Projet n°7 : HomeCredit – Note méthodologique

Raphaël PUIG

Abstract : Cette note constitue une synthèse de la démarche suivie et des résultats obtenus dans le cadre du 7ème projet de la formation de *Data Scientist* d'OpenClassrooms. Elle s'articule suivant un plan en cinq parties. Dans un premier temps est présentée la démarche de modélisation – les modèles testés, celui qui a été choisi et sur quels critères. Dans une deuxième partie est détaillée la fonction coût du modèle, sa métrique d'évaluation et son optimisation. L'interprétabilité du modèle et les hypothèses qui peuvent en être formulées sont abordés dans la troisième partie. Ses limites et ses améliorations possibles sont exposées dans la quatrième partie. Enfin, dans une cinquième et dernière partie, est donné un aperçu de l'application permettant la visualisation des variables principales du modèle par l'intermédiaire d'un tableau de bord ainsi que l'établissement de nouvelles prédictions sur des données fournies par l'utilisateur *via* un « prédicteur » faisant intervenir le modèle.

1 – Démarche de modélisation

L'agrégation des différentes bases mises à disposition pour aboutir à une base de données complète crée un très grand nombre de variables additionnelles : d'environs 120 variables que compte la base *application_train* le *feature engineering* accompagnant l'agrégation aboutit à une base de plus de 1900 variables. Une première réduction est réalisée en supprimant les variables trop peu renseignées[1]. Cette réduction est très modeste et ne concerne en tout et pour tout que 90 variables. La deuxième réduction consiste à supprimer les variables qui présentent une corrélation trop importante entre elles, autrement dit des variables faisant en quelque sorte double emploi. Si deux variables sont corrélées à plus de 90%, une des deux est supprimée. Cette réduction est plus sévère et un peu plus de 700 variables ont pu être abandonnées. La dernière démarche de sélection consiste à ne conserver que les variables les plus utiles au modèle. C'est à ce moment là que le choix du modèle est réalisé.

Ne sachant pas a priori quel modèle sera utilisé par la suite, plusieurs modèles sont testés sur la base *application_train*. Il faut une base de taille raisonnable et la base globale est à ce stade encore beaucoup trop vaste. Plusieurs contraintes gouvernent le choix de ce premier panel de modèles. Pour faciliter la mise en œuvre des modèles il faut qu'ils soient implémentés dans Scikit Learn ou facilement intégrable dans un notebook Jupyter. La *target* étant binaire[2] les modèles de classification sont de suite privilégiés. Voulant aboutir à la probabilité d'échec de remboursement les modèles doivent comporter la méthode *predict_proba*. Enfin, et c'est là le plus important pour la suite de la démarche de sélection, ils doivent également comporter la méthode *feature_importance*. Le choix des modèles est alors restreint à : AdaBoost Classifier, GradientBoosting Classifier, Light Gradient Boosted Machine Classifier (ci-après LGBMClassifier), DecisionTree Classifier, RandomForest Classifier et ExtraTrees Classifier. Ces modèles, ensemblistes à l'exception de DecisionTree, se basent tous sur des arbres de décisions et c'est un grand avantage dans la mesure où, même si la moitié des variables les plus corrélées ont été écartées pour allégées la base de données, il n'est pas exclu que des corrélations subsistent entre les variables restantes.

Dans un premier temps chacun de ces modèles tourne sur la base *application_train* avec ses hyperparamètres par défaut et via une validation croisée sur trois *folds* afin d'éviter d'éventuels biais consécutifs au choix des données d'entraînement (resp. validation). La performance d'un modèle est ensuite évaluée sur la base de trois critères : 1) la valeur moyenne du ROC AUC score obtenu sur la base de validation qui doit être préférentiellement la plus élevée possible, 2) la différence des ROC AUC scores moyens obtenus respectivement sur la base d'entraînement et la base de validation (ci-après le *gap*) et qui doit être préférentiellement la plus réduite possible, et enfin 3) le temps de calcul. L'intérêt du premier critère est assez évident dans la me-

1 plus de 75% de valeurs manquantes

2 échec de remboursement *vs* réussite de remboursement

sure où il s'agit de faire des prédictions les plus solides possible. Le but du deuxième critère, celui sur le *gap*, est d'évaluer le risque de surapprentissage (ou *overfitting*) lors de l'entraînement. Cette différence de ROC AUC scores est un critère à ne pas négliger et, dans la mesure du possible, il est préférable de privilégier un modèle au plus petit *gap*, quitte à devoir faire une concession sur les performances obtenus sur la base de validation. Le troisième et dernier critère, celui sur le temps de calcul, est également très important. Un temps de calcul trop important est extrêmement pénalisant et un modèle qui offre de très bonnes performances mais qui met énormément de temps à converger n'est pas souhaitable. Si deux modèles offrent des performances peu ou prou comparables il est préférable de privilégier le plus rapide des deux. C'est d'autant plus vrai pour cette première approche que la base utilisée est encore de taille réduite et qu'à terme il s'agit de traiter la base globale de 1099 variables (ne serait-ce qu'une fois pour réaliser la sélection des variables pertinentes pour le modèle).

Cette première approche des modèles a permis de sélectionner le modèle LGBMClassifier et de réaliser la dernière phase de sélection des variables. Comme on peut s'y attendre, entraîner et tester le modèle sur l'ensemble de la base de données est beaucoup plus long que lors de la sélection du modèle lui-même, mais ce n'est pas du temps mal investi : plusieurs centaines de variables ont une importance nulle pour le modèle et sont donc supprimées. Pour vérification, le modèle est à nouveau entraîné et testé sur la nouvelle base de données ainsi allégée. Contrairement à ce à quoi on aurait pu s'attendre, de nouvelles variables d'importance nulle font leur apparition et une nouvelle suppression est effectuée. L'opération est répétée jusqu'à ce qu'il n'y ait plus de nouvelles variables d'importance nulle. Ces suppressions successives ont permis de réduire la base de plus de moitié et de 1099 variables on aboutit à une base n'en comptant plus que 464.

A ce stade nous avons une base de données réduite et un modèle qui fonctionne relativement bien mais aux hyperparamètres[3] encore non ajustés. La toute dernière phase de la modélisation consiste à identifier les hyperparamètres optimaux qui permettent à la fois de réduire le *gap* entre les ROC AUC scores et, si possible, de gagner quelques précieux pourcents de performance supplémentaires sur la base de validation. Étant donné le nombre important d'hyperparamètres à ajuster, une première sélection d'une dizaine de paramètres est réalisée parmi lesquels le type de *boosting* utilisé, le nombre d'arbres considérés, le nombre de feuilles des arbres[4], les paramètres de régulation alpha et lambda, etc. Même après cette première sélection, le nombre de paramètres à ajuster est trop important pour tester toutes les valeurs possibles une à une[5]. Une telle recherche systématique, aussi appelée *grid search*, serait extrêmement coûteuse en temps de calcul et est donc exclue d'emblée[6]. Pour contourner ce problème tout en se permettant une exploration étendue des paramètres on opte pour une variante : la *random search*. Plutôt que d'essayer absolument toutes les combinaisons possibles de valeurs des paramètres on ne teste qu'un nombre limité de combinaisons composées aléatoirement à partir de chacun des domaines de définition. Cela ne permet peut-être pas d'identifier l'ensemble de paramètres absolument optimal mais cela permet d'identifier dans un temps raisonnable un ensemble de paramètres prometteurs.

Pour identifier cet ensemble de paramètres prometteur on applique deux critères : 1) le *gap* doit être inférieur à 0.05, valeur mesurée à l'issu de l'entraînement et du test d'un modèle aux paramètres par défaut ; 2) le ROC AUC score sur la base de validation doit être le plus élevé possible. D'abord réalisée sur une sélection aléatoire de 10'000 lignes, la *random search* est finalement réalisée sur une sélection aléatoire de 10% puis 20% de la base de données globale[7]. De même, d'abord réalisée avec 100 combinaisons, elle est ensuite réalisée avec 1000 combi-

3 Les termes *hyperparamètre* et *paramètre* seront régulièrement employés l'un pour l'autre indifféremment.

4 Ce nombre de nœuds terminaux impacte notamment la taille des arbres et donc leur capacité à tenir compte des éventuelles interactions entre les variables.

5 A ce stade, on n 'a pas la moindre idée des valeurs optimales et les intervalles de définition sont larges.

6 En considérant 100 secondes par entraînement il faudrait pas moins de 26 millions d'année pour tester les 8'235'676'350'000 combinaisons possibles.

7 10'000 lignes ne représente que 3,5% de la base de données globale, les résultats obtenus sont décevants.

naisons. Cette *random search* massive, même si elle représente déjà en soi un gain de temps appréciable comparée à une *grid search*, nécessite tout de même plusieurs heures de calcul (ce qui devient assez pénalisant pour la base de 20%). A titre d'exemple la base à 10% demande environ 4 heures pour tester les 1000 combinaisons et la base de 20% environ le double.

Une fois cet ensemble de paramètres prometteurs identifié, on passe à une toute dernière phase dite de *fine tuning* qui consiste, pour le coup, à effectuer une *grid search* autour des valeurs des paramètres numériques. Même en considérant des domaines de définition réduits au plus strict minimum[8], le nombre de combinaisons possibles croit très rapidement[9], le temps de calcul croît alors également très rapidement. Cela devient même assez handicapant lorsqu'il s'agit d'utiliser la base de 20%[10].

2 – Fonction coût, algorithme d'optimisation et métrique d'évaluation

a) Fonction coût

Le principe des modèles en *gradient boosting* est de faire appel à des apprenants faibles qui sont successivement améliorés d'entraînement en entraînement sur des échantillons des données. Par définition, la fonction de perte (ou *loss function*) calcule l'erreur lors d'un unique entraînement (donc sur un seul label) et la fonction de coût (ou *cost function*) et la moyenne des fonctions de perte calculée sur l'ensemble des entraînements. Par défaut lorsqu'il s'agit d'une classification binaire, la fonction de perte de LGBM est *logloss* (pour *logistic loss function*) et est définie par :

$$L_{\log}(y,\hat{y}) = -(y.\log(\hat{y}) + (1-y).\log(1-\hat{y}))$$

avec y les labels vrais et $\hat{y}$ les labels prédits La fonction coût qui en découle pour les N labels de la base d'entraînement est alors de la forme :

$$J(\boldsymbol{w}) = \frac{-1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

où $\hat{y}_i = g(\boldsymbol{w}\cdot\boldsymbol{x_i}) = \frac{1}{1+e^{-\boldsymbol{w}\cdot\boldsymbol{x_i}}}$ est la fonction logistique définissant la prédiction du i-ème label y_i avec $\boldsymbol{w}$ le vecteur pondérant chacune des valeurs du vecteur $\boldsymbol{x_i}$ qui correspond aux *features* fournies en entrée. L'objectif est de minimiser la fonction de perte (ou a minima d'en trouver un minimum local) lors des entraînements successifs du modèle en optimisant le vecteur poids $\boldsymbol{w}$ par descente de gradient[11]

b) Algorithme d'optimisation

1) *random search* sur des combinaisons aléatoires d'hyperparamètres ;
2) sélection des ensembles de paramètres permettant un *gap* inférieur à 0.05 ;
3) tri par ordre décroissant des ROC AUC scores obtenus sur les bases de validation et sélection de la combinaison de paramètres permettant d'obtenir le ROC AUC score de validation maximale ;

Facultatif

4) entraînement et validation d'un modèle aux paramètres ajustés suivant l'étape 3 sur l'ensemble de la base de donnée ;
5) première évaluation des performances du modèle par la valeur de ROC AUC score obtenu sur la base de validation et par mesure du *gap* ;

6) *fine tuning* des paramètres obtenus en étape 3 par une *grid search* : deux valeurs proches encadrent chacun des hyperparamètres numériques ;
7) identification de trois set de paramètres finaux (cf. notebook3 pour le détail).

Ces trois combinaisons sont ensuite comparées pour chacune des situations suivantes : 10% et

8 La valeur identifiée par la *random search* et deux valeurs proches de part et d'autre

9 En n^m où m est le nombre de paramètres à ajuster et n le nombre de valeurs possibles pour chacun des paramètres

10 Il faut compter environs 12h de calcul.

11 Méthode itérative du premier ordre consistant, qualitativement, à « aller dans la direction opposée du gradient », ou de manière équivalente « dans la direction de plus grande pente », d'une fonction dérivable afin d'en déterminer un minimum local.

20% de la base d'entraînement et *random search* effectuée sur 100 et 1000 essais, ce qui fait un total de douze combinaisons possibles de paramètres. Le fait est que l'on ne peut pas être gagnant sur tous les tableaux et qu'un compromis doit être accepté[12]. La combinaison retenue (5) constitue un tel compromis, à la fois intermédiaire entre la minimisation du gap et la maximisation du ROC UAC score sur la base de validation.

c) Métrique d'évaluation

S'agissant d'une classification binaire *réussite vs échec de remboursement*, le ROC AUC score est la métrique d'évaluation choisie. Cela correspond à l'aire sous la courbe ROC[13]. Initialement développé pour évaluer la qualité des détections radars, cet indicateur est particulièrement adapté pour évaluer la qualité d'un classificateur binaire en mettant en regard le taux de vrais positifs face au taux de faux positifs. Idéalement, lorsque le classificateur est parfait tous les vrais positifs sont effectivement détectés sans qu'il n'y ait le moindre faux positif et l'aire sous la courbe vaut 1. Si le classificateur est complètement aléatoire le classificateur se trompe une fois sur deux quelque soit la classe considérée et l'aire sous la courbe vaut 0,5. L'objectif est d'augmenter la valeur de la métrique pour la faire s'approcher le plus possible de l'unité.

3 – Interprétabilité de LGBMClassifier

LGBMClassifier est, comme son nom l'indique[14], un algorithme de classification du type *gradient boosting*. Les algorithmes en *gradient boosting* font parties des modèles dits ensemblistes séquentiels qui font intervenir des arbres de décisions de tailles réduites[15] successivement améliorés par la minimisation de la fonction la fonction coût (cf. 2-a). Ces algorithmes permettent en outre d'avoir accès aux variables qui ont été les plus utiles pour l'entraînement du modèle via la *feature_importance*. Par défaut LGBM évalue l'importance d'une variable par la fréquence de son usage dans les différents nœuds. Autrement dit, plus une variable sera présente dans les nœuds des arbres et plus son importance relative vis-à-vis des autres variables va croître[16]. Le nombre de variables considérées par le modèle étant important, l'importance relative des variables est assez faible et ce même pour les variables considérées comme étant les plus importantes. Cependant trois variables se détachent assez nettement : EXT_SOURCE_1, EXT_SOURCE_2 et EXT_SOURCE_3. Ces variables correspondent à des notations provenant de sources externes à HomeCredit et qui ont pour but d'évaluer la solvabilité des clients. Pour les besoins de l'application deux autres variables supplémentaires ont été sélectionnées : DAYS_BIRTH (âge du client) et CREDIT_AMT (quantité de fonds empruntés).

D'un manière générale, il est préférable que le client ait des scores de EXT_SOURCE les plus élevés possibles. Les *Kernel Density Estimator* (ci-après KDE) de chacune de ces variables montre une claire séparation entre les clients en échec de remboursement aux EXT_SOURCE faibles, et les autres clients aux EXT_SOURCE scores élevés. De même la KDE de DAYS_BIRTH semble assez clair : les échecs de remboursement sont plus fréquents chez les jeunes emprunteurs alors que les succès sont répartis de manière relativement équilibrés sur l'ensemble des âges considérés. L'âge pivot, où la densité d'échec passe en dessous de la densité de succès, se situe aux alentours de 40 ans. Peut-être s'agit-il d'un manque d'expérience du crédit qui induit une sur-estimation de la part du client de sa capacité de remboursement. Enfin, la dernière variable sélectionnée pour l'application, AMT_CREDIT, semble indiquée une fourchette de montants, entre 290k$ et 670k$, où le risque d'échec de remboursement est sensiblement plus élevé que le celui de réussite. En dehors de cette fourchette, les succès de remboursement sont certes plus fréquents, mais l'écart de densité est très faible et assez fluctuant égale-

12 Soit on minimise le *gap* entre l'entraînement et la validation, soit on maximise le ROC AUC score sur la validation.

13 AUC et ROC pour, respectivement, *Area Under the Curve* et R*eceiver Operating Characteristic*

14 LGBM signifie *Light Gradient Boosted Machine*

15 Le nombre de nœuds semble optimal lorsqu'il est compris entre 4 et 8. En dessous les possibles interactions entre les variables ne sont plus prises en compte et il est généralement peu probable que plus de nœuds soit nécessaire. Hastie et al., "10. Boosting and Additive Trees". *The Elements of Statistical Learning* (2nd ed.). New York: Springer. pp. 337–384

16 La somme totale des *feature importances* vaut 1.

ment. Cela peut peut-être s'expliquer par le fait que les plus petits crédits sont intrinsèquement plus faciles à rembourser et que les plus gros crédit sont peut-être mieux budgétisés puisque s'agissant d'investissements plus importants.

4 – Limites et améliorations

La principale limite de ce modèle réside dans le temps de calcul nécessaire à son optimisation. Malgré le choix de LGBMClassifier pour sa rapidité d'exécution, la phase finale de la recherche des hyperparamètres optimaux, la *grid search*, a demandé énormément de temps. Le gain de performance par rapport à ce qui a été obtenu à l'issue de la *random search* est, du reste, assez limité.

Une amélioration possible serait d'implémenter un *early stopping* afin de palier à ce défaut. L'idée serait d'interrompre l'entraînement du modèle si aucune amélioration n'est observée après un nombre donné de cycles. Le nombre de cycles à considérer reste cependant à déterminer. Une autre piste d'amélioration serait d'opter pour un autre algorithme d'optimisation des hyperparamètres. L'inconvénient des algorithmes tels que la *random search* et la *grid search* est qu'ils sont « aveugles » et « amnésiques » : chaque combinaison d'hyperparamètres est considérée indépendamment des autres, et ils ne tiennent pas compte des résultats précédemment obtenus pour affiner leurs recherches. Un algorithme tel que l'optimisation Bayésienne semblerait indiqué pour palier à ce défaut. L'idée serait d'automatiser de manière « intelligente » l'optimisation des paramètres du modèle.

5 – Application

L'application développée pour présenter les résultats obtenus dans le cadre de ce projet s'articule en deux fonctionnalités : 1) un *dashboard* permettant de visualiser les principales *features* que sont EXT_SOURCE_1, 2 et 3, l'âge du client, le montant du crédit et la probabilité d'échec de remboursement ; 2) un « prédicteur » affichant sous forme de tableau les principales *features* et la probabilité d'échec de remboursement pour un fichier client chargé par l'utilisateur[17].

17 Des échantillons de 10, 100 et 1000 clients sont disponibles dans le fichier app_samples du dépôt Git.

a) Dashboard

Cette première fonctionnalité permet de visualiser pour chacun des clients de la base de données (librement accessible depuis un menu déroulant) ses valeurs sur les variables EXT_SOURCE_1, 2 et 3, son âge, le montant de son crédit et sa probabilité d'échec de remboursement prédit par le modèle. Afin d'améliorer la lisibilité et de mieux situer le client par rapport aux autres, chacune de ses valeurs est également affiché dans des graphiques de *kernel density estimator* correspondant respectivement aux échecs et aux succès de remboursement. En d'autres termes, ces graphiques permettent de voir si les valeurs du client considéré sont plutôt propres aux réussites ou plutôt propres aux échecs et ainsi mettre en perspective, au moins qualitativement, sa probabilité d'échec telle qu'elle est prédite par le modèle. Sélectionner un nouveau client du menu déroulant met à jour les affichages en conséquence. Un client donné peut-être également directement recherché dans le menu, il suffit d'y écrire son SK_ID_CURR.

b) Predictor

Cette deuxième fonctionnalité permet d'afficher sous la forme d'un tableau les principales *features* sus-mentionnées et la probabilité d'échec de remboursement associée pour une base de données clients fournie au modèle *via* le téléchargement d'un fichier au format csv. Quelques fonctionnalités supplémentaires permettent d'améliorer la lisibilité des résultats : un code couleur montre à quel tiers de probabilité d'échec appartient le client[18], et la possibilité de classer par ordre croissant ou décroissant chacune des *features*. Cette dernière fonctionnalité permet notamment de retrouver plus facilement un client si son âge ou le montant de son crédit est particulièrement élevé ou au contraire réduit. Elle permet également d'évaluer l'importance relative des *features* si le classement leur données modifie plus ou moins l'ordre des probabilités. En ce sens l'importance particulièrement élevée de EXT_SOURCE_3 et EXT_SOURCE_2 est assez manifeste.

18 Vert si inférieur à 0.33, orange si compris entre 0.33 et 0.66, rouge si supérieur à 0.66