Note Méthodologique

Introduction

On trouvera ci-après une note méthodologique qui porte sur le travail de classification réalisé sur un jeu de donné appartenant à une banque. On a également travaillé sur la réalisation d'un Dashboard pour retrouver, communiquer et comprendre le résultat d'un client donné par notre modèle de classification entrainé.

Cette note a une dimension chronologique et se comprend aisément en son ordre naturel. Nous partons de l'objectif : entrainer un modèle de classification binaire (deux issues possibles) pour classifier des demandeurs de crédits d'une banque. On souhaite savoir si le client est ou n'est pas solvable. On a mis à notre disposition plusieurs fichiers .csv potentiellement joignables qui portent sur les informations de clients anonymisés et sur leur solvabilité (cadre supervisé).

Classe 0 = Solvable, Classe 1 = Non Solvable.

Plan :

Méthodologie d'entrainement du modèle

Application d'un Kernel Kaggle

Le modèle a été entraîné sur des données traitées. Le traitement de ces données se base sur un Kernel Kaggle qu'on trouvera à l'adresse suivante : https://www.kaggle.com/code/jsaguiar/lightgbm-with-simple-features.

Avant d'être appliqué le kernel a été compris : de façon générale le kernel en sus effectue des agrégations sur le SK_ID_CURR c'est-à-dire sur l'identifiant unique du client. Ces agrégations se déclinent sur plusieurs modalités (Moyenne, Mean, etc.). Ces agrégations sont très utiles car le jeu de données est originellement fragmenté en plusieurs fichiers .csv, en plusieurs tables au nombre de lignes différents (ce qui ne facilite pas leur union). Dès lors que ce Kernel est appliqué, il nous est possible d'effectuer plus aisément une jointure gauche de chacune des tables sur la table principale application_train sans craindre d'avoir plusieurs lignes pour un même individu.

Après application du Kernel Kaggle, la table comprend 307511 individus uniques et 677 features.

Découpage de la table

Il nous faut alors définir un découpage d'entrainement et de test afin d'éviter de choisir un modèle qui se généraliserait mal. On notera un point clef de ce jeu de données : les données cibles sont déséquilibrées. Il y a en effet beaucoup plus de clients solvables que non solvables. Il conviendra alors de faire un découpage stratifié ce qui permet de retrouver ce même déséquilibre à la fois dans le jeu de données d'entrainement et dans celui de test.

Choix de la métrique

On pourra se référer à la partie II pour en savoir plus sur la raison de la sélection de la ROC-AUC.

Définition d'une Baseline

La baseline est un outil très utile car elle permet de mesurer les écarts de performance entre un modèle dit « stupide » et un modèle entraîné. On a utilisé un DummyClassifier pour ce faire avec une stratégie constante afin de mettre en exergue le paradoxe de l'accuracy. Notre stratégie constante consiste à « prédire » que le client est toujours solvable. On obtient alors une ROC AUC à 0.50 (notre baseline) et un score accuracy de 0.91 qui est logique puisqu'il trouve son origine dans la nature déséquilibrée de nos cibles (on a 91% de client solvable – c'est le paradoxe de l'accuracy).

Définition d'une Pipeline

Nous transformons la donnée avant de l'appliquer à nos modèles afin au moins de leur permettre de fonctionner et au mieux d'améliorer leurs performances. Nous utiliserons pour cela une Pipeline.

On se rappellera qu'en l'état, il reste des valeurs manquantes à certaines variables et que notre table (après application du Kernel) contient encore quelques variables qualitatives. On définit alors un premier socle à notre Pipeline qui est un ColumnTransformer. Celui-ci, à l'aide du type des données contenus par les features va soit One Hot Encoder les features qualitatives soit imputer les valeurs manquantes par la médiane des features numériques à l'aide d'un SimpleImputer.

Le second socle de notre Pipeline se focalise sur le problème des cibles déséquilibrées. On souhaiterait dans l'idéal équilibrer les cas solvables et non solvables pour nos modèles. Pour ce faire nous recourrons au package imbalanced-learn qui nous va nous permettre de réaliser de l'oversampling et/ou de l'undersampling sur les données. L'intuition derrière ces procédés est relativement simple :

l'oversampling va créer des individus (synthétiser) de la classe la moins représentée en plus (n.b : les individus sont synthétiquement « créés » et ne sont pas simplement dupliqués) tandis que l'undersampling va enlever des individus de la classe la plus représentée afin d'équilibrer la représentation des classes.

On pourra remarquer ici que notre Pipeline n'est pas une Pipeline du package scikit-learn mais une du package imblearn car si l'oversampling et/ou l'undersampling est effectué sur les données d'entrainement il ne doit surtout pas être effectué sur les données de test (ce que permet, entre autres, la Pipeline imblearn).

On notera également la définition de certaines étapes de notre Pipeline avec un tuple ('Nom_estimateur', None) qui va nous permettre de grid searcher pas seulement des paramètres mais des estimateurs mis dans des listes. A l'étape deux par exemple nous « grid searchons » les méthodes SMOTE, ADASYN, RandomUnderSampler et ClusterCentroids.

La troisième étape est plus succincte : à l'aide d'un modèle nous effectuons une sélection des features c'est-à-dire que nous entrainons un modèle et que nous conservons les features qui semblent avoir le plus d'impact dans la prédiction à l'aide d'une threshold via un SelectFromModel.

Enfin, si le modèle est linéaire nous ajoutons une étape de mise à l'échelle sinon nous entrainons le modèle (car les autres modèles que nous utilisons sont des modèles de la famille des arbres décisionnels non sensibles à l'échelle).

Choix des modèles

On a choisi d'entrainer quatre modèles : une régression logistique, une random forest, un XGBOOST et un LightGBM.

Validation Croisée et Fine Tuning

On utilise l'outil GridSearch de scikit-learn pour faire deux choses : une validation croisée et fine tuner nos Pipelines décrites précédemment.

La validation croisée va permettre d'évaluer le modèle en tenant compte de sa capacité à se généraliser. Pour ce faire, la validation croisée crée des folds (5 dans notre cas), entraine 4 folds et l'évalue sur le dernier. Elle répète cette opération autant de fois qu'il y a de folds (5 dans notre cas) toujours en prenant soin d'utiliser un fold de test différent de ceux déjà testés. Elle moyennise enfin les résultats pour n'en donner plus qu'un.

On répète cette validation croisée autant de fois que nous créons de paramétrages pour notre modèle. On appelle cette étape le fine tuning : on donne des paramètres (hyperparamètres) aux modèles, différentes étapes dans la Pipeline également et l'on recherche via les propriétés combinatoires des paramètres fournit à la GridSeacrh la combinaison de paramètres la plus performante.

Modèle retenu

Après comparaison des scores on retient le modèle qui nous paraît être le plus performant.

On a retenu le modèle LightGBM.

Fonction coût métier, algorithme d'optimisation et métrique d'évaluation

On utilisera VP, VN, FP, FN pour exprimer respectivement, vrais positifs, vrais négatifs, faux positifs, faux négatifs.

Du fait du déséquilibre de nos cibles nous nous sommes écartés de l'accuracy car elle prend en compte les VP qui sont systématiquement les plus élevés nous empêchant d'appréhender clairement le pouvoir discriminant de nos modèles.

Nous souhaitons accorder assez de crédits pour générer de l'argent et avec assez de précaution pour ne trop en perdre. Ne pas en générer revient à en perdre. On ne peut donc pas choisir uniquement le Rappel [VP / (VP + FN)] ni uniquement la Précision [VP / (VP + FP)].

Il existe le F1-score qui est la moyenne harmonique du Rappel et de la Précision, le F-bêta score qui est la même moyenne harmonique mais pondérée (par le bêta) qui sont des mesures plus adaptées. Cependant ces mesures, confrontées au dilemme en sus, sont apparues être un peu flous pour sélectionner un modèle. Comment choisir entre deux modèles aux F1-Score semblables mais l'un avec plus de FP et l'autre plus de FN ? Et dans le cas du F-bêta score comment choisir la valeur du bêta sans savoir réellement ce qu'il vaut mieux : plus de FP ou plus de FN ?

La ROC est une fonction de seuils qui séparent les individus selon leur probabilité d'appartenance aux classes. Pour chaque seuil, on calcule le Rappel [VP / (VP + FN)] et le contraire de la Spécificité [FP / (FP + VN)] et on les relie en un point. La courbe tracée par ces points est la ROC, l'aire au-dessous de cette courbe est la ROC-AUC. Plus la ROC-AUC est élevée plus il y a de chances de trouver un point où le contraire de la Spécificité est faible et le Rappel est élevé (le cas parfait étant un Rappel à un pour le contraire de la Spécificité à zéro). Il ne reste alors plus qu'à trouver ce point pour définir le seuil optimal de séparation des classes après calcul de la probabilité d'appartenance aux classes des individus par l'algorithme sélectionné. Conséquemment nous avons retenus la ROC-AUC.

Dans un souci d'appréhension de la réalité on a également mis au point une fonction métier qui calcule selon le scénario réel l'argent gagné ou perdu des différents seuils de séparation des classes d'un modèle donné. Quoi que le gain (montant*1.03) et la perte (montant*0.5) soient arbitraires le modèle retenu semble bien être profitable à la banque.

Nous avons retenu le modèle LightGBM. Il s'agit d'un modèle de Gradient Boosting c'est-à-dire que ce modèle va générer séquentiellement des apprenants faibles (en l'occurrence des arbres décisionnels) afin d'optimiser une fonction de perte. En d'autres termes, LightGBM va générer des arbres décisionnels (apprenant faible avec un nombre d'embranchement maximum faible) et chaque nouvel apprenant généré viendra corriger l'apprenant qui lui précède (sauf l'apprenant initial qui n'a pas d'apprenant qui lui précède) afin d'optimiser la fonction de perte car nous avons sélectionné le type de boosting « traditionnel » pour ses performances. Nous avons gardé la fonction de perte par défaut du LightGBM : la perte logistique donnée par la formule $L(y,p(x))=-ylnp(x)-(1-y)ln(1-p(x))$ car elle convient parfaitement à l'exercice de la classification binaire.

On pourra souligner la performance de ce modèle en termes de vitesse d'apprentissage également. Par exemple il se distingue très largement du XGBOOST (modèle de Grandient Boosting également qui performe très bien) en termes de temps d'apprentissage ce qui a joué dans le choix de la sélection du modèle LightGBM.

Interprétabilité Locale et Globale du modèle

Il est possible d'interpréter globalement et localement notre modèle grâce à la théorie des jeux coopératifs. Ce pan de la théorie des jeux se penche, entre autres, sur la question de la contribution de chacun des joueurs dans une équipe (donc dans un mode coopératif) au résultat de l'équipe. Les valeurs Shapleys permettent d'évaluer ces contributions. Ces valeurs qui nous viennent de la théorie des jeux sont tout à fait transposables à la science des données puisqu'un modèle, à la façon d'un jeu coopératif, se voit donné des joueurs (des features) qui vont amener à un résultat. Calculer une valeur Shapley revient à calculer pour un individu donné la contribution de chacune des features sur le score que nous lui attribuons.

On a utilisé le package SHAP afin de manipuler et calculer ces valeurs avec python. On a sollicité le TreeExplainer pour notre LightGBM car celui-ci, nous l'avons vu, utilise des arbres décisionnels en tant qu'apprenants faibles.

On a utilisé le summary plot pour interpréter globalement notre modèle. On trouve sur ce summary plot les 20 premières features, celles qui contribuent positivement et négativement le plus aux résultats attribués. La palette de couleur de SHAP est contrintuitive : en bleu nous trouvons la part de la feature qui a contribué négativement aux résultats, en rouge positivement. On trouve donc deux informations globales sur notre modèle via le summary plot : 1. l'importance des features, 2. La part de la contribution positive et la part négative de ces features. En ce qui nous concerne, on a en top position (et loin devant) la feature EXT_SOURCE_2 c'est-à-dire le score attribué par une source externe.

Pour ce qui est de l'interprétation locale, on a utilisé un force plot auquel nous passons le paramètre link='logit' qui nous permet d'observer la contribution des features en probabilité (ainsi nous voyons clairement la contribution des features sur notre score : la probabilité d'appartenance aux classes). De la même façon que le summary plot nous y trouvons en rouge la contribution positive des features (c'est-à-dire que les features rouges augmentent la probabilité d'un individu d'être solvable) et en bleu la contribution négative des features. On se méfiera de généraliser les contributions des features d'un individu sur tous les autres car il s'agit d'une représentation locale, propre à l'individu qui lui est associé uniquement. Ce force plot a été intégré à notre Dashboard car il est certainement la représentation la plus accessible pour comprendre facilement le score donné par notre algorithme.

Limites et Améliorations possibles

. De toute évidence on souhaiterait avoir plus de samples de la classe non solvable pour résoudre le problème du déséquilibre de nos cibles.

. On aurait voulu avoir des statistiques sur les montants gagnés lorsque le prêt est remboursé et perdu lorsqu'il ne l'est pas pour mieux calibrer l'optimisation de la métrique sélectionnée voire créer une métrique métier par inférence de ces statistiques et optimiser cette métrique-là.

. On souhaiterait avoir une meilleure description des features de types EXT d'autant plus que l'EXT de la seconde source est la feature qui contribue le plus aux résultats de notre modèle.

. On pourrait combiner les performances relativement bonnes de tous nos modèles à travers un voting classifier afin d'optimiser nos performances si le fit-time n'est pas une priorité.

. On pourrait se pencher davantage, dans une seconde itération de ce travail, sur le feature engineering en prenant en compte l'importance des features mis en exergue par SHAP.

. Par rapport au Dashboard, implémenter une base de données (plutôt qu'un .csv) aurait été plus agréable et serait nécessaire avant le « déploiement » métier de cet outil. Par ailleurs cela permettrait via une logique CRUD d'ajouter des clients, de les mettre à jour, de les supprimer etc. Ce CRUD et cette base de données ouvriraient le champ à de nouvelles possibilités intéressantes pour une banque. Par exemple, cela permettrait d'attirer des nouveaux clients en proposant un outil de simulation, etc.

. Fine tuner et entrainer nos modèles dans le cloud nous aurait été utile afin de les optimiser dans les meilleurs conditions (matérielles et temporelles).

. On pourrait élaborer un Score plus « vulgarisé ». En effet, notre modèle utilise une threshold à 0.07. Cela signifie que si la probabilité estimée par notre modèle d'appartenir à la classe non solvable est supérieure à 7% alors le crédit n'est pas attribué. Nous avons utilisé la probabilité du score de la classe solvable pour notre score. Avoir 94% de chance d'appartenir à la classe solvable revient à avoir un score de 94. Mais si un client à un score de 92, le crédit n'est pas attribué car 8% > 7% et cela peut s'avérer être contrintuitif pour un client d'avoir un score de 92/100 et de ne pas se voir attribué de crédit !