**Note Méthodologique**

Projet 7 parcours Data Scientist : Implémentez un modèle de scoring

Wenzel Nicolas

**Introduction :**

Ce document vise à décrire les différentes étapes de la modélisation effectuée sur le projet 7 « implémentez un modèle de scoring » et sera divisé en plusieurs partie :

On abordera dans un premier temps abordée la préparation des données et la méthodologie d'entraînement des modèles. Un second point sera consacré à l’optimisation des modèles, leurs évaluations ainsi qu’à la fonction coût-métier considérée. Enfin, la dernière partie du document couvrira l’interprétabilité du modèle avant de conclure sur les limites et axes d’améliorations possibles.

**Méthodologie d'entraînement :**

Après avoir utilisé le Kernel de traitement de données proposé en ressource du projet pour effectuer les tâches de Data Engineering, plusieurs étapes de nettoyage des données ont été réalisées en amont de l’entraînement :

Tout d’abord, les colonnes avec trop de valeurs manquantes ont été éliminées de même que les colonnes contenant des informations redondantes (par analyse de corrélations).

Afin de préparer les données à l’entraînement, un scaling et un équilibrage des classes via SMOTE ont été effectués.

Après ces étapes : la stratégie d'entraînement et de sélection du modèle a été effectuée par cross validation sur 5 plis avec pour métrique de mesure la précision après avoir sélectionné un paramétrage pour chaque algorithme via gridsearch.

Ce choix de métrique semble naturel dans un contexte où les erreurs de prédiction de type faux positif ont un impact négatif important pour l’application du client.

Dans cette phase d’investigation, plusieurs modèles non optimisés ont été entraînés afin d’évaluer sommairement quel paradigme semble le plus pertinent pour notre cas d’usage. En particulier ont été entraînés :

* un classifieur probabiliste (type Gaussian Naive Bayes)
* un KNN
* une régression logistique
* une SVM
* un arbre de décision
* une RandomForest

Étant donné l’emphase mise sur l’explicabilité du modèle, j’ai fait le choix d’écarter les réseaux de neurones, moins adaptés à notre situation.

Après cette étape il est apparu clairement que les algorithmes type RandomForest étaient les algorithmes les plus performants.

En plus des performances de l’algorithme, les modèles de type RandomForest sont très lisibles, faciles à expliciter et sont donc particulièrement indiqués pour notre problématique.

**Optimisation Evaluation et fonction coût-métier**

La phase suivant la sélection du modèle consiste à l’optimiser afin de pouvoir l’adapter au mieux à notre cas d’usage. A cette fin, j’ai choisi d’utiliser le package Python “Optuna” permettant l’optimisation des modèles implémentés sur SickitLearn plus rapidement et flexiblement que le permet une optimisation par GridSearch par exemple.

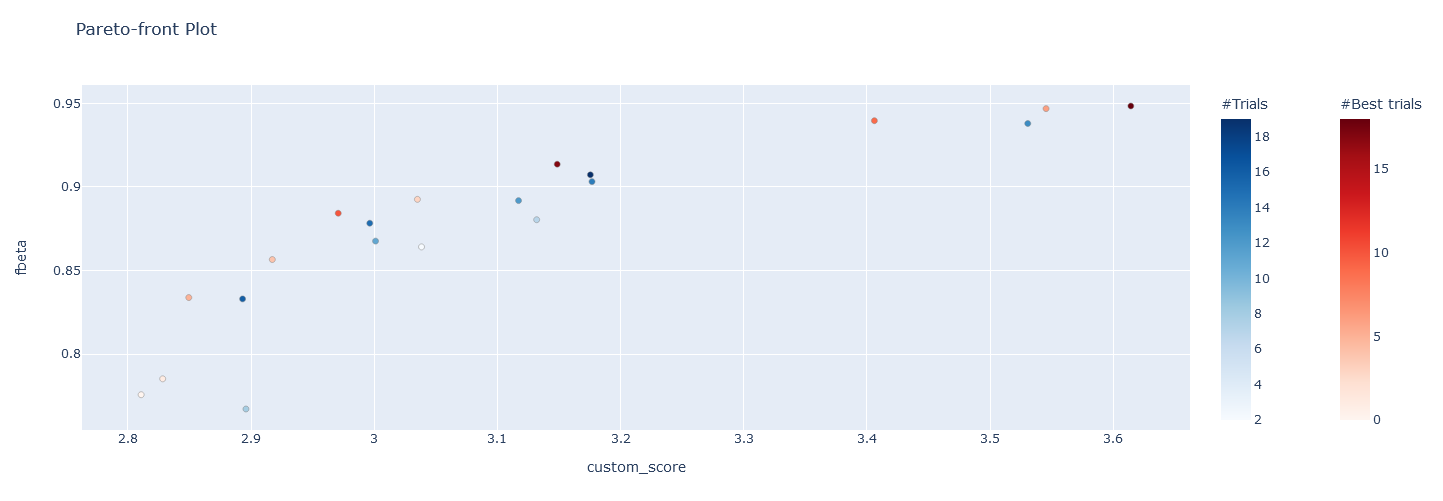
Afin d’effectuer cette optimisation, la première étape a été d'effectuer une recherche documentaire des paramètres pertinents à optimiser et des intervalles à utiliser pour cette optimisation. A l’issu de cette documentation le choix d’optimisation s’est porté sur les hyperparamètres et intervalles associés suivant :

* nombre d’estimateurs
* profondeur maximale de l’arbre
* nombre minimum d'échantillons par split
* nombre minimum de feuilles par split

Une fois les paramètres fixés, nous devons choisir une ou plusieurs métriques sur la base de laquelle optimiser notre système. Pour notre cas d’usage, j’ai décidé d’optimiser le modèle sur la base de deux métriques :

* d’une part le fbeta score: mesure prenant en compte à la fois la précision et le recall et paramétrable permettant ainsi de jouer sur l’optimisation pour faire varier les caractéristiques du modèle (notamment les features importantes). Ici, la fonction fbeta score a été paramétrée pour mettre l’emphase sur la précision
* Une métrique d’évaluation originale plus proche du cas d’usage. Cette fonction coût est définie de manière suivante : tout faux négatif génère 1 point d’erreur, tout faux positif en génère 5. Ce score est ensuite normalisé en le divisant par le nombre d'erreurs commises par le modèle.

Afin de sélectionner le paramètre optimal au regard de ces deux métriques, j’ai tracé un front de Pareto permettant de visualiser quels paramètrages offrent un compromis satisfaisant entre les deux fonctions d’optimisation.



La décision finale concernant le paramétrage de l’algorithme s’est porté sur la configuration suivante :

Max\_depth =32, nb\_estimator = 60, min\_sample split =14, min\_samples\_leaf =1

Cette configuration permet le compromis le plus satisfaisant entre nos deux métriques d’évaluation.

Il est aussi pertinent de noter que l’utilisation d’un front de Pareto et d’un score paramétrable comme métrique d’évaluation permet de proposer plusieurs solutions optimisées à un décideur lui permettant de se projeter dans différents scénarios et de choisir selon sa vision de l’application métier. Cette perspective peut s’avérer particulièrement intéressante si les différentes configurations de modèles utilisent des features différentes pour faire leur choix et permet d’orienter le décideur vers la solution la plus satisfaisante pour son application métier.

En plus de l’optimisation en fbeta score et fonction métier, les performances de l’algorithme ont aussi été mesurées en faisant varier le seuil de probabilité auquel la décision est prise. L’algorithme entrainé rend une prédiction binaire avec une probabilité associée >0.5 mais il est possible d’estimer qu’une prédiction à 0.53 de probabilité de remboursement n’est pas assez robuste pour notre application métier et cette évaluation permet encore une fois de présenter plusieurs scénarios de projection à un décideur.

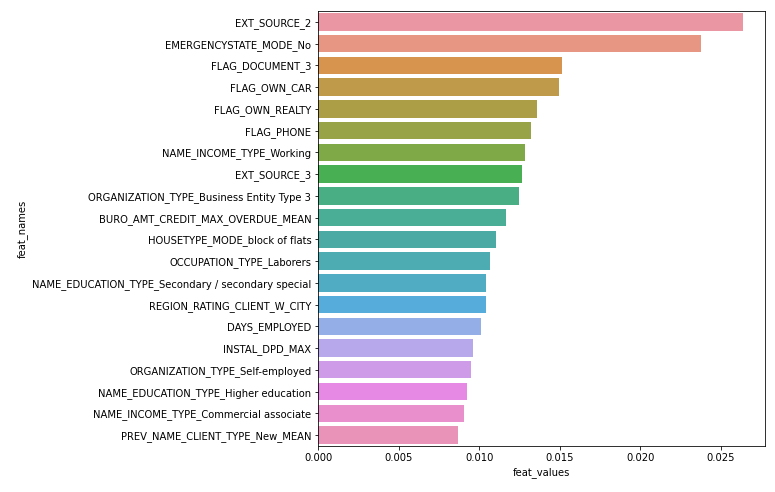
**Interprétabilité**

Maintenant que la configuration du modèle (ou du moins une version de celle-ci) est fixée, il nous faut nous intéresser à l'explicabilité de celui-ci.

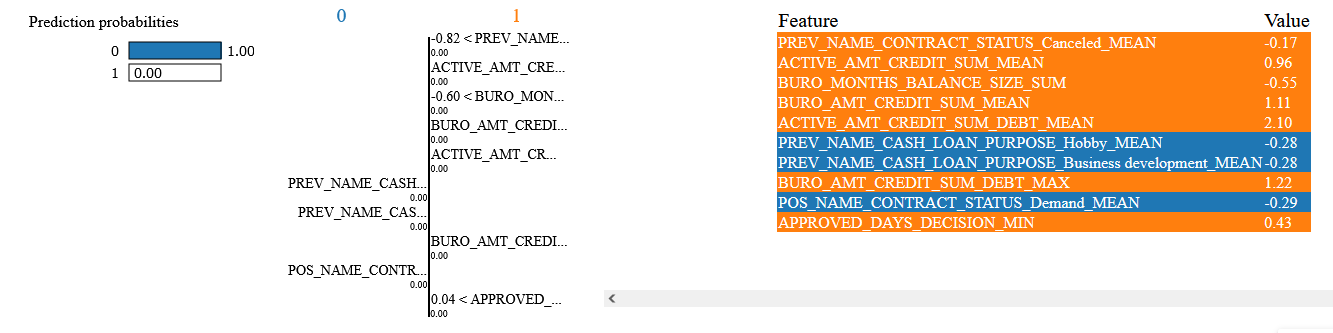
Deux niveaux d'intérêt sont à dégager de cette notion : l’explicabilité globale et l’explicabilité locale.

Le premier niveau consiste à extraire de notre modèle les critères généraux guidant ses décisions de classification. Ayant implémenté l’algorithme de classification via la librairie sickit-learn, les instances de modèles possèdent une méthode retournant l’importance des features pour la prise de décision. J’ai décidé d’encapsuler cette méthode dans une fonction permettant d’isoler les dix features prédominantes et de tracer sur un graphique leur importance relative sur l’interface graphique utilisateur.

D’autres méthodes inhérentes aux modèles sickit-learn sont aussi disponibles pour retourner les chemins de décision et visualiser les arbres utilisés mais j’ai choisi d’utiliser uniquement l’importance des features afin de garder une information claire et lisible suffisante pour un utilisateur néophyte.



Le niveau d’explicabilité locale quant à lui consiste à expliciter une décision de l’algorithme sur un échantillon particulier de notre base de données. Afin de pouvoir accéder à ces informations j’ai utilisé la librairie Python Lime spécialisée pour l’explication de modèles de machine learning. Grâce aux fonctionnalités de la librairie, il m’a été possible pour les clients de mon choix de générer une représentation graphique de la probabilité d’appartenance à chaque classe ainsi qu’un tableau illustrant quels ont été les features déterminantes pour son cas particulier ainsi que leurs valeurs et seuils associés.

****

**Limites et axes d’améliorations**

Le résultat obtenu après ce cycle de développement me paraît satisfaisant mais il est évident que plusieurs limites ont été atteintes et que plusieurs pistes d’améliorations restent à explorer afin de proposer la solution idéale.

Comme indiqué dans le projet, le feature engineering a été effectué via un kernel Kaggle et n’a pas été exploré par ailleurs. La première piste d’amélioration serait donc de reprendre le travail de feature engineering pour créer une base d'entraînement plus intelligente et plus créative.

Une autre limite rencontrée a été la puissance disponible pour entraîner et optimiser les modèles. Tout le processus de développement a été effectué sur google colab avec des ressources limitées. Dans ces conditions, l'entraînement et l’optimisation des algorithmes ont pris un temps considérable et afin de fournir un résultat dans un temps acceptable il a fallu faire des choix (sur le nombre de d’hyperparamètres à optimiser par exemple).

Cette limitation est une des raisons m’ayant poussé à optimiser uniquement un type d’algorithme et il est évident qu’un travail plus exhaustif permettrait d’obtenir une meilleure vision des capacités de chaque algorithme pour notre cas d’usage.

Le choix des métriques d’évaluation est lui aussi un choix pouvant être amélioré. La métrique métier utilisée est très caricaturale et il est évident qu’on pourrait l’améliorer pour mieux rendre compte de la réalité.

Enfin, les solutions de Deep Learning ont été laissées de côté lors de l’investigation des algorithmes en vertu de leur faible explicabilité. Cependant, les moyens d’expliciter les décisions de ces modèles sont de plus en plus accessibles et une exploration de ces solutions pourrait mener à de meilleures performances.