Note Méthodologique – Projet OpenClassrooms ***Implémentez un modèle de scoring***

[Parcours Data Scientist – P07 – Par Brice Ouizeman – Juin 2022]

1. **Objectif de la note et présentation de la mission**

Cette note méthodologique a pour objectif de présenter le modèle développé dans le cadre de notre mission, et son interprétabilité.

La mission consiste à accompagner *Prêt à dépenser* qui souhaite mettre en œuvre un outil de “scoring crédit” pour calculer la probabilité qu’un client rembourse son crédit, puis classifier la demande en crédit accordé ou refusé.

Le pré-traitement et la création des variables ont été élaborés à partir des kernels publiés par [Will Koehrsen](https://www.kaggle.com/code/willkoehrsen/start-here-a-gentle-introduction/notebook)

1. **Caractéristiques de notre jeu de données**

Caractéristiques des principaux traitements :

* L’encodage des variables catégorielles utilise le « Label encoding » jusqu'à 2 modalités et le « One-Hot Encoding » au-delà.
* Créatino de nouvelles variables, notamment polynomiales, pour enrichir la table principale
* Le centrage-réduction est réalisé au travers d’un « Standard Scaler ».
* 2 méthodes de réduction de dimension : par importance des variables d’un modèle de classification « Gradient Boosting » et par ACP.
* Définition de la graine sur le générateur aléatoire utilisée lors des séparations « train-test » pour rendre nos résultats déterministes.

Caractéristiques du jeu d’apprentissage :

* 20% du jeu d’entraînement initial est isolé pour constituer le jeu de validation.
* Le jeu d’entraînement est échantillonné aléatoirement permettant de passer de 246 000 à 5 000 individus (pour une question de ressources).
* Rééquilibrage de la variable cible avec la méthode SMOTE « Synthetic Minority Over-sampling Technique »
* 11 variables

1. **Entrainement du modèle**

Nous utilisons Pycaret pour sa fonctionnalité de comparaison de modèles côte-à-côte (setup, compare\_models), pour la possibilité d’améliorer facilement les hyperparamètres du modèle sélectionné et pour la qualité de ses visualisations (plot\_model).

Nous avons simplement précisé la liste des modèles à comparer, la métrique retenue pour le tri et le nombre de décimales à afficher. Les autres paramètres ont leur valeur par défaut.

1. **Fonction coût, algorithme d’optimisation et métrique d’évaluation**

L’objectif de *Prêt à dépenser* est avant tout de réduire les refus sur les bonnes demandes et minimiser les accords sur les mauvaises demandes. Pour réaliser le premier objectif, la fonction coût métier (proportionnelle au montant de la demande) pénalise donc 2 fois plus les refus sur les bonnes demandes que les accords sur les mauvaises demandes :

Une image contenant table

Description générée automatiquement

L'algorithme d'optimisation commence par la déclaration d’une fonction de calcul des coûts de chaque demande avec, en argument, le vecteur des classes cibles, le vecteur des classes prédites et un dictionnaire de vecteurs des paramètres supplémentaires kwargs (*le dictionnaire n’aura qu’une clé ‘amt’ car nous n’utilisons que le montant de la demande*).

Ensuite, nous déclarons une fonction de coût total comme étant la somme des coûts des demandes.

Avec la fonction « add\_metric » de Pycaret, nous ajoutons la fonction de coût total comme métrique d’évaluation tout en transmettant le dictionnaire des paramètres additionnels.

Nous pouvons lancer la fonction « compare\_model » de Pycaret avec sort='Cout' afin d’identifier le meilleur type de modèle. Nous réglons par la suite les hyper-paramètres avec la fonction « tune\_model » : optimize='Cout' (pour favoriser la minimisation de la fonction coût).

* Nous pouvons ainsi calculer la prédiction, la probabilité de défaut et le coût de chaque demande.

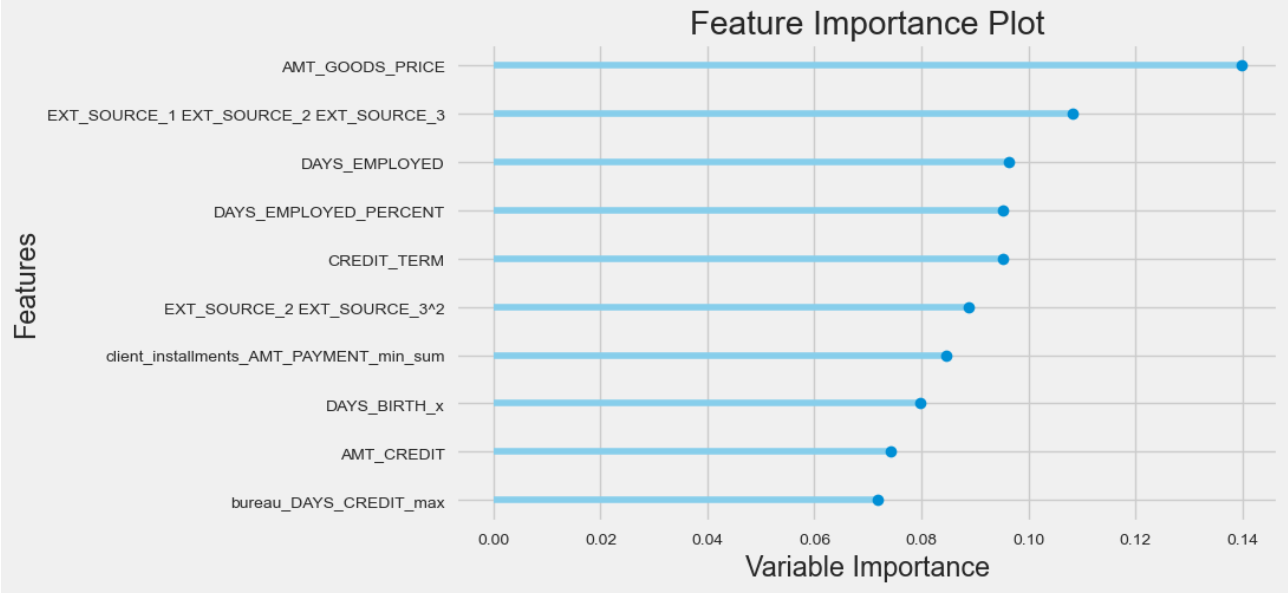
1. **Métriques d’évaluation**

Nous avons utilisé plusieurs métriques d’évaluation pour sélectionner notre modèle, notamment :

* **F1 score** : il permet d’évaluer la capacité d’un modèle de classification en faisant un compromis entre la Precision et le Recall.
* **AUC score** : l'aire sous la courbe (AUC) permet de déterminer l’efficacité d’un modèle. Ainsi, le modèle le plus efficace a une AUC égale à 1, et le modèle le moins efficace a une AUC égale à 0,5.
* **Kappa Score** : le coefficient de Kappa sert à mesurer l’accord entre deux méthodes de classification.
* **MCC Score** : Le MCC est un coefficient de corrélation pour la cible et les prédictions et varie entre -1 et 1 (meilleur accord).

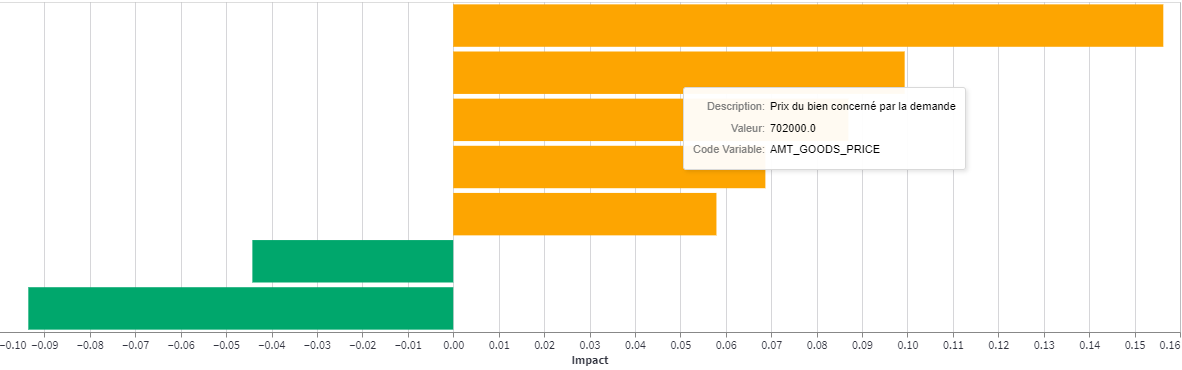
1. **Interprétabilité globale et locale du modèle**

**L’interprétabilité globale** traite de l’importance des variables issue du modèle optimisé. Sa représentation graphique est immédiate avec la fonction « plot\_model » de Pycaret et le paramètre plot='feature' :



L’interprétabilité locale permet d’expliquer la prédiction pour chaque client. **La contribution de chaque variable à la prédiction du client est la valeur normalisée de la variable multipliée par l’importance globale de la variable correspondante**.

Un impact positif, en orange, signifie que la variable fait accroître la probabilité de défaut et un impact négatif, en vert, signifie que la variable fait décroître la probabilité de défaut.



**5. Pistes d’amélioration :**

* Pour faciliter le travail sur notre modèle, j’ai volontairement – après plusieurs essais compliqués – réduit le nombre d’entrées. Avec plus de ressource ou de temps il aurait été possible d’améliorer la prédiction du modèle avec de nouvelles variables et un dataset d’entrainement plus important.
* La gestion des valeurs manquantes peut également être améliorée (nous avons éliminé les colonnes avec plus de 90% de données manquantes et remplacé les autres par la médiane).
* Il est possible d’améliorer la fonction cout en affinant la proportionnalité des combinaisons et avec l’aide du métier.