Projet 7 – Guessoum Adnène

Parcours Data Science

« Implémentez un modèle de scoring »

Juillet 2022

NOTE MÉTHODOLOGIQUE

Table des Matières :

I – Démarche de modélisation :

1) Contexte de l'étude et objectifs

objectif du projet :

Développement pour la société d'offre de crédit, « Prêt à dépenser », d'une solution de classification automatique de clients pour l'octroi de crédits :

- Déterminer automatiquement la probabilité qu'un client fasse défaut étant donné ses caractéristiques. Le classer dans crédit « accepté » ou « refusé » sur la base de cette probabilité (score).

- Développer et déployer pour utilisation un tableau de bord lisible et explicatif du score pour le professionnel et le client (non-spécialiste).

Contraintes du projet :

a) La détermination de cette capacité à rembourser repose sur des données comportementales et descriptives puisque l'on ne dispose pas nécessairement d'un historique d'emprunt conséquent pour chaque client. Données disponibles ici :
Home Credit Default Risk

b) Le modèle choisi doit à la fois être un bon prédicteur de la capacité à rembourser et être interprétable par le professionnel et le client.

c) Le modèle doit tenir compte de la spécificité métier et s'intégrer dans l'usage des professionnels dans le contexte d'une start-up (rendre compte des moyens d'améliorer sa capacité à emprunter, être plus sensible aux risques de défaut qu'au coût d'opportunité d'un emprunt non octroyé, être intuitif à l'utilisation et interactif/modulable...)

2) Leçons de l'analyse exploratoire de données : Qui sont les clients ?

La nature et les contraintes du projet impliquent un modèle choisi sur la base des données « clients » disponibles pour l'entraînement. Une analyse exhaustive des données est disponible ici : Extensive EDA.

On revient sur les points importants ayant informé la décision des modèles à entraîner et la méthodologie d'entraînement :

a) Les données d'entraînement sont déséquilibrées. La plupart des individus pour lesquelles on dispose des « targets » sont capables de rembourser leurs prêts et seuls à peu près 15 % de ces 331 408 clients font défaut. Il faudra tenir compte dans la méthodologie et la métrique choisie de cette dimension du problème. On dispose en tout de 282 665 clients de la classe négative, 48 743 clients de la classe positive et 24 823 clients sans Target (à prédire).

b) De nombreuses informations pertinentes sont catégorielles. Elles impliquent un « One hot encoding » en pré-traitement. Les modèles de régression (notamment logistique) impliquent de conserver toutes les catégories des variables « dummy » sauf une pour maintenir une forte rigueur dans l'explication des résultats. Cela peut poser un problème de dimension.

c) Les valeurs manquantes sont nombreuses et pas nécessairement les mêmes pour chaque individu. On ne peut pas éliminer de features sans perdre des informations importantes. On ne peut pas non plus se séparer des « outliers » mécaniquement puisque des caractéristiques significativement différentes de la moyenne peuvent être explicatives d'un défaut de paiement. Probable d'ailleurs que le manque d'information soit significatif pour les personnes ne pouvant pas rembourser. Les valeurs moyennes, maximales et minimales des variables numériques sont intéressantes et prise en compte dans la partie « feature engineering ».

Conclusion :

On privilégie les méthodes d'entraînement et les modèles qui sont peu sensibles aux « outliers » et qui ne nécessitent pas d'imputations pour les valeurs manquantes.

Imputer supposerait de faire une hypothèse sur les raisons du défaut de crédit. Cela nécessite des compétences domaines plus précises et est une voie d'amélioration du modèle.

3) la méthodologie d'entraînement : Démarche comparative dans une perspective métier.

Les modèles :

- **Dummy Classifier** : Modèle « Baseline » qui retourne une prédiction sans tenir compte des features sur la base de la distribution (déséquilibrée) de la « Target » (0 = pas de défaut de paiement du crédit ; 1 = défaut sur l'emprunt).

- **lightGBM** : Modèle « performance » optimisé dans un notebook kaggle consultable ici. Permet de jauger la performance de nos modèles cross-validés et l'efficacité du pré-traitement que l'on a opéré (différent de celui du notebook). Lgbm est en général bien adapté en termes de temps de calcul et de performance pour les grands jeux de données et tend à mieux éviter le sur-apprentissage que Xgboost.

- **XGBoost** : Modèle « boosting » choisie pour sa bonne gestion des valeurs manquantes et des données

déséquilibrées. Minimise les erreurs des apprenants (arbres de décision) par renforcement de gradient. Préféré à Lgbm pour des raisons pédagogiques sur ce projet : diversifier les modèles, gérer le sur-apprentissage et le temps de calcul, opérer par soi-même une cross-validation sur un modèle différent de la plupart des notebooks kaggle disponibles pour ce dataset.

- **Regression Logistique** : Modèle linéaire utilisé comme point de comparaison pour s'assurer du choix du modèle. Nécessite de l'imputation (Simple imputer avec médiane de la variable) et un prétraitement spécifique des déséquilibres entre les deux classes.

- **Random Forest** : Modèle simple de « bagging » pour comparer le gain effectif d'une méthode plus complexe. Nécessite également un prétraitement pour les valeurs manquantes.

Réduction de dimension :

- **Boruta-Shap** : Pour réduire le nombre de variables issues du prétraitement, on utilise un algorithme de sélection fondée sur la contribution de chaque variable au résultat final.

Comment fonctionne Boruta ?

On construit pour chaque variable considérée une « variable ombre » (« shadow feature ») qui est une permutation aléatoire des valeurs de la colonne du dataframe associée à la variable.

On utilise les variables et leurs « ombres » pour prédire la target considérée (On choisit le modèle Lgbm utilisé dans le notebook kaggle et performant pour les problèmes de ce type afin d'assurer que les « features » choisies sont effectivement utiles pour la suite).

On calcule les importances des features dans la prédiction avec Shap (shap values). On ne conserve que les variables qui ont mieux contribué aux résultats que la meilleure « shadow feature » (variable transformée aléatoirement à partir d'une variable existante).

Traiter le déséquilibre dans les données :

- **Class_weight** : On calcule le ratio nombre d'individus de la classe minoritaire sur nombre d'individus de la classe majoritaire pour déterminer une estimation du poids à affecter pour les algorithmes qui l'acceptent. On opère ensuite une optimisation d'hyper-paramètres proches pour s'assurer que le poids est bien choisi.

- **SMOTE et RandomUndersampler** : On transforme artificiellement les données en créant ou en supprimant des individus de chaque classe pour les rééquilibrer.

Train-validation-test split et Cross-validatiion des hyper-paramètres :

- « **Train-test split »** : On conserve un set de validation (20 % des données) pour lequel on dispose des vraies valeurs afin de comparer les différents modèles et de garder un œil sur les problèmes de sur-apprentissage. On prend garde à spécifier la stratification pour conserver une proportion similaire des deux classes dans le set de validation.

- **RandomizedSearchCV et GridSearchCV** : permettent de séparer le set d'entraînement en sections (10 « folds ») qui serviront successivement d'entraînement et de test pour l'ensemble des hyper-paramètres considérés. GridsearchCV opère une recherche exhaustive des hyper-paramètres tandis que RandomizedSearchCV tire aléatoirement et combine les différents hyper-paramètres. Il permet de réduire le temps de calcul en maintenant des résultats similaires.

- **StratifiedKfold et RepeatedStratifiedKfold** : permet de spécifier la nature des sections opérées par la cross-validation en répétant l'opération plusieurs fois (3 fois dans notre cas) et en maintenant les proportions entre les classes dans chaque « fold ». La procédure de « cross-validation emboîtée » (« nested cross-validation ») est supposée réduire le risque « d'overfitting » du modèle lors de l'optimisation.

- Hyper-paramètres (HP) pour le modèle choisi (XGBoost) :

Les plus importants pour le XGBoost dans un contexte déséquilibré et de sur-apprentissage rapide du modèle sont :

→ « scale_pos_weight » : défaut = 1 ; contrôle la balance entre classes dans le modèle. La documentation suggère de partir de la valeur suivante pour l'optimisation de cet HP :

« `sum(negative instances) / sum(positive instances)` »

→ « n_estimators » : le nombre d'apprenants (arbres).

→ « learning rate » : réduit le facteur de correction des erreurs du modèle des nouveaux apprenants (sensé réduire l'erreur résiduelle des prédictions faites par les apprenants existants).

→ « max_depth » : profondeur maximale de chaque arbre de décision individuel entraîné (un « apprenant »), arrête la partition des données (« apprentissage ») lorsqu'il atteint cette profondeur.

→ « min_child_weight » : l'arbre cesse la partition s'il atteint une feuille (« leaf node ») pour lequel une partition supplémentaire conduirait à une somme des

poids associés à chaque « enfant » potentiel inférieur à « min_child_weight » (« degré de pureté » minimal acceptable).

→ « Gamma » : réduction minimale de la fonction de perte nécessaire pour accepter une partition supplémentaire. Dans notre cas : fonction de perte associée à l'objectif de classification binaire logistique est par défaut la log loss.

→ « reg_alpha » : paramètre de régularisation L1 des poids. Pénalise l'apprentissage d'un arbre du modèle en ajoutant à la fonction de perte un terme qui augmente de « alpha » la perte à chaque fois que la valeur absolue d'un poids d'une feuille augmente de 1. Autrement dit, minimiser la fonction de perte revient à encourager les prédictions parcimonieuses en termes de « feuilles ». La fonction de perte augmente d'autant plus que l'arbre grandit, ie. que l'on opère des partitions dans nos données (augmentant le nombre de « leaf nodes » finales). La décision d'une nouvelle partition dépendant de la fonction de perte, un alpha plus grand arrête plus vite l'apprentissage et évite « l'overfitting ».

II – Optimisation du modèle : métrique d'évaluation et la fonction de coût

1) Fonction de coûts :

Nous n'avons pas modifié les fonctions de coût (« loss functions » à minimiser) par défaut des modèles :

– la Log-Loss pour classifieur binaire (logistique, Lgbm et XGBoost) :

$$L_{\log}(y, p) = -(y \log(p) + (1 - y) \log(1 - p))$$

avec y : la vérité pour l'individu considéré ; p la probabilité que y soit 1.

– l'impureté de Gini pour le RandomForest Classifier :

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk})$$

avec Qm : données à la « node » m, pmk : proportion de classe k à la « node » m.

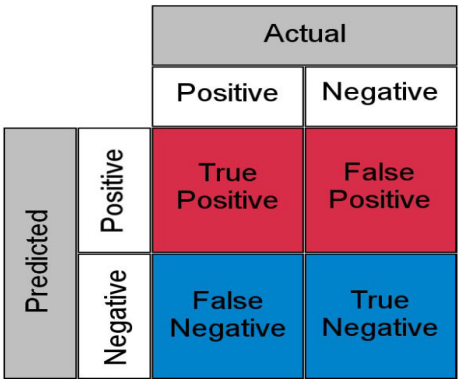
2) Risque métier et coût d'opportunité :

On commence par indiquer les spécificités qu'imposent le problème métier et le choix de l'XGBoost.

Il n'est pas équivalent pour la start-up de faire une erreur de prédiction sur la classe positive (personnes faisant défaut sur leur crédit) et la classe négative.

Il est préférable de ne pas donner un crédit à quelqu'un qui aurait été capable de le rembourser plutôt que d'en donner un à quelqu'un qui ne remboursera pas. La classe positive est 1 (il fera défaut), négative est 0 (il ne fera pas défaut).

Ainsi, si 1) quelqu'un est classé comme positif ('1', fera défaut) mais ne l'est pas ('0', ne fera pas défaut) (faux positif) **bien moins grave** que 2) classer comme négatif ('0', ne fera pas défaut) alors qu'il est positif ('1' - fera défaut) (faux négatif). On lui a accordé un crédit qu'il ne peut pas rembourser, on perd de l'argent (plutôt que l'opportunité d'en gagner plus). c'est la différence entre le risque de perte d'argent et de perte d'opportunité.



3) métriques pertinentes et « fine tuning » :

De ce point de vue, on peut considérer plusieurs métriques pour juger de la pertinence de nos modèles :

– L'AUC - aire sous la courbe ROC :

Métrique principale qui permet de juger de l'efficacité d'une classification, en particulier dans un contexte de données déséquilibré où l'accuracy (calcul de similarité entre valeurs de la prédiction et réalité) sera trivialement haute.

La courbe ROC indique le taux de vrai positif (recall, sensitivity) en ordonnée et le taux de faux positif en abscisse (1 – spécificité). Ainsi, la classification parfaite (sans aucun faux positif ou faux négatif) conduit à une aire sous la courbe de 1 tandis qu'une classification aléatoire qui à une chance sur deux de bien classer un individu (dans le cas d'une classification binaire) conduit à une AUC de 0.5.

– Le fbéta-score (avec un béta de 2) :

C'est la moyenne harmonique pondérée entre la précision (proportion de vrai positifs parmi tout ce que le modèle a prédit positif) et le « recall » (proportion de

positifs détectés parmi tout ce qui était vraiment positif). Comme on cherche à minimiser les faux négatifs et qu'ils diminuent le « recall », on cherche à favoriser l'effet du « recall » sur le fbéta_score sans abandonner la précision pour autant. On choisit donc un score de 2.

– *un score métier* :
« credit_score » dans lequel un faux négatif contribue à l'augmentation de la métrique 10 fois plus qu'un faux positif (cs = 10*fn + fp).

- le « recall » : tp / (tp + fn)

- la spécificité : tn / (tn + fp)

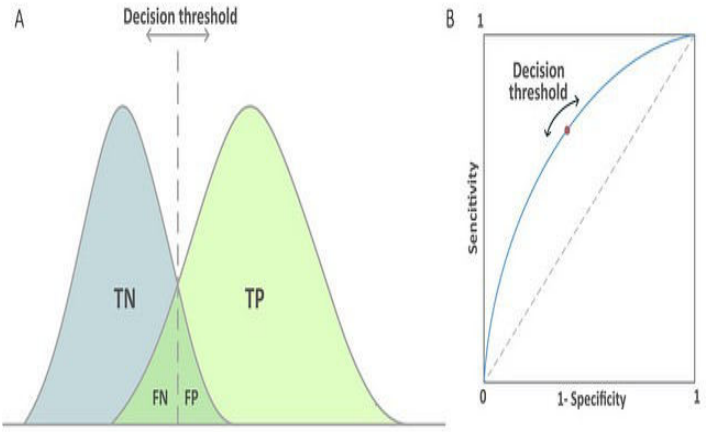
- la précision : tp / (tp + fp)

On garde à l'esprit également que le modèle XGBoost entre très vite en sur-apprentissage. C'est un problème significatif pour une start-up que de nombreux projets kaggle ne prennent pas en compte. Le modèle Lgbm, largement utilisé sur la plateforme pour ce problème, a également un sur-apprentissage très fort (auc entraînement à 0.9, contre 0.79 pour validation). Cela s'explique par la nature des compétitions kaggle dans lesquels toute amélioration de la prédiction est bonne à prendre.

Pour un produit robuste et qui ne nécessite pas de ré-entrainer le modèle trop souvent, dans un contexte où l'entreprise peut-être amener à changer d'échelle ou de niche de clients, il faut tenir compte de la proximité entre les valeurs des scores sur les données d'entraînement et de validation.

4) la question du seuil de prédiction :

Les quelques explications précédentes montrent assez vite qu'il existe un arbitrage à faire entre recall et spécificité. La détection de tous les faux négatifs implique d'accepter un taux de faux positif élevé.



Le score crédit et l'expertise du domaine peuvent permettre de faire ce type de choix. Si on considère qu'un faux négatif est 10 fois plus coûteux qu'un faux positif ; alors il suffit de chercher à minimiser le score de crédit en déplaçant le seuil de prédiction (seuil d'assignation à une classe basée sur la probabilité prédite, par défaut : 0.5)

Pour la procédure d'optimisation, nous avons donc procédé ainsi :

– On conserve le modèle XGBoost qui obtient le meilleur AUC avec un sur-apprentissage plus faible que le modèle lgbm (AUC équivalent, et modèle tiré de Kaggle donc pas très intéressant d'un point de vue pédagogique).

– On opère une cross-validation des hyper-paramètre toujours fondé sur l'AUC pour tenter d'améliorer le modèle sous deux dimensions : réduire le sur-apprentissage, améliorer si possible l'AUC.

– Ensuite, en deux temps :

1) On cherche sur les données d'entraînement à définir, après la cross-validation du modèle et la prédiction des probabilités, le seuil de prédiction de la classe qui minimise le score crédit.

Puis sur la base de ce seuil, on prédit les catégories des individus du set de validation.

Enfin, on compare les prédictions sur la base des métriques définies ci-dessus (seuil par défaut et le nouveau seuil).

2) On tente deux nouvelles recherches d'hyper-paramètres en définissant comme score à maximiser/minimiser successivement le f2_score puis le credit_score lui-même. On utilise les mêmes valeurs de paramètres que pour l'optimisation avec l'AUC.

On répète dans les deux cas la recherche du seuil de prédiction avec le crédit score.

Conclusion :

L'optimisation sur la base du f2_score et du credit_score n'ont pas significativement amélioré la prédiction sur le set de validation. Du point de vue du sur-apprentissage par contre, il est pratiquement éliminé dans le cas de la cross-validation avec AUC. Il vaut donc mieux choisir le modèle XGBoost paramétré avec l'AUC.

Enfin, à propos du changement de seuil pour le XGBoost – auc. On constate que le seuil déterminé sur le set d'entraînement ne permet pas d'améliorer la métrique métier sur le set de validation. Le nouveau seuil est trop proche de 0.5 pour un gain significatif lors de la prédiction sur de nouvelles données.

Les poids visant à rééquilibrer classe majoritaire et minoritaire, paramétré avec l'AUC, ont donc déjà relativement bien jouer leur rôle de sorte que le modèle fait la partition entre "bons" et "mauvais" clients de manière optimale et sans être trop dépendant de la structure des données des clients d'entraînement (overfitting réduit).

III – Interpréter les résultats du modèle

1) Meilleur modèle et le problème de la boite noire :

Le modèle XGBoost choisit obtient donc les scores :

AUC training set : 0.808 ; AUC validation set : 0.787

	precision	recall	f1-score	support
0.0	0.96	0.74	0.83	56533
1.0	0.19	0.69	0.29	4965
accuracy			0.73	61498

credit_score validation : 30141
f2_score validation : 0.45037629350893693
auc_score validation : 0.7148089640769664
Precision : 0.1872555410691004
recall : 0.6942598187311179
specificity : 0.735358109422815

Les prédictions sont donc tout à fait honorables d'après les différentes métriques disponibles.

Cependant, comment le modèle est-il parvenu à ces résultats de prédictions ? Quelle relation chaque variable entretient-elle avec la prédiction finale ?

Le problème de l'interprétation est crucial pour l'entreprise qui doit pouvoir expliquer au client pourquoi elle refuse ou accepte son crédit et doit, en particulier s'assurer que le modèle respecte certaines considérations éthiques et juridiques (RGPD) de transparence. Il serait par exemple condamnable par le droit de discriminer un client sur la base de son genre. Il faut pouvoir s'assurer localement et globalement de la contribution relative de chaque « feature » à la prédiction.

Si dans les modèles de régression linaires et logistiques, ces interprétations peuvent se tirer mathématiquement de la structure du modèle (odds ratio, coefficients linéaires, …). Les méthodes ensemblistes fonctionnent souvent comme des boîtes noires qui « apprennent » les poids associés à leurs décisions dynamiquement. Une méthode d'interprétation de l'importance des « features » est celle que propose la librairie SHAP.

2) Fonctionnement et signification de SHAP

Dans la perspective de SHAP, chaque « feature » est un contributeur dans un jeu coopératif qui a pour objectif de prédire la target.

La contribution relative de chaque variable est déterminée par rapport à une prédiction de base ; si l'on ne connaissait aucune des variables du modèle (la prédiction du modèle dummy). Il suffit ensuite de comparer cette prédiction à la prédiction obtenue par un modèle contenant la variable d'intérêt pour obtenir sa contribution relative à la prédiction. L'ordre d'introduction des variables dans le calcul ayant son importance, plutôt que de calculer l'ensemble des permutations possibles dans l'introduction successive des variables, le package shap utilise un algorithme récupérant les poids à chaque feuille successive pour déterminer l'ordre d'introduction souhaitable. La somme des contributions de chaque variable est égale à la différence entre la prédiction de base (moyenne) et la prédiction du modèle.

En dernière instance cependant, on obtient bien pour chaque variable sa contribution relative à la détermination de la prédiction de chaque individu, et sa contribution relative à la prédiction générale du modèle (moyenne des contributions de la variable pour chaque individu du dataset, à laquelle on ajoute la valeur de la « contribution » à la prédiction de base).

On peut proposer des graphiques de ces contributions au niveau du modèle (interprétation globale) ou pour une prédiction particulière pour un individu (interprétation locale).

Les graphiques indiquent les importances de chaque variable et s'ils ont contribué positivement ou négativement à l'établissement de la prédiction (ou au modèle).

3) Interprétation locale et globale - Exemples de graphiques utilisée dans le dashboard pour l'interprétation :

FIG 1 : Summary Plot – interprétation globale de notre modèle avec les variables les plus importantes et le sens de leur influence :

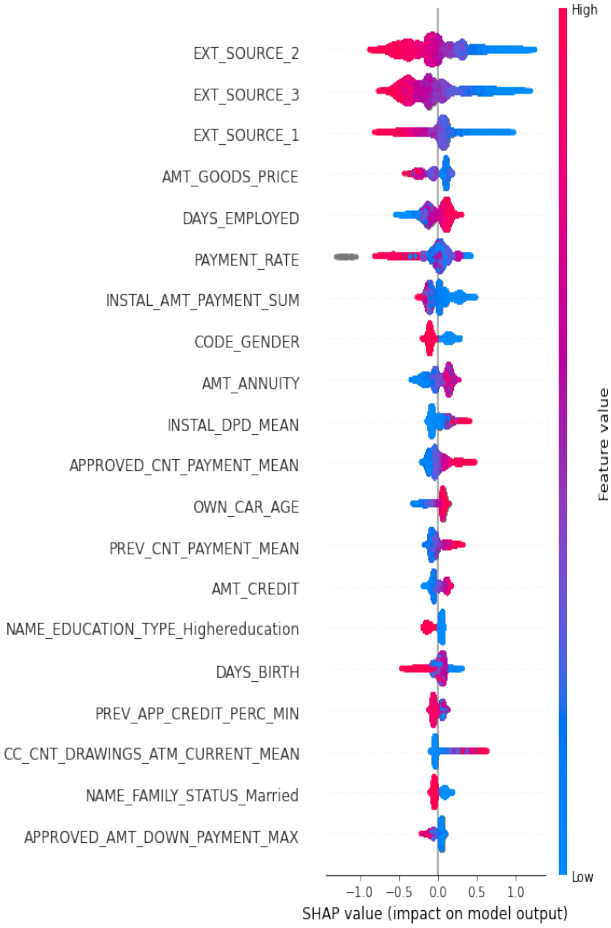


FIG 2 :Decision plot – interprétation locale du processus de décision pour quelques individues du dataset :

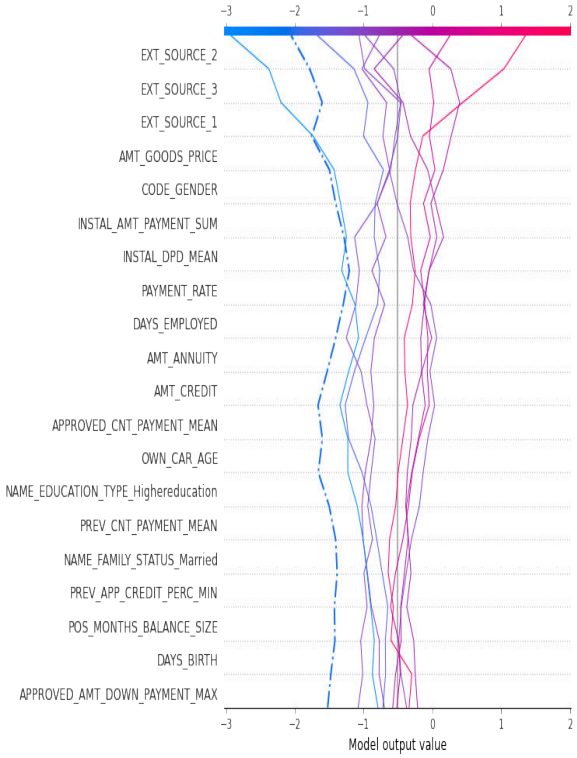
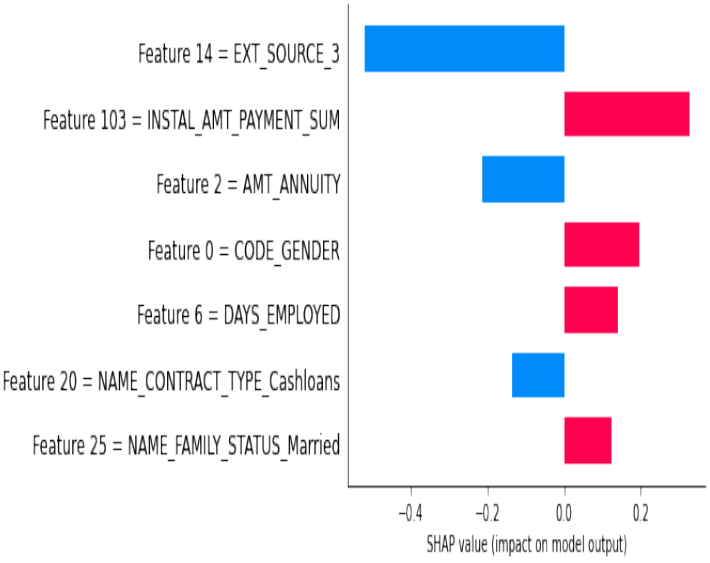


FIG 3 :Bar plot – l'influence spécifique des features pour un individue avec le sens de leurs contributions :



IV – Limites et Perspectives d'amélioration

1) Données, connaissance du domaine et temporalité :

Nous avons bien atteint avec les données disponibles les scores limites obtenues sur kaggle (autour de 0.79) tout en réduisant significativement le nombre de dimensions du problème (nombre de features) et le sur-apprentissage du modèle.

Les voies d'amélioration sont donc à chercher du côté des informations disponibles et collectées sur les clients ou du côté de modèles plus avancées (deep learning, …).

Une dimension significative qui est manquante dans les données et celle de la temporalité. En effet, il peut tout à fait arriver que les causes d'un défaut soit moins dû à la valeur d'une variable à l'instant t qu'à une évolution des caractéristiques clients dans le temps.

De plus, le contexte macro-économique ou la santé générale du marché du crédit (niveau des taux d'intérêts, contexte inflationniste, taux de chômage qui modifie la propension des clients pour le risque…) sont autant de dimensions qui peuvent préciser la prédiction.

Trouver et analyser des séries temporelles économiques et sociales peut peut-être améliorer significativement la prédiction, ou la connaissance du domaine et donc le feature engineering et l'optimisation des hypers paramètres.

Enfin, une expertise issue du domaine du crédit pourrait confirmé l'opportunité d'une métrique comme celle du crédit ou même permettre de construire un score spécifique plus étoffé.

2) le problème du « software engineering » :

L'application (Dashboard) construite est relativement lente et lourde d'usage. Son code source, ainsi que les notebooks, gagneraient à être retravaillé (objets, fonctions…) pour assurer la maintenance du travail à moyen et long terme.