**Note Méthodologique du Projet 7 : Implémenter un modèle de scoring**

1. **Méthodologie d'entraînement du modèle**
2. **Préparation des fichiers train et test**

La préparation des données qui nous concerne ici consiste :

* En premier temps, à effectuer une jointure des fichiers train et test afin qu’ils aient subi les mêmes manœuvres.
* Sélectionner les variables pertinentes permettant de prédire la faillibilité du client.
* Mettre en évidence les valeurs aberrantes ou atypiques puis les traiter.
* Traiter des valeurs manquantes.
* Encoder des variables catégorielles.
* Effectuer du feature engineering manuel sur certaines variables à l’aide du noyau Kaggle fourni par Openclassrooms.

**Explication :**

* **Valeurs aberrantes ou atypiques :**

**Variables numériques :** Le traitement concerne principalement les variables de type temps. Il s’agit de les mettre dans la bonne grandeur : conversion en nombre positif et en années.

Le deuxième volet de cette section revient à corriger l’attribution de TARGET 1 à des clients aux revenus très élevés donc absence de risque de faillite.

**Variables Catégorielles :** Le traitement concerne que le **« CODE GENDER »,** qui compte trois modalités F, M et XNA dont « XNA » apparait que quatre fois seulement. Du coup les emprunts correspondants ont été supprimés.

* **Valeurs manquantes :** Il faut signaler qu’en data science, le phénomène des valeurs manquantes est très courant et se les débarrasser est une perte d’information immense ; de ce fait nous les avions traités par la méthode Iterrative imputer en fixant zéro comme valeur minimale.
* **Encodage des catégorielles :** Pour cette section, toutes les variables ont été encodé à chaud avec **get\_dummies** en binaire 0 ou 1 à l’aide du noyau de Kaggle.
* **Features engineering :** Quelques variables spécifiques au caractère financier et la durée de la vie active qui semblaient importantes ont été construites manuellement comme la proportion du temps travaillé par rapport à l’âge du client, ou le ratio crédit/revenu ou encore revenus par personne, le pourcentage de jours employés par rapport à l'âge des clients etc. Il est cependant important de remarquer nous avions construit ces variables en s’appuyant toujours sur le noyau de KAGGLE.

1. **Entrainement et sélection de modèle**

En apprentissage automatique, il existe plusieurs modèles, des plus simple au plus complexes, des moins performants aux autres.

Sur ce, nous avons choisi six modèles qui sont les plus classiques à savoir :

**DummyClassifier,**

**LogisticRegression,**

**RandomForestClassifier,**

**GradientBoostingClassifier,**

**XGBClassifier,**

**LGBM,**

Puis après optimisation des hyper params et entrainement de ces modèles, on choisira le meilleur en se basant sur le score AUC.

* **Entrainement :**

**Préparation des données :** Cette première étape consiste à sélectionner aléatoirement 30 % des jeux de données nettoyées sur lesquelles on a ensuite équilibré artificiellement avec SMOTE les différents états de faillibilité des clients en nombre égal. Puis séparer les données en X et Y où Y représente la cible et X les autres.

**Remarque** : Dans le seul but de réduire le temps d’exécution, nous avions sélectionné ces 30% pour représenter l’ensemble de nos données.

**Création du preprocessor :** Encore une fois, en apprentissage automatique, trouver les moyens de faire le travail sereinement en quelque ligne de code doit être rêve de beaucoup, et nous estimons que Pipeline fait partir de ces moyens. Sur ce, nous avions créé un preprocessor qui va être solliciter pour chaque modèle afin d’appliquer une normalisation à toutes nos données ou encore permettre d’extraire les noms de features pour les importances.

**Séparation en train et test :** Training data avec 70 **%** pour l’entrainement du modèle et test data 30% pour la validation.

**Optimisation des paramètres :** Pour chaque modèle excepté DummyClassifier, nous avions optimisé les meilleurs paramètres sur une Grille GridSearch avec Cross Validation.

* **Sélection de modèle :**

Le choix du meilleur modèle réside dans sa capacité à s’adapter aux données qui lui ont été envoyées. Ici, il ressort après optimisation et calcule du score AUC établi après la construction d’une fonction coût également optimisé, que le modèle LogisticRegression semble mieux apprendre et donc plus performant.

1. **Fonction coût métier**

**Construction :**

Concernant la fonction coût métier qui sera définie par rapport au gain et perte que cela rapporte à la banque après ses décisions de prêt. On remarque également que dans notre cas, les bons clients prédits mauvais coûtent plus chers que les mauvais clients prédits bons. Du coup, nous avions pénalisé dans la fonction coût, ces Faux positifs et Faux Négatifs. Par conséquent nous théorisons les poids des variables ainsi :

**Tn\_value= 0, Tn\_value = 10, Fp\_value = 1, Tp\_value = 0**

Chaque **vrai négatif** (TN, bons client classé bons) signifie que nous pouvons vendre **X dollars** de produits de crédit supplémentaires à des clients qui ne feront pas défaut équivalent à un poids de 0 car ils ne génèrent pas de perte à la banque.

Chaque **faux positif** (TP, mauvais classé bon) nous coûte **Y dollars** car nous aurions pu engager ces clients, mais au lieu de cela, nous les avons évités, équivalent à un poids positif de 1 car ils engendrent une perte.

Chaque **faux négatif** (FP, bons classé mauvais) nous coûte **Z dollars** parce que nous n'avons pas réussi à intervenir et à empêcher ces clients de faire défaut équivalent à un poids positif de 10, car ils peuvent générer un gros manque à gagner.

Chaque **vrai positif** (TP, mauvais classé mauvais) nous aide à prévenir les défaillances de crédit et chacun de ces clients génèrent **β dollars** de nouveaux revenus équivalent à un poids 0, ils n'engendrent pas de perte.

**Optimisation :**

Dans un premier temps, nous stockons tous les seuils de probabilités dans thresholds où devons par la suite calculer, pour chacune des valeurs correspondantes, les valeurs prises par la fonction. On récupère la valeur minimale de cette fonction coût qui nous servira également à retourner le seuil optimal de probabilité correspondant.

**Seuil de faillibilité ou solvabilité :**

Le modèle retourne un score entre 0 et 1 et par défaut il attribue la classe 1 lorsque le score est supérieur à 0.5 et 0 sinon. Ainsi pour déterminer à quel seuil un client sera susceptible d’honorer son crédit ou non. Par conséquent, le seuil optimal déterminé vaut **0.5382**, c’est-à-dire que si le score retourné par le modèle est supérieur au seuil optimal, le client est déclaré en faillite sinon non en faillite.

1. **L’interprétabilité globale et locale du modèle**

Concernant l’interprétabilité du modèle, on remarque qu’une augmentation de la complexité implique l’importance de 'interprétabilité du modèle. Nous avions utilisé la librairie Lime pour rendre interprétable le modèle LogisticRegression localement.

1. **Interprétabilité globale :**

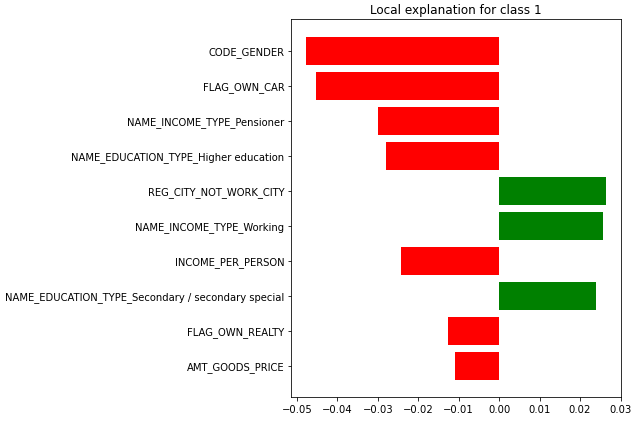
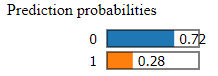
L’interprétabilité globale cherche à identifier les variables les plus importantes du modèle notamment ceux qui ont plus d’impact sur la prédiction en les attribuant un coefficient selon la contribution de chaque variable sur les données de sortie du modèle. Ici, l’interprétabilité globale permet à la banque de comprendre finement les critères et les variables que le modèle utilise pour estimer le risque de crédit associé à tel ou tel type de profil sans pour autant préciser la nature de l’importance autrement positivité ou négativité pour risque de crédit.

1. **Interprétabilité locale (Lime) :**

* **Principe de fonctionnement :**

LIME fonctionne en modifiant localement l'entrée du modèle. Ainsi, au lieu d'essayer de comprendre l'ensemble du modèle en même temps, une instance d'entrée spécifique est modifiée et l'impact sur les prédictions est surveillé. Dans le contexte de la classification des clients, cela signifie que certains des variables sont par exemple remplacés, pour déterminer quels éléments de l'entrée ont un impact sur les prédictions.

* **Exemple de sortie de Lime :**

****

**Interprétation :**

Ce client sera considéré comme étant **non en faillite** au regard des information sur le graphique. Ainsi :

Les variables qui jouent plus en sa défaveur sont :

* **REG CITY NOT WORK CITY** : Ne pas travailler dans la ville qu'il habite
* **NAME INCOME TYPE Working** : Type de contrat de travail.
* **NAME EDUCATION TYPE SECONDARY / secondary special** : Niveau d'éducation spécial

Et les variables qui jouent plus en sa faveur sont entre autres :

* **CODE GENDER** : Un homme
* **FLAG OWN CAR**
* **NAME INCOME TYPE PENSIONER** : Revenu de pension

1. **Les limites et les améliorations possibles**

* **Performance du modèle :**

Si la performance d’un modèle de score ne peut pas beaucoup s’améliorer en augmentant le nombre de variables explicatives, il serait judicieux de faire la partition (ou stratification) du modèle qui consiste à faire **précéder la modélisation** d’une classification de la population, puis à construire un modèle différent pour chacune des classes, avant d’en faire la synthèse.

Cette partition est intéressante car il permet **de travailler sur des groupes homogènes,** nécessitant moins de variables pour les décrire.

De même, on peut songer aux agrégations par **bagging ou boosting** qui améliorent les performances et réduisent la variance.

* **Interprétabilité du model :**

SHAP est une approche utilisée dans la théorie des jeux ; il relie l’explication locale de l’allocation optimale de crédits à l’aide des valeurs de Shapley et peut donc compenser les défauts de la méthode LIME comme son instabilité et le fait qu’un modèle local simple n’est pas une bonne approximation du modèle global.

* **Transformation des variables :** Il semble peut-être important de discuter avec un expert du domaine concernant les variables à transformer permettant de bien décrire la situation et donc réduire les fausses prédictions ou encore faciliter la prédiction des clients difficiles.

**Mot de fin :** Si comprendre un modèle est un impératif dans toute démarche scientifique, les contraintes juridiques imposent quelque fois de ne pas prendre une décision basée uniquement sur le résultat d’un algorithme automatique, donc le bon sens de l’esprit humain doit valider les décisions prises par les machines.