Projet 7 - Implémentez un modèle de scoring - Note méthodologique

L'objectif de cette note méthodologique est de présenter la démarche de modélisation, permettant de prédire les clients non-solvables, tout en garantissant un gain minimal pour la société. Les données d'entrées sont les informations des clients provenant des données variées (données comportementales, données provenant d'autres institutions financières, etc..), tandis que leur solvabilité ou non-solvabilité représente la sortie du modèle.

Table des matières

1. La méthodologie d'entraînement du modèle approprie

L'objectif du projet est de développer un algorithme de classification binaire, ayant comme spécificité le caractère déséquilibré des classes des données.

Avant l'étape de modélisation, j'ai procédé à la constitution de 2 datasets qui servirons à l'entrainement et prédiction, afin de m'assurer que les données de prédiction, celles utilises dans l'API, n'ont jamais servi à l'entrainement du modèle. Les 2 datasets ainsi constitué sont:
- df_train : utilisé pour entrainement des modèles et optimisation des hyperparamètres,
- df_test_500 : constitué avec la fonction « sample » avec laquelle j'ai constitué un dataset représentatif des données, qui contient les caractéristiques pour 500 clients et qui sera utilisé seulement pour les prédictions effectué à travers l'API.

En s'agissant des données déséquilibrées, un cadre systématique de modélisation comprend les étapes suivantes :

a) Sélection d'une métrique permettant d'évaluer les modèles candidats :
- c'est l'étape la plus importante, car la métrique est l'instrument de mesure qui permet d'évaluer et de comparer tous les modèles. Le choix d'une mauvaise métrique peut signifier le choix d'un mauvais algorithme. La métrique doit capturer les détails sur un modèle ou ses prédictions qui sont les plus importants pour le projet.
- Mon choix a été de prédire les probabilités d'appartenance à une classe en utilisant l'aire sous la courbe (ROC AUC), qui représente la probabilité que le modèle distingue les observations de deux classes et donne un score unique pour un classificateur sur toutes les valeurs de seuil de probabilité. L' ROC AUC est comprise entre 0 et 1. La courbe ROC trace la courbe des faux résultats d'un classificateur binaire.
- Ensuite on converti les probabilités en étiquettes de classe par le bisais d'un seuil défini. Cela permet ainsi de maximiser le taux de vrais positifs et de minimiser le taux de faux positifs.

b) Test d'une série d'algorithmes, impliquant l'utilisation d'une validation croisée de type k-fold pour estimer la performance d'un modèle donné sur les données d'entrainement:

- Les algorithmes choisis sont:

- Naïve type: 'DummyCls' - il est important de vérifier ponctuellement un algorithme d'apprentissage type DummyClassifier sur la classification déséquilibrée, car il n'est pas performant dans cette cas, et ainsi nous fournit une base de référence en matière de performances à laquelle les modèles plus spécialisés peuvent être comparés et doivent être plus performants ;
- Linear algorithme: 'LogisticRegression' – l'apprentissage est souvent rapide et il est souvent très performant.
- Ensemble algorithmes: 'LGBM', 'RandomForest', 'XGBoost' - les algorithmes d'arbres de décision, sont connus pour être très performants en pratique surtout sur les problèmes de classifications.

- La comparaison des modèles effectués via une Cross-Validation sur 5-folds :

- en utilisant la mesure l'aire sous la courbe 'roc_auc', le f1-score et le temps de traitement pour le fit et prédiction ;

- La tactique choisie pour le déséquilibre des classes a été d'utiliser l'argument class_weight='balanced' pour pénaliser les erreurs sur la classe minoritaire. La class_weight ajuste la fonction de coût (cost function) du modèle de sorte que le fait de mal classer une observation de la classe minoritaire soit plus fortement pénalisé que le fait de mal classer une observation de la classe majoritaire. Cette approche peut contribuer à améliorer la précision du modèle en rééquilibrant la distribution des classes. Ainsi, les poids de classe sont ajustés automatiquement selon la formule: n_samples / (n_classes * np.bincount(y)), ce qui donne dans notre cas un poids plus élevé à la classe1-minoritaire (w1=6,2) et poids plus faible à la classe0-majoritaire (w0= 0.54) ;

➔ Les meilleures performances ont été obtenues avec l'algorithme : LightGBMClassifier,

c) Tuning du modèle le plus performant (optimisation des hyperparamètres) pour la classification déséquilibrée :

- la méthode de Gradint Boosting arbre de décision est implémentée pour LightGBM. Les arbres sont construits de manière séquentielle : le premier arbre apprend à s'adapter à la variable cible. Le deuxième arbre apprend à s'adapter au résidu (différence) entre les prédictions du premier arbre et la vérité terrain. Le troisième arbre apprend à s'ajuster aux résidus du deuxième arbre et ainsi de suite. Tous ces arbres sont formés en propageant les gradients d'erreurs à travers le système.
- les principaux paramètres pour optimisation sont : 'num_leaves', 'max_depth', 'min_data_in_leaf'.
- Les données déséquilibrées sont traités à travers le paramètre : is_unbalance = True, qui ajuste automatiquement les poids des classes ;
- L'optimisation des paramètres a été effectuée dans le cadre du RandomizedSearchCV sur 5-plis avec la métrique 'roc_auc', et en sélectionnant 25 paramètres aléatoirement ;
- Pour chaque pli, le score est obtenu sur les 4 plis d'entrainement (moyenne) et le pli test ;
- Le meilleur score obtenu (0.97), permettant de sélectionner les meilleurs paramètres, a été choisi sur l'entrainement ;
- La stabilité du modèle est vérifiée aussi à travers les scores sur l'entrainement et test.

➔ Le meilleur modèle est : LGBMClassifier(is_unbalance=True, max_depth=7, metric='auc', min_data_in_leaf=100, num_iterations=500, num_leaves=71, objective='binary', random_state=13)

d) Réglage du seuil de classification pour les données déséquilibrées:

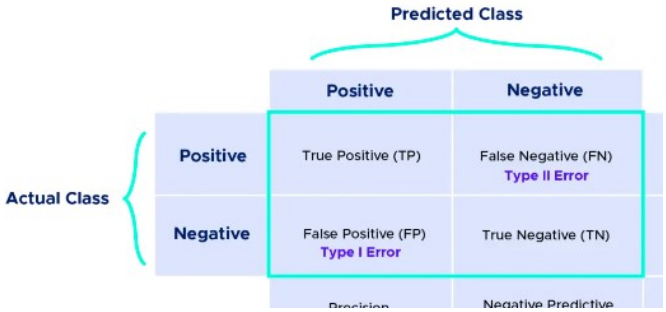
- Les probabilités sont mises en correspondance avec les étiquettes de classe en utilisant une valeur seuil de probabilité/classification. Toutes les probabilités inférieures au seuil sont affectées à la classe 0, et toutes les probabilités égales ou supérieures au seuil sont affectées à la classe 1.

- Afin de régler le seuil de classification, j'ai utilisé plusieurs métriques d'évaluation : la courbe ROC, la courbe Précision-Rappel/Sensibilité, avec la matrice de confusion ;

- Matrice de confusion est un tableau montrant les prédictions correctes et les types de prédictions incorrectes :

- Précision (FPR): le nombre de vrais positifs divisé par toutes les prédictions positives, il s'agit d'une mesure de l'exactitude d'un classificateur. Une faible précision indique un nombre élevé de faux positifs ;

- Rappel/Sensibilité (TPR): le nombre de vrais positifs divisé par le nombre de valeurs positives dans les données de test, il s'agit d'une mesure de la complétude d'un classificateur. Un rappel faible indique un nombre élevé de faux négatifs. Le rappel n'est pas concerné par le nombre de faux positifs et vise plutôt à minimiser le nombre de faux négatifs (FN). Le rappel est approprié lorsque le nombre de faux négatifs est plus critique.

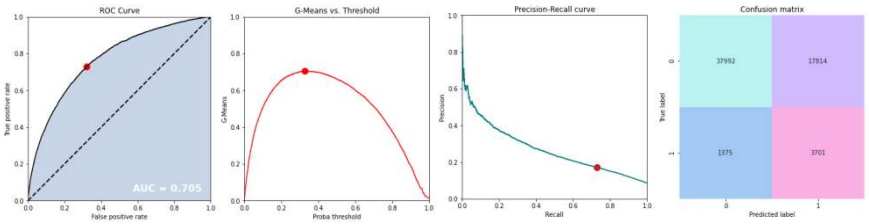


Les mesures utilisées pour le réglage de seuil de classification :

- **G-mean** : est la moyenne géométrique de la Sensibilité/Rappel et de la Spécificité.

$$G-Mean = \sqrt{Recall * Specificity}$$
$$= \sqrt{TPR * \frac{TN}{FP+TN}} \text{ where } \frac{TN}{FP+TN} = \left(1 - \frac{FP}{FP+TN}\right) = \sqrt{TPR * (1-FPR)}$$

➔ Résultat: Threshold = 33%, G-Mean= 70%, AUC = 70%, Recall= 73%, f1_score = 28%
TN = 3701, FP = 1375, FN = 17814, TP = 37992

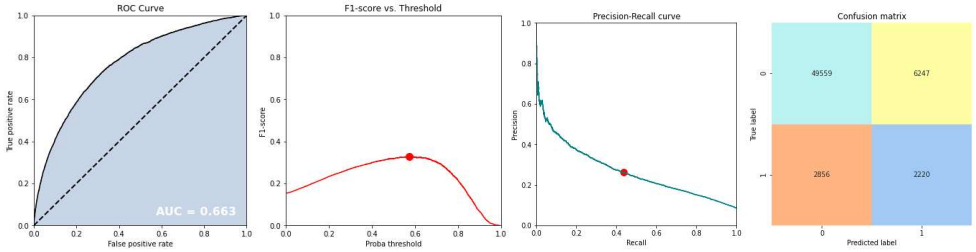


=> En utilisant la G-mean comme mesure d'évaluation non biaisée et comme objectif principal de détection du seuil, on obtient un seuil optimal pour la classification binaire de 0,33.

- **F1-score** : la moyenne pondérée de la Précision et du Rappel. J'ai choisi F1-score, car je souhaite prédire avec précision la classe positive avec un minimum de faux positifs et de faux négatifs.

$$F - Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

➔ Résultat: Threshold = 57%, AUC = 66%, Recall= 44%, f1_score_optimal = 33%
TN = 2220, FP = 2856, FN = 6247, TP = 49559



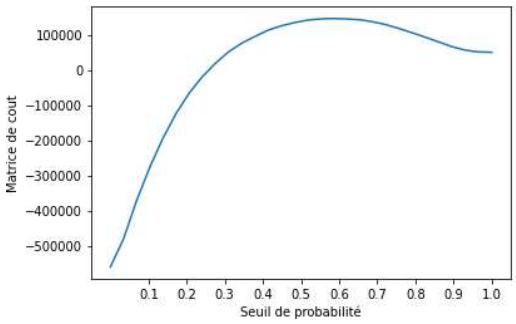
Conclusion: l'ajustement du seuil à l'aide de la courbe ROC, de la courbe Précision-Rappel est une solution alternative pour gérer la distribution déséquilibrée. Dans le cadre du projet, le réglage du seuil en utilisant F1-score donne une meilleure classification binaire.

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

Lorsque nous travaillons sur un problème de classification déséquilibrée, nous sommes généralement plus préoccupés par la prédiction correcte de la classe minoritaire.
Dans le cadre du projet, quel seuil de classification privilégier? Que-ce qu'il faut réduire en priorité ? Dans notre cas, afin de palier la connaissance métier, j'ai défini une matrice de coût qui me permet d'estimer le gain total obtenu par la banque, mon objectif étant de maximiser ce gain à travers la sélection du seuil de classification.

		Classe prédite 0 (+)	Classe prédite 1 (-)
Classe réelle	0 (+)	TP tp_rate=10	FN fn_rate=-10
Classe réelle	1 (-)	FP fn_rate=-100	TN tn_rate=0

Matrice de coût



Le gain pour la banque selon le seuil de classification

J'ai défini ainsi un poids de valeur pour chaque prédiction relativement aux valeurs réelles et qui me permet de calculer le gain total pour la banque, selon le seuil de classification choisi. Ces valeurs sont arbitraires et il est tout à fait possible de changer ces valeurs à la convenance de l'expert métier.
Les coefficients de la matrice de cout sont :

- tp_rate=10, pour chaque client solvable, la banque gagne 10 unités ;
- tn_rate=0, pour chaque client non-solvable qui ne reçois pas le prêt bancaire, la banque ne gagne et ne perd rien ;
- fn_rate= -10, pour chaque client solvable qui ne reçoit pas le prêt, la banque perds 10 unités ;
- fp_rate= -100, pour chaque client non-solvable qui reçoit le prêt, la banque perds 100 unités, car il y a un risque plus important d'impayés de la part du client;

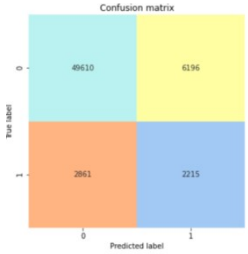
Le gain total obtenu par la banque est défini selon la formule :
G = TP*tp_value + TN*tn_value + FN*fn_value + FP*fp_value
ou TP, TN, FN, FP sont issue de la matrice de confusion.

Sachant que la matrice de confusion dépends de seuil de classification, alors j'ai choisi de modifier le seuil de classification jusqu'à l'obtention d'un gain maximal pour la banque.

Avec la définition de la matrice de coût et le gain pour la banque, j'ai relancé l'optimisation des hyperparamètres du modèle en utilisant la matrice de cout et maximisation du gain : 'make_scorer(matrice_cout, greater_is_better=True)' ;

- Les meilleurs paramètres ont été sélectionnés sur le gain maximal :
  - best_lgbm_model_cout_train = LGBMClassifier(is_unbalance=True, max_depth=7, metric='auc', min_data_in_leaf=100, num_iterations=500, num_leaves=80, min_child_weight=0.001, objective='binary', random_state=13);

- Un Gain maximal de 148040 unités a été obtenu pour un seuil de classification fixé à 57%.
- La matrice de confusion ainsi obtenu est indiquée dans l'image :



Matrice de confusion pour le seuil optimal

3. L'interprétabilité globale et locale du modèle

Feature importance fait référence aux techniques qui attribuent un score aux caractéristiques d'entrée en fonction de leur utilité pour prédire une variable cible. Les scores d'importance des features jouent un rôle important dans un projet de modélisation prédictive, notamment en fournissant un aperçu des données et la sélection des caractéristiques qui peuvent améliorer l'efficacité d'un modèle prédictif sur le problème.

Afin de pouvoir répondre à une problématique métier et comprendre pourquoi, lors d'une classification, un client est considéré comme bon ou mauvais, on doit connaitre les informations pertinentes qui ont contribué à cette classification. Plus particulièrement, ont dois connaitre les caractéristiques principales qui ont contribués à l'élaboration du modèle et les caractéristiques spécifiques à chaque client qui ont entrées en jeu dans le calcule de son propre score de classification.

Ainsi, 2 approches ont été utilisées pour l'interprétabilité des features :

a) Features GLOBALES spécifiques au modèle (LGBMClassifier) sélectionnées en utilisant la fonction : « .feature_importances_ »

Les algorithmes traditionnels d'importance des caractéristiques nous indiquent quelles sont les caractéristiques les plus importantes pour l'ensemble de la population. La permutation d'importance est calculée sur un ensemble de données et il est nécessaire de connaitre la sortie pour calculer ainsi la mesure d'évaluation. Le résultat contient le nombre de fois où la caractéristique est utilisée dans le modèle. Cependant, cette approche globale ne s'applique pas à chaque client individuellement. Ainsi, un facteur qui est important pour un client peut ne pas l'être pour un autre. En ne considérant seulement les tendances globales, ces variations individuelles peuvent être perdues. Pour palier à ce problème, on utilise les valeurs SHAP décrites au point suivant.

b) Features GLOBALES et LOCALES avec SHAP :

SHAP (SHapley Additive exPlanations) est une approche de la théorie des jeux permettant d'expliquer les résultats de tout modèle d'apprentissage automatique. SHAP traite la contribution collective ou individuelle des caractéristiques à la variable cible et donne des explications locales, pour une donnée.

Comment ca fonctionne : nous devons entrainer le meilleur modèle obtenu précédemment. Pour calculer les valeurs SHAP du modèle, nous devons créer un objet explicateur basé sur arbre (TreeExplainer) et l'utiliser pour évaluer un échantillon ou l'ensemble de données. Pour un problème de classification binaire, TreeExplainer.shap_values() renvoie une liste de taille 2. Chaque objet de cette liste est un tableau de taille [n_samples, n_features] et correspond aux valeurs SHAP pour la classe respective. Ainsi, shap_values[1] correspond aux valeurs SHAP pour la classe 1 & shap_values[0] correspond aux valeurs SHAP pour la classe 0. La valeur de SHAP représente la contribution de chaque entité (ligne) dans la prédiction des sorties. Si la valeur SHAP est une valeur fortement positive ou fortement négative, le point de données contribue à prédire la classe positive ou négative.

Importance Features Globales : Pour un problème de classification, shap.summary_plot() trace les valeurs moyennes de valeurs shap pour chaque classe. Les caractéristiques sont répertoriées sur l'axe des ordonnées dans l'ordre de classement, la première étant celle qui contribue le plus aux prédictions et celle du bas étant celle qui contribue le moins ou zéro. Les valeurs shap sont fournies sur l'axe des x.

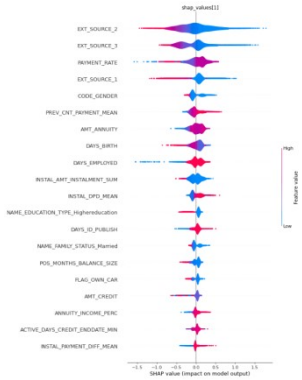


Figure : Summary plot pour shap_values[1]

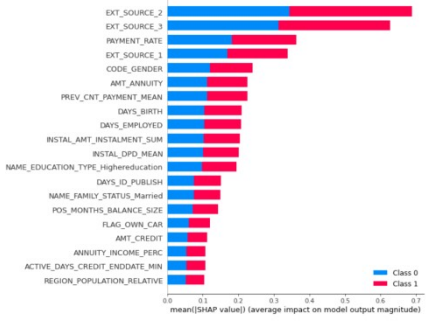


Figure : Summary plot pour toutes les shap_values

Interprétabilité : Les valeurs élevées de la variable « EXT_SOURCE_2 » ont une forte contribution négative sur la prédiction de la classe1, tandis que les valeurs faibles ont une forte contribution positive. Pour la même classe1, la variable « PREV_CNT_PAYMENT_MEAN » a une contribution positive très élevée lorsque ses valeurs sont fortes, et une faible contribution négative lorsque ses valeurs sont faibles.

Importance Features Locales : Pour obtenir l'importance de features locales, c'est-à-dire l'effet de chaque caractéristique sur la prédiction, pour une observation donnée, on utilise la représentation « local bar plot » ou « force plot », qui nous montre quelles sont les principales caractéristiques affectant la prédiction d'une seule observation. Le graphique de force prend une seule ligne et montre dans un ordre de classement comment chacune des caractéristiques a contribué à la prédiction. Plus le bloc d'une fonctionnalité est large, plus la contribution est importante.

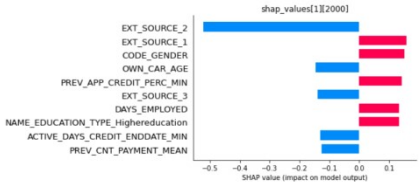


Figure: Local bar plot pour l'observation idx=2000

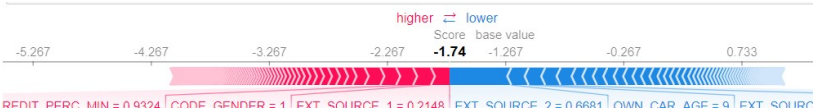


Figure: Force plot pour l'observation idx=2000

Important : alors que SHAP montre la contribution ou l'importance de chaque caractéristique sur la prédiction du modèle, il n'évalue pas la qualité de la prédiction elle-même.

4. Les limites et les améliorations possibles

La classification binaire dans le cadre du projet a nécessité de nombreuses techniques de Machine Learning : création de nouvelles variables, sélection de variables afin de rendre la modélisation plus facile, traitement de données déséquilibrées, choix des métriques et spécifique au métier, réflexion sur la matrice de cout et projection sur le contexte bancaire, sélection du seuil de décision. Tout ce travail présente des limites car il est basé sur ma propre réflexion en tant que datascientist.

Afin de pouvoir améliorer le modèle et ainsi la prise de décision, un travail collaboratif avec les experts métiers, les chargés de clientèle est nécessaire afin de mieux cibler l'intérêt de la banque et son interaction avec les clients. Cet échange nous permettra ainsi de crée des variables les plus adaptés, d'avoir plus de compréhension sur les données externes est d'avoir plus de transparence vis-à-vis du client. En plus, la définition d'une matrice de coût plus proche de la réalité bancaire nous permettra d'améliorer le modèle, et ainsi affiner les hyperparamètres du meilleur algorithme.

Côté client, afin d'assurer la transparence, un travail en amont avec une implication des clients est nécessaire afin de pouvoir améliorer le Dashboard, les représentations techniques, le besoin client.