Note méthodologique

Sommaire

1. Méthodologie d'entraînement du modèle

Tout d'abord, le jeu de données que nous utilisons pour la modélisation est issue de la table "application_train.csv" car il contient la variable cible qui nous intéresse ("TARGET"). Ce jeu de données a des valeurs manquantes, des variables avec des échelles différentes, ainsi qu'un déséquilibre des classes de la variable cible. Par conséquent, pour entraîner un modèle et réaliser des prédictions avec, nous devons effectuer des étapes de prétraitement. Pour cela, nous créons un pipeline composé des étapes suivantes :

1. Imputation des valeurs manquantes par la médiane (SimpleImputer)
2. Normalisation par un algorithme MinMax
3. Traitement du déséquilibre des classes (SMOTE)
4. Modèle

Puisque la variable cible est catégorielle, nous cherchons à résoudre un problème de classification. Ainsi, nous essayons 3 modèles de classification différents : le dummy classifier, la régression logistique (linéaire) et le lightGBM (non-linéaire). Le dummy classifier nous sert de base de référence pour comparaison avec les autres classifieurs. Ce modèle prédit toujours la même valeur. Il s'agit d'un modèle constant qui n'apprend pas (contrairement aux autres).

A partir de notre jeu de données, nous découpons un jeu d'entraînement et un jeu de test, représentant respectivement 80% et 20% des données. Le jeu d'entraînement sert à entraîner un modèle, tandis que le jeu de test sert à évaluer les performances d'un modèle sur de nouvelles données et comparer ces performances avec celles d'autres modèles.

Ensuite, pour chaque modèle que nous essayons, nous intégrons le pipeline dans une grille de validation croisée. Cela nous permet d'utiliser l'intégralité du jeu d'entraînement, pour l'entraînement et la validation du modèle. En effet, nous découpons ce jeu en 4 folds (parties). Tour à tour, chaque fold sert de jeu de validation (représentant 20% du jeu de données entier). Lorsqu'un fold est utilisé comme jeu de validation, les autres sont utilisés comme jeu d'entraînement (représentant 60% du jeu de données entier).

Le jeu de validation permet d'optimiser des hyperparamètres d'un modèle, en comparant ses performances pour différentes valeurs d'hyperparamètres. Pour évaluer les performances d'un modèle, nous moyennons les performances obtenues sur les folds (lorsqu'ils ont servi de jeu de validation). Ainsi, nous sélectionnons le modèle ayant les performances les plus élevées, c'est-à-dire celui ayant les hyperparamètres optimaux (parmi les différentes valeurs essayées).

En outre, pour évaluer les performances d'un modèle, nous utilisons les critères suivants :

- accuracy : proportion de prédictions correctes
- AUC (Area Under the Curve) : aire sous la courbe ROC (taux de vrais positifs en fonction du taux de faux positifs)
- score métier

C'est en privilégiant le score métier que nous choisissons le modèle optimisé par validation croisée. Ensuite, nous évaluons les scores (selon les mêmes critères) de ce modèle optimal, sur le jeu de test.

Par conséquent, après avoir optimisé les hyperparamètres de nos 3 classifieurs, nous comparons leurs scores sur le jeu de test. Puis pour choisir le modèle le plus adapté à notre problématique, nous accordons de nouveau plus d'importance au score métier. Toutefois, nous prenons également en compte le temps qui a été nécessaire à l'entrainement de chaque modèle, ainsi que le temps pris par chacun pour calculer les prédictions.

2. Traitement du déséquilibre des classes

Pour la variable cible, nous avons un total de deux classes, avec 0 la classe majoritaire (91.93%) et 1 la classe minoritaire (8.07%).

Pour traiter le déséquilibre des classes, nous employons SMOTE (Synthetic Minority Oversampling TEchnique). Il s'agit d'une méthode de sur-échantillonnage (oversampling). Afin d'équilibre les classes, SMOTE génère de nouveaux individus appartenant à la classe minoritaire. Les individus générés ressemblent aux autres individus de la classe minoritaire, sans être des clones identiques. Ainsi, l'échantillon d'individus minoritaires est densifié de manière plus homogène avec SMOTE, qu'avec un clonage aléatoire de certains de ces individus.

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

Le F-score est une mesure de la performance d'un modèle de classification. Il représente la moyenne harmonique de la précision et du rappel (sensibilité). La précision est la proportion de prédictions correctes parmi les individus qui sont prédits positifs. Le rappel est le taux de vrais positifs, c'est-à-dire la proportion d'individus positifs qui sont correctement identifiés. Dans notre cas, les classes (de la variable cible) sont :

- 1 : client ayant des difficultés de paiement (mauvais client)
- 0 : tous les autres cas (bon client)

Pour notre modèle, nous considérons que la classe 1 est la classe positive, tandis que la classe 0 est la classe négative.

Toutefois, nous devons prendre en compte le déséquilibre du coût métier entre un faux négatif (FN) et un faux positif (FP). Un FN, c'est-à-dire un mauvais client prédit bon client, représente une perte en capital. Un FP, c'est-à-dire un bon client prédit mauvais, représente un manque à gagner en marge. Par conséquent, un FN a un coût plus élevé qu'un FP.

Pour pallier au déséquilibre du coût métier, nous utilisons le Fbeta-score, qui est une généralisation du F-score introduisant le paramètre beta. Ce paramètre permet de donner plus de poids à la précision (beta < 1) ou au rappel (beta > 1). En favorisant le rappel, nous accordons plus d'importance aux FN, ce qui est notre but. Alors fixons par exemple la valeur de beta à 10.

En conséquence, le F10-score constitue notre score métier. Alors nous demandons à notre grille de validation croisée, de choisir le modèle ayant la valeur du F10-score la plus élevée. Cela revient à minimiser les FN. La valeur de ce score varie entre 0 (pire valeur) et 1 (valeur optimale). En parallèle, nous comparons ce score métier à des mesures d'évaluation classiques (accuracy et AUC).

Après avoir optimisé les hyperparamètres du modèle, nous pouvons essayer de minimiser encore plus le coût métier (ce qui revient à augmenter la valeur du F10-score). Pour cela, nous pouvons tenter d'optimiser le seuil déterminant à partir d'une probabilité la classe 0 ou 1. Par défaut, ce seuil est de 50%. Ainsi, nous calculant le F10-score pour différents seuils allant de 0% à 100%, avec un pas de 10%. Cependant, nous déterminons également pour chaque seuil, la proportion d'individus prédits comme appartenant à la classe 1. Cela dans le but de s'approcher le plus de la proportion de 1 dans le jeu de données. Par conséquent, nous cherchons un seuil augmentant la valeur du F10-score, tout en se rapprochant le plus de la proportion de 1 dans le jeu de données.

4. Tableau de synthèse des résultats

model	fit_time	pred_time	acc_val	acc_test	auc_val	auc_test	fbeta_val	fbeta_test
Dummy	10.5588	1.2766	0.9190	0.9204	0.5000	0.5000	0.0000	0.0000
Logistic	24.5975	1.3124	0.6971	0.6947	0.7369	0.7362	0.6326	0.6226
LGBM	223.8890	30.4411	0.9178	0.9187	0.7353	0.7415	0.0521	0.0497

Pour comparer les classifieurs entre eux, nous nous intéressons particulièrement aux scores sur le jeu de test, ainsi qu'aux temps (en secondes) d'entraînement et de prédiction.

Parmi les trois classifieurs, la régression logistique a le score métier (fbeta_test) le plus élevé. De plus, son accuracy est décente et son AUC est bonne. D'ailleurs, il semble que nous n'ayons pas de problème de sur-apprentissage car l'AUC n'est pas supérieure à celle du premier du challenge Kaggle (0.82). De surcroît, le temps d'entraînement et le temps de prédiction de la régression logistique sont relativement courts. C'est donc ce modèle que nous choisissons.

Pour finir, nous remarquons que les scores sur le jeu de test sont très proches de ceux sur le jeu de validation. Ainsi, la régression logistique est prête à être mise en production.

5. Interprétabilité globale et locale du modèle

La feature importance est un score calculé pour une variable du modèle de prédiction. Ce score reflète l'importance de la variable pour la prédiction. La feature importance peut être globale ou locale. Elle est globale lorsqu'elle prend en compte toutes les prédictions, et locale lorsqu'elle se concentre sur une prédiction spécifique.

Pour la régression logistique (également appelée "modèle logit"), les feature importances globales correspondent aux coefficients des variables dans la fonction de décision. Chaque coefficient traduit le changement de la fonction logit, par unité de la variable associée.

Pour déterminer la feature importance locale sur un individu du jeu de données, nous pouvons estimer les valeurs de Shapley de cette instance. La valeur de Shapley d'une variable correspond à sa contribution à l'écart entre la valeur prédite et la moyenne des prédictions du modèle.

Lorsqu'une variable a une importance (globale ou locale) positive, cela indique qu'elle augmente la probabilité que la classe prédite par le modèle soit 1 (mauvais client). Au contraire, lorsqu'elle a une importance négative, cela indique qu'elle diminue cette probabilité.

6. Limites et améliorations possibles

Pour équilibrer les classes de la variable cible, nous avons utilisé SMOTE en laissant le nombre de plus proches voisins par défaut qui est de 5. Or le nombre de voisins peut également être optimisé au sein d'une grille de validation croisée.

D’ailleurs, pour nos trois classifieurs, nous avons essayé un nombre limité de valeurs pour les hyperparamètres. Nous a même laissé la valeur par défaut pour plusieurs d’entre eux. Or, en intégrant plus de variation pour les hyperparamètres dans les grilles de validation croisée, nous pouvons potentiellement trouver des modèles plus performants.

De même, nous avons essayé un nombre limité méthodes de prétraitement (dans le pipeline) et de classifieurs. D’abord, en employons d’autres méthodes de prétraitement, il se peut que nous améliorions les performances de la régression logistique. Ensuite, en expérimentant d’autres classifieurs, il est possible que nous trouvions mieux que la régression logistique.

Dans le but d’augmenter interprétabilité de la régression logistique, nous pouvons comparer la valeur du biais (intercept) à celles des coefficients (feature importances globales). Egalement, nous pouvons convertir chaque coefficient en probabilité d’une variable d’augmenter ou de réduire la chance de prédire la classe 1 (mauvais client).

En ce qui concerne le score métier que nous avons élaboré (F10-score), il ne reflète pas parfaitement la vraie évaluation métier. Par ailleurs, nous avons gardé le seuil par défaut (50%) déterminant à partir d’une probabilité la classe 0 ou 1. En effet, il est impossible d’optimiser le F10-score et la proportion de 1 en même temps. Enfin, il est plausible qu’une autre valeur de beta pour le Fbeta-score soit plus adaptée à notre problème. Cette valeur de beta pourrait éventuellement permettre d’optimiser le Fbeta-score et la proportion de 1.

7. Analyse du Data Drift

A l’aide de la librairie “evidently”, nous cherchons à détecter éventuellement du Data Drift sur les principales features, entre les datas d’entraînement et les datas de production. Les datas d’entraînement correspondent au jeu de données que nous utilisons pour la modélisation issue de la table “application_train.csv”. Les datas de production correspondent au jeu de données de la table “application_test.csv”, et ils nous permettent de simuler l’arrivée de nouveaux clients une fois le modèle en production.

Pour déterminer les principales features, nous choisissons les 10 variables ayant les plus grandes importances (globales) de notre régression logistique. Parmi ces 10 variables, nous détectons du Data Drift pour trois d’entre elles. Par conséquent, il sera nécessaire d’assurer une maintenance du modèle en production afin de garantir des prédictions performantes.