Matthieu Cazier

Note Méthodologique

Pour le projet 7 : Implémentez un modèle de scoring

OC / CentraleSupélec

Objectifs :

- La méthodologie d'entraînement du modèle
- La fonction coût, l'algorithme d'optimisation et la métrique d'évaluation
- L'interprétabilité du modèle
- Les limites et les améliorations possibles

Table des matières

1. Préambule 3

2. La méthodologie d'entrainement du modèle 3

a. Séparation des données 3

b. Cross validation 3

c. Les différents modèles utilisés 4

3. La fonction coût, l'algorithme d'optimisation et la métrique d'évaluation 5

a. La fonction coût, algorithme d'optimisation 5

b. La métrique d'évaluation 6

4. L'interprétabilité du modèle 7

5. Les limites et les améliorations possibles 8

1. Préambule

Cette note méthodologique est un travail demandé dans le projet 7 « Prêt à dépenser » de la formation Data Scientist chez OpenClassroom. L'objectif est de décrire les différentes étapes lorsque nous entrainons des modèles. Notre méthode de travail doit répondre le plus possible au cahier des charges. Ici la société « prêt à dépenser » me demande de développer un modèle de scoring sur le « défaut de paiement du client ». C'est une société financière qui veut avoir un modèle performant pour prendre la décision de financer des clients.

2. La méthodologie d'entrainement du modèle

Les données utilisées ont déjà eu une analyse exploratoire et la création de features n'est pas l'objectif de ce projet. Les données sont donc disponibles sur le site de Kaggle.

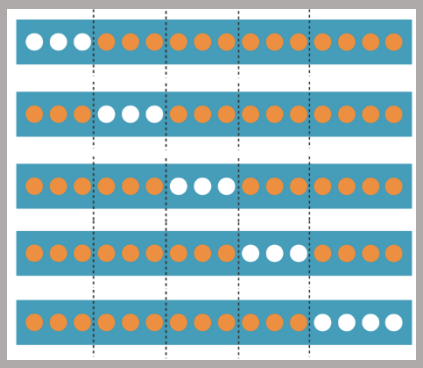
*a. Séparation des d*onnées

Avant toutes choses le jeu de données doit être impérativement séparé avec un trainset et un testset. Lors de cette séparation il faut faire attention à plusieurs choses. La première chose que nous avons pu observer c'est que le jeu de données est déséquilibré au niveau de la target, cest-à-dire qu'il y a beaucoup de réponses positives à un prêt que l'inverse. Il faut donc faire attention à bien le conserver avec « stratify=y ». Dans un deuxième temps, conventionnellement, le trainset représente 80% du jeu de données et le testset doit représenter 20%. Le testset ne doit être en aucun cas être utilisé pour évaluer nos modèles, car si nous entrainons un modèle sur les données qui vont servir à l'évaluer, le résultat sera complétement erroné.

b. Cross validation

Lors de l'évaluation d'un modèle il faut prendre le trainset, le partager pour l'entrainer sur une partie et l'évaluer sur l'autre partie. Cependant on ne peut jamais être sûr que la découpe des deux parties soit parfaite et qu'il n'existe pas une partie plus compliquée que l'autre à évaluer. C'est pour cela

qu'il faut utiliser une cross validation qui va découper le trainset en plusieurs parties, exemple : cinq parties. Le modèle va alors s'entrainer sur 4 parties et va être évalué sur la cinquième partie.



Comme nous pouvons le voir sur le schéma, le trainset est coupé en cinq parties, l'entrainement va être sur les groupes avec les ronds orange et le test sur les ronds blancs. Ensuite il suffit de faire une moyenne des cinq résultats pour avoir l'évaluation de notre modèle.

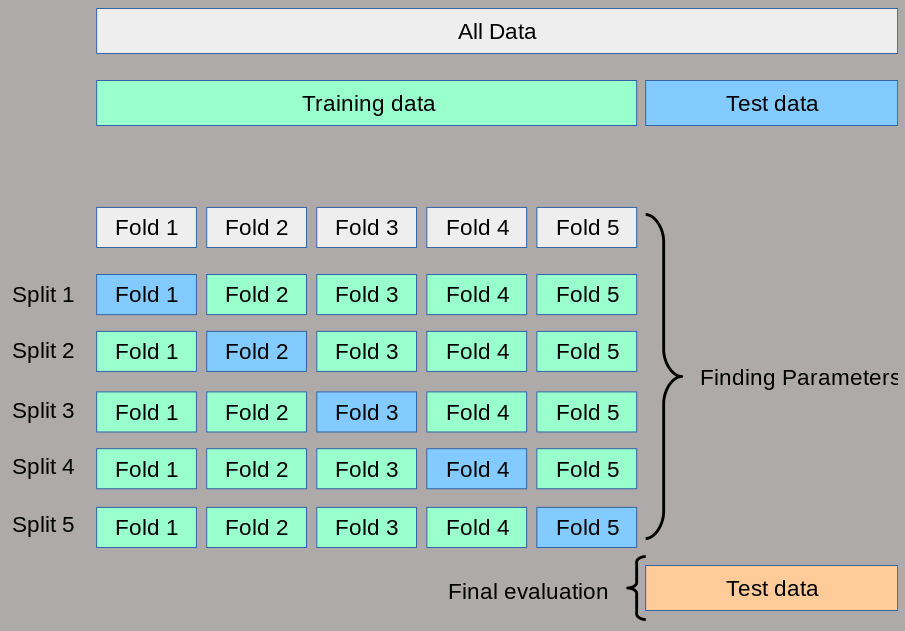


Figure 1 : Schéma global d'un dataset

c. *Les différents modèles utilisés*

Chaque problématique va avoir un modèle plus adapté pour son cas. Les problématiques peuvent être multiples lors d'un choix d'un modèle : rapidité, performance, complexité … ; etc. C'est pour cela qu'il faut tester plusieurs modèles de différents types. Pour notre problématique nous avons besoin de modèles classifiés.

- RandomforestClassifier
- Knn
- LogisticRegression
- LGBMClassifier

Les modèles doivent aussi conserver le même ratio au niveau du résultat de la target, il faut donc rajouter « class_weight='balanced' »

3. La fonction coût, l'algorithme d'optimisation et la métrique d'évaluation

a. La fonction coût, algorithme d'optimisation

La fonction coût est une fonction de perte, elle permet d'évaluer les performances de l'algorithme. Il faut, dans une classification binaire, minimiser la différence entre les deux distributions.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

La fonction coût est expliqué à travers cette formule ci-dessus. Dans notre cas plus la différence entre p(yi) et yi est grande plus l'algorithme a une mauvaise performance.

Lors de l'entrainement des modèles, j'ai utilisé la fonction cross validation (comme expliqué ci-dessus) et ensuite j'ai évalué différentes combinaisons d'hyperparamètres pour trouver la meilleure combinaison possible par rapport à la métrique que j'ai faite (expliqué ci-dessous). Les combinaisons d'hyperparamètres sont recherchées grâce à la fonction RandomSearchCV qui va tester un certain nombre de combinaisons (n_iter) afin de trouver les meilleures performances (scoring=metrique). Il faut dans un premier temps élargir les choix possibles autour de la valeur standard puis affiner par la suite par rapport aux résultats.

b. *La métrique d'évaluation*

L'évaluation du modèle doit suivre une logique d'entreprise. Nous n'allons pas chercher la même chose suivant les objectifs de chaque entreprise, il est donc nécessaire d'adapter sa méthode de scoring. Pour cela deux solutions existent, soit nous pouvons utiliser une métrique qui existe déjà (si elle répond aux problématiques de l'entreprise), soit il faut créer sa propre métrique afin de répondre au plus près possible de la problématique de l'entreprise. Ici, l'entreprise veut un modèle de scoring de « la probabilité de défaut de paiement du client » donc ce qui va faire perdre de l'argent c'est quand le modèle va prédire que le client est régulier alors qu'il ne va pas payer (défaut de paiement) c'est ce qu'on appelle un faux négatif.

		Prédiction	
		1	0
Réalité	1	Vrai positif	Faux négatif
	0	Faux positif	Vrai négatif

0 = client régulier

1 = client défaut de paiement

Lorsque je crée ma métrique, je mets plus de poids lorsque le modèle me donne 0 (le client est régulier) alors que le résultat est 1 (défaut de paiement)

Pour notre métrique nous allons mettre 5 à chaque erreur de type 1 (le faux négatif) est ce qu'on appelle la Précision c'est une fonction qui s'écrit :

$$Précision = \frac{Vrai\ Positif}{Vrai\ Positif + Faux\ Positif}$$

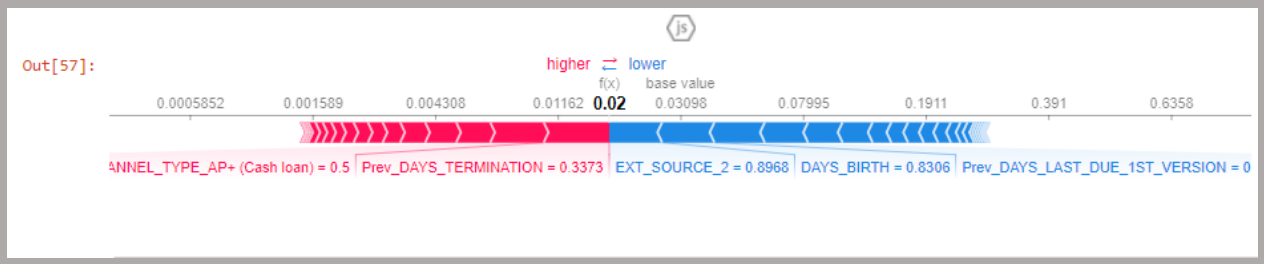
Notre métrique devra également prendre en compte les erreurs de type 2 (le faux positif) c'est ce qu'on appelle le Recall (rappel) :

$$Rappel = \frac{Vrai\ Positif}{Vrai\ Positif + Faux\ Négatif}$$

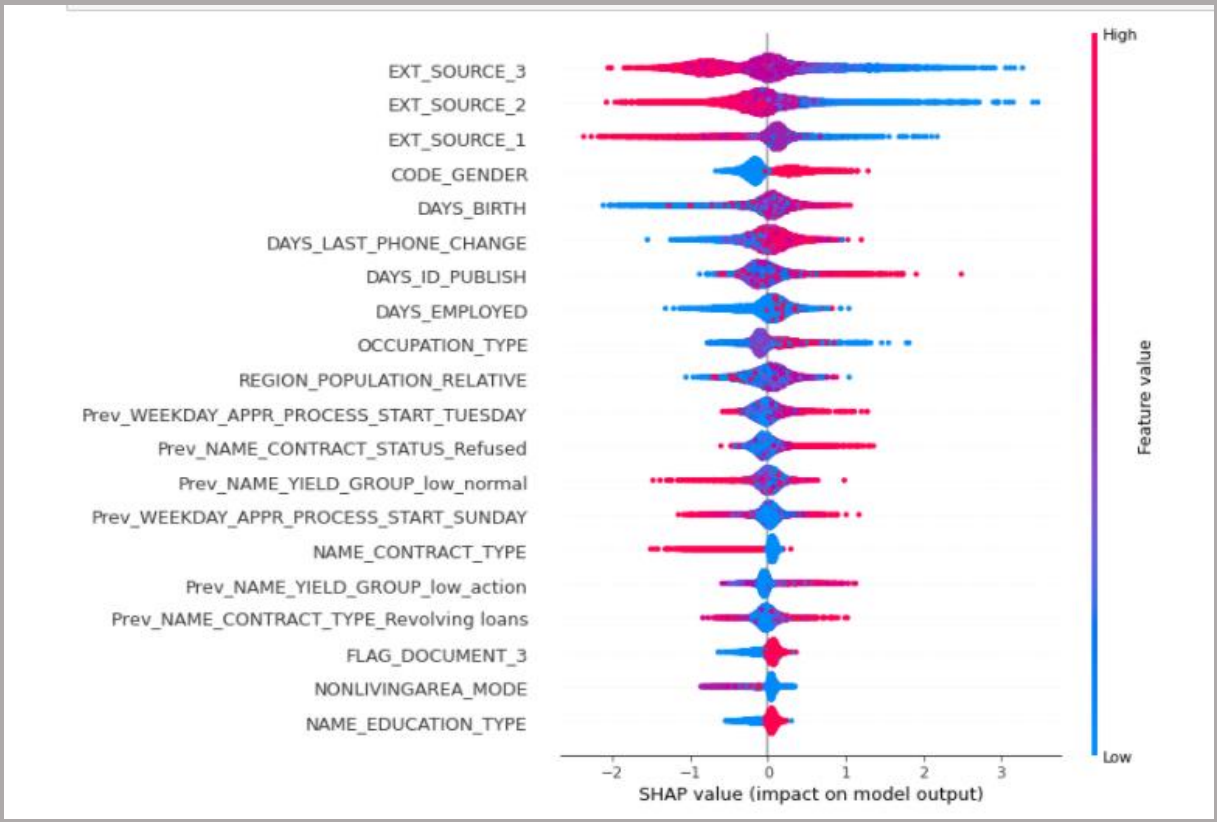
Pour notre métrique nous allons affecter la valeur de 1 à chaque erreur de ce type.

4. L'interprétabilité du modèle

Pour l'interprétabilité du modèle, j'ai choisi d'utiliser shap qui permet d'expliquer les sorties de tout modèle d'apprentissage automatique. Le premier graphique que j'utilise (shape.force_plot) me permet d'identifier les features qui font augmenter et/ou baisser ma probabilité de prédire un client régulier ou un client avec un défaut de paiement.



Pour le deuxième graphique j'ai choisi de regarder les features importantes (shap.summary_plot) qui vont me permettre d'avoir sur l'axe des abscisses, l'impact de la feature sur la sortie et en ordonnée le nombre de valeur de cette feature.



5. Les limites et les améliorations possibles

Comme nous pouvons le constater, l'évaluation des modèles peut complètement varier suivant le scoring que nous utilisons. Plus particulièrement, la métrique qui a été créée sous une hypothèse que l'entreprise va perdre de l'argent si le modèle prédit que le client est régulier mais que finalement le client va avoir un défaut de paiement. L'entreprise va devoir rembourser le crédit non payé et donc perdre de l'argent.

Cette prise de position pourrait être différente si nous pouvions avoir une discussion directe avec le directeur marketing ou les objectifs de l'entreprise. L'information pourrait changer toute la vision du projet et avoir comme conséquence le changement du modèle choisi.

Nous pourrions également essayer plus de modèles ou avoir un jeu de données plus équilibré pour encore mieux cibler les personnes qui ont un défaut de paiement.