Note Méthodologique - Projet 7

Pierre Schifflers - OpenClassrooms 2020

Problématique et Objectifs

En tant que Data Scientist pour l'entreprise "Prêt à Dépenser", une entreprise qui propose des crédits pour des personnes ayant peu ou pas du tout d'historique de prêt, il nous est demandé de développer un modèle de scoring de la probabilité de défaut de paiement d'un client pour aider dans la décision d'accorder ou non un prêt à un client potentiel.

Un des points importants de la mission est d'assurer une certaine transparence quant à la décision afin de permettre au responsable client de mieux comprendre pourquoi la demande d'un de ses clients est acceptée ou refusée. Ce modèle doit être accessible via un dashboard interactif.

Méthodologie d'entraînement du modèle

Problème et Données

Le problème auquel nous sommes confrontés ici est un problème de classification binaire. Nous devons pouvoir prédire si un client appartient à la catégorie 0 (paiement sans défaut) ou la catégorie 1 (défaut sur le paiement). Afin de parvenir à faire cette prédiction, nous avons un jeu de données de 307,511 clients ainsi que des informations sur 121 features différentes. Chacune de ces observations contient une colonne cible indiquant si le client appartient à la catégorie 0 ou 1.

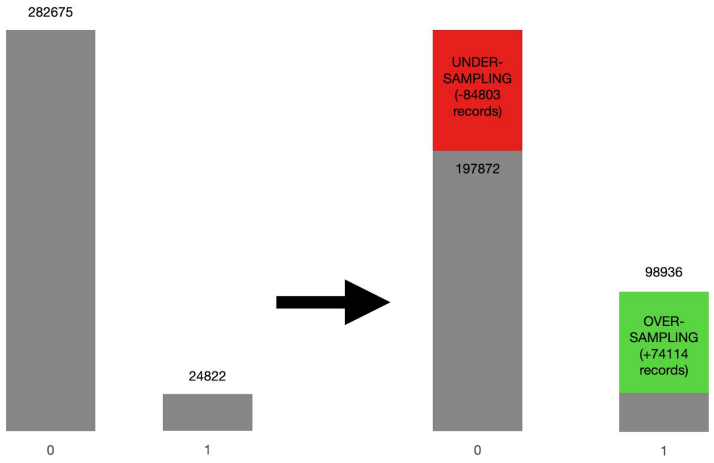
Exploration des Données

Une première exploration des données révèle un problème important auquel il faudra remédier. Nous voyons en effet que la distribution de la feature cible est très inégale avec **92%** des observations se situant dans la catégorie 0.

Afin de résoudre ce problème, la technique **SMOTE** (Synthetic Minority Oversampling Technique) fut employée. SMOTE est une technique d'augmentation des données (over-sampling) qui permet de synthétiser des nouvelles données dans la classe minoritaire à partir d'observations existantes.

Une méthode alternative au over-sampling est le **under-sampling**, qui consiste à réduire la taille de la catégorie majoritaire pour arriver à un ratio plus égal. Dans notre contexte, pour arriver à un ratio de 1:2 uniquement en utilisant le under-sampling, il aurait fallu réduire la catégorie 0 à +- 50,000 observations ce qui voudrait dire éliminer plus de 80% des données. Pour éviter une telle perte d'information tout en ne synthétisant pas trop de nouvelles données, nous avons décidé de combiner le over et under-sampling.

Pour ce faire, lors de l'apprentissage de nos modèles nous avons augmenté la part de la catégorie minoritaire pour qu'elle atteigne 35% des données et nous avons ensuite réduit la part de la catégorie majoritaire pour atteindre un ratio de 1:2. Cela nous permet d'atteindre un ratio satisfaisant tout en limitant la perte d'information et en limitant le nombre d'observations synthétiques créées.



Entraînement des algorithmes

Avec ce jeu de données plus adapté, nous avons séparé les données en un jeu d'entraînement avec 70% des données et un jeu de validation de 30% des données.

Nous avons testé 4 algorithmes:

- Régression Logistique
- Random Forest Classifier
- Light Gradient Boosting
- XG Boost

Pour chacun de ces algorithmes, nous avons effectué un Grid Search sur le jeu de données test avec une cross-validation de 4 folds afin d'optimiser certains hyper-paramètres clé. La fonction de scoring utilisée pour optimiser les paramètres est le **F-Beta** score expliqué plus en détail dans la section "métrique d'évaluation". Pour chacun des algorithmes, nous avons évalué le F-Beta score sur le jeu de validation mais aussi sur le training set pour éviter d'utiliser un algorithme en sur-apprentissage.

Basé sur cette évaluation, le modèle de régression logistique nous donne clairement les meilleures performances sur le jeu de validation. On notera aussi la nette amélioration des performances une fois qu'on commence à utiliser SMOTE. En effet, la première ligne est une baseline de régression logistique sans l'utilisation de SMOTE et celle-ci nous donne des performances bien moins bonnes que tous les autres modèles.

	model	score	train_score	test_score
0	Logistic Regression - No Smote	F-Beta	0.013918	0.015908
1	Logistic Regression	F-Beta	0.352568	0.354126
2	XG Boost	F-Beta	0.190002	0.117907
3	Light Gradient Boosting	F-Beta	0.282701	0.261707
4	Random Forest Classifier	F-Beta	0.989360	0.040362

Métrique d'évaluation

Fonction Coût

Les fonctions coût que l'on tente de minimiser avec ces algorithmes sont les suivantes:

Modèle	Fonction Coût
Régression Logistique	Sigmoid Function
XG Boost	Régression logistique pour classification binaire
Random Forest Classifier	Minimisation du Gini impurity index pour chaque noeud
Light Gradient Boosting	Régression logistique pour classification binaire

Métrique

Comme expliqué plus tôt, nous avons utilisé le F-Beta score comme métrique pour ce problème spécifique. Voyons pourquoi cette métrique est appropriée pour ce problème:

Nous avons à faire à un problème de classification binaire et nos prédictions peuvent donc se situer dans l'une des 4 catégories de la matrice de confusion ci-dessous:

		Classe Réelle	
		Non-Défaut	**Défaut**
Classe Prédite	**Non-Défaut**	Vrai Négatif	Faux Négatif
	Défaut	Faux Positif	Vrai Positif

Selon cette matrice de confusion, la question est de déterminer ce qui est le plus important pour l'entreprise. Est-il plus coûteux de:

- Ne **pas** octroyer de crédit à un client n'aurait pas été en défaut (perte de revenus potentiels)? **Faux Positif**
- Octroyer un crédit à un client qui finira en défaut sur son paiement (coûts additionnels)? **Faux Négatif**

Idéalement, cette question devrait être l'objet d'une analyse poussée non disponible dans le cadre de cet exercice. Cela dit, nous pouvons émettre l'hypothèse qu'il est important de prendre les deux métriques en compte mais qu'il est plus coûteux d'octroyer un crédit qui ne sera pas repayé (faux négatif).

Nous voulons donc maximiser les vrais positifs (recall):

$$recall = \frac{Vrais\ positifs}{Vrais\ positifs + Faux\ Négatifs}$$

En même temps, nous ne voulons pas refuser des prêts à chaque client, nous voulons donc également maximiser la précision

$$precision = \frac{Vrais\ positifs}{Vrais\ positifs + Faux\ Positifs}$$

Une métrique permettant d'optimiser ces deux valeurs est le F-Score:

$$FScore = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Cependant, avec ce score, la précision et le recall ont tous deux le même poids. Comme évoqué plus tôt, nous voulons mettre plus de poids sur le recall, pour ce faire, nous utilisons le F-Beta score où Beta est le poids du Recall comparé à la précision:

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

Dans le cadre d'une analyse poussée, il serait important de déterminer la valeur de Beta selon une analyse de perte de revenus et de coûts additionnels. Dans ce projet, nous avons utilisé une valeur de 2 pour donner 2 fois plus d'importance au recall qu'à la précision.

Interprétabilité

Pour l'interprétabilité du modèle, nous avons choisi d'utiliser LIME (Local interpretable model-agnostic explanations). LIME est particulièrement intéressant dans notre cas car cette méthode, étant locale, permet d'expliquer les prédictions individuelles d'un modèle. C'est à dire que nous pouvons prendre une observation (dans notre cas un client) et mieux comprendre

quelles features ont le plus d'impact dans pour déterminer la catégorie à laquelle cette observation appartient.

Pour chaque observation, le dashboard nous permet donc de voir les 5 features ayant le plus d'impact sur la prédiction et nous permet de comprendre l'influence de cette feature. Par exemple: "Quand ***feature X*** a une valeur > 0.7, le risque de défaut du client **augmente**".

La possibilité de comprendre les features est d'autant plus important dans le contexte d'une régression logistique car nous ne pouvons pas nous fier à des mesures de 'feature importance' comme c'est le cas avec un modèle Random Forest, par exemple.

Nous avons aussi rajouté des descriptions claires des features au sein du dashboard pour que l'utilisateur final comprenne bien ce que représente la feature sans devoir se fier au nom technique du feature. Par exemple, le feature "FLAG_DOCUMENT_3" sera décrit comme "Did the client provide document 3?"

Limitations et Améliorations

Un nombre de points pourraient être améliorés dans ce projet avec plus de temps et de ressources:

- **Performance des modèles** - Le gros point d'attention de ce projet étant le dashboard et le déploiement en ligne, moins d'attention fut consacré à l'optimisation des modèles pour obtenir les meilleures prédictions possibles. De plus, le jeu de données étant particulièrement lourd (300,000 observations sur 240 colonnes), le temps de calcul sur mon ordinateur personnel était trop long que pour effectuer une optimisation approfondie des hyper-paramètres.
- **Feature Engineering** - M'étant basé sur une manipulation des données et un feature engineering déjà existant sur Kaggle, je n'ai pas eu l'occasion de pousser le feature engineering aussi loin que ce que je l'aurais fait d'habitude. Il y a donc très probablement des modifications à apporter qui pourraient augmenter la qualité de la prédiction finale.
- **Interprétabilité à l'échelle** - Ayant eu recours à un scaler avant l'entraînement des modèles, toutes les valeurs des features montrées dans le dashboard sont à l'échelle. Par exemple, prenons le feature "nombre d'enfants", les valeurs du jeu de données initial se trouveraient certainement quelque part entre 0 et 7. A l'échelle, cependant, les valeurs sont comprises entre 0 et 1. Cela enlève une partie de l'interprétabilité du modèle et rend l'interactivité moins intuitive. Avec plus de temps et de ressources, cela serait un point d'amélioration.