MEMOIRE METHODOLOGIQUE

Mars 2020

Project 7 - Parcours Data Scientist «Implémentez un modèle de scoring »

1. Contexte

Ce mémoire constitue l'un des livrables du projet «Implémentez un modèle de scoring » du parcours Data Scientist d'Openclassrooms. Il présente le processus de modélisation et d'interprétabilité du modèle mis en place dans le cadre du projet.

Le projet consiste à développer pour la société « Prêt à Dépenser », une société de crédit de consommation, un modèle de scoring de la probabilité de défaut de paiement d'un client avec pas ou peu d'historique de prêt.

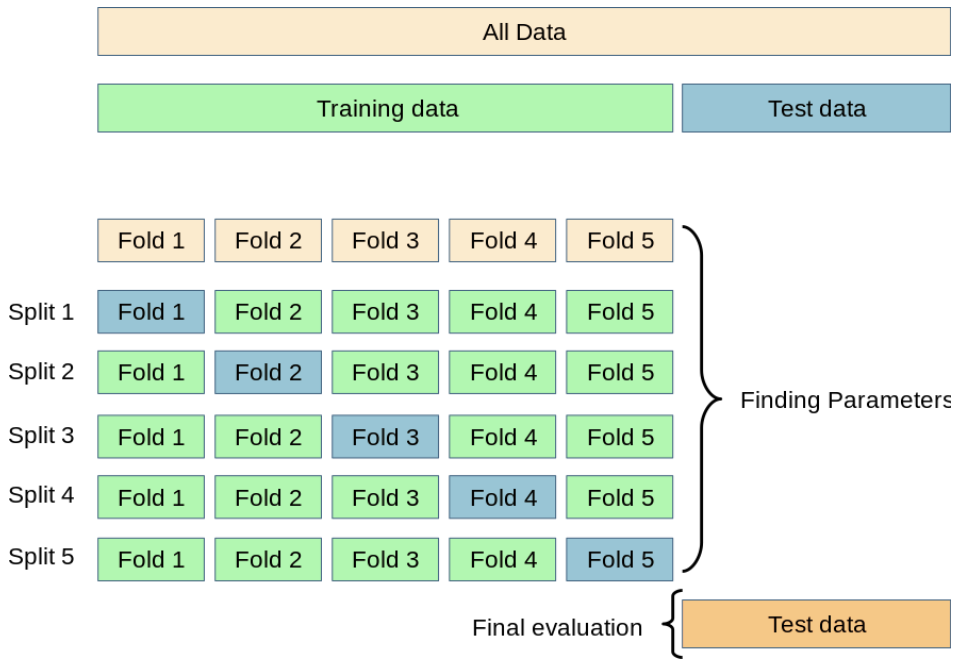
Les données utilisées pour ce projet sont une base de données de 307 000 clients comportants 121 features (âge, sexe, emploi, logement, revenus, informations relatives au crédit, notation externe, etc.)

2. Méthodologie d'entraînement du modèle

Le modèle entraîné dans le cadre de ce projet a été entraîné sur la base du jeu de données après analyse exploratoire et création de nouvelles features. Le notebook utilisé est consultable sur le site Kaggle.

Le jeu de données initial a été séparé en plusieurs parties de façon à disposer :

- D'un jeu de training (75% des individus) qui a été séparé en plusieurs folds pour entraîner les différents modèles et optimiser les paramètres (cross validation) sans overfitting.
- D'un jeu de test (25 % des individus) pour l'évaluation finale du modèle



Le problème est un problème de classification binaire avec une classe sous représentée (9 % de clients en défaut contre 91 % de clients sans défaut). Ce déséquilibre des classes doit être pris en compte dans l'entraînement des modèles puisqu'un modèle « naif » prédisant systématiquement que les clients sont sans défaut aurait une accuracy (justesse) de 0.92 et pourrait être considéré à tort comme un modèle performant alors qu'il ne permettrait pas de détecter les clients à risque. Deux approches pour rééquilibrer les deux classes ont été testées : Sample Weights et Smote :

- SMOTE permet de créer des données synthétiques à partir dees données existantes ;
- Sample Weights permet de modifier les poids associés aux observations de sorte qu'une observation mal classée dans la classe minoritaire pénalise davantage la fonction de perte qu'une observation mal classée dans la classe majoritaire.

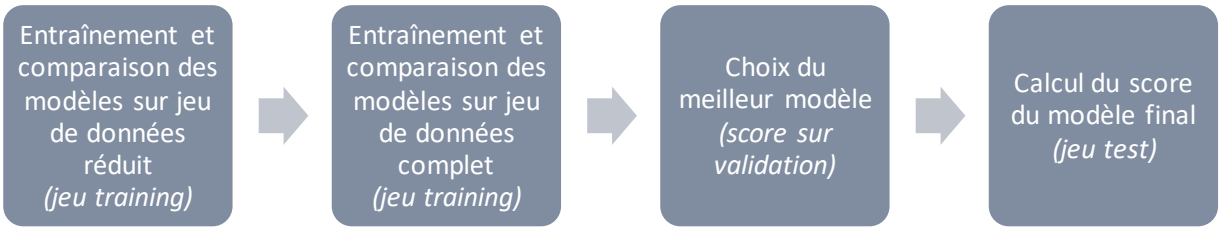
Le jeu de données post feature-engineering a été exporté et utilisé dans un premier temps sur la base d'un échantillon de 10 % du jeu de données initial. le but étant de comparer avec un temps de calcul réduit les approches SMOTE) et Sample Weights.

Pour chaque approche, différents modèles ont été testés avec recherche d'hyperparamètres et cross validation (5 folds) :

- Random Forest Classifier ;
- XGBoost ;
- Multi Layer Perceptron Classifier (réseau de neurone peu profond) – avec SMOTE uniquement, Sample Weights n'étant pas implémenté pour ce modèle ;
- Modèle Stacking combinant les 3 modèles précédents (Algorithme XGBoost) : ce modèle prend en entrée la combinaison des probabilités a posteriori issues des algorithmes précisés ci-avant ainsi que le jeu de données de base. Le modèle stacking combine donc les approches de bagging (random forest) et boosting (xgboost).

Le travail sur jeu réduit a montré que les 2 approches présentaient des résultats d'ordre de grandeur comparable sur le jeu de validation. Elles ont donc été conservées pour un entraînement sur le jeu complet. Le processus a ensuite été réalisé sur l'intégralité du jeu de données.

Le choix du meilleur modèle a été effectué en retenant le modèle avec le meilleur score sur le jeu de validation.



Logigramme de méthodologie d'entraînement du modèle

3. Fonction coût, algorithme d'optimisation et métrique d'évaluation

3.1. Fonctions coût

Les modèles ont été entraînés dans la fonction de cross validation et testés suivant différentes combinaisons d'hyperparamètres. Les fonctions de coût pour les algorithmes entraînés sont les suivantes :

Algorithme	Fonction de coût
XGBoost Classifier **XGBoost Classifier Stacking**	Régression logistique pour classification binaire
Random Forest Classifier	Minimisation du coefficient de GINI pour chaque noeud

Multi Layer Perceptron Classifier	Entropie croisée

3.2.Métrique d'évaluation

Choix de la métrique

Dans le jeu de données de base, il y a une part de 92 % des clients qui n'ont pas d'incident de paiement, tandis que 8 % des clients ont eu des incidents.

La matrice de confusion est la suivante :

	Clients prédits en défaut	**Clients prédits sans défauts**
Clients réellement en défaut	Vrais positifs	**Faux négatifs**
Clients sans défaut	Faux positifs	Vrais négatifs

Du point de vue d'une banque, on cherchera à éviter de mal catégoriser un client avec un fort risque de défaut (pertes financières et frais de recouvrements qu'on imagine importants). On cherche donc à minimiser le pourcetage de faux négatifs (erreur de type II) et à maximiser le pourcentage de vrais positifs.

Autrement dit, on va chercher à maximiser le recall

$$Recall = \frac{vrais\ positifs}{frais\ positifs + faux\ négatifs}$$

Par ailleurs on cherche à maximiser le nombre de clients potentiels donc à ne pas tous les classer en défaut. On cherche donc à éviter d'avoir un trop grand nombre de faux positifs (erreur de type I).

On cherche donc à maximiser la précision

$$Precision = \frac{vrais\ positifs}{frais\ positifs + faux\ positifs}$$

Pour notre problématique métier, le RECALL est plus important que la PRECISION car on préfèrera vraisemblablement limiter un risque de perte financière plutôt qu'un risque de perte de client potentiel.

On cherche donc une fonction qui optimise les 2 critères en donnant plus d'importance au recall. Fonction permettant de faire cela : F Beta Score : https://en.wikipedia.org/wiki/F1_score) avec Beta le coefficient d'importance relative du recall par rapport à la précision.

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{\beta^2 \cdot precision + recall}$$

$$F_\beta = (1 + \beta^2) \cdot \frac{true\ positive}{(1 + \beta^2) \cdot true\ positive + \beta^2 \cdot false\ negative + false\ positive}$$

On cherchera à **maximiser** cette métrique.

Quantification de l'importance relative entre precision et recall (Beta)

Il faut pour cela estimer :

- le coût moyen d'un défaut
- le coût d'opportunité d'un client potentiel accidentellement écarté

La consigne du projet ne permet pas de répondre à ce problème, on va donc prendre des hypothèses « réalistes », qu'on pourrait ajuster avec les interlocuteurs métiers sur un projet réel:

- que chaque défaut entraîne la dépense de 30 % du montant du crédit en pertes et frais de recouvrement ;
- qu'un client a 10 % de chance de soucrire au crédit quand il en fait la demande à un conseiller, et donc que le coût d'opportunité pour un client potentiel accidentiellement écarté est de 10 % du montant du crédit.

On obtient les coefficient suivants:

- Coefficient RECALL : de 30 % * le montant moyen de crédit des personnes en défaut
- Coefficient PRECISION : 10 % * le montant moyen de crédit des personnes sans défaut

On obtient $Beta = \frac{coefficient\ Recall}{coefficient\ precision} \approx 2{,}758$

3.3.Algorithme d'optimisation

La meilleure combinaison d'hyperparamètres a été retenue pour chaque algorithme. Un modèle stacking a été réalisé en combinant tous les modèles pour chaque approche (Sample Weights et Smote). Le modèle ayant le meilleur score en cross validation sur le jeu de training a été retenu : il s'agit du modèle XGBoost de stacking combinant les modèles XGBoost et Random Forest Classifier. Ce modèle dispose d'une bonne performance en combinant des modèles de machine learning différents : Random Forest Classifier avec des arbres de décisions complets entraînés en parallèle contre XGBoost avec des arbres de décisions faibles (weak learners) mais entraînés de façon itérative.

4. Interprétabilité du modèle

Le modèle étant destiné à des équipes opérationnelles devant être en mesure d'expliquer les décisions de l'algorithme à des clients réels, le modèle est accompagné d'un module d'explicabilité.

Pour réaliser ce module, la première perspective envisagée était d'utiliser l'importance des features issues des différents modèles utilisés mais cette approche n'est pas optimale car :

- Les features importances en sortie de modèle sont difficiles à interpréter lorsqu'il y a des variables issues de One Hot Encoding ;
- L'alternative de réaliser une méthode par permutation pour déterminer l'importance des variables est lourde en terme de temps de calcul.

L'approche retenue a été d'utiliser LIME qui est une librairie s'appliquant à n'importe quel modèle de machine learning et permettant de comprendre comment évolue la prédiction d'un modèle et perturbant les variables en entrée du modèle.

5. Limites et améliorations possibles

La modélisation effectuée dans le cadre du projet a été effectuée sur la base d'une hypothèse forte : la définition d'une mérique d'évaluation : le F Beta Score avec Beta fixé suivant certaines hypothèses non confirmées par le métier. L'axe principal d'amélioration serait de définir plus finement la métrique d'évaluation en collaboration avec les équipes métier.

L'interprétabilité du modèle pourrait être étoffée en considérant les variables issues du one hot encoding comme une seule et même variable dans la perturbation (un client ne pouvant cumuler plusieurs caractéristiques dans la logique du jeu de données initial.

Par ailleurs, la partie de traitement préalable du jeu de données a été abordée de façon superficielle en réutilisant un notebook issu de Kaggle qui se base uniquement sur une table du jeu de données. Il y a très probablement l'opportunité d'améliorer la modélisation en utilisant d'autres features des données fournies, ainsi qu'en créant de nouvelles features en collaboration avec les équipes métier.