Note méthodologique :

Ce projet a pour objectif de proposer à la fois un modèle de scoring de la probabilité de défaut de paiement du client et un dashboard interactif qui facilite les taches des chargés de relation client.

1- Les données :

J'utilise un jeu de données disponible sur le site de Kaggle. Il contient 10 fichiers de type CSV. Il propose des informations diverses qui constituent une base intéressante pour apporter des éléments de réponses aux objectifs de notre projet.

Pour gagner du temps, j'ai utilisé un Kernel Kaggle qui a déjà traité la partie « Ingénierie des fonctionnalités » et a obtenu un bon score.

J'ai gardé seulement la partie du jeu de données contenant la variable *TARGET*. J'ai également réalisé quelques traitements basiques à propos des données manquantes avant la standardisation et la division de mes données entre la partie Entrainement (70% des données) et la partie Test (30% des données). J'ai utilisé le paramètre *stratify* pour m'assurer que les deux classes sont équitablement réparties entre les parties.

2- Stratégie de modélisation

J'ai choisi l'algorithme *DummyClassifier* comme Baseline et *roc_auc_score* comme métrique d'évaluation.

Conscient du problème du temps de traitement lorsqu'on utilise un jeu de données aussi important comme le mien, j'ai adopté une stratégie qui se divise en 3 étapes. Dans chaque étape, je commence par entrainer le modèle avec les données d'entrainement, ensuite je réalise une prédiction avec les données de test et puis j'affiche ma métrique avec le temps d'exécution pour l'apprentissage du modèle.

Afin d'optimiser les hyperparamètres du modèle, j'utilise *GridSearchCV* avec un *KFold* = 5 et *scoring= roc_auc*. Je fais ici également la même chose que le modèle de base concernant l'apprentissage et la prédiction.

Réalisé et présenté par : Mohamed EL MOCTAR ELLAH
Le : 18/01/2021

Les trois étapes de ma stratégie se résument ainsi :

La première étape : je teste tous mes algorithmes en utilisant 1% du jeu de données.

Les algorithmes testés :

- *svm.SVC*
- *LogisticRegression*
- *RandomForestClassifier*
- *GradientBoostingClassifier*
- *LGBMClassifier*

Voici deux tableaux qui résument le travail et les résultats pour cette étape :

Algorithme	svm. SVC	Logistic Regression	RandomForest Classifier	GradientBoosting Classifier	LightGBM
Paramètres optimisés	C, gamma	C, penalty	max_depth, max_features, min_samples_leaf, min_samples_split	n_estimators, max_depth, max_features, min_samples_leaf, min_samples_split	n_estimators, max_depth, learning_rate, num_leaves

	Temps_normal	Roc_nrmal	Temps_grid	Roc_grid
svm.	8.07s	0.69	635.99s	0.71
L_Regression	0.89s	0.57	894.78s	0.50
RandomForest	2.61s	0.61	121.35s	0.61
GradientBoosting	15.92s	0.69	1914.10s	0.67
LGBM	4.25s	0.71	2707.60s	0.70

Les meilleurs algorithmes :

- *svm.SVC*
- *RandomForestClassifier*
- *LGBMClassifier*

A partir de cette étape, je sélectionne les trois meilleurs algorithmes pour l'étape suivante.

La deuxième étape : je teste les trois meilleurs algorithmes avec 10% des données. A la fin de cette étape je retiens le meilleur algorithme comme modèle final.

Les algorithmes testés :

- *svm.SVC*
- *RandomForestClassifier*
- *LGBMClassifier*

Le meilleur algorithme :

- *LGBMClassifier*

Voici le résultat de cette étape :

➢ svm.SVC ➢ RandomForestClassifier ➢ Lightgbm	• Auc = 0.68, Time = 2155,60 s • Auc = 0.69, Time = 57,24 s • Auc = 0.77, Time = 7661.16 s

La troisième étape : dans cette étape j'utilise toutes mes données afin de tester mon modèle final.

L'algorithme testé :

- *LGBMClassifier*

Le résultat de la dernière étape est :

Modèle finale : Lightgbm

- Auc = 0,7865
- Time = 229,10s

3- Enregistrement du modèle final

Le modèle final est sauvegardé à l'aide de *joblib*.

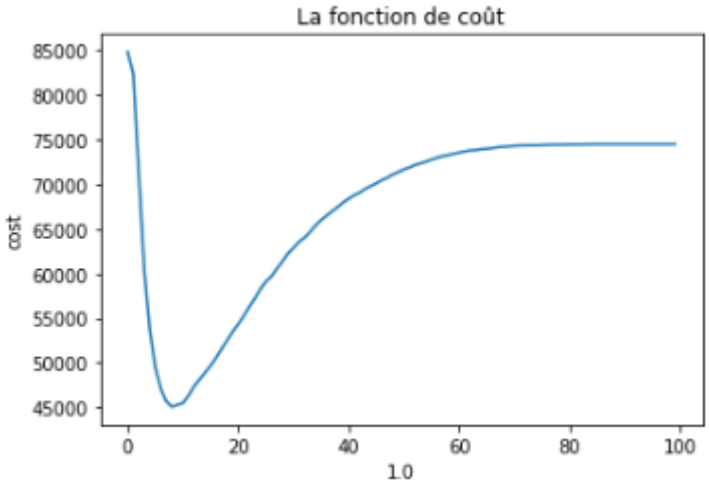
```
import joblib
# save the model to disk
filename = 'finalized_model.sav'
joblib.dump(lg, filename)
```

4- Fonction de coût

A partir de la matrice de confusion, je propose une fonction de coût qui prend en compte les faux positifs et les faux négatifs. Je suppose qu'un client faux positif représente un coût dix fois plus important qu'un client faux négatif.

Coût banque (seuil) = proportionnel à (10*FN + FP)

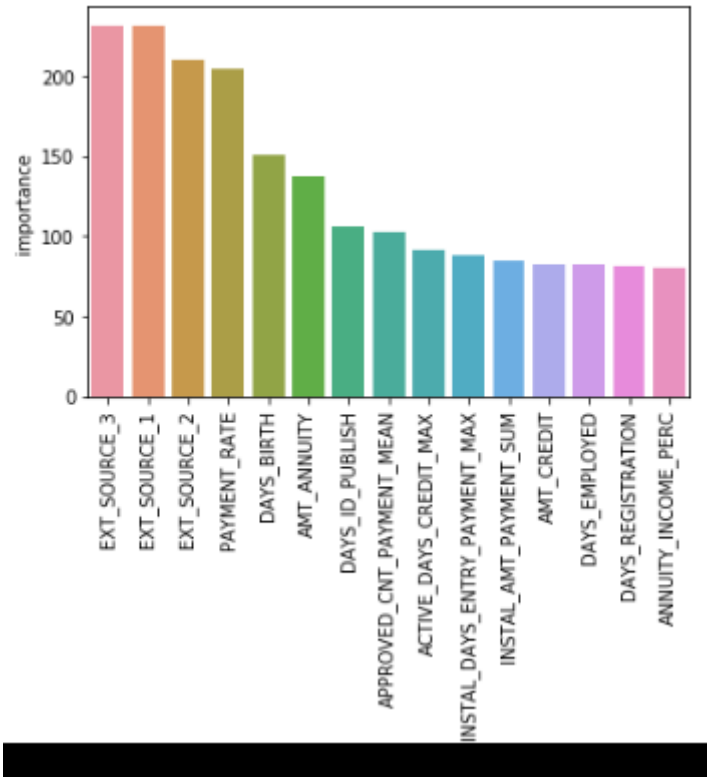
Cette fonction nécessite d'être minimisée pour pouvoir identifier le seuil qui permet de mieux connaitre les bons et les mauvais clients. Ce seuil est égal à 0,08.



5- Interprétabilité du modèle

J'ai affiché dans cette section les fonctionnalités selon leurs importances dans le modèle final.

Globalement, la plupart d'entre elles contribuent dans le résultat du modèle avec des niveaux d'importance variable.



6- Les limites et les améliorations possibles

- Les limites : j'ai testé la majorité des algorithmes avec un échantillon de données.
- Les améliorations possibles : il est possible de tester plus d'algorithmes et d'augmenter le nombre de paramètres à optimiser.
- Utiliser des méthodes telles que LIME, Anchors et SHAP pour bien interpréter les modèles.

Réalisé et présenté par : Mohamed EL MOCTAR ELLAH
Le : 18/01/2021