# OC DS P7 - Implémentez un modèle de scoring

## Note méthodologique

Grégoire MANSIO

### Méthodologie d'entraînement du modèle

Comme proposé par notre manager, le travail se base sur le notebook Kaggle d’Olivier G disponible à l’adresse suivante (<https://www.kaggle.com/code/ogrellier/lighgbm-with-selected-features/log>). Après exécution de celui-ci (hormis la partie modélisation), on obtient un dataset complet (tous les sous-jeux de données ont été merge), et prêt à être nettoyé. On fait un traitement des valeurs infinies (pas des NaN) avant de séparer le dataset en 3 sous-set : train, validation, et test (ce dernier ayant déjà été distingué à la source). On va donc entrainer et valider nos modèles sur le duo train/validation.

Etant donné le nombre de valeurs manquantes très important dans le dataset (30%), on implémente une méthode d’imputation SimpleImputer qui va remplacer chaque NaN par la médiane de la variable à imputer. En effet certaines variables sont très peu renseignées et/ou présentent un maximum et un écart-type très importants. Pour imputer, nous préférons donc utiliser la médiane que la moyenne. En outre la présence de ces outliers nous invite à choisir un RobustScaler pour standardiser les séries car celui-ci y est peu sensible, contrairement au StandardScaler qui va utiliser la moyenne et l’écart-type, ou le MinMaxscaler qui (comme son nom l’indique) tiendra compte de l’intervalle complet. A noter que ces opérations sont toutes deux intégrées dans un pipeline avec le modèle de classification. Concernant les caractéristiques de l’entrainement initial des modèles, nous avons choisi une stratégie de cross-validation (KFold) en 5 sous-échantillons, et notre métrique de comparaison est l’AUC.

Nous avons préalablement retenu 3 modèles de classification afin de faire un pré-sélection plus rapide parmi ces trois-là. Tout d’abord un RandomForest comme modèle baseline. Nous aurions pu partir d’une régression logistique mais il était évident que le Random Forest performerait déjà mieux que la régression. A noter que c’est le modèle de présélection le plus long avec 2329 secondes d’entrainement (près de 40mins). En second et troisième modèles nous avons retenu un XGBoost et un LGBM. Ces deux derniers sont assez ressemblants (modèles de gradient-boosting) mais sont parmi les modèles de classification (et de régression) les plus performants. Le seul inconvénient des modèles basés sur les arbres de décision (nos trois modèles) est la probabilité d’overfitting. Cependant, nos résultats initiaux (pré-optimisation des hyperparamètres) semblent exclure cette éventualité. A noter que XGB et LGBM sont tous deux bien plus rapides que RandomForest avec des temps d’entrainements respectifs de 814 et 183 secondes. Le dernier modèle : Light Gradient Boosting Machine est le plus performant en terme d’AUC et de temps d’entraînement, et nous retenons donc celui-ci comme modèle principal. A noter que pour la sélection du meilleur modèle, il n’était à nos yeux pas nécessaire d’effectuer une recherche d’hyperparamètres puisqu’il s’agissait simplement d’obtenir un regard global sur le modèle à choisir avant d’aller plus loin.

### Méthodologie d'optimisation du modèle

Dans cette section nous allons aborder le traitement du déséquilibre des classes, le choix d’optimisation du modèle, la fonction coût métier, ainsi que la métrique technique d’évaluation.

À la suite du choix du modèle LGBM, nous avons mis en place une fonction de recherche d’hyperparamètres intégrant notre pipeline précédemment expliqué (RobustScaler + SimpleImputer) mais aussi un algorithme SMOTE pour poursuivre le prétraitement. Il semblerait que le dataset présente un fort déséquilibre entre les classes minoritaires et majoritaires : certains individus « à part » sont sous-représentés et seront donc mal expliqués par le modèle. Une solution est donc d’appliquer une Synthetic Minority Over-sampling Technique qui va créer des combinaisons linéaires entre les k-plus proches voisins des individus minoritaires, et ce jusqu’à respecter un ratio de représentation : Individus\_de\_classes\_minoritaires / Individus\_de\_classes\_majoritaires = X, X étant choisi par l’utilisateur. Nous avons retenu 0.2, signifiant que nous voulons après re-sampling un nombre d’individus de classes minoritaires 5 fois moins grand qu’un nombre d’individus de classes majoritaires.

Nous avons utilisé le package ‘hyperopt’ afin de procéder à l’optimisation de notre modèle. L’avantage de ce package est de proposer une recherche d’hyperparamètres dans un intervalle fermé, et non dans une liste de paramètres discrets et souvent proposés par défaut par l’algorithme de régression/classification. Mis à part le learning\_rate où nous choisissons une fonction de répartition logarithmique pour définir notre espace de recherche, les espaces des autres paramètres sont tous définis par une fonction de répartition uniforme. Enfin, la recherche du meilleur paramètre de chaque espace est faite automatiquement par un estimateur appelé Tree-structured Parzen Estimator (ou TPE) qui va travailler en arrière-plan en optimisant lui-même la recherche.

Toute cette recherche de paramètres en encadrée par une minimisation d’une fonction objectif. On en propose deux différentes dont on va comparer les résultats. Dans un premier temps, on optimise le modèle selon le critère du F1-Score. Ce dernier juge de la capacité du modèle à bien prédire les individus positifs à la fois via la précision et le recall. A noter que le temps d’entrainement de ce modèle a été de 7192 secondes. Dans un second temps on optimise notre modèle selon la fonction coût métier. Celle-ci est duale. Nous avons d’une part une maximisation du profit c’est-à-dire le nombre de bons clients effectivement sélectionnés (TN[[1]](#footnote-1)) auquel on retire les mauvais clients effectivement sélectionnés (FN) et auquel on retire aussi les bons clients malheureusement non sélectionnés (FP). Cette première partie peut donc être résumé par : . En outre, nous avons été informés que retenir un individu mauvais-payeur est dix fois plus dommageable pour l’entreprise que ne pas retenir un bon payeur. Ceci équivaut à l’équation . On optimise donc une nouvelle fois le modèle mais avec ce système d’équations comme fonction objectif. Le temps d’entrainement est toujours très élevé avec 5129 secondes.

Nous discuterons dans la section suivante des résultats et des limites de notre travail.

### Résultats, interprétabilité, limites et améliorations possibles

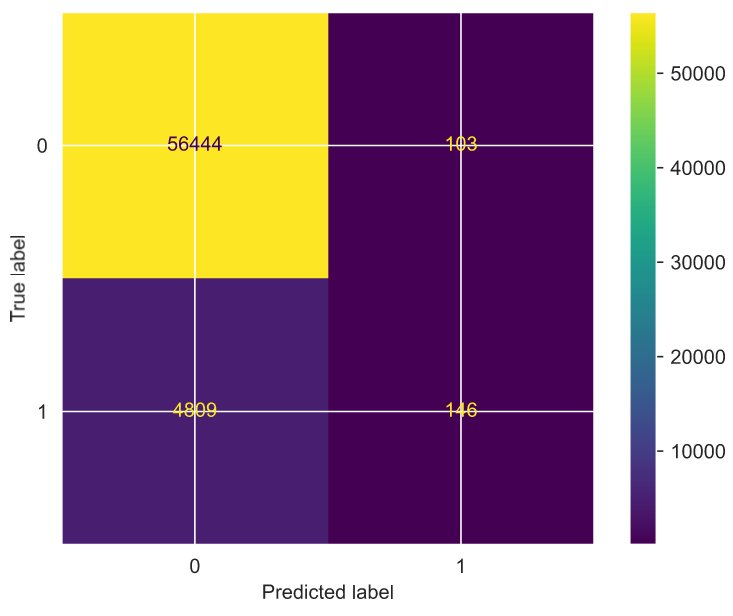
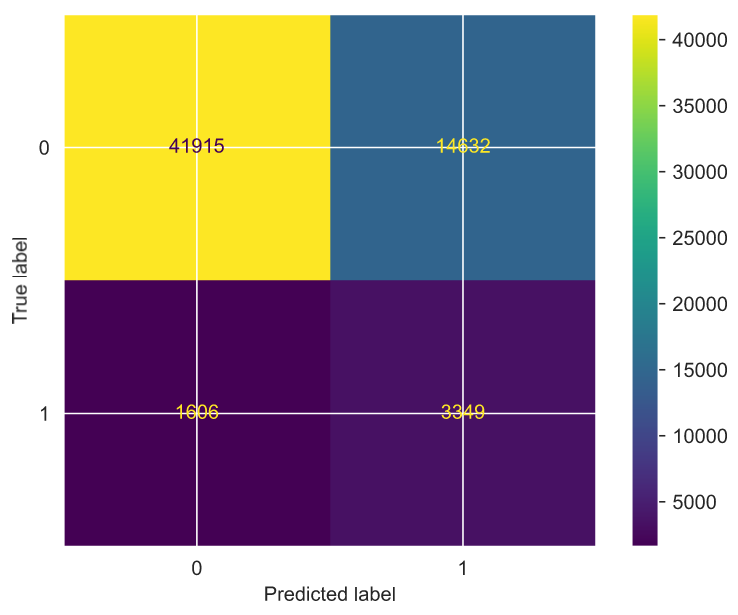
Voici ci-contre les matrices de confusion de nos deux modèles. Il est probant de constater que le passage de la première à la deuxième métrique améliore grandement les résultats désirés. En effet, avec cette deuxième fonction objectif, le ratio de FN / FP passe de 46 à 0.11 (soit en effet un quasi respect de la deuxième équation), et le nombre de mauvais payeurs retenus passe de 4809 à 1606. A fortiori, le modèle perd en précision puisqu’on passe de 92% à 73%, et le nombre de crédits accordés est sensiblement diminué (-25%) : on refuse davantage de clients (notamment de bons clients), mais on limite les pertes, tout en maximisant les bénéfices. Il est important de noter qu’un bon payeur (TN) s’est vu arbitrairement attribuer une valeur de +2, quand un mauvais payeur malheureusement retenu (FN) a une valeur de -10, et un bon payeur malheureusement non-retenu (FP) a quant à lui une valeur de -1. En modifiant ces valeurs, par exemple en augmentant la valeur d’un bon payeur TN de +2 à +3, il est fort probable de voir le nombre TN augmenter. Il serait d’ailleurs intéressant de voir comment TN suit FN quand la valeur de TN augmente.

Figure Matrice de confusion - Score F1

C’est là la principale limite de notre approche. Les résultats dépendent fortement des valeurs retenues, et nous n’avons retenu qu’un cas de figure étant donné les très longs temps d’entraînement de notre LGBM. Cela dit, pour un besoin métier, ces types de poids dépendent de l’état actuel de l’entreprise/du marché et ne changent normalement pas quotidiennement. En outre, il eu été intéressant de tester plus en profondeur un modèle de type XGBoost, puisque ses performances de pré-sélection étaient très proches du LGBM. Mais souhaitant déployer notre modèle sur Microsoft Azure, LGBM semble plus approprié puisqu’initialement développé par Microsoft.

Figure Matrice de confusion – Fonction métier

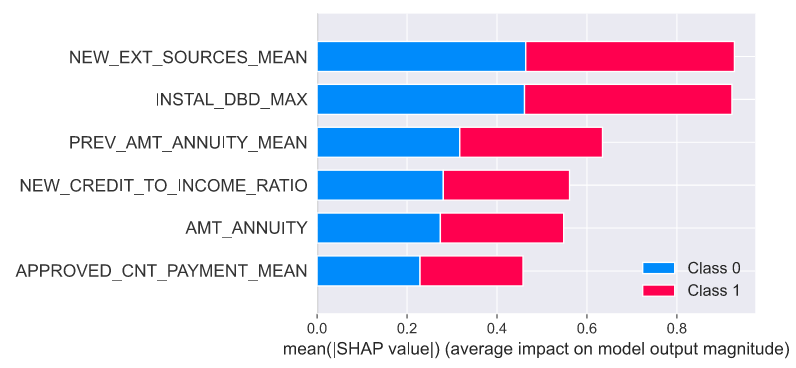
La figure 3 ci-dessous propose l’analyse globale de l’importance des features réalisée avec le package SHAP. Seules les 6 variables les plus importantes sont représentées, sur 531 au total. Les poids moyens des variables sont exprimés en valeur absolue, signifiant qu’elles influent tant positivement pour certain individus que négativement pour d’autres. La variable influençant le plus la classification est la synthèse des EXT\_SOURCES 1, 2 et 3 qui correspondent à *: Normalized score from external data source* . Nous n’avons pas d’informations supplémentaire sur leur nature si ce n’est qu’elles ont été encodées numériquement dans le dataset. La seconde variable la plus importante correspond aux nombre de jours passés depuis la date du dernier crédit, tandis que les autres représentent majoritairement des mesures monétaire (montant du crédit, ratio crédit/revenu etc..).

Figure Feature importance globale

Au niveau local de l’analyse, nous avons arbitrairement retenu l’individu numéro 4, et la figure 4 (!) représente les features les plus influents sur son score. On voit par exemple que EXT\_SOURCES\_MEAN n’a pas été aussi influent pour ce client que pour les autres, et que ce sont surtout des variables monétaires qui ont influencé la décision (en l’occurrence le refus) de crédit.

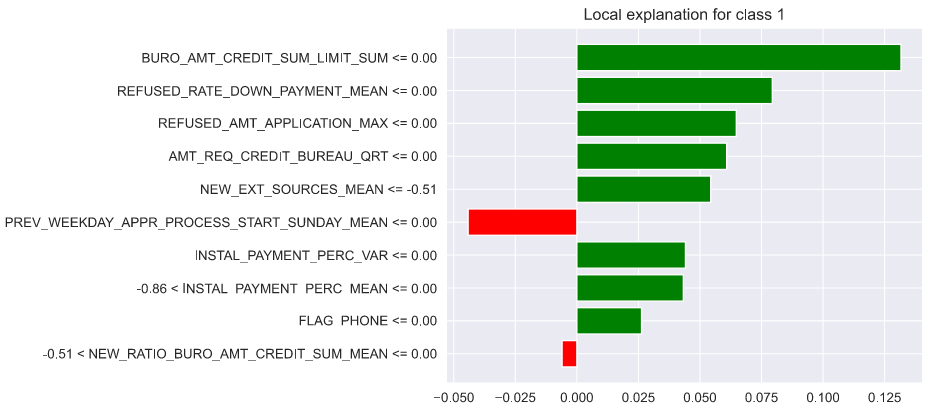


Figure Feature importance locale

1. Pour rappel, voici la répartition d’une matrice de confusion

   |  |  |
   | --- | --- |
   | TN | FP |
   | FN | TP |

   [↑](#footnote-ref-1)