédictif robuste pour évaluer le méthodique pour l'encodage et la normalisation, nous avons pu construire un ensemble de données cohérent et pertinent pour l'entraînement de modèles de classification.

L'application de techniques avancées de cross-validation et d'optimisation des hyperparamètres a permis de sélectionner le modèle XGBoost comme le plus performant, avec une excellente accuracy et un score ROC AUC supérieur. Ce modèle a été optimisé davantage en ajustant le seuil de classification pour équilibrer la sensibilité et la spécificité, garantissant ainsi une prédiction fiable et équilibrée.

L'analyse du data drift a révélé des changements dans les distributions des caractéristiques, soulignant l'importance d'une surveillance continue pour maintenir la performance du modèle dans le temps. Le déploiement réussi du modèle via une API Flask et l'intégration avec un dashboard interactif Streamlit offrent à l'entreprise un outil puissant pour la prise de décision en matière de crédit, avec la possibilité d'interpréter les prédictions grâce aux valeurs de Shapley et aux graphiques SHAP.

En conclusion, ce projet a non seulement fourni à "Prêt à dépenser" un système de scoring efficace pour évaluer le risque de crédit, mais a également établi une base solide pour des améliorations futures et une adaptation continue aux évolutions du marché. La mise en œuvre de ce modèle de scoring devrait conduire à une meilleure gestion des risques financiers et à une optimisation des décisions de prêt, contribuant ainsi à la stabilité et à la rentabilité de l'entreprise à long terme.