OpenClassRooms Centrale Supélec / Parcours Data Scientist / Projet 7

Implémentez un modèle de scoring

Note Méthodologique

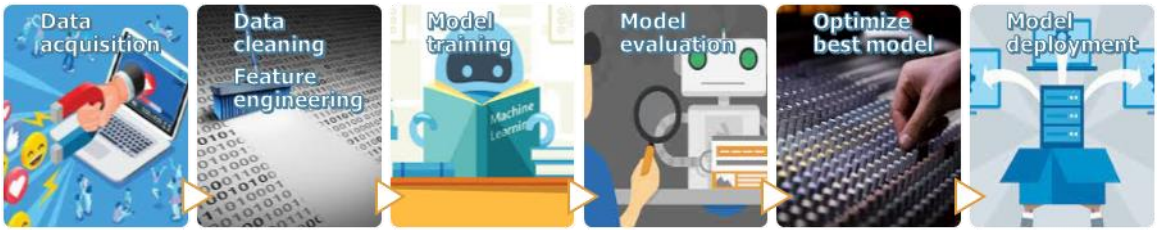
Sébastien De Rosa
07/10/2022
GitHub|https://github.com/SDR69250/OpenClassRooms_P7

I | Contexte

Cette note méthodologique est l'un des livrables du projet 7 « Implémentez un modèle de scoring » du parcours Data Scientist d'OpenClassRooms. Le propos est ici de présenter le processus de modélisation et d'interprétabilité du modèle mis en place dans le cadre du projet.

Il s'agit de mettre en œuvre un outil de "scoring crédit" pour **calculer la probabilité** qu'un client rembourse son crédit, de **classifier** la demande en crédit accordé ou refusé, et d'**expliquer** de manière transparente au client les décisions d'octroi de crédit.

Les étapes successives suivantes ont été suivies pour la modélisation et sont détaillées ci-après :



II | Méthodologie d'entraînement du modèle

1. Acquisition des données

Le jeu de données d'entrainement .csv disponible sur Kaggle comporte 307 000 observations qui présentent 121 caractéristiques (âge, sexe, emploi, logement, revenus, informations crédit, notation externe, etc.). Elles sont complétées par 6 fichiers de données .csv et 1 de description, tous chargés dans le notebook et rapidement parcourus pour d'identifier le type d'information disponible.

2. Data cleaning et Feature engineering

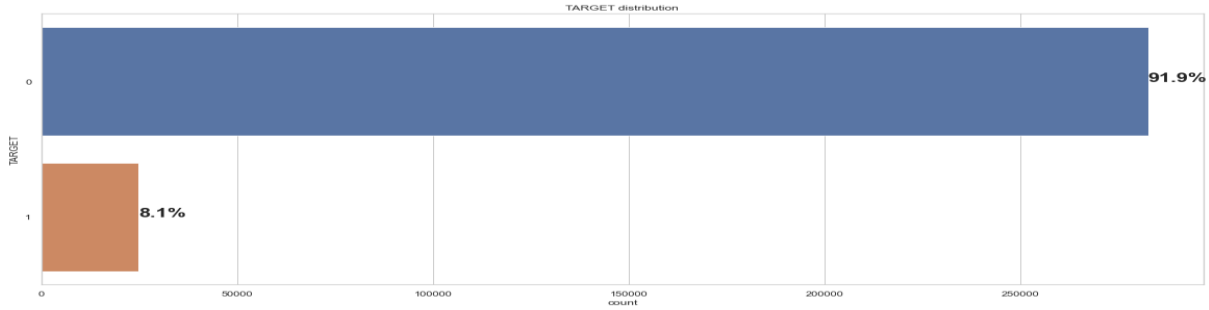
Les données étant éparses et incomplètes, cette étape consiste à traiter séparément d'abord les données de chaque tableau, encoder les features catégorielles, créer des features spécifiques métier à partir des données source, puis assembler les tableaux en une seule base, pour enfin éliminer certaines features, traiter les valeurs aberrantes et imputer les manquantes (médiane).
La démarche est inspirée du kernel LightGBM with Simple Features, disponible ici :
https://www.kaggle.com/code/jsaguiar/lightgbm-with-simple-features/script
Cette étape aboutit à un jeu de données contenant 307 507 observations et 114 features.

3. Construction et entrainement des modèles

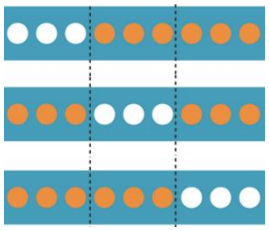
Typiquement un modèle de prédiction est une fonction qui prend en entrée des données et renvoie en sortie une prédiction sous la forme d'une prévision de défaut de paiement dans notre cas.

Une partie seulement du jeu de données doit être fournie aux modèles pour leur entrainement, car l'autre partie doit permettre de tester de manière indépendante leur performance. Le jeu d'entrainement comprend 80% des données d'origine, et le jeu de validation 20%.

La classe minoritaire qui nous intéresse (clients en défaut de paiement, ou TARGET = 1) est sous-représentée, l'échantillon est dit déséquilibré, au risque de faire « surapprendre » la classe 0 :



Nous prenons donc soin de respecter cette proportion d'origine dans la constitution des échantillons train et test, afin de pouvoir traiter par la suite ce déséquilibre avec les outils adéquats.



Chaque modèle a vu certains paramètres testés selon une **validation croisée GridSearchCV** (chaque partie sert alternativement de jeu d'entrainement et de validation), et c'est au départ de ce processus que le sur-échantillonnage a lieu : le nombre d'individus minoritaires (en défaut) est **augmenté par SMOTE** (suréchantillonnage synthétique : on crée de nouvelles observations virtuelles en s'appuyant sur la technique des k plus proches voisins) pour que ceux-ci aient plus d'importance uniquement durant la modélisation.

Les différents modèles testés sont 6 au total :

- **Baseline** : Dummy Classifier (most frequent)
- **Classique** : Logistic Regression
- **Forêt aléatoire** : Random Forest Classifier
- **Boosting** : Cat Boost Classifier, XGB Classifier, Light Gradient Boosting Model Classifier.

III | Fonction coût métier / Algorithme d'optimisation / Métrique d'évaluation

1. Fonction coût métier

Pour le métier il est important de minimiser les coûts liés au fait d'identifier correctement ou pas les clients qui se révèleront en défaut. Or, les modèles génèrent deux types d'erreurs de prédiction :

- **Faux Négatif** = client identifié à tort pas en défaut. Accord de crédit, et perte en capital.
- **Faux Positif** = client identifié à tort comme en défaut. Refus de crédit, et perte de marge.

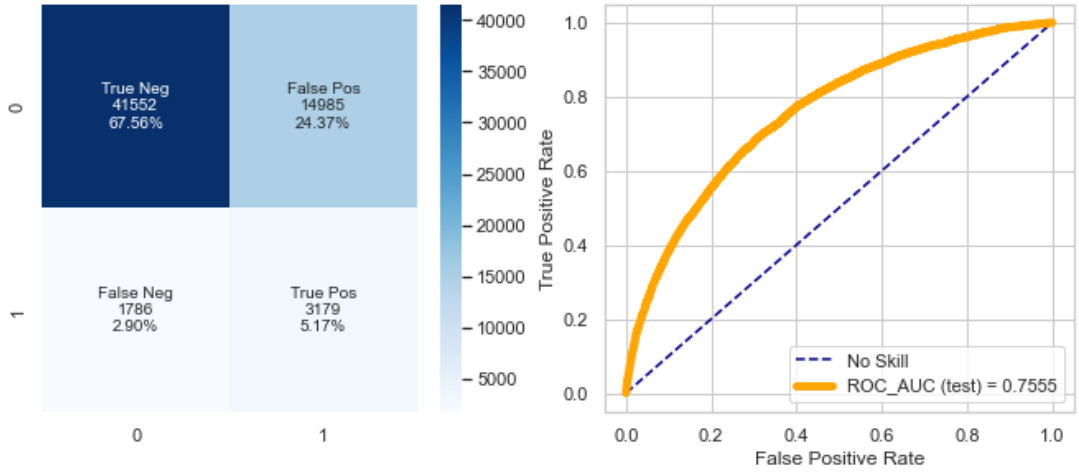
Le coût de la perte en capital à la suite d'un FN est environ 10 fois supérieur à celui de la perte de marge à la suite d'un FP. En effet, si nous partons d'un capital de 100 :

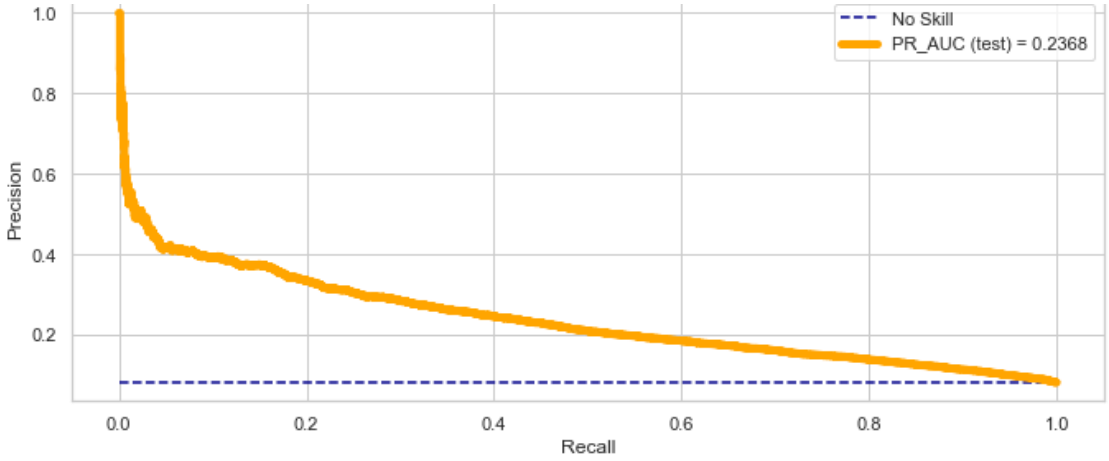
- **FN** | Perte de 50% du capital prêté en moyenne → **Perte de 50**
- **FP** | Manque à gagner de 1% par an sur 10 ans en moyenne
  (Le capital passe de 100 à 0, soit une moyenne de 50) donc 10% de 50 → **Perte de 5**

Il faut minimiser les observations **FN** et **FP** menant à ces fausses prédictions. Nous souhaitons donc réduire la fonction **10 FN + FP / taille échantillon** (dénominateur choisi pour obtenir un ratio entre 0 et 1) en sortie de la matrice de confusion (comparaison prévision vs réel) pour chaque modèle testé. C'est selon ce score que les modèles sont évalués dans l'étape de GridSearchCV et finalement sélectionnés. D'autres scores (temps, ROC_AUC, etc) sont également calculés.

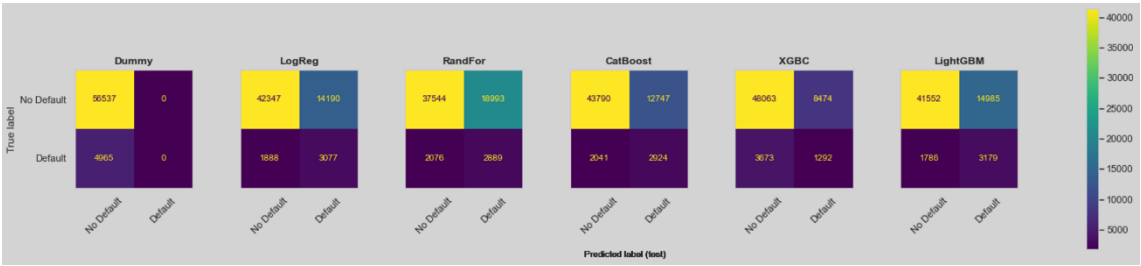
2. Optimisation des modèles

Pour chaque modèle retenu par le GridSearchCV il a été effectué une **optimisation du seuil**, afin de déterminer la **valeur minimale de la probabilité** de défaut à partir de laquelle une observation doit être **classée** en défaut. Avant cette optimisation le seuil standard est de 0,5. Cette opération est effectuée à l'aide d'une boucle qui itère sur 100 valeurs comprises entre 0 et 1, et qui retient celle pour laquelle la fonction coût métier **(10 FN + FP) / taille échantillon** est minimale.

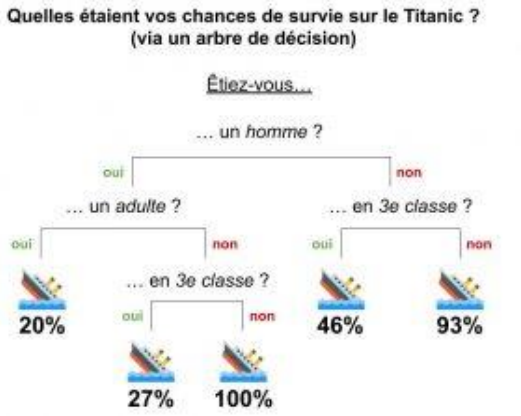




	Data	Model	Custom_score	Time	ROC_AUC	PR_AUC
11	Test	LightGBM	0.534	0.2961	0.7555	0.236795
3	Test	LogReg	0.5377	0.203	0.7534	0.228894
7	Test	CatBoost	0.5391	0.2131	0.7512	0.231547
13	Test	LightGBM_NS	0.5395	0.2521	0.7539	0.232267
5	Test	RandFor	0.6464	0.2841	0.6688	0.145183
9	Test	XGBC	0.735	0.2631	0.6199	0.111155
1	Test	Dummy	0.8073	0.041	0.5	0.540365

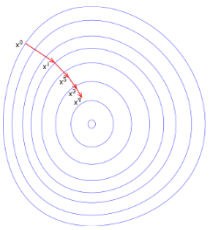


Le **modèle retenu est le LightGBM**, qui obtient le meilleur custom score, c'est à dire qui minimise le coût métier. Il présente les plus grandes aires sous la courbe ROC_AUC et PR_AUC.



LightGBM est un algorithme qui repose sur des arbres de décision, qui se déroulent et proposent des intersections. Une question y est posée, qui oriente vers l'une ou l'autre branche selon la réponse. Ici un homme non adulte en 3e classe n'avait que 27% de chances de survivre à l'accident du Titanic.

Le point de départ est la racine (root) et le bout des branches, les feuilles (leaf) montre les probabilités. La croissance des arbres s'effectue au niveau des feuilles.



Il s'agit aussi d'un **Gradient Boosting Machine**, qui combine plusieurs apprenants dits « faibles » (ici des arbres à peine meilleurs qu'un classifieur aléatoire), de manière à minimiser en suivant le gradient une fonction de perte différentiable qui permet de résoudre le problème métier.

Chaque ajout d'arbre supplémentaire permet de descendre le gradient.

Interprétabilité globale et locale du modèle

Le modèle est destiné à être utilisé par des professionnels de la banque auprès des clients. Il faut être capable d'**expliquer les décisions** d'octroi de crédit de manière claire et transparente.

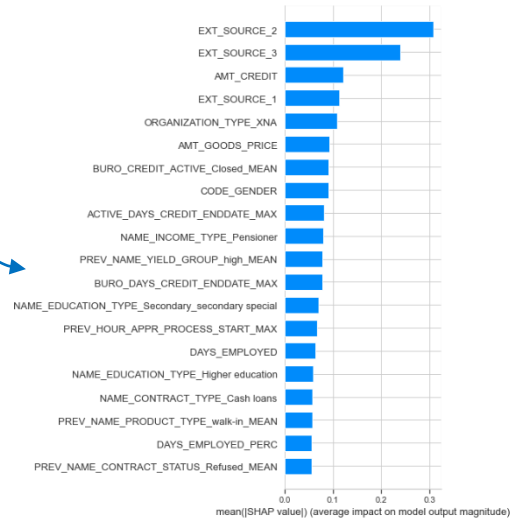
Or les modèles ont la particularité de se comporter comme des boites noires, qui ne permettent pas de comprendre les relations de cause à effet.

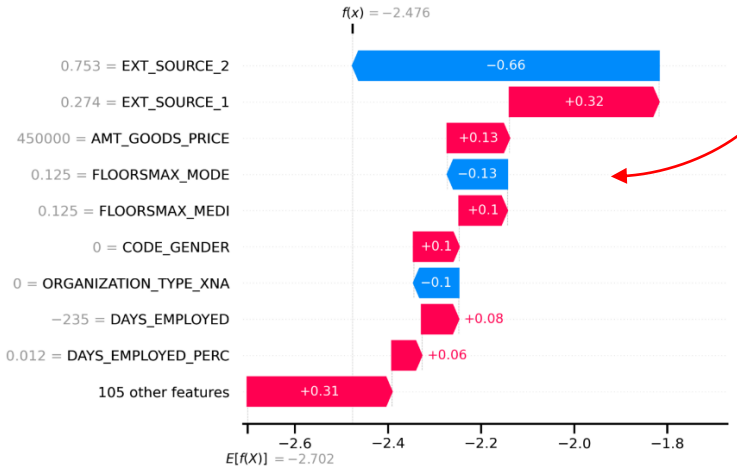
Pour contourner cet obstacle, nous calculons les **Shap values** (**SH**apley **A**dditive ex**P**lanations), c'est-à-dire l'impact additionnel de chaque feature dans la prédiction de chaque probabilité.

Il devient donc possible d'expliquer non seulement globalement le poids de chaque feature, mais également de comprendre localement le poids de chaque feature dans la prédiction associée à une observation spécifique, en partant de la valeur de base attendue, et en sommant les SHAP values jusqu'à obtenir la probabilité prédite par le modèle.

Interprétation Globale : liste des features par impact décroissant sur l'ensemble des prédictions

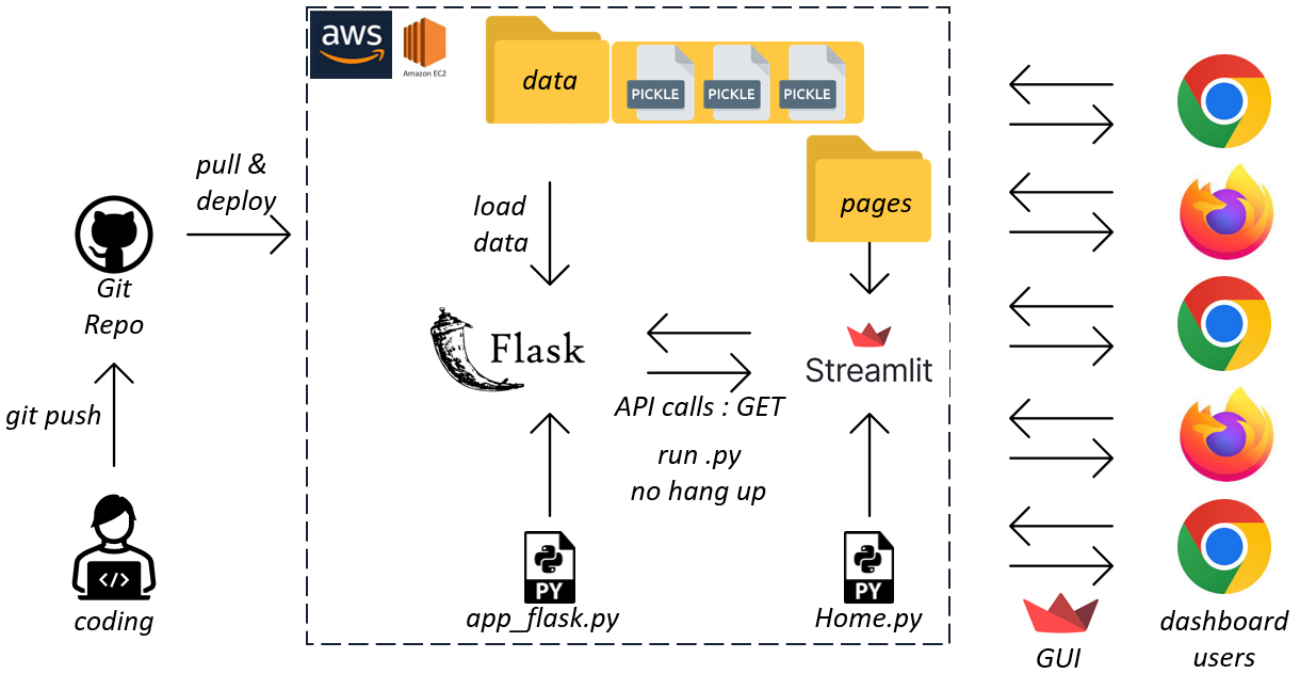
Interprétation Locale : par observation, on part d'une base value pour expliquer 1 prédiction





Déploiement

Afin qu'il puisse être utilisé par les chargés de relation client ce modèle de prédiction a été embarqué dans un tableau de bord Streamlit, et déployé dans le cloud sur un serveur EC2, avec une API Flask, la gestion des versions de code a été effectuée à l'aide de Git et GitHub : dashboard



Limites et améliorations possibles

Des améliorations semblent possibles à ce stade sur 3 niveaux :

1. La sélection du meilleur modèle est strictement basée sur la **fonction coût métier**. Les hypothèses de construction de cette fonction coût métier restent à valider avec le métier, voire à améliorer.
2. Nous avons établi que **3 sources externes** participent pour une part significative de la décision d'octroi de crédit (60%). Il faudrait pouvoir étoffer ces critères et être ici également transparent. De quoi ces informations sont-elles composées ?
3. Le **pré-traitement des données** ayant été récupéré depuis un **kernel Kaggle**, sans avoir apporté de modifications significatives, ni de remise en cause (ce n'était pas l'objet de ce projet), il est probable qu'un travail plus minutieux de nettoyage et de feature engineering permettrait d'obtenir de meilleurs résultats en termes de qualité de prédiction.