Projet 7 : OpenClassroom

Implémentez un modèle de scoring

Note Méthodologique

Introduction

Dans ce rapport on va présenter la méthodologie d'entraînement pour le projet 7 d'Openclassroom : "Implémentez un modèle de scoring"

Le but du projet est de prédire la probabilité de défaut des clients pour une société financière qui donne des crédit à la consommation "Prêt à dépenser", pour ce faire, on possède différentes données sur les clients (Âge, sexe, ville, cercle social ...), et aussi des données sur le crédit actuel ou passés.

Pour le nettoyage des données on a utilisé, comme recommandé par l'énoncé, une suite de notebooks Kaggle déjà existant, ils sont consultables à cette adresse (Ou dans les notebooks de cleaning)

La pré exploration s'est déroulé en 3 étapes principales:

- Agrégation des différentes données dans une seule dataframe
- Nettoyage des valeurs peu représentées
- Utilisation d'un "Label encoding" pour les variables catégorielles avec deux valeurs, et du "One Hot Encoding" pour les autres variables catégorielles
- Nettoyage des valeurs fortement corrélées
- Nettoyage des valeurs en utilisant les "features importance" de Light GBM en gardant que les features expliquant 95% du modèle.

On obtient à la fin une unique source de données avec 338 features, et 307511 lignes.

A. Méthodologie d'entraînement du modèle:

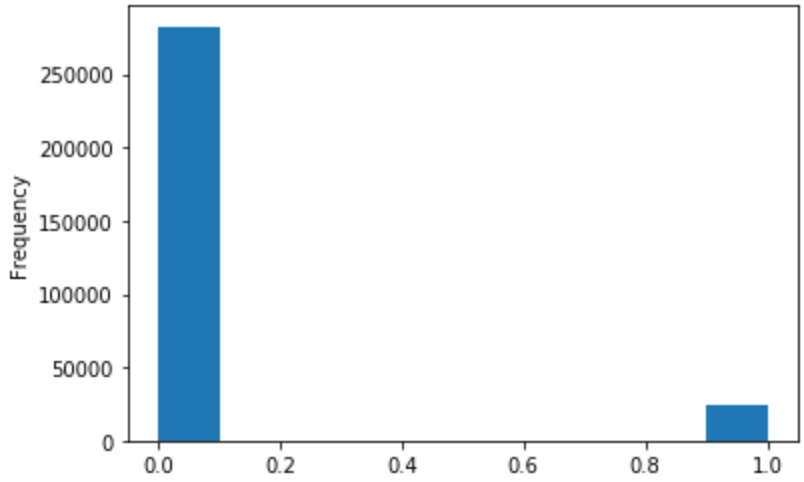
Première étape, qui va nous servir pour tous nos modèles, c'est de séparer le jeu d'entraînement avec le jeu de test, on va prendre 75% des données qu'on va utiliser pour l'entrainement, et 25% pour le test.

Or comme on va le voir par la suite, la valeur qu'on doit prédire (une valeur binaire 0 ou 1) est très mal répartie, pour que les échantillons de test, et d'entraînement soit représentatifs des données, on va utiliser un train test split stratifié, le même pourcentage de 0 et de 1 va se refléter sur les jeux de test et d'entraînement.

a. Utilisation d'une baseline

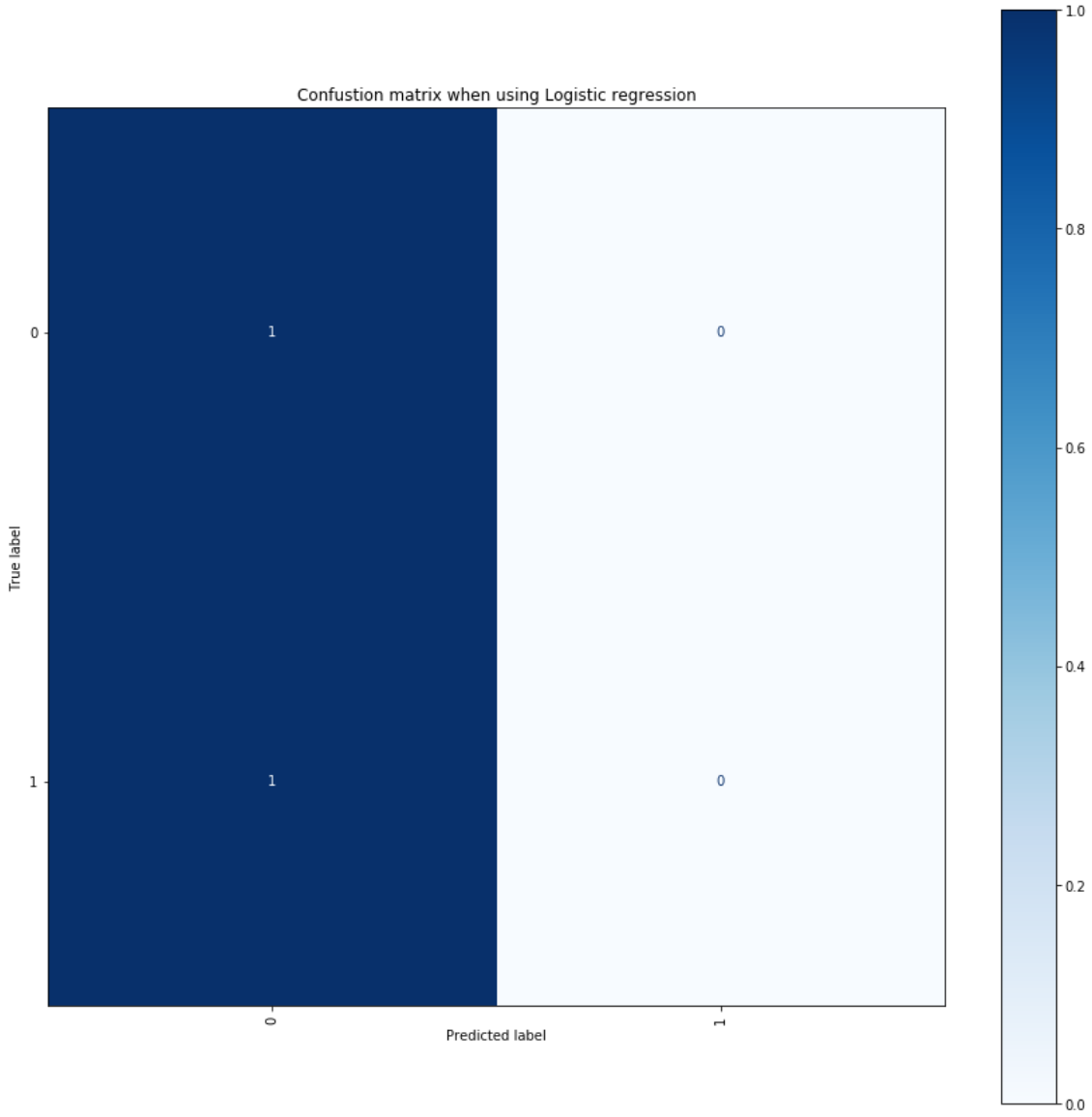
Pour entraîner notre modèle, il nous faudrait une "baseline", un modèle de base qui va nous donner un premier score, et qui nous servira de point de référence pour améliorer le modèle. On prend pour cela un modèle simple sans forcément essayer de l'optimiser au départ. On est donc partis sur une régression logistique qui nous donne une "accuracy" de 91% sur le jeu de test.

Cela peut être considéré à tort comme un bon modèle, regardons d'abord la répartition des données entre mauvais / et bon client :



On remarque déjà que la répartition des données entre les gens qui sont "des bons payeurs", et les "mauvais payeurs" dans le jeu de données et très déséquilibrée (91% vs 9%), un modèle qui prédirait que tous les clients sont de bons clients, aurait une accuracy de 91%, et serait quand même un très mauvais modèle.

Regardons la matrice de confusion qui nous permettra de voir les faux positifs, vrais positifs, faux négatifs, et vrais négatifs de notre modèle :

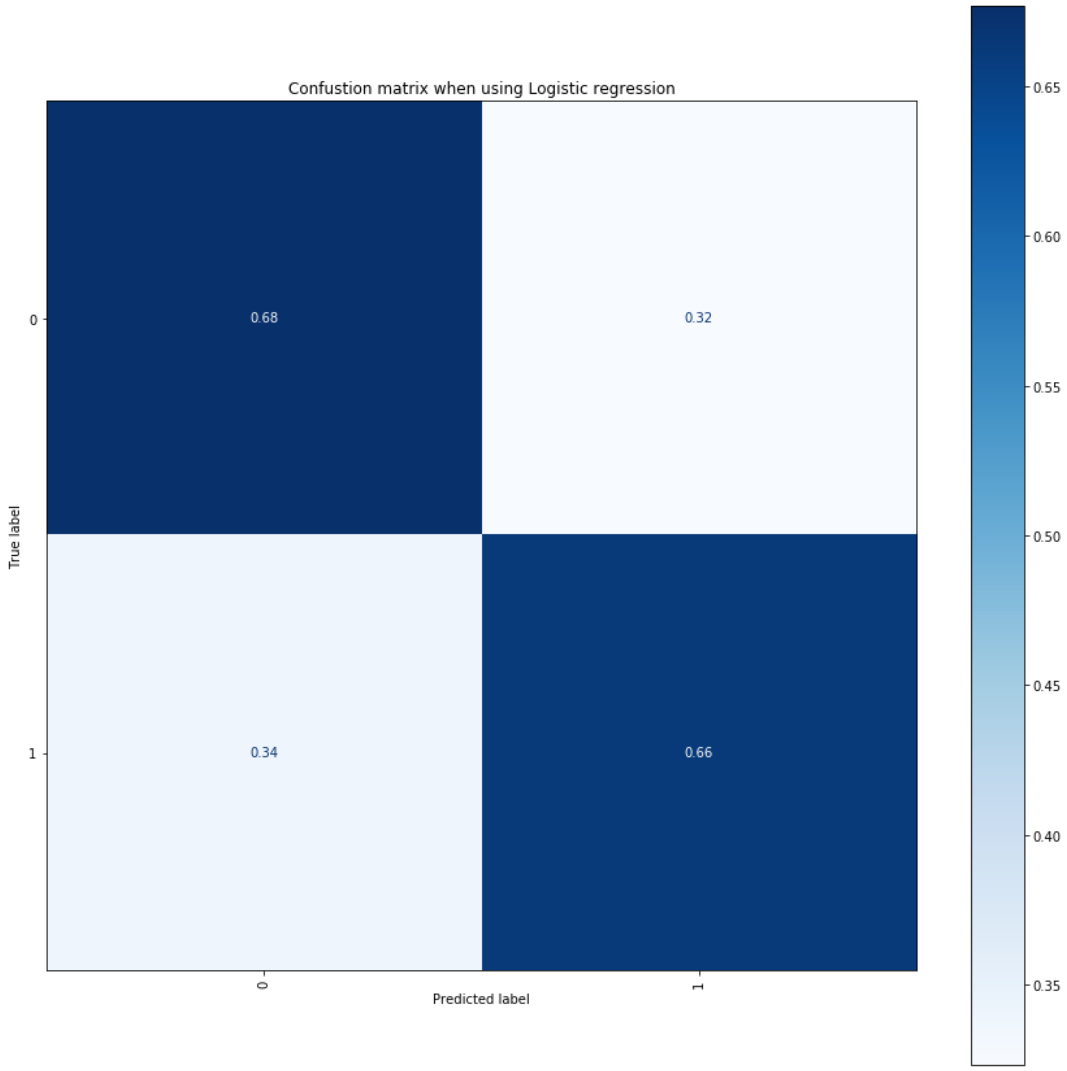


On remarque qu'effectivement notre modèle renvoie toujours une valeur target de 0, et propose d'accorder le crédit à tout le monde, d'un point de vue métier ce modèle risque de provoquer des pertes énormes pour la banque, accorder un crédit à un client qui va pas le payer est beaucoup plus coûteux que ne pas l'accorder à un client qui va probablement le rembourser.

Une métrique qui nous permet de voir cette mauvaise détection est l'AUC score = 0.5 pour cette baseline pour le jeu de test.

b. Gérer l'échantillon minoritaire:

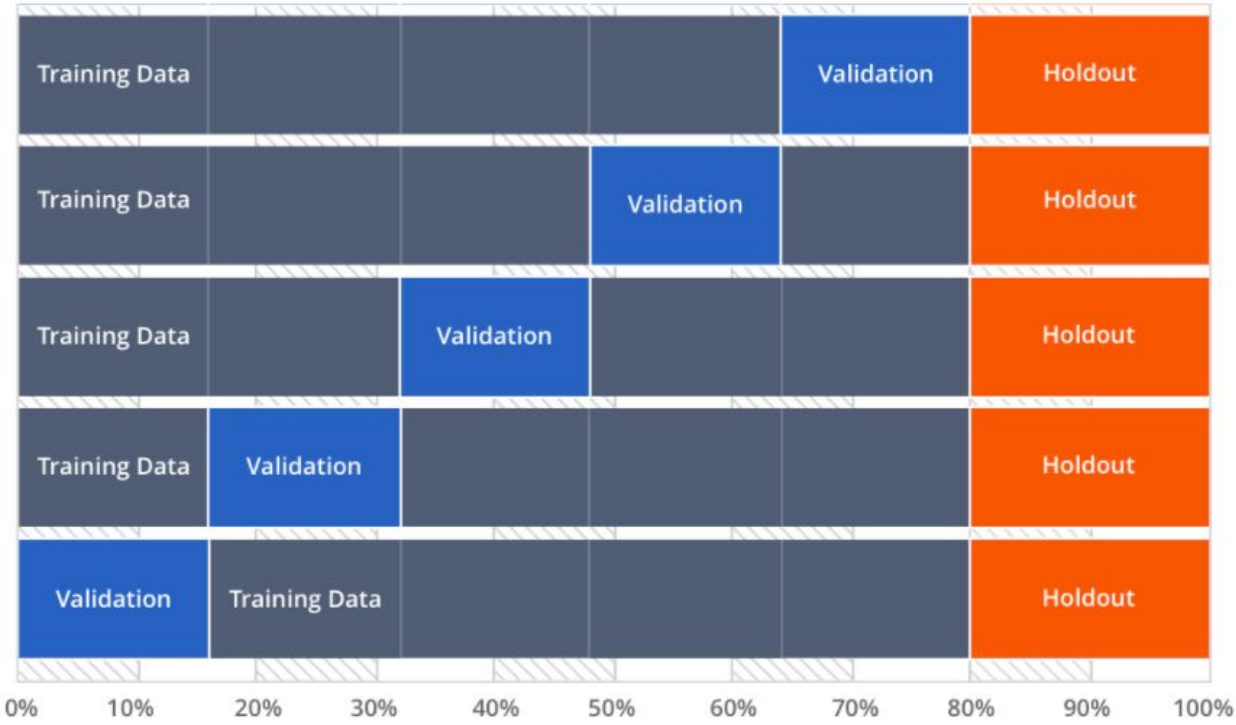
Afin d'améliorer notre modèle, on va juste améliorer un peu la baseline, en ajoutant des données pour étoffer l'échantillon minoritaire, pour celà on va utiliser SMOTE , qui en s'appuyant sur l'algorithme des plus proches voisins, va générer des échantillons pour la classe minoritaire, uniquement pour le jeu d'entraînement, c'est de l'*Over-Sampling*. En lançant la même régression logistique que la baseline on obtient, une accuracy moins importante (67%), par contre on passe sur un score AUC meilleur (0.67)



c. Entraîner un meilleur modèle:

On a vu que ça améliorerait le score d'utiliser SMOTE pour gérer l'Over Sampling dans notre cas. Donc pour trouver un meilleur modèle on va :

- Partir sur le jeu d'entraînement et de test généré précédemment
- Générer des échantillons de la classe minoritaire en utilisant SMOTE (s'appuyant sur l'algorithme des plus proches voisins)
- Pour chercher les meilleurs paramètres des algorithmes données en entrée on va utiliser Hyperopt, qui est plus rapide qu'une recherche sur une grille (en passant par l'algorithme TPE) et surtout qu'il accepte des valeurs continues en paramètre,
- Comme algorithme, on a utilisé que le Light GBM, qui est un algorithme de Gradient Boosting, utilisant peu de mémoire et plus rapide. (Ceci est aussi pour des raisons de machine peu puissante)
- Pour éviter l'overfitting, on va utiliser une validation croisée: on partage le jeu d'entraînement en 5 groupes différents, à chaque itération de l'algorithme, celui ci va utiliser un jeu pour la validation, et 4 autres pour l'entraînement, et c'est la moyenne de ces scores qu'on va utiliser pour trouver le meilleur modèle



- Les paramètres qu'on a utilisé en entrée de hyperopt pour le Light GBM sont les suivants:

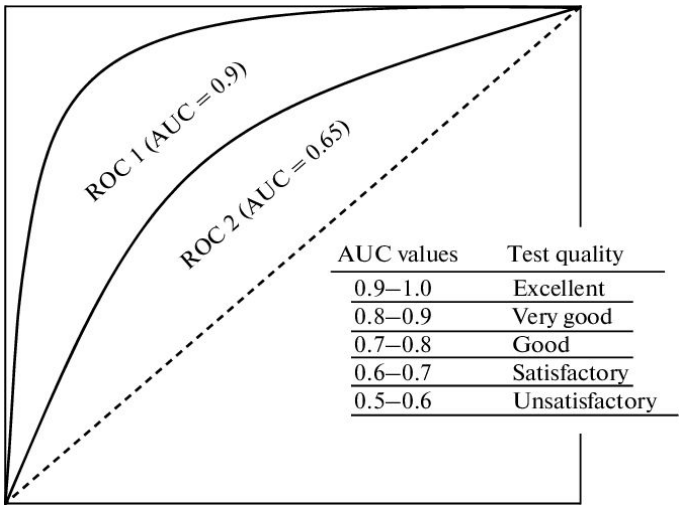
```
'model':            LGBMClassifier,
'num_leaves':       hp.choice('lgbm_num_leaves', range(6,50)), # max de feuilles par arbre
'min_child_samples': hp.choice('lgbm_min_child_samples', range(20,100)), # min de données par abre
'min_child_weight':  hp.uniform('lgbm_min_child_weight', 0.00001, 20),  # min de poids par arbre
'colsample_bytree':  hp.uniform('lgbm_colsample_bytree', 0.2, 1), # le ratio des colonnes tree
'reg_alpha':        hp.uniform('lgbm_reg_alpha', 0, 1), # régularisation L1
'reg_lambda':       hp.uniform('lgbm_reg_lambda', 0, 1), # régularisation L2
'n_estimators':     10000, # nombres d'arbres boostées à utiliser
'objective':        'binary', # objectif pour une classification binaire
'boosting_type':    'goss', # plus rapide que l'algorithme classique
'class_weight':     'balanced'
```

B. La fonction coût, l'algorithme d'optimisation et la métrique d'évaluation:

a. La fonction coût:

En effectuant une validation croisée pour notre algorithme, on peut directement définir une fonction de coût pour le Light GBM, on a utilisé la mesure AUC:

La mesure AUC, est l'aire sous la courbe ROC, qui représente le graphe des Vrais positifs (TP) en fonction des faux positifs (FP). Plus y a de vrais positifs, plus l'air sous la courbe sera grande, et plus la mesure AUC sera proche de 1.



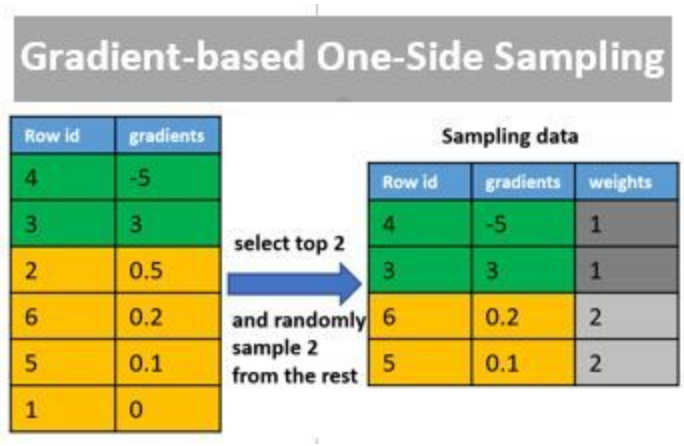
AUC values	Test quality
0.9–1.0	Excellent
0.8–0.9	Very good
0.7–0.8	Good
0.6–0.7	Satisfactory
0.5–0.6	Unsatisfactory

On a aussi utilisé une fonction de coût représentant les pertes potentielles de la banque, cette métrique sera détaillée dans la partie c) de cette section

b. Algorithme d'optimisation:

Pour optimiser notre modèle quelques implémentations ont été faites :

- Pour le Light GBM utilisation de l'algorithme GOSS : Gradient-based One Side Sampling, permettant d'enlever des features avec des gradients plus faibles, ce qui fait que le modèle tourne plus vite:



- Utilisation d'un early stopping, utilisant la mesure auc, et l'estimation de la perte de la banque pour éviter l'overfitting, ce qui veut dire que l'algorithme ne va pas utiliser tous les arbres boostés précisés dans le paramètre n_estimators du Light GBM, mais va s'arrêter quand le score n'évoluera plus (pendant 100 arbres)
- Finalement pour hyperopt, on arrête aussi les itérations dès que le meilleur score n'aura pas bougé pendant 10 itérations, ce qui permet d'économiser du temps de calcul.

c. Métrique d'évaluation:

Pour faciliter les explications, on va considérer la valeur cible 0 comme valeur positive, c'est à dire que le crédit est quelqu'un qu'on considère comme fiable, la valeur cible 1 est donc négative.

Un faux positif (FP) est quelqu'un qu'on considère comme quelqu'un qui va payer son crédit, mais qui le fait pas, et donc très dangereux pour la banque.

Un faux négatifs (FN) est quelqu'un qu'on considère comme mauvais client, mais à tort, ça fait perdre de l'argent à la banque moins que les FP.

Quelle mesure choisir ?

On pourrait utiliser le F**β** score :

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

$$F_\beta = \frac{(1 + \beta^2) \cdot \text{true positive}}{(1 + \beta^2) \cdot \text{true positive} + \beta^2 \cdot \text{false negative} + \text{false positive}}$$

Et en donnant au **β** des valeurs inférieurs à un, pour donner plus de poids aux faux positifs, ça peut être un bon indicateur, mais ce n'est pas une mesure métier qui pourrait parler à n'importe quel utilisateur de l'application, d'autant plus que tous

les faux positifs n'ont pas le même"poids", un client qui ne rembourserait pas la somme de 100 euros, est beaucoup moins impactant qu'un client qui ne rembourserait pas la somme de 100 000 euros.

Donc on va définir plutôt la métrique d'évaluation suivante :

- Si le client est un faux positif, il fait perdre potentiellement tout le prêt à la banque, dans ce cas notre métrique enverra -1 * montant total du crédit
- Si le client est un vrai positif, la banque va gagner sur les taux d'intérêt, notre métrique enverra + marge sur les taux gagnés
- Si le client un est faux négatif, la banque ne gagnera pas de marge sur les taux d'intérêt notre métrique enverra -1 * marge sur les taux gagnés
- et enfin pour un vrai négatif, on a détecté que le client est un mauvais payeur, et en lui donnera 0 en crédit, donc on enverra 0

Pour la marge sur les taux, une recherche sur internet sur les taux de marché, par rapport aux taux bancaires pratiquées donne une moyenne de 0.5%, donc on simule un coût de crédit avec cette marge et avec la formule ci dessous.

Avec ça on obtient un coût de crédit entre 10 et 20% du montant total, ce qui veut dire que la perte engendré par un FP est 5 à 10 fois supérieur à celle pour un FN.

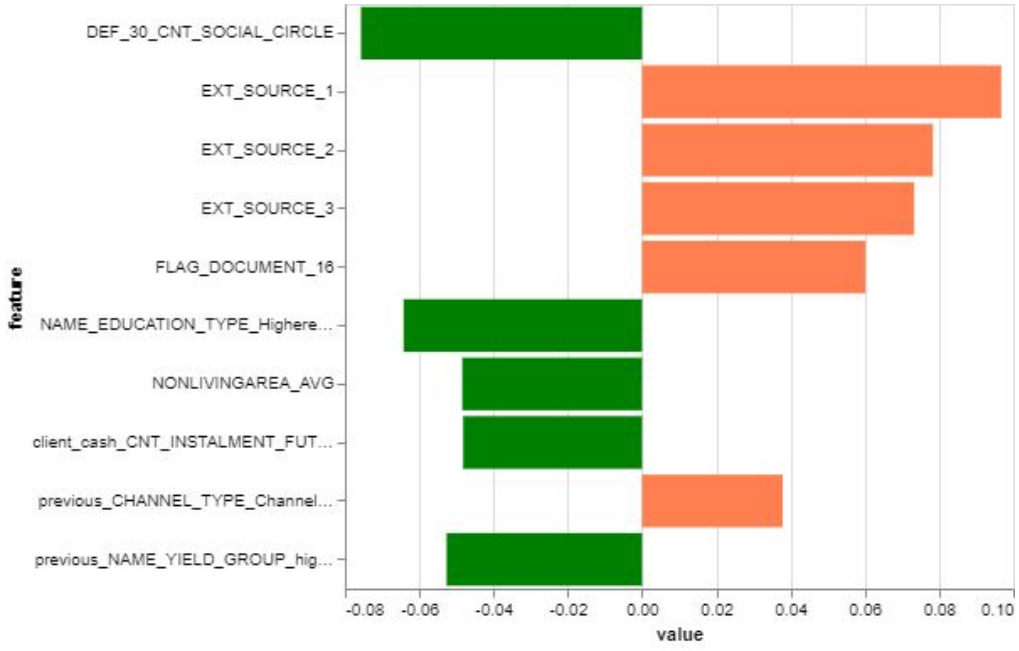
C= Capital emprunté (en €)
m = mensualité (en €)
d = Durée du prêt (nombre de mois de remboursement)
t = TAEG (Taux Annuel Effectif Goblal)

$$\textbf{Coût du crédit} = d \times m - C \quad \text{avec} \quad m = \frac{C \times t/12}{1 - (1+t/12)^{-d}}$$

C. Interprétabilité du modèle:

On a interprété le modèle en utilisant les feature importances du modèle, mais celles ci ne sont pas toujours adaptées, dans le cas du Light GBM par exemple, certaines features intégrées le sont à cause de la méthode GOSS justement, garde des features aléatoirement avec un gradient faible.

Une autre méthode est de considérer le modèle comme étant une boîte noire, c'est pour ça qu'on a utilisé LIME, qui est totalement agnostique au modèle, et qui utilise des surrogate models pour approximer le modèle dans une blackbox , LIME entraîne des surrogate models localements pour avoir des explications sur un individu en particulier, et c'est exactement ce qu'on voulait pour une application de crédit.



D. Limites et améliorations:

- La partie features engineering n'a pas été bien traité vu qu'elle se base purement sur des notebooks Kaggle, avec plus de connaissances métiers et un peu plus de temps on pourrait pousser cette partie plus loin ce qui devrait probablement améliorer notre modèle.

- A cause des restrictions matérielles, on n'a pas pu explorer beaucoup de modèles, ni utiliser des approches du genre du model stacking. Avec beaucoup plus de ressources on aurait pu pousser la recherche beaucoup plus loin, en utilisant des machines plus puissantes ou le cloud par exemple
- On n'a pas pu intégrer SMOTE dans un pipeline pour essayer de chercher les paramètres optimaux, toujours à cause de cette restriction optimale, il n'y a pas eu de vrai optimisation sur la partie SMOTE
- Dans la continuité du point précédent, on pourrait tester plus de paramètres, (comme le GBDT pour le Light GBM), et même différents paramètres de SMOTE / SMOTEEN
- Le modèle Light GBM malheureusement peut être compliqué pour l'interprétabilité, on peut pas en dégager facilement le pourcentage d'impact de chaque feature, encore une fois on pourrait résoudre ça en poussant la recherche dans les modèles encore plus loin
- LIME utilise des surrogate models, donc on n'interprète pas le modèle originale et on risque d'avoir une perte de données
- Notre métrique d'évaluation reste pas très précise, elle se base sur une approximation, on n'a pas le taux d'intérêt exact du crédit, on pourrait sortir une métrique beaucoup plus précise