Projet N°7: Conception et déploiement d'un modèle de scoring

Note Méthodologique

Soumaya Khila

OpenClassrooms – Formation Data Scientist

Table of Contents

Avant Propos

Ce projet concerne une société financière qui propose des crédits à la consommation pour des personnes ayant peu ou pas du tout d'historique de prêt.

L’entreprise souhaite développer un modèle de scoring de la probabilité de défaut de paiement du client pour étayer la décision d'accorder ou non un prêt à un client potentiel en s’appuyant sur des sources de données variées (données comportementales, données provenant d'autres institutions financières, etc.).

De plus, les chargés de relation client ont fait remonter le fait que les clients sont de plus en plus demandeurs de transparence vis-à-vis des décisions d’octroi de crédit. Cette demande de transparence des clients va tout à fait dans le sens des valeurs que l’entreprise veut incarner.

Elle décide donc de développer un dashboard interactif pour que les chargés de relation client puissent à la fois expliquer de façon la plus transparente possible les décisions d’octroi de crédit, mais également permettre à leurs clients de disposer de leurs informations personnelles et de les explorer facilement. Mission

- Construire un modèle de scoring qui donnera une prédiction sur la probabilité de faillite d'un client de façon automatique.
- Construire un dashboard interactif à destination des gestionnaires de la relation client permettant d'interpréter les prédictions faites par le modèle et d’améliorer la connaissance client des chargés de relation client.

Le manager incite à sélectionner un kernel Kaggle pour faciliter la préparation des données nécessaires à l’élaboration du modèle de scoring. L'idée est d'analyser ce kernel pour l'adapter aux besoins de la mission.

Introduction

Le Machine Learning est une science moderne permettant de découvrir des répétitions (des patterns) dans un ou plusieurs flux de données et d'en tirer des prédictions en se basant sur des statistiques. En clair, le Machine Learning se base sur le forage de données, permettant la reconnaissance de patterns pour fournir des analyses prédictives.

Notre projet de Machine Learning comprend les étapes suivantes :

- Prétraitement des données
- Modélisation
- Choix d'une fonction coût et d'une métrique d'évaluation
- Entrainement du modèle
- Prédiction sur des données non apprises

Dans ce document, nous allons expliquer comment sont déployées ces différentes étapes.

1. Prétraitement des données

L'entrainement du modèle doit être fait impérativement après avoir prétraiter les données. Une fois c'est fait, il faut choisir le modèle approprié et l'entrainer sur les données nettoyées.

Afin de faciliter cette tache, nous nous sommes inspirés de deux kernel Kaggle. Ci-dessous les liens:

- https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction
- https://www.kaggle.com/danilz/merge-all-data-base-glm-vs-xgb-explained-0-763

Le prétraitement des données nécessite, en premier lieu, de charger les données qui consistent à 8 fichiers ".csv". Nous avons mis l'accent sur les deux fichiers "application_train.csv" et "application_test.csv". C'est principalement ces deux fichiers qui contiennet les features dont on a besoin et la variable "TARGET" qui fournit la décision prise par rapport à l'acceptation ou le refus de la demande du prêt.

La deuxième étape c'est la gestion des outliers. Nous avons remarqué, par exemple, la présence des valeurs négatives dans la variable 'DAYS_BIRTH'. Ceci est parce que ces valeurs sont enregistrées par rapport à la demande de prêt en cours. . Nous les avons rectifiées (cellule 24 du notebook). D'autres valeurs aberrantes ont été aussi détectées sur la variable 'DAYS_EMPLOYED' . Nous les avons remplacé par NaN.

La troisième étape c'est le feature engineering. Nous avons créé 5 nouvelles variables métiers:

- **CREDIT_INCOME_PERCENT**: Pourcentage du montant du crédit par rapport au revenu d'un client
- **ANNUITY_INCOME_PERCENT**: Pourcentage de la rente de prêt par rapport au revenu d'un client
- **CREDIT_TERM**: Durée du paiement en mois
- **DAYS_EMPLOYED_PERCENT**: Pourcentage des jours employés par rapport à l'âge du client

- **INCOME_PER_PERSON**: pourcentage des revenues des clients par rapport aux membres de la famille.

La quatrième étape est l'encodage des variables catégorielles. Pour ce faire, nous avons utilisé la fonction "pd.get_dummies()". Une fois de nouvelles variables sont créées (correspondant chacune à une valeur unique de la variable catégorielle), nous avons aligné les deux dataframes, test et entrainement, tout en supprimant les colonnes différentes.

La cinquième étape est l'imputation des NaN. Nous avons utilisé la méthode la plus simple qui est l'imputation par la médiane.

La sixième étape est la standardisation des données. Nous avons utilisé la technique "Robust_scaler". Cette technique se base sur le même principe de mise à l'échelle que MinMaxScaler(). Néanmoins, elle utilise l'intervalle interquartile au lieu du min-max, ce qui la rend plus fiable vis à vis des outliers.

Une fois nos données sont nettoyées et prétraitées. Nous les avons sauvegardé dans des fichiers ".csv".

2. La méthodologie d'entraînement du modèle

Il existe différents modèles de ML plus ou moins performants et plus ou moins complexes, donc plus ou moins interprétables. Parmi ceux communément utilisés, nous avons sélectionnés les suivantes : Dummy, Random Forest, XGBoost et LightGBM.

Afin d'avoir une première idée des performances possibles, la modélisation par DummyClassifier permettra d'obtenir une baseline. Nous nous basons sur les résultats trouvés comme base de comparaison.

Pour les autres modèles, nous avons cherché à déterminer la meilleure combinaison d'hyper-paramètres suivant la fonction coût déterminée par le problème.

<u>Démarche de modélisation</u>

La première étape a consisté à gonfler artificiellement le jeu de données pour assurer un équilibre dans les classes de la variable TARGET. En effet, environ 90% des prêts existants dans le jeu de données d'entrainement ont été remboursés. Nous avons donc utilisé pour ce faire la librairie SMOTE .

Pour déterminer quelle combinaison d'hyper-paramètres donne les meilleurs résultats sur notre jeu de données, l'algorithme d'optimisation « hyperopt » a été utilisé.
L'algorithme « hyperopt » utilise une approche Bayésienne pour déterminer les meilleurs paramètres d'une fonction. A chaque étape, il essaye de construire un model probabiliste de la fonction et choisi les paramètres les plus promettant pour la prochaine étape. Couplé à l'algorithme d'optimisation « hyperopt », un système de validation croisée a été mis en place. Cette technique permet d'éviter le sur-apprentissage (overfitting).

l'efficacité du modèle est déterminée en observant l'aire sous la courbe AUC (ou Area Under the Curve). Ainsi, le modèle le plus efficace a une AUC égale à 1, et le modèle le moins efficace a une AUC égale à 0,5. Le F-score est aussi utilisé comme métrique d'évaluation.

Après avoir entrainer nos modèles, nous avons les conclusions suivantes:

- Baseline : DummyClassifier --> F-score = 0.5 , AUC= 0.5
- Random Forest --> F-score = 0.86 , AUC= 0.67
- XGBoost --> F-score = 0.92 , AUC= 0.748
- LightGBM --> F-score = 0.92 , AUC= 0.759

--> les meilleurs résultats sont donnés par XGBoost et LightGBM. Pour faire le choix entre les deux, nous pouvons comparer leur temps d'exécution. On mettra comme hypothèse que les ressources mémoire et processeur peuvent être beaucoup plus performants que ceux utilisées pour ce projet. Et donc on ne va pas considérer le critère de temps. Nous allons étudier, par contre, les pourcentages de faux-positifs et des faux-négatifs donnés par chaque algorithme:

- XGBoost: Faux-positifs: 0.49 % , Faux-négatifs: 7.65 %
- LightGBM: Faux-positifs: 0.25 % , Faux-négatifs: 7.81 %

--> XGBoost génère moins de faux négatifs que LightGBM. Par contre, LightGBM donne moins que Faux-positifs que XGBoost.

Le Faux-négatif est plus important à éviter puisqu'il signifie qu'un prêt est donné pour un client qui ne peut pas le rembourser, donc la perte touche la capitale de la banque. Par contre, le Faux-positif signifie qu'un client qui pourra rembourser son prêt a eu un refus de la part de la banque, donc la perte touche la marge. --> le faux négatif est plus dangereux et donc nous allons choisir le modèle qui résulte moins de faux-négatif.

--> Nous choisissons donc XGBoost avec les hyperparamètres: XGBClassifier(colsample_bytree= 0.36, gamma= 0.18 , max_depth= 8)

3. Choix d'une fonction coût et d'une métrique d'évaluation

La problématique « métier » est de prendre en compte qu'un faux positifs (bon client considéré comme mauvais = crédit non accordé à tort, donc manque à gagner de la marge pour la banque) n'a pas le même coût qu'un faux négatif (mauvais client à qui on accorde un prêt, donc perte sur le capital non remboursé). Par manque d'expertise métier, nous mettons comme hypothèse q'un faux négatif est environ 10 fois plus coûteux qu'un faux positif. Les mesures techniques tels que le f1 score ne le prennent pas en compte. Nous proposons donc cette fonction pour calculer le nouveau fscore à minimiser:

fscore = poids*fn +fp

avec:

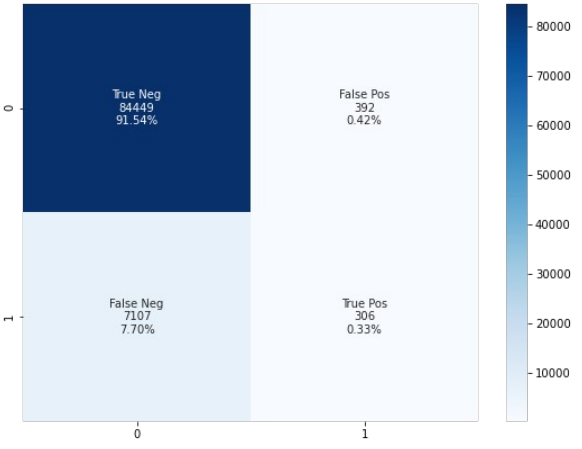
- fp: faux positif
- fn: faux négatif
- poids: importance de fn par rapport à fp. Nous avons choisi poids = 10.

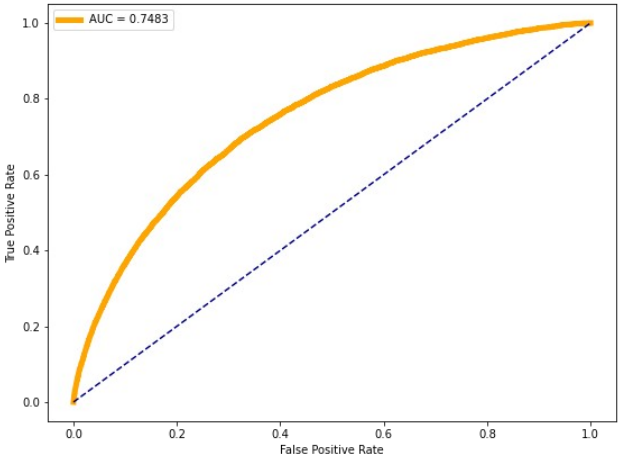
Notre approche: Une fois le modèle choisi et les hyperparamètres optimisés finement d'un point de vue technique via l'AUC, nous avons calculé une fonction de coût métier "fscore" et nous allons chercher son minimum. Nous allons donc effectuer une nouvelle recherche des hyper-paramètres se basant sur la fonction métier proposée, de cette façon, ils seront choisis de sorte à minimiser la perte pour l'entreprise.

Les résultats trouvés:

- Avant définition de la fonction de coût métier :
  - auc = 0.759
  - fp = 0.25 %
  - fn = 7.81 %
  - accuracy = 0.92
- Après définition de la fonction de coût métier :
  - auc = 0.748
  - fp = 0.42 %
  - fn = 7.70 %
  - accuracy = 0.92

--> Nous avons constaté une légère amélioration du pourcentage des Faux-négatifs (7.81% --> 7.7%)





4. L'interprétabilité globale et locale du modèle

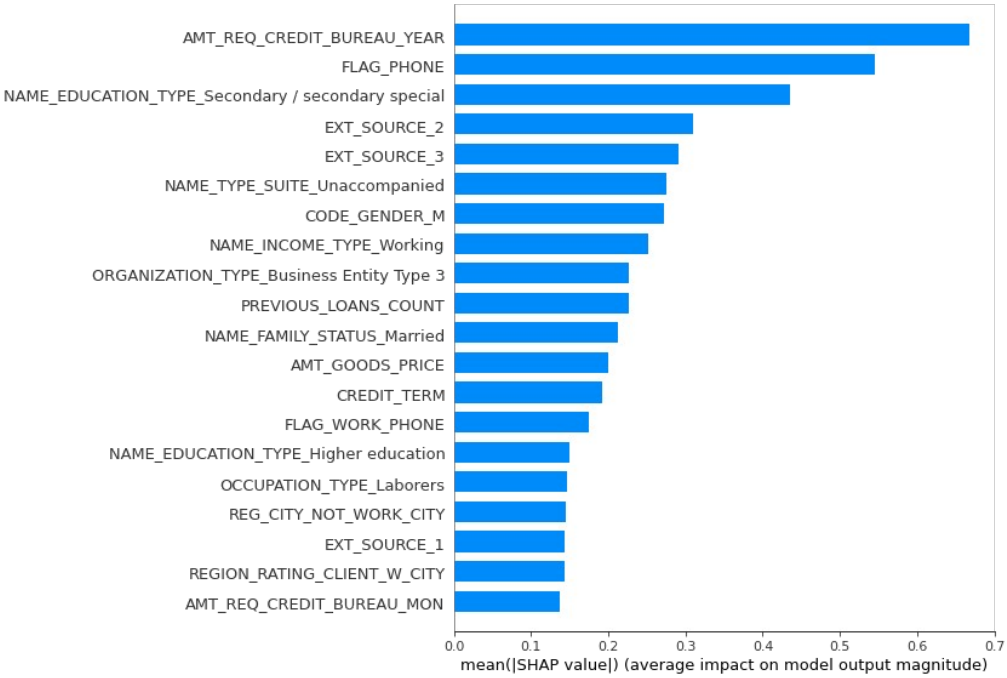
Pour extraire des informations du modèle, la première étape est sûrement l'approche globale qui consiste à définir l'importance des variables du modèle de manière globale. La seconde étape consiste à changer d'échelle afin d'extraire des informations locales pour des exemples spécifiques de notre dataset.

LIME (Local Interpretable Model-agnostic Explanations) et SHAP (SHapley Additive exPlanations) sont deux bonnes méthodes pour expliquer les modèles. En théorie, SHAP est la

meilleure approche car elle fournit des garanties mathématiques pour la précision et la cohérence des explications. En pratique, l'implémentation de SHAP (KernelExplainer) est lente. Ce problème de vitesse est beaucoup moins préoccupant si vous utilisez un modèle basé sur une arborescence. Pour LIME, l'inconvénient majeur c'est le biais dû à une hypothèse de non-corrélation entre features. En effet, l'échantillon d'entraînement de données généré à partir de l'instance qui nous intéresse, ne prend pas en compte les corrélations potentielles entre les features du modèle, ce qui peut donner des combinaisons de valeurs aberrantes, par exemple un appartement de 20m2 composé de 4 pièces.

--> Pour ces raisons, nous allons utiliser SHAP

Feature importance globale

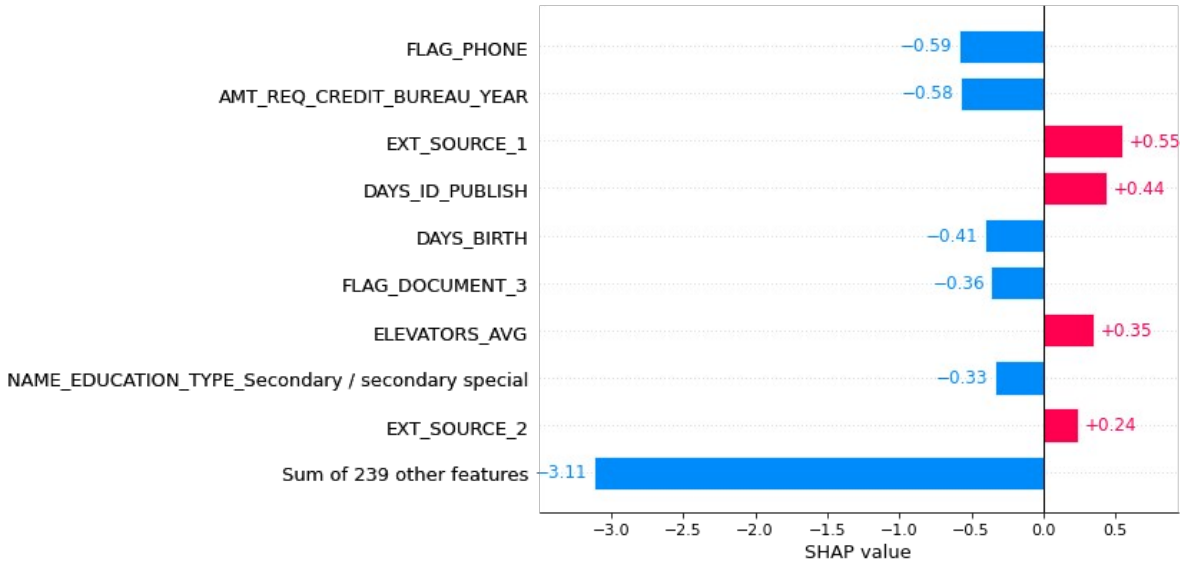


Nous remarquons que la variable "AMT_REQ_CREDIT_BUREAU_YEAR" (0.67) a la plus grande influence sur le résultat de la prédiction du modèle, suivi par "FLAG_PHONE"(0.55) et "Name_education_type_secondary" (0.44).

Feature importance locale

Afin d'assurer plus de transparence au client par rapport à la prise de décision du système, nous lui offrons un graphe de feature importance correspondant à ses données afin de lui exlpiquer quels sont les données qui ont influencées le plus ce résultat (accord/ acceptation du demande du prêt).

Prenons l'exemple d'un prêt refusé:



Sur cet exemple, les valeurs de Shapley nous indiquent que la demande du prêt est refusée en premier lieu à cause de son EXT_SOURCE_1, et dans une moindre mesure à cause de DAYS_ID_PUBLISH et ELEVATORS_AVG et EXT_SOURCE_2.

A contrario, flag_phone, AMT_REQ_CREDIT, DAYS_BIRTH, et FLAG_DOCUMENT_3 et autres features du clients favorisent l'acceptation du prêt sans toutefois le compenser totalement.

5. Déployement du modèle

Avant de déployer le modèle sur le web, nous l'avons sauvegardé. Ensuite nous avons créé un nouveau projet Flask sur PyCharm afin de créer l'API. Cette API a pour rôle d'importer le modèle et de retourner la probablité qu'un prêt soit accordé à partir de l'identifiant de l'utilisateur. Ceci présente la partie back-end.

La deuxième étape c'est de créer un dashbord (front-end) afin de permettre l'interaction entre modèle et client via le web. Ce dashbord fait appel à l'API qu'on a déjà créé afin de présenter au client la décision d'accord ou refus de sa demande de prêt. Notre dashbord a été développé sur PyCharm en utilisant la bibliothèque Streamlit.

Plusieurs autres fonctionnalité sont aussi déployées:

- Consultation des données utilisateurs
- Permettre de visualiser des informations descriptives relatives à un client (via un système de filtre).
- Permettre de visualiser le score et l'interprétation (feature importance) de ce score pour chaque client de façon intelligible pour une personne non experte en data science.

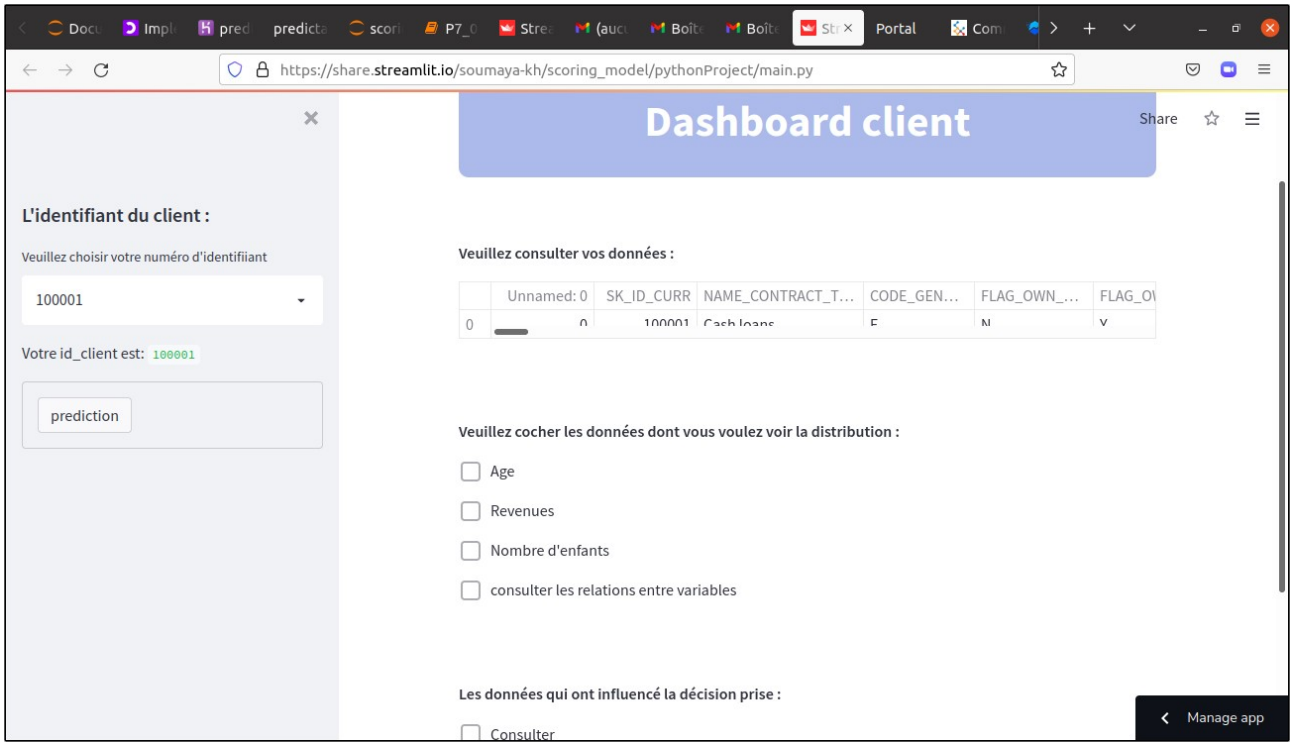
Afin d'assurer l'hébergement sur le web, nous avons déployé l'API avec Heroku et le dashbord avec streamlitshare.

◆ Lien vers l'API (exemple du client dont ID= 100001):

https://predictappli.herokuapp.com/predict/100001

◆ Lien vers le dashbord :

https://share.streamlit.io/soumaya-kh/scoring_model/pythonProject/main.py



Capture écran du dashbord

6. Les limites et les améliorations possibles

À cause des limites en ressources (mémoire et processeur) ainsi qu'au défaut du temps consacré pour ce projet, nous notons quelques limites qui peuvent être palliées par des améliorations que nous proposons. Plusieurs améliorations peuvent être apportées soit au niveau du modèle soit au niveau du dashbord. Nous citons , ci-dessous, les limites ainsi que les solutions proposées pour les pallier.

Au niveau du modèle sont :

- Consacrer plus de temps à la partie nettoyage des données. Rappelons que dans notre cas, nous nous sommes basé sur un kernel déjà existant. Le faite d'étudier plus profondément les données, favorisera leur compréhension et leur prétraitement.

- Pousser les recherches pour l'équilibrage des classes de Targets. Rappelons que dans ce projet nous les avons équilibré en utilisant une méthode de data-level (créant des échantillons synthétiques- SMOTE). D'autre techniques peuvent être utilisé afin de tester leur efficacité avec nos données, telque les méthodes algorithm-level. Ces méthodes effectuent des modifications aux modèles tel qu'accorder un poids plus important aux erreurs de la classification des points de la classe minoritaire.

- Nous manipulons des dataframes avec 248 features. Sélectionner les features les plus pertinents nécessite un expert métier du domaine. Une telle amélioration devra optimiser l'espace mémoire ainsi rendre le modèle plus performant.

- Il sera aussi interessant d'améliorer la fonction coût-métier à l'aide d'un expert métier.

Au niveau du dashbord :

- Rajouter une section pour que le client puisse consulter les données des autres clients qui lui sont proches (semblables).

- Offrir la possibilité de la visualisation des distributions de toutes les variables à l'aide d'un système de filtrage (liste déroulante)