Projet 7 : Implémentez un modèle de scoring

Note méthodologique

Présenté par :
Bourama FANE
Etudiant Data Scientist

Dirigé par :
Babou M'BAYE
Mentor chez OpenClassrooms

Décembre 2023

Table des matières

1 Introduction

1.1 Problématique

La société financière, nommée **"Prêt à dépenser"**, propose des crédits à la consommation pour des personnes ayant peu ou pas du tout d'historique de prêt.

L'entreprise souhaite mettre en œuvre un outil de **scoring crédit** pour calculer la qu'***un client rembourse son crédit***, puis classifie la demande en crédit **accordé** ou **refusé**. Elle souhaite donc développer un algorithme de classification en s'appuyant sur des sources de données variées.

- ✓ identité,
- ✓ données comportementales,
- ✓ données provenant d'autres institutions bancaires,
- ✓ etc.

Les données financières des clients anonymisés ont été fournies en sept tables, (descriptions et data : `https://www.kaggle.com/c/home-credit-default-risk/data`) avec un colonne cible '**TARGET**' informant si le client a remboursé son prêt (0) ou était défaillant (1).

1.2 Objectifs

- ☞ Analyse d'un jeu de données :
  - — Nettoyage du jeu de données
  - — Mise en place d'une métrique adaptée pour la banque
  - — mettre en place un modèle de régression comme base puis une comparaison d'algorithmes de classification,
  - — déterminer celui qui sera le plus adapté à notre problématique (en prenant en compte le fait qu'il y ait un déséquilibre de **TARGET**)
  - — montrer l'importance des features puis séléctionner les features pertinentes pour les visualisations sur notre **dashboard**.

- ☞ API
- ☞ Dashboard interactif
  - — Visualisation score et interprétation ;
  - — Informations du client ;
  - — Interprétation prédiction modèle.

2 Méthodologie

2.1 Prétraitement des données

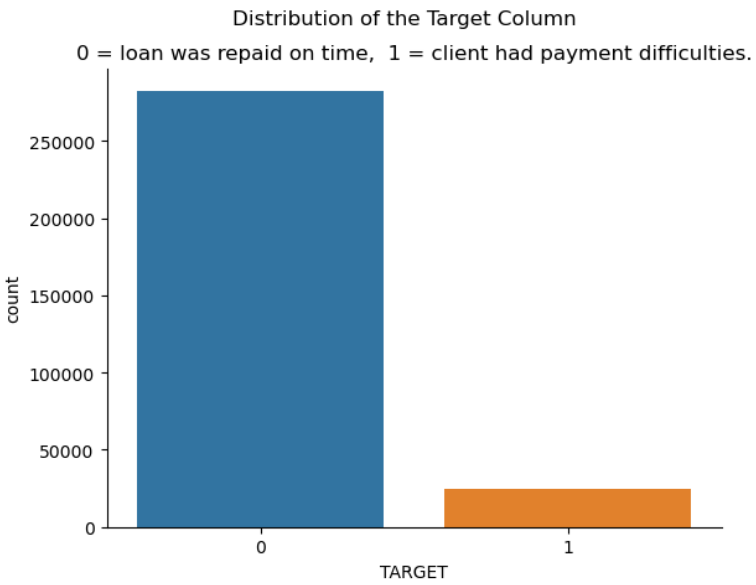
Les tables financières (demande de prêt, l'historique de remboursement, des prêts antérieurs, données externes) ont été fusionnes par JOINTURE sur la clé du client (SK_ID_CURR), avec un script qui a déjà produit des bons résultats de classification. (https://www.kaggle.com/jsaguiar/lightgbm-with-simple-features/script), avec quelques adaptations légères.

- ☞ Elimination des valeurs aberrantes (valeurs infinies, DAYS > 100 ans remplacés par NaN) ;
- ☞ Agrégations sur les tables secondaires (MIN, MAX, MEAN, VAR, SUM, COUNT, PERC, DIFF) ;
- ☞ Label Encoder pour colonnes avec deux modalités (ex. GENDER) ;
- ☞ One Hot Encoder des colonnes catégoriques ;
- ☞ Elimination des colonnes avec un seule valeur (pas de variation) ;
- ☞ Suppression des colonnes avec plus de **40% de missings** ;
- ☞ Supression des colonnes constantes (avec un seule valeur) ;
- ☞ conversion des ages (nombre de jours) en **années** ;
- ☞ changement des valeurs négatives en valeurs positives ;
- ☞ Imputation avec la **méthode interpolate** ;

2.2 Equilibrage des classes

Le problème est un problème de classification binaire : client à risque (classe 1) vs client fiable (classe 0) avec une classe sous représentée (8% clients à risque contre 92% de clients fiable).

Plusieurs méthodes sont envisagées pour équilibrer les classes dans l'ensemble d'entraînement. Elles se distinguent en deux catégories : les méthodes de sur-échantillonnage et de souséchantillonnage.



Stratégies de sur-échantillonnage

SMOTE est une des méthodes de sur-échantillonnage utilisée. En créant des observations synthétiques, on est capable de balancer les classes jusqu'à un rapport de 1. On utilisera la fonction **SMOTE** de la librairie **imblearn**, dont le paramètre **sampling_strategy** permet de déterminer la proportion cible entre la classe positive et négative. Les mêmes paramètres sont disponibles pour la méthode de sur-échantillonnage aléatoire avec la fonction **RandomOverSampler** de la même librairie.

Stratégies de sous-échantillonnage

Pour le sous-échantillonnage aléatoire, on utilisera la fonction RandomUnderSampler. Cette fonction partage aussi le paramètre **sampling_strategy** qui permet de déterminer l'équilibre des classes désiré. Le sous-échantillonnage avec liens Tomek et la méthode Edited Nearest Neighbors sont disponibles avec les fonctions **TomekLinks** et **EditedNearestNeighbours** respectivement. Ces méthodes ne permettent pas d'atteindre une proportion de classe cible mais suppriment des instances avec des considérations de distance.

Combinaisons de stratégies de ré-échantillonnage

Certaines méthodes de sur-échantillonnage et de sous-échantillonnage peuvent être combinées afin de déterminer la meilleure stratégie.

- Ensemble de base non-équilibré (BASE)
- Random Over-Sampling (ROS)
- Random Under-Sampling (RUS)
- SMOTE (SMT)

— SMOTE + Liens Tomek (SMT+TMK)
— SMOTE + Edited Nearest Neighbors (SMT+ENN)

Afin de déterminer la valeur optimale du paramètre **sampling_strategy** des méthodes Random Under−Sampling, Random Over−Sampling et SMOTE, différentes valeurs sont testées pour chaque compagnie et chaque découpage sur un modèle de base. La valeur du paramètre qui donne les meilleures performances est conservée pour chaque découpage et chaque compagnie.

2.3 Selection de features

La sélection de variables consiste, étant donné des données dans un espace de grande dimension, à trouver un sous-sensemble de variables pertinentes. Il faut donc minimiser **la perte d'information** venant de la suppression de toutes les autres variables.

Plusieurs techniques ont été utilisées :

- ☞ **Filtrage** : suppression des variables colinéaires.
- ☞ **Embedded** : des méthodes intégrées qui apprennent quelles variables contribuent le mieux à la précision du modèle pendant sa création. Une valeur est calculée et liée à chaque variable du jeu de données servant à entraîner le modèle.
- ☞ **Automatiques** : basées sur des librairies python (RFE, RFECV, permutation importance avec scikit-learn)

Explicitement, nous avons appliqué les différentes techniques ci dessous :

- ✓ **Pearson Correlation :** Utilisation de la corrélation de Pearson pour évaluer les relations linéaires entre les variables.
- ✓ **SelectKBest :** Utilisation de la méthode SelectKBest pour sélectionner les k meilleures variables en fonction de certaines métriques (par exemple, ANOVA pour les caractéristiques continues).
- ✓ **RFE (Recursive Feature Elimination) :** Utilisation de la méthode RFE pour éliminer récursivement les variables moins importantes.
- ✓ **Logistics Regression L1 :** Utilisation de la régression logistique avec la pénalité L1 pour favoriser la sparsité dans la sélection des variables.
- ✓ **Random Forest :** Utilisation de l'algorithme Random Forest pour identifier l'importance des variables.
- ✓ **LightGBM :** Utilisation de l'algorithme LightGBM pour sélectionner les variables importantes.
- ✓ **Comparaison des 100 'best' features sélectionnées :**.

2.4 Split des données

Dans le cas d'une prédiction supervisée, l'entrainement d'un modèle nécessite une étape de division du jeu de données entre le prétraitement des données et la modélisation. Cette division permet de

créer un échantillon d'apprentissage (train) qui représente 80% des données et un échantillon test qui représente 20% des données.

Ainsi, le jeu de données nettoyé (après sélection des 100 meilleures features) a été en train (80%) et test (20%). Les données ont été mises à l'échelle avec **StandardScaler**.

2.5 Choix des algorithmes

Un modèle de prédiction prend en entrée des données et donne en sortie une conclusion. Pour déterminer l'algorithme optimal de classification adapté à la problématique, quatre algorithmes ont été testés.

- ☞ Dummy Classifier
- ☞ Logistic Regression
- ☞ SVC
- ☞ Decision Tree
- ☞ Random Forest
- ☞ XG Boost
- ☞ Light GBM

- ✓ **Dummy Classifier** : ne prend pas en compte les caractéristiques du jeu de données et se contente de faire des prédictions en utilisant des règles simples. Ici, on renvoie l'étiquette de classe la plus fréquente dans l'argument y observé.
- ✓ **Logistic Regression** : le but est de trouver une relation mathématique entre les variables d'entrée (features) et la variable de sortie (classe prédite). Cette relation est généralement exprimée sous la forme d'une fonction logistique qui transforme la sortie en une probabilité.
- ✓ **SVC** : L'algorithme fonctionne en trouvant un hyperplan optimal qui sépare les données d'entraînement en différentes classes. L'hyperplan est déterminé de manière à maximiser la marge entre les points de données de chaque classe. Les points de données les plus proches de l'hyperplan sont appelés vecteurs de support.
- ✓ **Decision Tree :** algorithme d'apprentissage automatique qui prend un ensemble de données en entrée et construit un modèle prédictif sous forme d'arbre hiérarchique. Chaque nœud l'arbre représente une caractéristique de l'ensemble de données, chaque branche représente une règle de décision basée sur cette caractéristique, et chaque feuille représente une classe ou une valeur prédite.
- ✓ **Random Forest** : algorithme d'apprentissage automatique qui combine plusieurs arbres de décision pour effectuer des prédictions. Chaque arbre de la forêt donne une prédiction et la classe prédite est déterminée par un vote majoritaire.

✓ **XG Boost** : utilise un ensemble de modèles d'arbres de décision pour effectuer des prédictions. L'algorithme fonctionne en itérations successives pour minimiser une fonction de perte spécifiée et ajouter des arbres qui réduisent cette perte.

✓ **Light GBM** : algorithme d'apprentissage automatique basé sur le gradient boosting qui est conçu pour offrir une exécution rapide et des performances élevées. Il utilise une technique d'échantillonnage basée sur le gradient pour sélectionner les échantillons les plus informatifs pendant le processus d'apprentissage.]

3 Évaluation du modèle

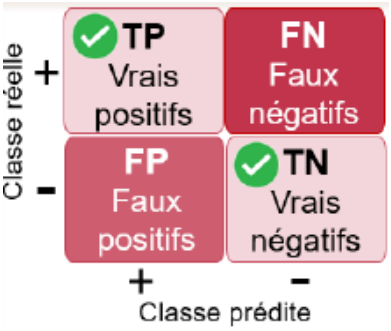
3.1 Métriques de performance

Le choix de la métrique est primordial, dépend de la problématique et permet d'évaluer la performance du modèle prédictif et de **garantir la qualité du modèle** de classification. Ces métriques permettent de comparer les classes réelles aux classes prédites par le modèle.

Pour notre problématique, nous devons minimiser les pertes d'argent pour la banque. De ce fait, notre modèle doit donc :

a.) ne surtout pas prédire un client comme non-défaillant alors qu'il est défaillant (minimiser le nombre de faux négatifs (erreur de type II). Dans ce cas, le **groupe Home Crédit** aura perdu toute la somme prêtée à l'emprunteur.

b.) s'efforcer de ne pas prédire comme défaillant alors que le client est non défaillant (minimiser les faux positifs (erreur de type I). La banque fait un déficit si elle n'accorde pas de prêt à des clients qui peuvent rembourser le prêt.



Nous avons considerer les métriques suivantes :

☞ **Accuracy** : somme de tous les vrais positifs et vrais négatifs qu'il divise par le nombre total d'instances. Il permet d'apporter une réponse à la question suivante : de toutes les classes positives et négatives, combien parmi elles ont été prédites correctement ? Des valeurs élevées de ce paramètre sont souhaitables. Precision : indique le rapport entre les prévisions positives correctes et le nombre total de prévisions positives. Ce paramètre répond donc à la question suivante : sur tous les enregistrements positifs prédits, combien sont réellement positifs ?

☞ **Recall/Rappel** : paramètre qui permet de mesurer le nombre de prévisions positives correctes sur le nombre total de données positives. Il permet de répondre à la question suivante : sur tous les enregistrements positifs, combien ont été correctement prédits ? La mesure vise à minimiser les faux négatifs.

☞ **F1 score** : moyenne harmonique de la précision et du rappel. Il équivaut au double du produit de ces deux paramètres sur leur somme. Sa valeur est maximale lorsque le rappel et la précision sont équivalents. Il est particulièrement difficile de comparer deux modèles avec une faible précision et un rappel élevé. Le contraire est également vérifié. Dans ces conditions, le score F1 permet de mesurer ces deux paramètres simultanément.

☞ **Fbeta score** : généralisation de la F-measure qui ajoute un paramètre de configuration appelé beta. Une valeur bêta par défaut est 1.0, ce qui est identique à la mesure F. Une valeur bêta plus petite, telle que 0.5, donne plus de poids à la précision et moins au rappel, tandis qu'une valeur bêta plus grande, telle que 2.0, donne moins de poids à la précision et plus de poids au rappel dans le calcul du score. Ici nous donnons plus de poids au rappel qui minimise les faux négatifs.

☞ **ROC AUC score** : mesure de façon globale la performance d'un modèle de classification. Il indique à quel point le modèle est capable de faire la distinction entre les classes. Il est égal à 1 pour un modèle parfait et à 0.5 pour un modèle non-informatif.

✓ **maximiser** : AUC, TP, F1, recall

✓ **minimiser** : FN

3.2 Définition d'une métrique "métier"

Il faut créer une métrique pertinente concernant notre problématique pour comparer nos algorithmes. Une société de crédit cherche à "**maximiser**" ses gains.

— Accorder un crédit à quelqu'un ne pouvant pas le rembourser par la suite (FN) est synonyme de perte pour l'entreprise

— Accorder un crédit à un client qui le remboursera par la suite (TN) est un gain.

— Ne pas accorder le prêt et que le client ne peut pas rembourser (TP) n'est ni une perte, ni un gain.

— Ne pas accorder le prêt mais que le client pouvait rembourser (FP) est une perte de client donc d'argent.

On va donc **créer un score en pondérant les différents cas possibles** qui sera normalisé entre 0 et 1 par une normalisation min-max feature scalling avec :

$$gain\_total = (TN * coeff\_TN + FP * coeff\_FP + FN * coeff\_FN + TP * coeff\_TP)$$

$$gain\_min = (TN + FP) * coeff\_FP + (TP + FN) * coeff\_FN$$

$$gain\_max = (TN + FP) * coeff\_TN + (TP + FN) * coeff\_TP$$

$$gain = \frac{(gain\_total - gain\_min)}{(gain\_max - gain\_min)}$$

3.3 Validation croisée

Pour chaque algorithme à tester, une recherche d'hyper paramètres doit être effectuée pour que le modèle soit le plus adapté aux données. L'échantillon d'apprentissage est l'échantillon principal qui permet à un modèle d'ajuster sa prédiction par le biais d'une validation croisée.

L'optimisation pour chaque modèle a été réalisée à l'aide de **GridSearchCV**. Les hyperparamètres sélectionnés seront ceux qui permettent d'obtenir le meilleur score pour l'AUC, Recall et la métrique métier, en utilisant la validation croisée avec 5 plis ($CV = 5$). Le meilleur modèle, i.e celui presentant le es meilleurs scores pour les métriques a été le lightGBM. Ce dernier sera considéré et permettra de classer tout nouveau client en tant que client fiable ou client à risque, et ainsi de décider d'accorder ou non le crédit.

```
model_performance = pd.concat(models_perf, axis=0)
model_performance.sort_values(by        = ['Recall class 1','AUC', 'score Gain', 'FN'],
                              ascending = [False,            False,  False,        True],
                             inplace    = True, )

model_performance = (model_performance.loc[model_performance.Modele.str.contains('Model', case=False),:]
                     .drop_duplicates())
model_performance.reset_index(drop=True, inplace=True)
model_performance
```

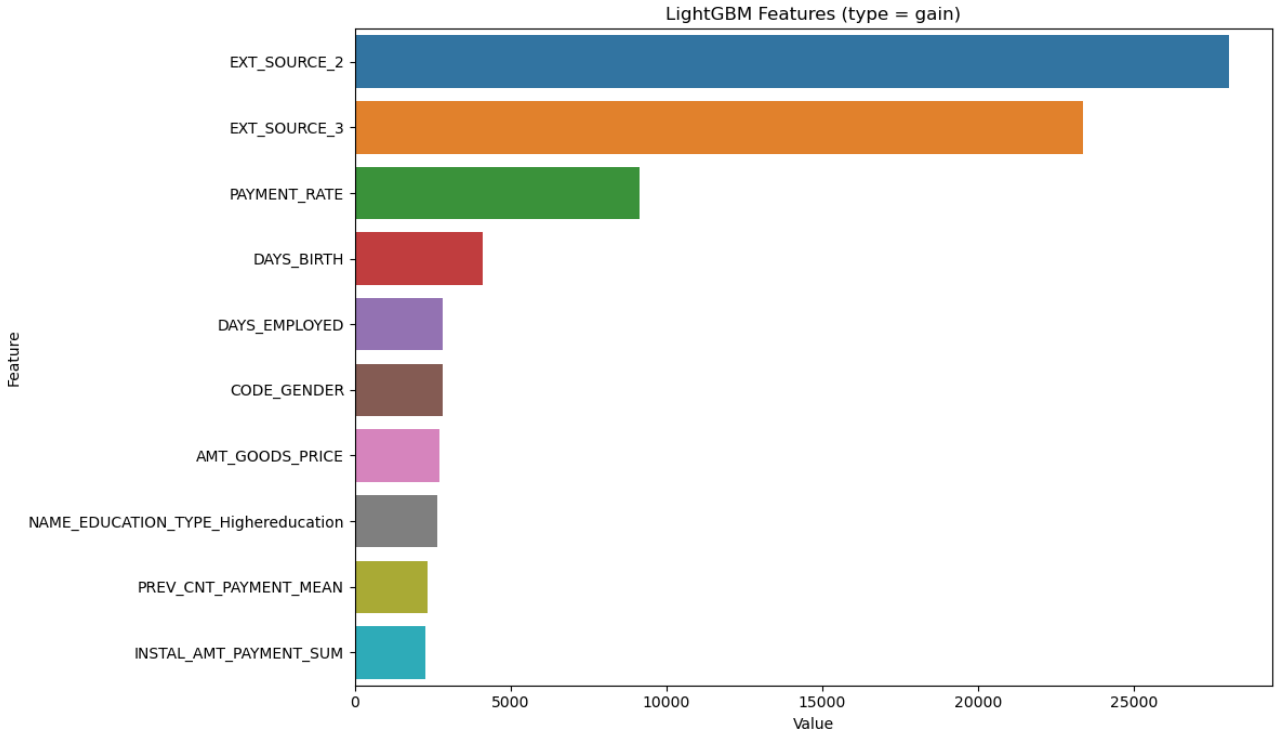
	Modele	Accuracy	AUC	Recall class 1	F1	fbeta	TP	Precision	FN	score Gain	train_time	predict_time
0	Model_LGBMClassifier	0.699815	0.765181	0.696073	0.272405	0.429114	3456	0.169337	1509	0.698336	6.136364	1.406532
1	Model_LGBMClassifier	0.699815	0.765181	0.696073	0.272405	0.429114	3456	0.169337	1509	0.698336	6.233220	1.624582
2	Model_SVC	0.688986	0.752176	0.689829	0.263685	0.418981	3425	0.162994	1540	0.689319	2533.136927	300.204419
3	Model_LogisticRegression	0.691717	0.751976	0.681974	0.263174	0.416718	3386	0.163047	1579	0.687868	7.234787	1.064942
4	Model_XGBClassifier	0.687929	0.745665	0.676737	0.259329	0.411684	3360	0.160397	1605	0.683507	10.105386	1.399076
5	Model_RandomForestClassifier	0.691376	0.740535	0.667472	0.258815	0.409095	3314	0.160531	1651	0.681932	98.769776	3.561240
6	Model_DecisionTreeClassifier	0.586290	0.588409	0.590937	0.187404	0.317484	2934	0.111360	2031	0.588126	14.551581	0.949493

4 Résultats du modèle

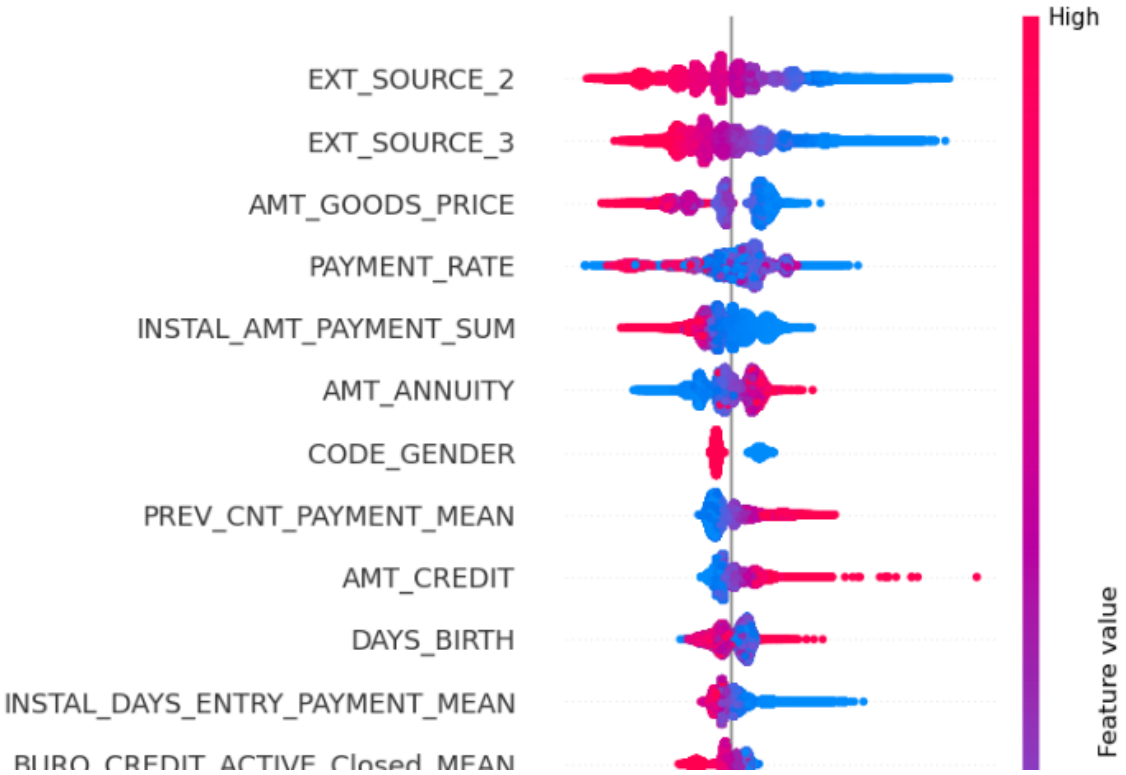
4.1 Interpretabilité globale

Le modèle **LightGBM optimisé final** contient une méthode permettant pour chaque variable de calculer l'importance de cette variable pour le modèle. Une fois normalisées, nous pouvons comparer l'importance relative de chacune des variables et par un simple tri, afficher les 10 premières variables les plus importantes.

Avec cette représentation, on peut dire que les features les plus importantes pour la prédiction d'accord d'un prêt sont les sources extérieures 2 et 3 qui sont des scores normalisés créés à partir de sources de données externes.



Pour plus de précision et pour connaître le sens d'influence de chacune des variables, il est possible d'utiliser la librairie SHAP (SHapley Additive exPlanations) qui explique la sortie de tout modèle d'apprentissage automatique en utilisant la théorie des jeux.



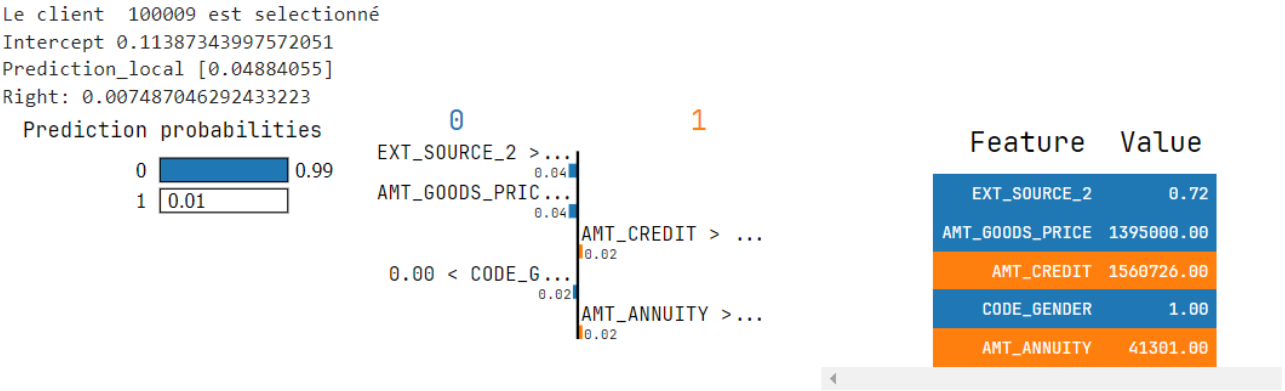
Pour **EXT_SOURCE_2**, **EXT_SOURCE_3** on peut voir que de faibles valeurs augmentent de manière significative la sortie de probabilités du modèle et donc le fait d'être non solvable et le non

accord du prêt. Nous observons le contraire pour **PREV_CNT_PAYMENT_MEAN**.

4.2 Interpretabilité locale : LIME

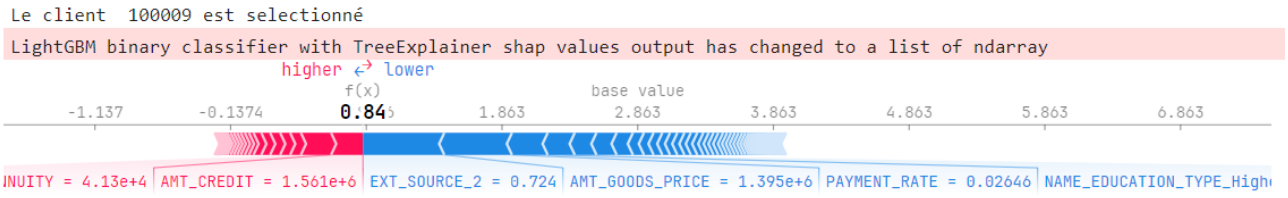
LIME est une technique qui permet de créer un modèle simple autour de la prédiction que nous voulons expliquer et utilise ce modèle simple pour donner une explication. LIME peut être utilisé avec n'importe quel modèle de machine learning, pas seulement avec des arbres de décision ou des forêts aléatoires.

L'idée de base de LIME est assez simple. Pour une prédiction donnée, LIME perturbe les entrées de la prédiction et essaie de comprendre comment ces perturbations affectent la sortie du modèle. Les attributs qui affectent le plus la sortie lorsqu'ils sont perturbés sont considérés comme ayant une importance élevée pour cette prédiction spécifique.



4.3 Shapley Values

La valeur de Shapley, dans le contexte de l'interprétation de modèles d'apprentissage automatique, permet d'évaluer l'impact de chaque caractéristique sur une prédiction donnée. Elle fait cela en calculant la contribution moyenne de chaque caractéristique à la prédiction sur toutes les combinaisons possibles de caractéristiques.



5 Limitations & Améliorations

La partie de prétraitement et de création des variables a été réalisée de façon superficielle en se basant sur le **kernel Kaggle** fourni dans les ressources. Un travail plus approfondi, en collaboration avec les équipes métier, pourrait éventuellement améliorer les résultats de prédiction.

Afin d'alléger le travail de modelisation, nous avons procedé à une selection de variables. Nous avons retenu **100 Variables (de façon arbitraire)**. Une collaboration prealable avec les équipes métier aurait permis de faire un choix beaucoup plus optimal.

En ce qui concerne les **coefficients attribués aux FN et FP**, une hypothèse forte a été de considerer le coût d'un faux négatif comme **10 fois supérieur** à celui d'un faux positif. Cette hypothèse nécessite, également, une meilleure exploration en collaboration avec les équipes métier.