Note Méthodologique

Projet 7

Méthodologie d'entraînement :

Le modèle a été basé quasi entièrement sur les deux notebooks suivants de la compétition Kaggle associée aux jeux de données fournis pour le projet 7 :

- Pour le choix des attributs et du modèle :

https://www.kaggle.com/tahmidnafi/cse499

- Pour l'optimisation du modèle :

https://www.kaggle.com/willkoehrsen/intro-to-model-tuning-grid-and-random-search#Four-parts-of-Hyperparameter-tuning

Le premier a été sélectionné en partie pour son modèle (Light Gradient Boosting Machine) qui est à la fois le plus répandu pour les données structurées de taille moyenne et le plus performant sur la plupart des compétitions actuelles. Une autre raison était son score de compétition publique et privé qui était dans les meilleurs. Enfin, le code était relativement simple et compréhensible.

Le second notebook a été sélectionné pour sa grande pédagogie et la clarté des explications fournies.

Le modèle utilisé est donc un Light Gradient Boosting Machine (LGBM). C'est un modèle ensembliste séquentiel supervisé. Le modèle va se baser sur une multitude de petits arbres de décision successifs qui vont chacun tenter de classifier nos données. Chaque arbre va donner un poids plus important sur les observations mal classifiées par l'arbre précédent. La minimisation de l'erreur (ou fonction de perte) va s'effectuer de manière successive selon une descente de gradient d'où le nom du modèle.

Grâce à ce modèle, on va chercher à savoir si un client peut avoir l'obtention d'un prêt ou non. On va donc avoir à faire à une classification des clients dans laquelle chaque client va se voir attribuer une classe (prêt accepté ou refusé) à laquelle on associera une probabilité.

Ainsi on pourra avoir un client dont le modèle pense qu'on devrait lui accepter le prêt avec une de 78 % et de lui refuser avec une probabilité de 22 %.

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

Pour optimiser notre modèle, nous allons chercher à optimiser ses hyperparamètres. Pour cela, nous utiliserons une validation croisée en évaluant la ROC AUC, c'est à dire l'air sous la courbe du ratio des vrais positifs en fonction du ratio des faux positifs. C'est la métrique d'évaluation la plus répandue également car elle ne dépend pas de l'échelle ni du seuil de validation.

La validation croisée nous permet de réduire le sur-apprentissage et d'utiliser l'intégralité du jeu de donnée pour la validation. Cela permet d'avoir une meilleure estimation de la performance de notre modèle.

Pour tester différents hyperparamètres, nous ferons une recherche aléatoire sur une dizaine d'hyperparamètres pertinents. Nous ne faisons pas de recherche en grille car cela prendrait trop de temps pour tout essayer. La recherche aléatoire permet de balayer une plus grande plage d'hyperparamètres et d'avoir un ensemble de valeurs correctes pour ces derniers.

Interprétabilité du modèle :

Comme dit précédemment, le modèle va nous fournir une classification des clients avec une probabilité d'appartenir à chacune des deux classes « prêt accepté » ou « prêt refusé ».

Ceci permettra au conseiller bancaire de donner une réponse au client et selon la probabilité, d'avoir une certitude du résultat et donc de pouvoir juger lui-même et adapter sa réponse en fonction de sa connaissance du client. Par exemple si un client a un prêt refusé avec une certitude de 97 %, il n'y a pas de doutes à avoir mais si un client a un prêt refusé avec une certitude de 56 %, le conseiller bancaire va pouvoir estimer lui-même si un acte commercial ne peut pas être envisager envers un client qui le mérite.

Grâce à la fonction de scikit-learn « permutation_importance », il est de plus possible de connaître à quel point un attribut a de l'importance pour la prise de décision de la classification. Cela permet de pouvoir adapter encore une fois le résultat et de l'interpréter en regard du profil du client et des changements possibles de son côté. Par exemple il sera compliqué pour un client de modifier son âge si c'est un des facteurs principaux du refus de son prêt mais il sera par contre possible de modifier la somme du crédit demandé afin de l'ajuster en fonction de ses revenus.

Limites et améliorations possibles :

Pour gagner en interprétabilité, il pourrait être intéressant de limiter le nombre d'attributs utilisés par le modèle. Il faudra dans ce cas bien sûr être attentif à l'impact de ce choix sur la performance du modèle.

Pour rester sur le thème des attributs, ceux qui ont été créés l'ont été manuellement à partir du savoir d'une personne dans le domaine d'utilisation du modèle. Il est possible d'automatiser la création de ces attributs pour en créer un plus grand nombre comme décrit dans ce notebook :

https://www.kaggle.com/willkoehrsen/tuning-automated-feature-engineering-exploratory

Une autre piste d’amélioration est celle du choix des hyperparamètres. Comme dit précédemment, l’optimisation a été effectuée à partir d’une recherche aléatoire. Il existe cependant des méthodes plus efficaces. On peut penser à un mélange de recherches aléatoires pour avoir une idée des valeurs des hyperparamètres suivies de recherches en grille plus spécifiques autour de ces valeurs. Une autre méthode consiste à automatiser la tâche grâce à des techniques telle que l’optimisation Baysesienne.

Une dernière possibilité d’amélioration serait de changer de modèle en passant par un réseau de neurones. Ceux-ci ont l’avantage de pouvoir être très performant dans des tâches de classification. Cependant leur mise en place est souvent plus complexe et leur interprétabilité moindre.