Openclassrooms Parcours Data Science Projet 7

Dashboard sur les données bancaires pour évaluer la solvabilité client

REFERENCE MODELE

Nicolas Martinez

# Introduction

Cette note a pour but d’expliquer les différentes étapes du projet.

Le but du projet est de mettre à disposition un tableau de bord permettant d’évaluer la solvabilité de particulier et ainsi proposer une recommandation au secteur de la banque pour l’attribution de prêt bancaire à ce potentiel client. Ce tableau de bord interroge une API (Application Programming Interface).

Ces deux objets sont hébergés sur des plateformes cloud afin d’être accessible partout sur le web.

Pour la partie modélisation des données, il s’agit d’un problème de classification. Les résultats sont classés dans les 4 états suivants :

* Vrais positifs : les clients que le modèle estime solvable et qui remboursent leur prêt sans délai.
* Vrais négatifs : les clients que le modèle estime à risque et qui ont eu des retards avec leurs échéances.
* Faux positifs : les clients que le modèle estime solvable mais qui ont eu des retards avec leurs échéances.
* Faux Négatifs : les clients que le modèle estime non solvable mais qui le sont en réalité.

Le cœur du sujet se situe sur les 2 derniers aspects (faux positifs & faux négatifs) car ce sont ceux qui coutent à la banque par leur retard de paiement ou représentent un manque à gagner car ils auraient été de bons clients.

Le modèle que l’on construit est donc paramétré en tenant compte de ce point clé.

Contents

[Introduction 2](#_Toc92990471)

[1. La méthodologie d'entraînement du modèle 3](#_Toc92990472)

[2. La fonction coût métier, l'algorithme d'optimisation et la métrique d'évaluation 5](#_Toc92990473)

[3. L’interprétabilité globale et locale du modèle 6](#_Toc92990474)

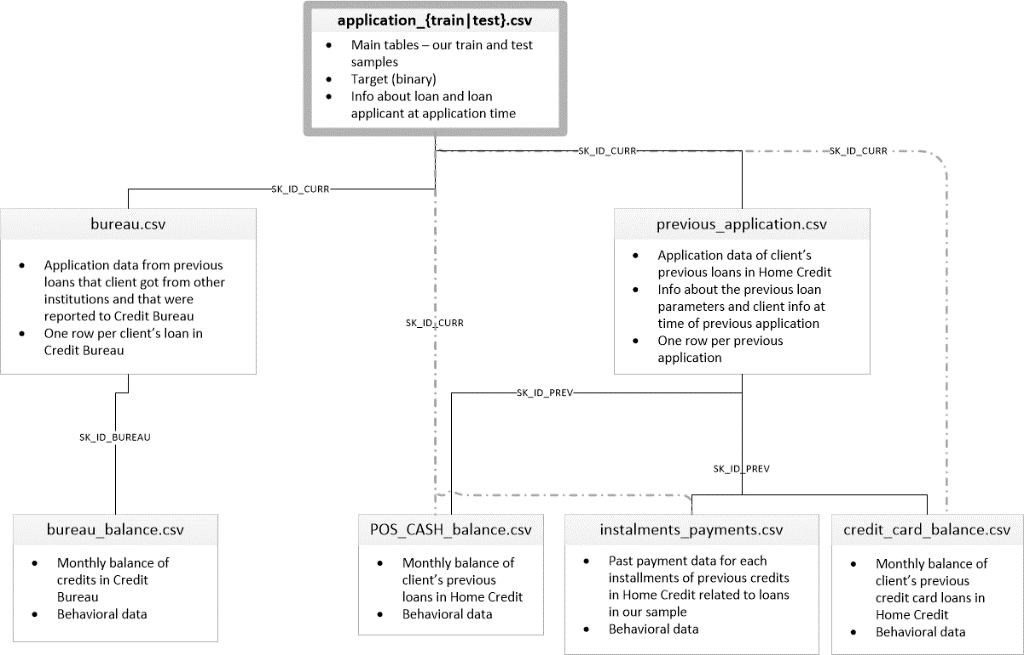
[4. Les limites et les améliorations possibles 7](#_Toc92990475)

# La méthodologie d'entraînement du modèle

L’entrainement du modèle se déroule en 5 étapes :

1. Appropriation des données

Le jeu de données est une série de fichier CSV contenant des données bancaires et financières de particuliers.



Structure de la base de données

Dans cette partie on effectue les opérations suivantes :

* Chargement des données csv dans des dataframes Pandas
* Analyse univariée des données
* Remplissages des données manquantes par le mode pour ne pas modifier la distribution et subir les effets de points aux valeurs extrêmes (outliers). Des opérations différentes sont utilisées pour les données textes, catégories et numériques discrètes d’une part, et les valeurs numériques continues d’autre part.
* Fusion des différents dataframes afin de rassembler les informations pour le modèle.

1. Préparation des données

Les données sont de bonnes qualité (faible pourcentage de données manquantes, pas d’erreurs) car elles proviennent d’établissements bancaires / financiers qui se sont assurés de l’intégrité.

Cette partie se résume alors aux opérations suivantes :

* Pour les données catégorielles et numériques discrètes : Remplissage des cellules vides par mode, utilisation d’un label encoder pour remplacer les labels par des données numériques acceptées par les modèles.
* Pour les données numériques continues : remplissage des cellules vides par moyenne.

1. Feature engineering

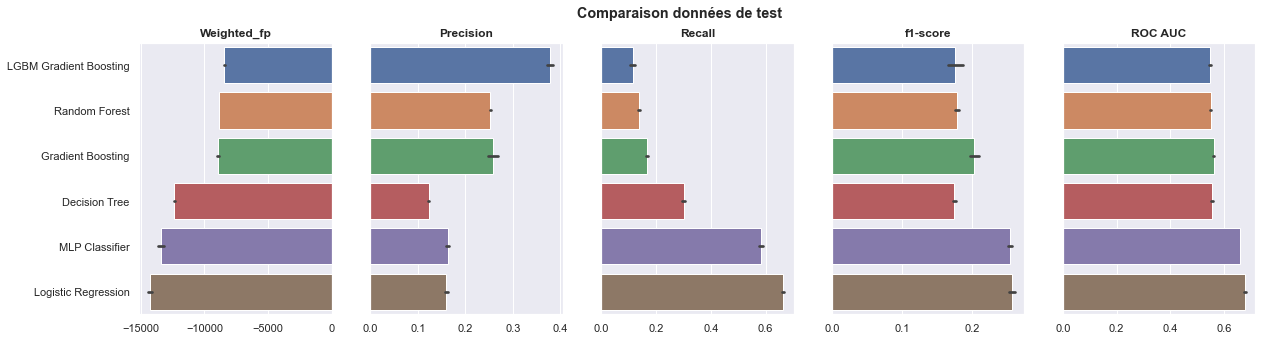
Dans cette partie on utilise les données présentes pour constituer de nouvelles features pertinentes à l’analyse. On retire ensuite les données qui ont une corrélation trop élevée afin d’éviter le problème de fuite des données (data leakage).

1. Recherche du modèle le plus efficace

Les opérations usuelles de preprocessing sont appliquées :

* Standardscaler() : on normalise les données afin que toutes les colonnes aient le même poids avant entrainement
* Séparation des données en jeu de données d’entrainement et test
* Opération d’équilibrage des données : comme les clients non solvables sont fortement majoritaires on emploie des techniques d’équilibrages (ici SMOTE) afin de synthétiser des données aux proportions de cible raisonnables.

On essaie ensuite les diffèrents modeles de classification afin de voir le(s)quel(s) sont les plus adaptés à ce jeu de données. On entraine avec leurs paramètres par défaut pour voir la réaction au jeu de données. On regarde notamment les scores de précision et recall qui donne une indication sur les faux positifs & négatifs.

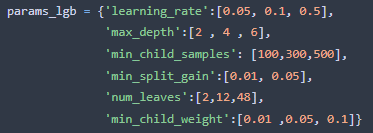
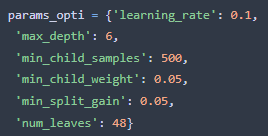


Le modele LGBMclassifier a les meilleurs scores : c’est celui que nous choisissons.

Également, on essaie d’entrainer le fichier initial application\_train et le fichier data que l’on a constitué en fusionnant les différentes sources de données : le second fichier donne bien de meilleurs résultats.

1. Optimisation du modèle

Afin d’optimiser notre modèle, on utilise la fonction de validation croisée de Scikit-learn sur les hyperparamètres suivants.

A gauche les modèles testés (486 combinaisons), à droite le modèle choisi.

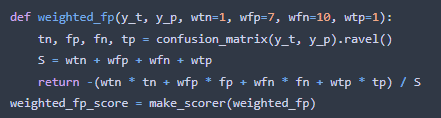
Plusieurs combinaisons obtiennent des résultats similaires en termes de score. On sélectionne celui qui permet d’obtenir les prédictions les plus rapides afin de proposer une meilleure « expérience client ».

On entérine ainsi l’utilisation du LGBMclassifier et du fichier data que l’on a constitué.

On valide l’utilisation de ce modèle pour la suite des travaux.

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

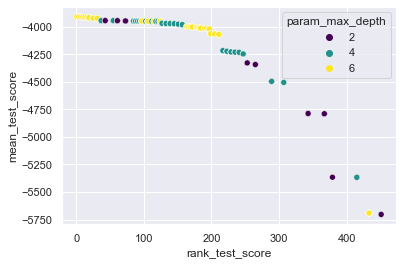
Afin de maximiser l’influence des faux positifs et négatifs, on crée une fonction score qui va leur donner une pondération plus importante. On récupère ces valeurs clés dans la matrice de confusion.



Définition de la fonction pondération et transformation en score

Ici on attribue respectivement un poids de 7 et 10 par rapport à l’unité pour les vrais. Ces valeurs 7 et 10 choisies arbitrairement doivent être le cœur de la discussion avec le métier et constituent une des pistes d’amélioration du modèle (cf. 4.)

On ajoute un signe moins afin de respecter la norme de Scikit-learn qui consiste à avoir des scorers qu’il faut maximiser[[1]](#footnote-1). Ici, on cherche à minimiser les faux, ce qui va donc conduire à un score le plus petit possible en valeur absolu. On ajoute un moins pour maximiser ce score. Cette fonction, une fois transformée en score avec la fonction make\_score peut ensuite être intégrée à la recherche des hyperparamètres par validation croisée afin d’être le facteur de décision. C’est ce que nous avons fait.



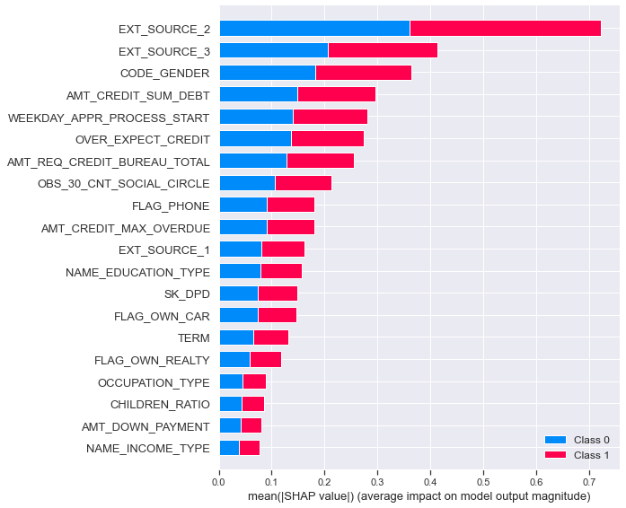
Score en fonction de l’hyperparametre param\_max\_depth

Les hyperparamètres ont différents niveaux d’impact sur le modèle. Ici une profondeur maximale de 6 donne clairement de meilleurs résultats que les 2 autres valeurs. On sélectionne cette valeur et on répète l’opération pour les différents hyperparamètres.

# L’interprétabilité globale et locale du modèle

Une fois le modèle optimisé, on peut observer les variables qui impactent le plus les prédictions qu’il procure.

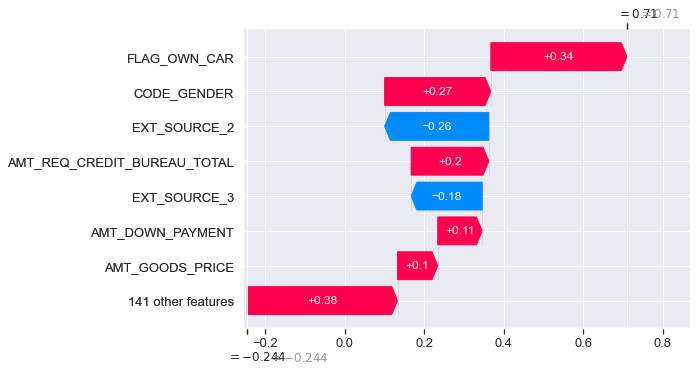
On peut obtenir des estimations de ces résultats par différents modules / techniques.



Impact des features d’après le module Shap

Le module Shap a l’avantage de donner l’impact pour la classe 0 et 1 séparément. Dans les 3 cas (Scikit, LGBM & Shap) on retrouve l’impact prépondérant du genre, du montant des crédits, et les données masquées Ext\_Source\_2, 3 et 1.

On peut ensuite observer pour chaque client individuellement, les facteurs qui ont le plus contribué à sa prédiction.



Waterfall chart de Shap : pour chaque feature, l’impact sur le dossier du client

Pour ce client, la possession d’une voiture personnelle a un plus grand impact (+0.34) que son genre (+0.27) ou sa donnée Ext\_source\_2 (-0.26).

Ce genre d’analyse nous permet de comprendre en détail le fonctionnement du modèle pour chaque client. Le fonctionnement du modèle est rationalisé pour l’utilisateur et évite d’être considéré comme une « boîte noire ».

# Les limites et les améliorations possibles

Le travail effectué permet sans aucun doute une utilisation du modèle pour les fins de la mission.

Il pourrait cependant être amélioré sur plusieurs points que l’on pourra considérer lors de prochaines évolutions :

* Optimisation de la fonction scoring :

La fonction de scoring a été codé afin de répondre à la problématique générale du problème qui est la minimisation des cas de faux positifs & négatifs qui coutent au client. Cependant, les coefficients de cette fonction ont été choisis arbitrairement.

Pour affiner le choix de ces coefficients, une discussion avec le métier est nécessaire pour comprendre les clefs du problème et les couts impliqués. Cela permettrait d’optimiser la fonction cout métier et en découlerait une meilleure fonction scoring pour ce projet.

La première étape de cette amélioration est donc une rencontre avec le client.

* Définition d’un seuil de prédiction :

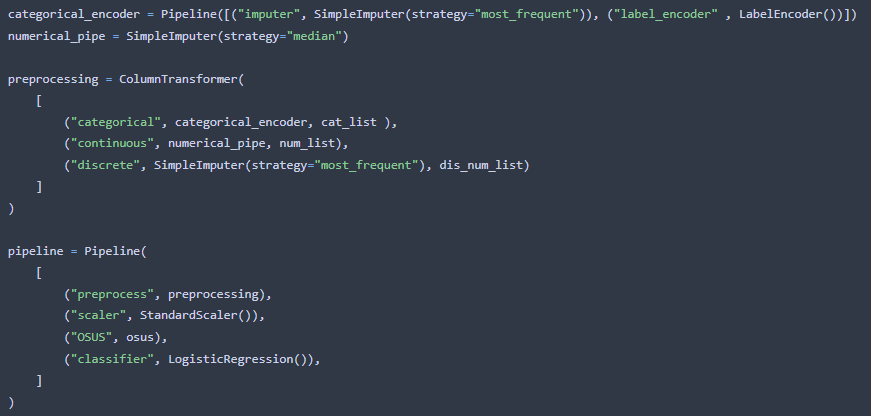
Les clients sont actuellement considérés solvables au-delà de 0,5 ; non solvable dans le cas contraire.

Ce seuil de prédiction pourrait être modifié en fonction du risque jugé comme acceptable par le client dans l’optique de l’abaisser et de prendre plus de client. Une nouvelle fois cette conversation doit avoir lieu avec le client car lui seul est décisionnaire.

* Inclure toutes les opérations dans un pipeline

Actuellement seul le modèle LGBM est utilisé par l’API. Les opérations décrites en partie 1.3 et 1.4 de préparation des données sont effectuées en amont. Cela signifie que quand on utilise le modèle, il faut lui renseigner des données après qu’elles étaient été réduites / centrées et passé par un LabelEncoder qui retire les objets pour les transformer en chiffres. On perd donc la notion d’échelle pour les données numériques continues, et de sens pour les données discrètes.

On peut remédier à ce problème en incluant toutes les opérations dans un pipeline tel que présenté ci-dessous. Un problème de compatibilité entre le *fit()* de scikit-learn et le *fit\_sample()* de SMOTE a empêché de mener à bien cette étape.



1. “All scorer objects follow the convention that **higher return values are better than lower return values**. “ https://scikit-learn.org/stable/modules/model\_evaluation.html#common-cases-predefined-values [↑](#footnote-ref-1)