NOTE METHODOLOGIQUE

METHODOLOGIE D'ENTRAINEMENT DU MODELE

DONNEES UTILISEES

Pour le modèle final, toutes les données disponibles dans le dataset Kaggle ont été employées (219 variables), de même que toutes les demandes de prêt.

Un pré-traitement a été effectué en s'inspirant du kernel de jsaguiar :
https://www.kaggle.com/jsaguiar/lightgbm-with-simple-features
Ce kernel ajoute 215 variables issues d'un feature engineering (transformations des principales features métier), pour un total de 434 variables. Il remplace de nombreuses valeurs aberrantes du dataset par des NaN.

ENCODAGE DES DONNEES CATEGORIELLES

Dans l'optique de travailler avec LGBM (Light Gradient Boosting Machine), nous avons réalisé un encodage différent de celui du kernel de jsaguiar (one-hot encoding).
Nous avons privilégié un encodage ordinal des données catégorielles car LGBM permet d'utiliser cet encodeur (le one-hot encoding quant à lui double le nombre total de variables). Ceci a permis un gain de temps et de RAM durant l'entrainement, sans impacter les prédictions du modèle.

TESTS DE REEQUILIBRAGE DE LA CLASSE CIBLE

Différentes approches ont été testées pour rééquilibrer la classe cible TARGET en utilisant la métrique AUC-ROC :

Méthode de rééquilibrage	Variation AUC-ROC	Commentaire
Aucune (référence)	**0 (réf)**	**-**
Random over-sampling	+0.001	Temps de calcul en hausse
Random under-sampling	-0.005	Temps de calcul en baisse
SMOTE	-0.070	Imputation des NaN obligatoire fait chuter le score
Paramètre LGBM : is_unbalance=True	+0.002	Doc LGBM : "while enabling this should increase the overall performance metric of your model, it will also result in poor estimates of the individual class probabilities"
Paramètre LGBM : scale_pos_weight=3 (varié de 1 à 10 (optimum pour 3))	+0.002	

Tests réalisés comparativement avec LGBM (paramètres par défaut). Les data provenaient uniquement des datasets "application_[train|test].csv" de Kaggle. Scoring effectué par une soumission du dataset « test » à Kaggle (pour garantir l'absence de data leak).

Les différentes méthodes testées n'apportant pas d'amélioration significative au score AUC-ROC, nous avons poursuivi l'entrainement du modèle sans rééquilibrer les données.

ENTRAINEMENT DE LGBM

Les données utilisées ont été séparées entre jeux d'entrainement et de validation avec un ratio 9 / 1.
4 hyperparamètres fondamentaux de LGBM ont été variés par une approche "grid search" : num_leaves, num_iterations, min_data_in_leaf et learning_rate.
L'optimisation a été réalisée sur 5 plis stratifiés avec une validation croisée.

FONCTION COUT METIER, METRIQUE D'EVALUATION ET ALGORITHME D'OPTIMISATION

FONCTION COUT METIER

Nous avons créé une fonction de coût pour la société "Prêt à dépenser". Les hypothèses retenues pour les candidats au crédit sont les suivantes :

Objet	Coût par client	Classe
octroi de crédit à un client qui fait défaut	100	Faux négatif (FN)
octroi de crédit à un client qui ne fait pas défaut	-10	Vrai négatif (TN)
refus de crédit à un client qui aurait fait défaut	0	Vrai positif (TP)
refus de crédit à un client qui n'aurait pas fait défaut	0	Faux positif (FP)
frais généraux pour chaque client	1	-

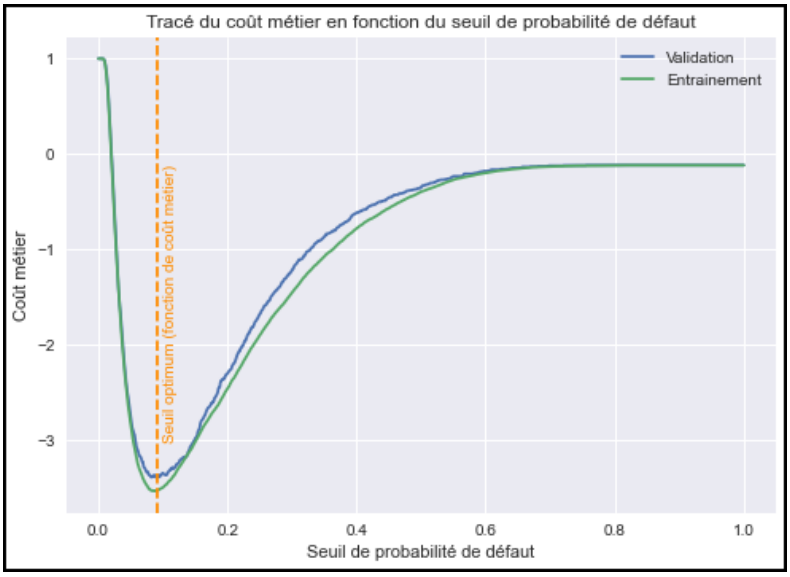
D'où la fonction de coût (rapportée à un client) :

$$Coût = \frac{100*FN-10*TN+1*(TP+TN+FP+FN)}{TP+TN+FP+FN}$$

METRIQUE D'EVALUATION

La métrique d'évaluation globale de notre modèle est la fonction de coût métier. L'optimisation a donc consisté en l'optimisation parallèle des hyperparamètres de LGBM et de l'hyperparamètre "Seuil optimum de la probabilité de défaut".

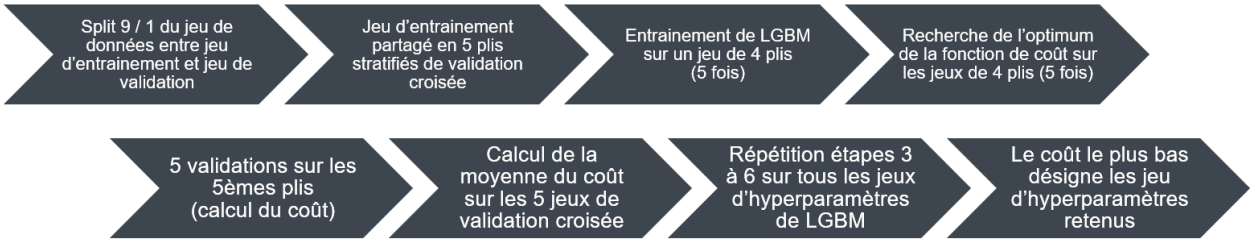
Le graphique suivant représente la fonction de coût obtenue pour différents seuils de probabilité de défaut sur les jeux d'entrainement et de validation. La ligne verticale en pointillés oranges représente l'optimum de ce seuil, càd la minimisation de la fonction de coût métier.



ALGORITHME D'OPTIMISATION

Un scorer contenant la fonction coût métier a été mis en place. Les étapes suivantes ont été répétées pour chaque pli et chaque jeu d'hyperparamètres de LGBM :
1) entrainement d'un modèle LGBM (métrique locale : fonction de perte logloss).
2) test des seuils de probabilité de défaut par pas de 0.001 sur le modèle LGBM entrainé à l'étape 1 → obtention d'une matrice de confusion pour chaque seuil.
3) recherche de la matrice de confusion (parmi celles obtenues à l'étape 2) qui minise la fonction de coût.

SCHEMA SYNOPTIQUE DU PROCESSUS D'ENTRAINEMENT

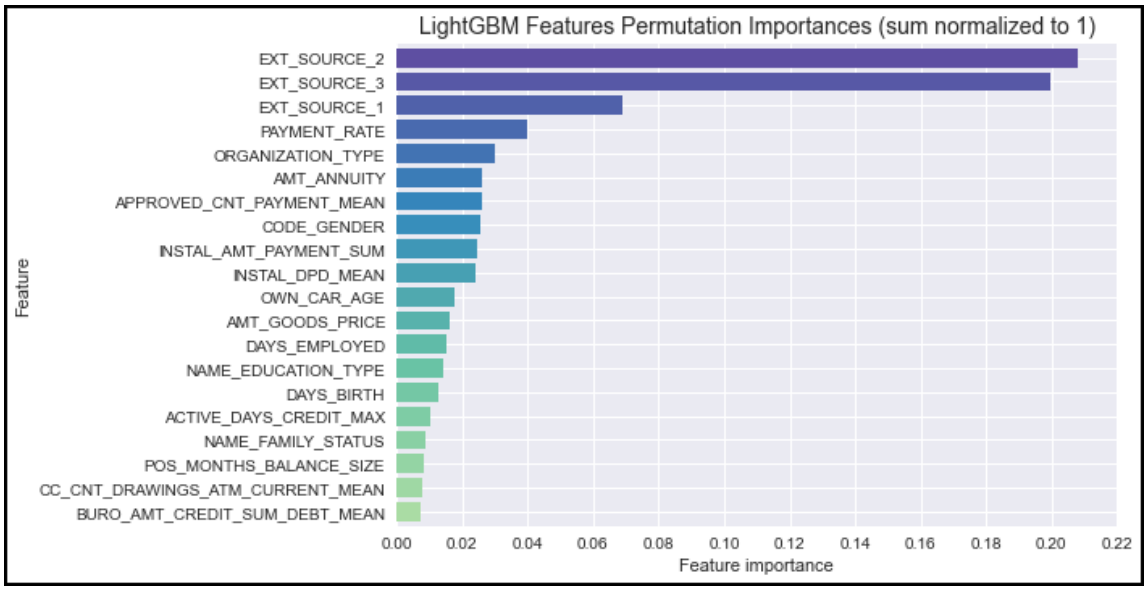


INTERPRETABILITE GLOBALE ET LOCALE DU MODELE

Nous avons retenu des interprétations qui permettent une lecture aisée pour les chargés de relation clientèle.

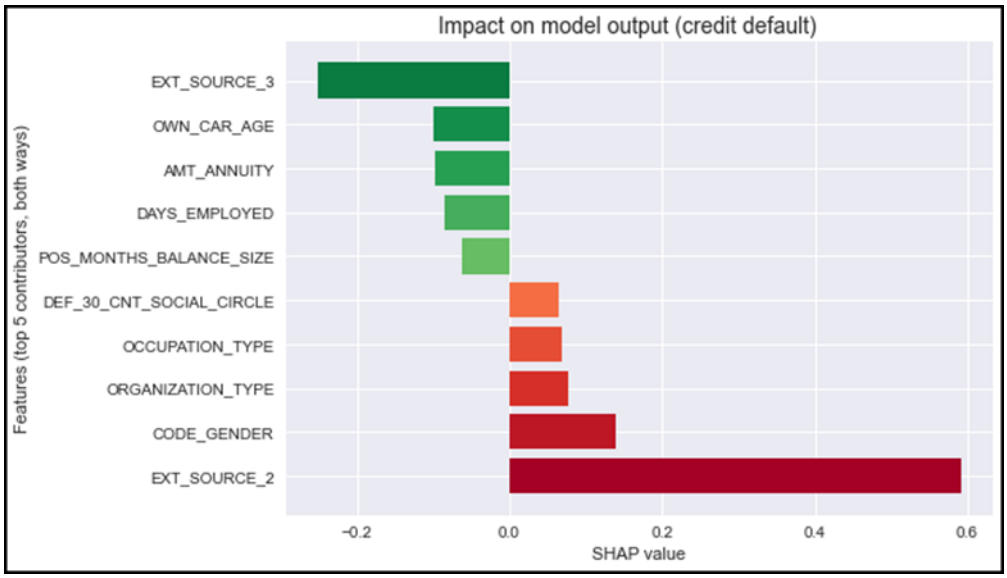
FEATURE IMPORTANCE GLOBALE PAR PERMUTATION

La technique des permutations calcule les variations de score d'une feature quand toutes les valeurs de cette feature sont mélangées. Plus cette variation est importante, plus la feature est importante. Pour notre modèle, les features les plus importantes sont regroupées dans le graphique suivant (somme des variations de probabilité normalisée à 1) :



FEATURE IMPORTANCE LOCALE PAR SHAP

La technique SHAP locale explique la prédiction pour chaque client en calculant la contribution de chaque feature à la prédiction. Par exemple pour le client 324806, les contributions des dix features qui impactent le plus sa probabilité de ce client sont les suivantes :



Note : les valeurs SHAP sont des logarithmes d'odds ratios ; elles sont donc additives.

LIMITES ET LES AMELIORATIONS POSSIBLES

Pour accélérer le boosting de LGBM et conduire au meilleur coût optimum, il faudrait connaître le gradient et le hessien de notre fonction de coût métier. Une des limites de l'optimisation de notre modèle est l'emploi d'une fonction de coût non dérivable par rapport aux variables du jeu de données (note : tout comme la quasi totalité des scorers). Pour contourner ce problème, nous avons utilisé la fonction de perte logloss pour la construction des arbres de LGBM.

Un axe d'amélioration est de poursuivre l'optimisation d'hyperparamètres :
- sur un plus grand nombre d'hyperparamètres de LGBM.
- en incluant les techniques de rééquilibrage de la classe cible en tant qu'hyperparamètre.
- en réalisant une optimisation bayesienne (module hyperopt) plutôt qu'un grid search.

LGBM a été préféré comme framework de machine learning en regard des très bonnes performances obtenues dans des problèmes de classifications avec des données tabulaires, tant en terme de précision que de vitesse d'exécution. Un autre axe d'amélioration serait de tester comparativement des frameworks tels que XGBoost ou CatBoost sur notre jeu de données.