Note méthodologique

La méthodologie d'entraînement du modèle

Rappel de la problématique :

La société financière « Prêt à dépenser » propose des crédits à la consommation. Elle souhaite se doter d'un algorithme à même d'identifier les clients solvables de ceux présentant des risques de défaut de paiement.

Il est proposé de mettre en place un modèle de machine learning afin de répondre à la problématique client.

Comme tout modèle de machine learning, notre modèle doit être entrainé afin d'ajuster ses paramètres à notre problématique. Au-delà des jonctions des tables de données, du nettoyage et de l'encodage qui sont des taches nécessaires préalablement à tout travail sur des données, nous avons conduit une méthodologie particulière résultant de la spécificité de nos données.

Celles-ci ayant la particularité d'être déséquilibrées, complexes et lourdes, il a fallu mettre en œuvre des outils adaptés à ces caractéristiques afin d'apporter une réponse pertinente prenant en compte les spécificités métiers.

Equilibrage des données avec SMOTE

La principale difficulté du jeu de données fourni était le déséquilibre entre les deux catégories de résultat. En effet, les clients ayant eu une réponse positive à leur demande de crédit étaient 11 fois moins nombreux que ceux dont la demande avait été rejetée.

Ce ratio de déséquilibre, de 1 :11, classe nos données dans la catégorie des déséquilibres sévères (plus de 1 :4) qui nous obligent à effectuer un prétraitement des données. Sans cela nos algorithmes de prédiction, qui sont construits pour traiter de jeux de données équilibrés, pourraient ne se concentrer que sur la classe la plus abondante, négligent la classe minoritaire alors même que c'est celle-ci qui est la plus importante (comme dans tous les jeux de données déséquilibrés).

Il n'y a pas de remède universel pour ce problème. Néanmoins, il existe différentes manières de corriger le biais que le déséquilibre cause. Outre les méthodes de undersampling, qui consiste à réduire la classe majoritaire pour qu'elle soit égale à la classe minoritaire, et d'oversampling, qui consiste à dupliquer la classe minoritaire autant de fois que nécessaire pour qu'elle soit égale en nombre à la classe majoritaire, il existe la méthode SMOTE.

SMOTE, pour Synthetic Minority Over-sampling TEchnique, est une méthode d'oversampling mais qui utilise l'algorithme des KNN pour créer des échantillons synthétiques à partir des

données minoritaires. C'est cette technique que nous avons utilisé pour notre problématique.

Ainsi, après avoir divisé notre jeu de données en données d'entrainement et données de test, nous avons appliqué l'algorithme SMOTE à nos données d'entrainement. Cela a permis d'avoir au final 226145 données positives et le même nombre de données négatives.

Recherche d'un algorithme pertinent

Une fois ce travail de rééquilibrage effectué, nous avons recherché quel modèle donnait les meilleurs résultats pour la classification supervisée de nos données avec comme contrainte un jeu de données volumineux (plus de 1GB).

Pour ce faire, nous avons présélectionné un model linéaire et deux modèles ensemblistes qui tous ont des outils de mesure d'importance des caractéristiques, ce afin de permettre d'expliquer globalement les prédictions faites dans un contexte de données complexes.

Pour les évaluer nous avons mis au point une métrique personnalisée qui privilégie la sensibilité et la précision (plus de détail disponible au paragraphe sur la fonction coût).

La régression logistique n'a pas réussi à converger ce qui écarte l'hypothèse de linéarité du modèle.

La forêt aléatoire nous donne une AUC de 0.71 et un score issu de notre métrique de 0.91, avec une sensibilité de 1.85% et une précision de 31.83%.

Le LightGBM nous donne les meilleurs résultats avec une AUC de 0.76 et un score 0.91 également mais avec une sensibilité de 4.05% et une précision de 49.39%.

Ces résultats apparaissent comme meilleur, et d'autant plus qu'ils ont été calculés en 7 fois moins de temps.

Nous privilégions naturellement le dernier modèle.

Amélioration de l'algorithme Halving Grid Search CV

Maintenant que nous avons choisi un algorithme, il nous faut l'optimiser. Pour ce faire, nous choisissons le Halving Grid Search CV. Cet algorithme reprend la méthode de recherche d'hyperparamètres en grille avec validation croisée mais il effectue des rounds sur des portions de plus en plus importante du jeu de données, en ne sélectionnant entre chaque round que les configurations ayant obtenu les meilleurs résultats. Au vu de la taille relativement du jeu de données, cette méthode apparaît comme opportune.

Cette optimisation nous permet un léger gain puisque nous obtenons une AUC de 0.77, un score de 0.91, une sensibilité légèrement moins bonne de 3.56% et une précision en hausse à 57.10%.

Recherche d'un seuil pertinent

Finalement, et une fois nos hyperparamètres arrêtés, nous recherchons quel peut-être le seuil optimal. En effet, notre algorithme, comme tout algorithme de classification génère une probabilité d'appartenance à une classe en fonction d'un seuil (par défaut 0.5). Le choix de ce seuil peut avoir un impact sur les faux positifs et faux négatifs.

Dans le cas de notre modèle, on s'aperçoit que deux seuils sont pertinents (d'après leur ratio de faux positifs, faux négatifs et vrai positifs), 0.4 et 0.5.

Le seuil de 0.4 aura une bien meilleure sensibilité que celui de 0.5 (quasiment x2) mais une précision légèrement moins bonne.

Du coup, la compagnie pourra choisir en fonction de ses priorités et d'un arbitrage entre faux positifs et faux positifs vrais positifs.

	Temps	AUC	Sensibilité	Précision	Score
Regression logistique	1min 29sec	0.68	65.6%	16.4%	-0.32
Forêt aléatoire	5min 05sec	0.71	1.9%	31.8%	0.91
Light GBM	42.5sec	0.76	4.1%	49.4%	0.91
Modèle optimisé seuil 0.4	1min 38sec	0.76	8.3%	44.8%	0.91
Modèle optimisé seuil 0.5	1min 38sec	0.76	4.1%	54.2%	0.91

La fonction cout

Comme évoqué précédemment, l'énoncé proposé est un problème de classification binaire. Ainsi, chaque client se voit attribuer une probabilité qui en fonction du seuil est converti en 0 ou 1.

0 correspond au refus d'octroi d'un crédit et 1 à son acceptation.

Ainsi le modèle peut faire deux types d'erreurs : accorder un crédit à une personne potentiellement insolvable ou refuser un crédit à une personne solvable.

Dans le premier cas, on parlera d'un faux positif, dans le second d'un faux négatif.

Si pour certaines problématiques, ces erreurs se valent, dans notre situation, elles doivent être traitées de manière différente.

En effet, ces deux erreurs n'emportent pas les mêmes conséquences pour l'entreprise puisque la première comporte un risque financier là où la seconde se limite à un

risque commercial dans le cas où le client débouté décide de s'adresser à un autre établissement.

Aussi, la métrique qui évaluera notre modèle doit tenir compte de cette spécificité inhérente au métier en proposant une mesure qui pénalise de manière bien plus importante les faux positifs.

Parallèlement, il faut, à mon avis, que la métrique prenne également en compte le nombre de vrai positifs correctement classifiés. Autrement nous pourrions privilégier des modèles qui ne détectent quasiment pas de positifs.

Ainsi, nous construisons notre propre métrique selon la formule suivante :

$$1 - \frac{10 * faux\ positif + faux\ négatif - 5 * vrais\ positifs}{nombres\ d'individus}$$

Tous les éléments précédemment exposés sont repris avec une pénalité de 10 attribuée aux faux positifs. Concrètement un faux positif est 10 x plus grave qu'un faux négatif.

Ce chiffre de 10 est arbitraire et peut-être révisé en fonction des critères de la compagnie.

En complément de cette métrique, nous devons être attentifs à la sensibilité et la précision des matrices de confusions résultant de nos prédictions.

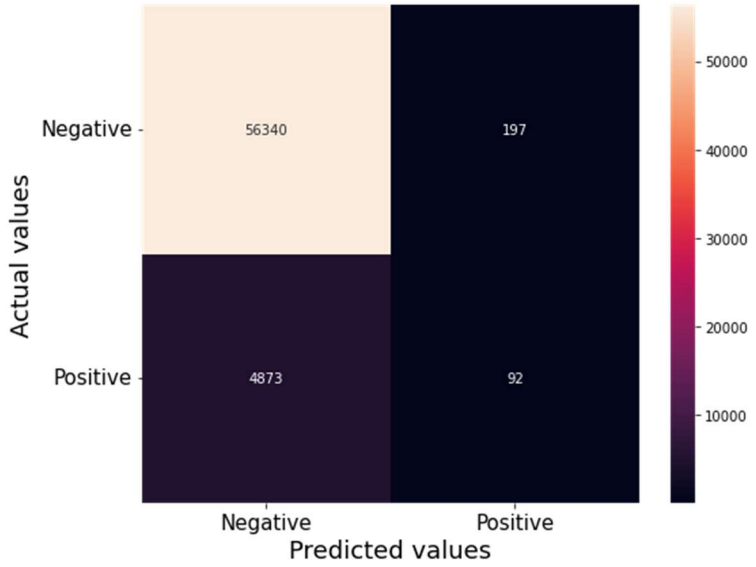
Concrètement, la sensibilité se définie dans notre cas comme les personnes réellement positives correctement identifiées comme tel par le modèle. Elle se calcule comme suit :

$$\frac{vrais\ positifs}{vrai\ positifs + \boldsymbol{faux\ négatifs}}$$

La précision en revanche se regarde comme le taux de personnes réellement positives parmi celles identifiées par le modèle. Elle se calcule comme cela :

$$\frac{vrais\ positifs}{vrai\ positifs + \boldsymbol{faux\ positifs}}$$

Matrice de confusion de la forêt aléatoire



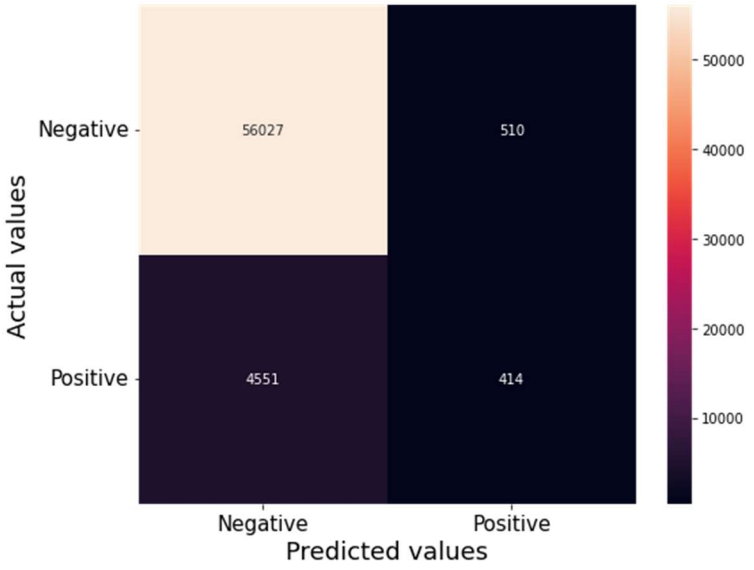
Sensibilité

$$\frac{92}{92+4873} = 1.85\%$$

Précision

$$\frac{92}{92+197} = 31.83\%$$

Matrice de confusion de notre modèle optimisé avec un seuil de 0.4



Sensibilité

$$\frac{414}{414+4551} = 8.34\%$$

Précision

$$\frac{414}{414+265} = 60.97\%$$

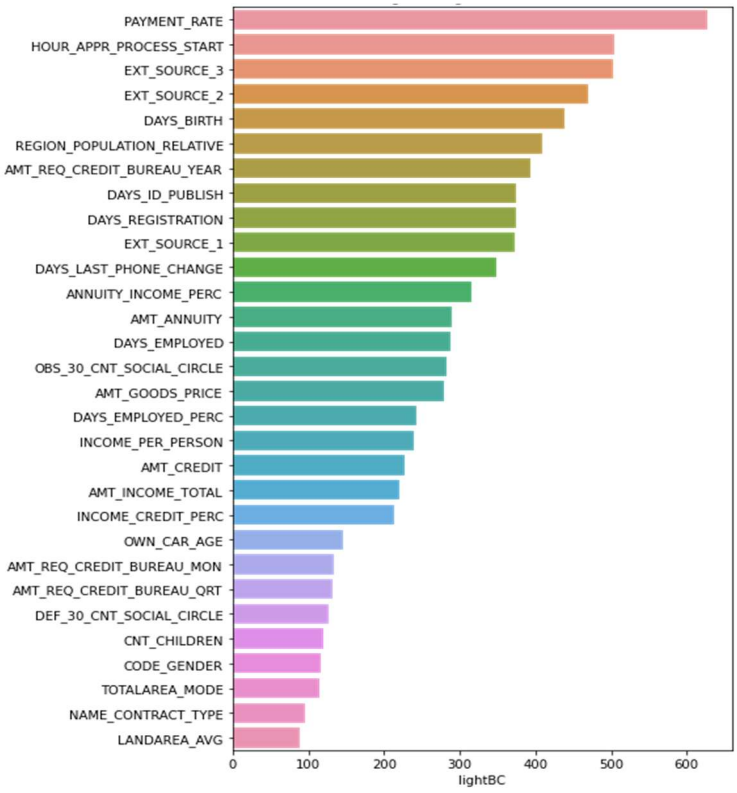
Dans les deux cas le score obtenu avec la métrique personnelle est relativement bon (0.91) mais les valeurs de sensibilité et précision diffèrent énormément et cela nous aide à choisir l'algorithme pertinent.

L’interprétabilité globale et locale du modèle

Au final, nous avons vu que nos données avaient 392 paramètres. Ainsi chaque individu s’est vu attribuer, lors du prétraitement des données, 392 valeurs afin que le modèle puisse établir une prédiction.

Pour interpréter le modèle et établir quels critères ont été déterminants, nous pouvons regarder les approximateurs globaux ainsi que locaux.

Les approximateurs globaux



Les approximateurs globaux expliquent le poids de chaque paramètre pour l’ensemble des individus du modèle.

Les algorithmes que nous avons essayé lors de l’élaboration de notre modèle ont tous une fonction de calcul des importances des paramètres.

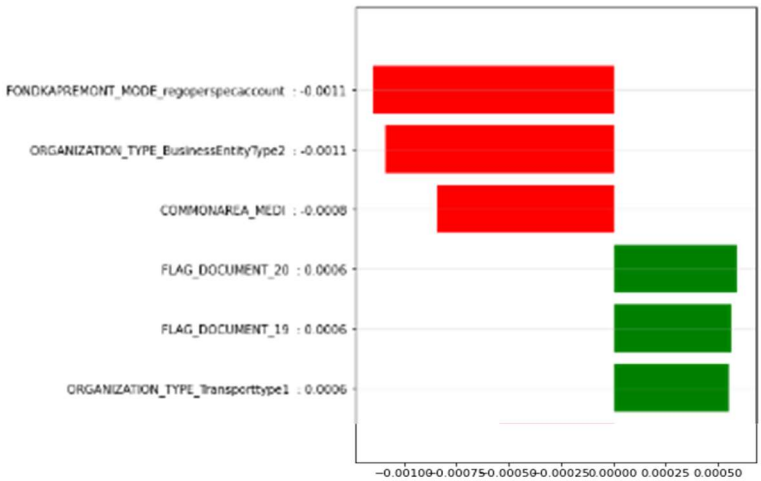
Ici vous pouvez donc voir les 30 paramètres les plus importants de notre modèle.

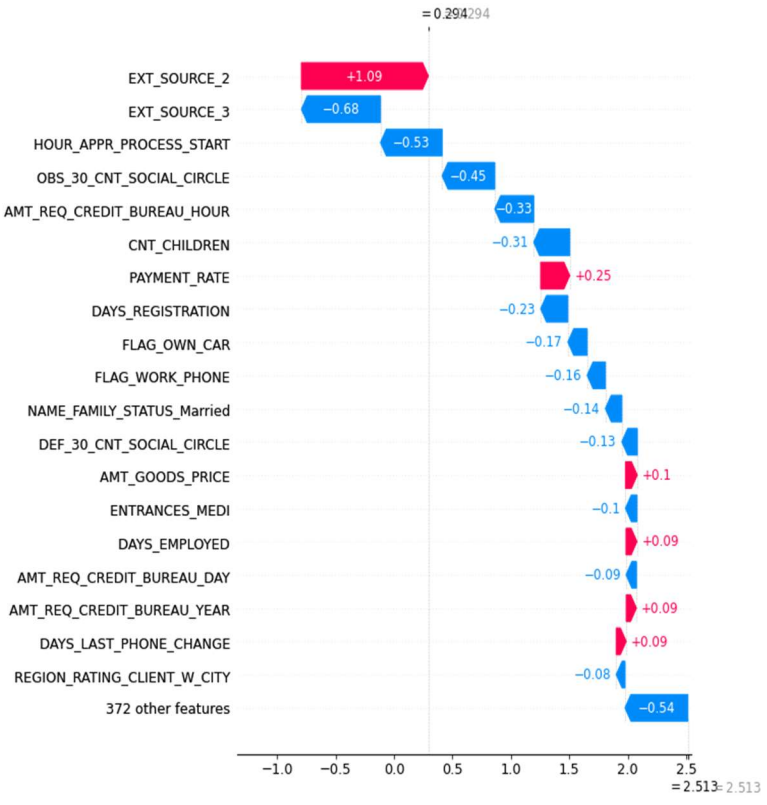
Les approximateurs locaux

A la différence des globaux, les approximateurs locaux ne s'intéressent qu'à un ou plusieurs individus.

LIME

L'algorithme LIME (Local Interpretable Model-agnostic Explanations) crée un modèle autour d'une prédiction donnée afin de l'approximer localement. Autrement dit, il génère de nouvelles données, proche de la prédiction à expliquer puis les apprends avec un modèle interprétable (régression linéaire ou arbre de décision).





SHAP

La méthode SHAP (SHapley Additive exPlanations) propose un classement des contributions des features selon des principes issus de la théorie des jeux.

La valeur de Shapley est la contribution moyenne marginale d'un paramètre par rapport à toutes les autres.

Dans le graphique présenté, vous pouvez voir la valeur de Shapley des 20 paramètres les plus importants. Mis bout à bout dans ce graphique en cascade, ces valeurs permettent de déterminer quelles features ont le plus pesé dans la prédiction.

Les limites et les améliorations possibles

Limites techniques

Des limites techniques se sont très vite imposées à nous, cela a eu pour conséquence de produire un modèle perfectible.

La taille conséquente du jeu de données (plus de 1.4Gb) a rendu leur manipulation complexe. Cela a amené à privilégier des solutions économes en calcul.

Le choix de LightGBM en est une illustration car c'est un algorithme qui optimise la rapidité et l'usage de la mémoire.

Avec une puissance de calcul plus grande, nous aurions pu par exemple faire une recherche d'hyperparamètres plus poussée, même si le Halving grid search nous a quand même aidé.

Nous aurions tout simplement pu également rechercher un autre modèle comme un réseau de neurones.

Trouver les bons features

L'autre point qui peut être amélioré est la sélection des features et l'extraction d'informations en découlant.

Dans notre travail, nous nous sommes grandement inspirés d'un kernel Kaggle comme il était recommandé de le faire. Ce travail de qualité, dont nous nous sommes servis nous a permis de mettre en œuvre les 392 features avec lesquelles nous avons entraîné et vérifié notre modèle.

Trouver d'autres features et en enlever certaines autres auront une influence sur les résultats, mais pour que cette dernière soit positive, il va falloir un gros travail puisque des centaines de personnes ont déjà essayé.

À noter que les features utilisés dans les modèles proposés vont d'une centaine à plus de 1800.

La pertinence des informations récoltées

Un travail sur ces deux axes aurait peut-être permis d'améliorer les performances de notre modèle, car même si celui-ci apporte une solution à la problématique posée, cette solution n'est pas optimale.

En effet, dans notre meilleur configuration (modèle avec un seuil de 0.5), nous avons toujours vingt fois plus de faux négatifs que de vrais positifs.

Ainsi concrètement, nous n'arrivons à identifier qu'un client qui peut bénéficier d'un crédit sur 20 ce qui représente un gros manque à gagner.

Dans l'absolu, pour avoir un modèle plus fiable, il faudrait des données plus pertinentes. En effet, les données fournies, qui sont celles collectées par les banques semblent ne pas être

en mesure d'expliquer clairement la différence entre un client qui remboursera son prêt et un qui fera un défaut de paiement.

Cela est corroboré par l'importance de indicateurs 'EXT_SOURCE_1' , 'EXT_SOURCE_2' et 'EXT_SOURCE_3' qui sont le résultat de scores établis par des 'sources extérieurs' à priori non chiffrables.

Concrètement, cela démontre selon moi qu'il n'y a pas de corrélation évidente entre les informations récoltées et la capacité à rembourser. Cela s'explique sans doute par la complexité du cheminement qui entraîne un individu à ne pas rembourser son prêt et qui est influencé par des facteurs totalement extérieur à sa volonté et à son contrôle.