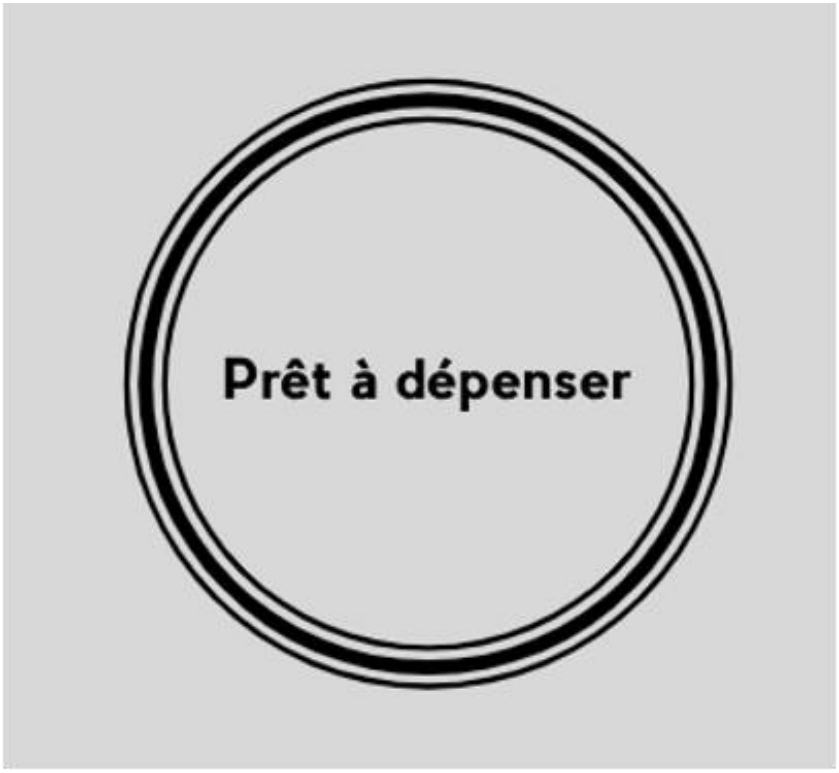


NOTE METHODOLOGIQUE :

IMPLEMENTATION D'UN MODELE DE SCORING

Haitem EL AAOUANI

Table des matières

1- Problématique :

Cette analyse a été conduite dans le cadre d'un projet dédié à l'évaluation du « Score de crédit ». L'objectif principal de cette initiative est de créer un modèle de classification qui permet de prédire la solvabilité des clients, et ce en passant par le calcul de la probabilité de leurs défauts de paiement. Parallèlement, le projet vise à développer un tableau de bord interactif pour expliciter de manière transparente les décisions relatives à l'attribution ou rejet des demandes de crédit.

Cette note méthodologique détaille les processus de modélisation et met en lumière les aspects associés à l'interprétabillité du modèle élaboré.

2- Exploration des données et features engineering :

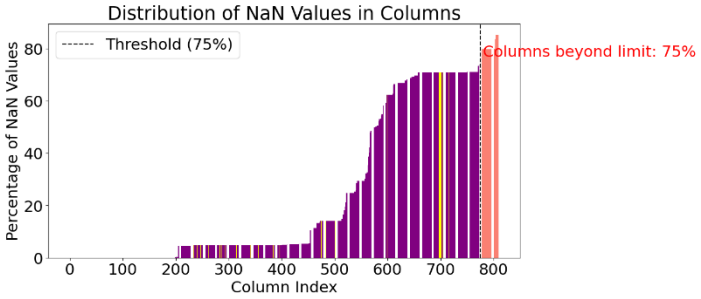
a. Exploration des données

Le dataset est constitué de 9 fichiers au format « csv », qui contiennent des données financières de 307 511 clients anonymes. Nous retrouvons dans la table principale "application_train.csv" la colonne 'TARGET', indiquant si le client a remboursé son prêt (0) ou a été défaillant (1). Cette donnée sera donc la variable cible servant à entraîner un modèle de classification.

L'analyse de la distribution de la variable cible révèle un déséquilibre de classes; en effet, 8 % des prêts n'ont pas été honorés. Ce déséquilibre devrait être pris en compte lors de la modélisation afin d'éviter que la classe minoritaire ne soit « écrasée » par le modèle de classification.

b. Features engineering :

Cette étape a été brillamment menée par J. Aguiar qui a mis à disposition du public le kernel Kaggle LightGBM with Simple Features que nous avons adapté aux spécificités techniques du projet. Ainsi, nous avons obtenus un dataset combinés de 810 variables.



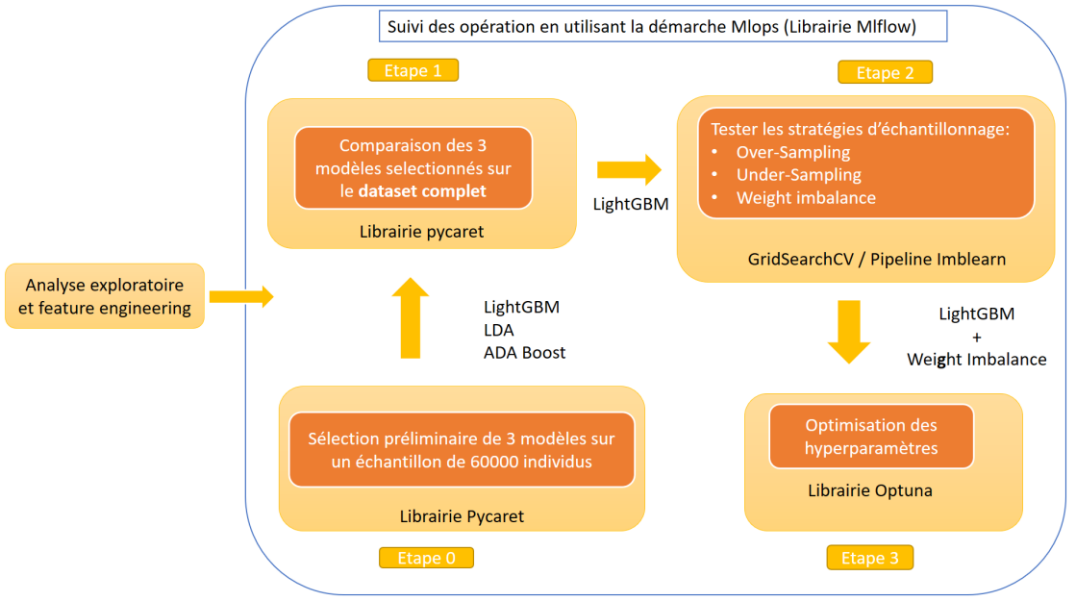
En complément, nous avons éliminé les variables ayant un taux de valeurs aberrantes supérieur à 75% et celles qui ont des valeurs uniques. Les valeurs manquantes (NAN) des variables numériques ont été imputées par leurs médianes et les valeurs infinies par la +/- max(abs(des valeurs)). Concernant les variables catégorielles l'imputation s'est faite en remplaçant les valeurs manquantes par les valeurs les plus fréquentes.

Enfin, nous avons bien pris en considération l'encodage des variables catégorielles et la standardisation des variables numériques.

Ainsi, nous avons récupéré un dataset d'entrainement nettoyé de 307511 individus et 775 variables.

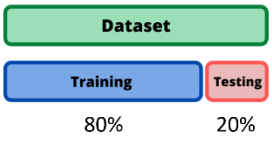
A noter que le dataset de test a été traité de la même manière.

3- Méthodologie d'entraînement du modèle de prédiction :



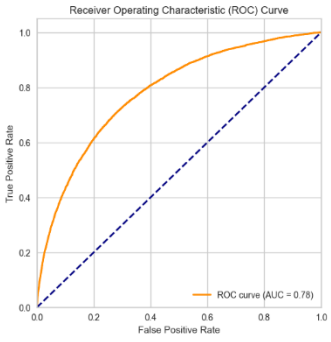
a. Train-Test Split et Preprocessing:

La répartition en du dataset s'est faite en 80% pour la phase d'entrainement et 20% pour le test. Pour ce faire, nous avons utilisé la fonctionnalité train_test_split() de Scikit-Learn.



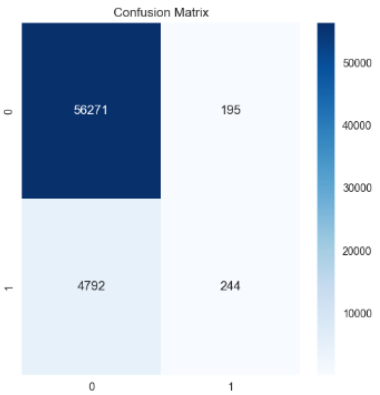
b. Métriques pertinentes et fonction coût :

AUC-ROC : L'aire sous la courbe ROC est une mesure synthétique de la performance d'un modèle de classification binaire. Elle évalue la capacité du modèle à discriminer entre les classes, en représentant graphiquement le taux de vrais positifs par rapport au taux de faux positifs. Une AUC-ROC élevée (proche de 1) indique une meilleure performance du modèle.



La matrice de confusion qui évalue les performances d'un modèle de classification en résumant les prédictions en quatre catégories :

TP (True Positives) : Correctement prédits comme positifs.
TN (True Negatives) : Correctement prédits comme négatifs.
FP (False Positives) : Incorrectement prédits comme positifs.
FN (False Negatives) : Incorrectement prédits comme négatifs.

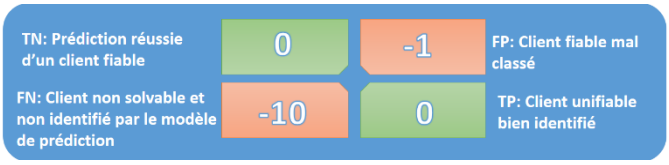


Recall : Proportion de vrais positifs identifiés correctement $\frac{TP}{TP+FN}$

Precision : Proportion de prédiction positives étant correcte $\frac{TP}{TP+FP}$

F1 score : Combinaison de la Precision et du Recall $2\frac{Precision \times Recall}{Precision+Recall}$

➢ **Le coût métier :** Il est pertinent de prendre en considération le coût métier entre un faux négatif (FN) et un faux positif (FP). En effet, un client non solvable prédit dans la classe négative peut engendrer des conséquences financières importantes sur la performance de l'entreprise. De même, un client solvable à qui on refuse la demande de crédit constitue un manque à gagner.



De ce fait, la fonction coût métier a été modélisée comme ainsi :

$$Coût\ métier = \frac{10FN+FP}{LEN(Dataset\ d'entraînement)}$$

c. Choix de 3 modèles candidats (Bibliothèque Pycaret)

Le choix du modèle de classification s'est déroulé en plusieurs étapes. Nous avons d'abord sélectionné 4 modèles de classification dont nous avons comparé la peformance. Cette sélection primaire s'est faite avec la fonction **compare_models** de la bibliothèque **Pycaret**. La métrique de comparaison est le score AUC 'que nous définissons est défini

Par soucis d'optimisation du temps de calcul, nous avons procédé en 2 phases :

➢ **Phase 1** : Nous avons sélectionné aléatoirement un échantillon de 60000 clients. Ensuite, avec la fonction **compare_models** de la bibliothèque **Pycaret**, nous avons récupéré les 3 meilleurs modèles de prédiction qui fournissent les meilleurs scores AUC et ayant montré un temps de calcul raisonnable. Cette démarche a permis donc de présélectionner les 3 modèles suivants : LightGBM (Light Gradient-Boosting Machine), LDA (Linear Disciminant Analysis) et ADA (Ada Boost Classifier). En plus nous avons retenu DummyClassifier pour servir de baseline.

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
gbc	Gradient Boosting Classifier	0.9198	0.7697	0.0317	0.5209	0.0596	0.0508	0.1128	426.751
lightgbm	Light Gradient Boosting Machine	0.9192	0.7645	0.0415	0.4683	0.0760	0.0639	0.1204	23.345
lda	Linear Discriminant Analysis	0.9156	0.7516	0.0832	0.3868	0.1368	0.1114	0.1492	14.641
ada	Ada Boost Classifier	0.9188	0.7514	0.0498	0.4588	0.0896	0.0750	0.1301	87.503
rf	Random Forest Classifier	0.9196	0.7055	0.0003	0.1000	0.0006	0.0005	0.0052	123.017
et	Extra Trees Classifier	0.9196	0.7005	0.0003	0.1000	0.0006	0.0004	0.0043	58.341
nb	Naive Bayes	0.8302	0.6778	0.2882	0.1829	0.2110	0.1282	0.1366	3.592
dt	Decision Tree Classifier	0.8525	0.5354	0.1576	0.1369	0.1465	0.0662	0.0664	56.784
qda	Quadratic Discriminant Analysis	0.2878	0.5119	0.7787	0.0828	0.1490	0.0051	0.0156	8.178
knn	K Neighbors Classifier	0.9163	0.5017	0.0071	0.1313	0.0134	0.0050	0.0121	11.510
dummy	Dummy Classifier	0.9196	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	2.937
lr	Logistic Regression	0.9196	0.4917	0.0000	0.0000	0.0000	0.0000	0.0000	3.916
ridge	Ridge Classifier	0.9196	0.0000	0.0036	0.4500	0.0070	0.0059	0.0337	3.866
svm	SVM - Linear Kernel	0.6567	0.0000	0.3139	0.0325	0.0566	0.0005	0.0002	27.206

➢ **Phase 2** : Après avoir sélectionné 3 modèles candidats, nous avons fait une 2ème itération mais cette fois-ci sur l'ensemble des individus du dataset.

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lightgbm	Light Gradient Boosting Machine	0.9197	0.7793	0.0384	0.5414	0.0717	0.0616	0.1282	66.517
lda	Linear Discriminant Analysis	0.9175	0.7699	0.0624	0.4236	0.1087	0.0898	0.1380	47.333
ada	Ada Boost Classifier	0.9189	0.7613	0.0323	0.4666	0.0604	0.0505	0.1060	323.691
dummy	Dummy Classifier	0.9193	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	7.257

Cette 2ième itération a montré que le modèle **LightGBM** pour la suite de l'étude.

d. Problématiques de déséquilibre et fuite de données

Après avoir retenu LightGBM comme baseline, et avant de procéder à l'optimisation des hyperparamètres, il était important de voir de plus près la problématique de **déséquilibre des données.** En effet, l'analyse exploratoire a mis en évidence que les crédits impayés (classe positive) sont uniquement de 8.1%. Ce fort déséquilibre influencera la pertinence des prédictions, i.e le modèle risque de ne prédire que la classe négative au détriment de la classe positive. Afin de répondre à cette problématique, il a fallu avoir appliquer des techniques d'échantillonnage et d'équilibre des poids.

Techniquement parlant, nous avons utilisé le pipeline de la bibliothèque **imbalanced -learn** qui permet d'éviter tout problème de **fuite de données (data leakage).**

- **Sous-échantillonnage (Undersampling)** en utilisant **RandomUnderSampler** de la bibliothèque **imblearn** est une méthode rapide et facile pour équilibrer les données en sélectionnant de manière aléatoire un sous-ensemble de données pour les classes ciblées.
- **Sur-échantillonnage (Oversampling)** en utilisant la méthode SMOTE (**S**ynthetic **M**inority **O**versampling **T**echnique) : Cette approche génère des données de manière synthétique en combinant un algorithme des k plus proches voisins avec l'ajout de bruit de fond (noise). Là encore, nous avons utilisé la fonctionnalité **SMOTE** de bibliothèque **imblearn**.
- **Equilibrage par poids (Weight imbalance ratio)** : Cette dernière **scale_pos_weight** qui est un hyper-paramètre du modèle LightGBM permettant de rééquilibrer le jeu de donnée d'entrainement. Nous avons commencé dans un premier temps par l'appliquer comme ainsi (cf ce tutorial):

$$scale\_pos\_weight = \frac{N\ individus\ de\ la\ classe\ négative}{N\ individus\ de\ la\ classe\ positive}$$

La comparaison des matrices de confusion issues des 3 approches ainsi que la métrique AUC ont favorisé la solution avec équilibrage par poids (Weight imbalance ratio). Optimisation du modèle baseline (LightGBM)

Method	Accuracy	Precision	Recall	F1_Score	AUC	Business_Cost
SMOTE	0.919645	0.534722	0.031036	0.058667	0.766572	0.520308
RandomUnderSampler	0.705408	0.173127	0.702136	0.277764	0.774056	0.508683
ImbalanceRatioAdjustment	0.733895	0.185007	0.674929	0.290409	0.780745	0.50000

e. Procédure d'optimisation :

L'optimisation du modèle de prédiction a été accomplie sur la base de l'indicateur métier. Un modèle idéal permettra de retrouver un coût métier = 0 (séparation parfaite des classes TP et TN). De ce fait, nous avons sélectionné les hyperparamètres sur la base de minimisation du coût métier.

Techniquement, nous avons utilisé la bibliothèque Optuna qui permet d'offre une gestion efficace du choix des hyperparamètres et la possibilité d'annuler des essais non concluants grâce à l'algorithme Tree-structured Parzen Estimator (TPEsampler), basé sur une approche bayésienne.

Nous avons procédé comme suivant :

- **Préparation des données** : Les données sont séparées en ensembles d'entraînement et de test tout en maintenant la stratification par classe pour gérer le déséquilibre.

- **Variation du seuil de prédiction :** Nous avons également fait varier le seuil de prédiction lié au modèle. En effet, les modèles de prédiction fournissement la probabilité de solvabilité d'un prêt octroyé (predict_proba). Cette probabilité est basée sur un seuil de solvabilité à 0.5 qui n'est pas forcément le seuil optimal. Il était donc important de chercher à minimiser le coût métier en variant le seuil de prédiction.

```python
def evaluate_and_plot_best_model(best_model, X_val, y_val, plot_file_path='Best_plot_otpuna.png'):
    # Predict on the validation set using the best model
    y_val_pred_prob = best_model.predict_proba(X_val)[:, 1]
    y_val_pred = best_model.predict(X_val)

    # Choose the threshold that minimizes the business cost
    thresholds = np.linspace(0, 1, 100)
    business_costs = [business_cost_scorer(y_val, y_val_pred_prob, threshold) for threshold in thresholds]
    best_threshold = thresholds[np.argmin(business_costs)]

    # Calculate metrics
    fpr, tpr, _ = roc_curve(y_val, y_val_pred_prob)
    auc_score = auc(fpr, tpr)

    accuracy = accuracy_score(y_val, (y_val_pred_prob >= best_threshold).astype(int))
    precision = precision_score(y_val, (y_val_pred_prob >= best_threshold).astype(int))
    recall = recall_score(y_val, (y_val_pred_prob >= best_threshold).astype(int))
    f1 = f1_score(y_val, (y_val_pred_prob >= best_threshold).astype(int))
    business_cost = business_cost_scorer(y_val, y_val_pred_prob, best_threshold)
```

Variation du seuil de probabilité

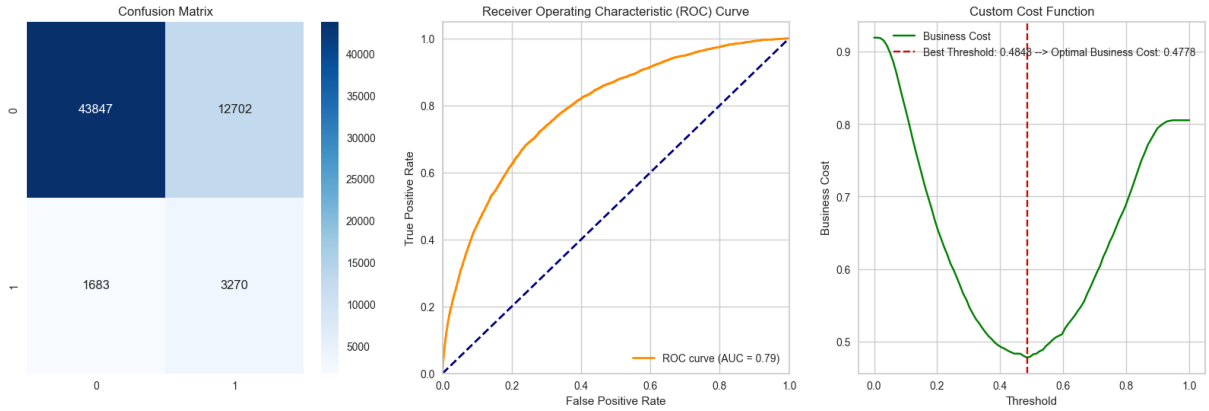
Calcul des métriques

- **Création du modèle** : Un modèle est créé avec des paramètres de base adaptés aux données déséquilibrées, ajusté pour prédire une classe selon un seuil de décision personnalisé (0,4).

```python
# -- Setting up mlflow callback
mlflc = MLflowCallback(create_experiment=True,
                       metric_name="Profit", mlflow_kwargs={"nested": True})

# -- Define the objective function
@mlflc.track_in_mlflow()
def objective(trial):
    # Define imbalance ratio
    imbalance_ratio = y_train.value_counts()[0] / y_train.value_counts()[1]

    # Params grid
    lgbm_params = {'learning_rate': trial.suggest_loguniform('learning_rate', 0.001, 0.02),
                   'n_estimators': trial.suggest_int('n_estimators', 100, 4000),
                   'scale_pos_weight': trial.suggest_float('scale_pos_weight', imbalance_ratio - 1, imbalance_ratio + 1),
                   'min_child_samples':  trial.suggest_int('min_child_samples', 2, 256),
                   'reg_alpha':  trial.suggest_loguniform('reg_alpha', 1e-8, 1e-3),
                   'reg_lambda':  trial.suggest_loguniform('reg_lambda', 1e-8, 1e-3)
                   }

    # Set up LightGBM classifier with hyperparameters to tune
    lgbm_classifier = LGBMClassifier(**lgbm_params)

    # Create a pipeline
    pipeline = Pipeline([('clf', lgbm_classifier)])

    # Fit the model on the data
    pipeline.fit(X_train, y_train)
```

- **Optimisation par Optuna** : À chaque exécution, un ensemble d'hyperparamètres est sélectionné. Le modèle est ensuite entraîné par validation croisée (5 folds) en minimisant la fonction coût métier. Les expérimentations ont été tracé avec l'approche MLOps (bibliothère MLFlow).

f. Synthèse des résultats de l'optimisation



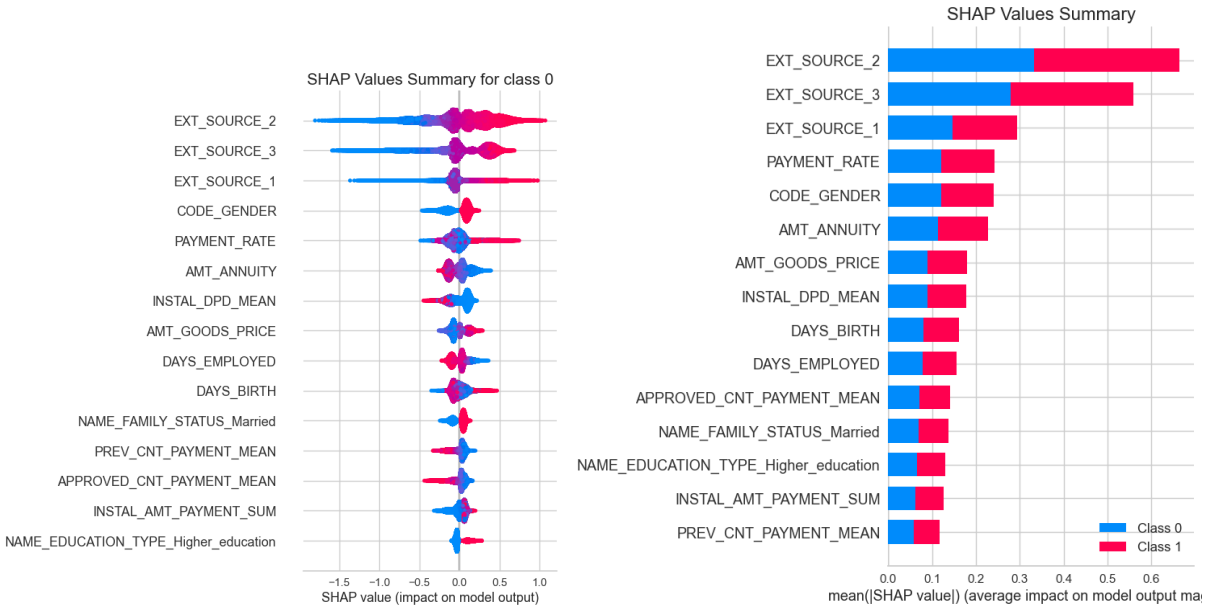
4- Importance des variables et leurs Interprétabilités globale et locale.

a. Importance globale et locale des variables

Afin d'assurer la transparence et la justification des décisions d'acceptation ou rejet des demandes de crédits, il est crucial de pouvoir interpréter les prédictions du modèle. Alors que les modèles linéaires offrent une compréhension statistique des coefficients de chaque variable sur la prédiction, certains modèles basés sur un algorithme de gradient boosting tels que le LightGBM, ne fournissent pas assez d'éléments pour faciliter le travail d'interprétabilité.

Pour évaluer l'importance des caractéristiques, la méthode SHAP (SHapley Additive exPlanations) a été employée. Les valeurs SHAP permettent de quantifier l'influence de chaque variable au niveau local (la contribution d'une variable sur la prédiction finale). Par conséquent, la moyenne des valeurs SHAP de tous les individus permet de calculer l'effet de chaque variable sur la réponse au niveau global.

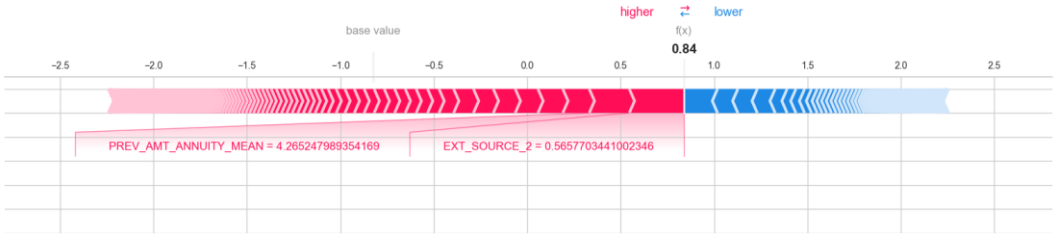
i. Importance globale



La distribution des valeurs des variables a été visualisée pour comprendre l'impact sur la prédiction.

ii. Importance locale

À l'échelle locale, les valeurs SHAP peuvent être calculées individuellement pour chacun des individus, ce qui permet de quantifier l'impact de chaque variable sur la prédiction finale. Les valeurs négatives favorisent l'inclusion dans la classe 1 (difficultés de remboursement), tandis que les valeurs positives favorisent l'inclusion dans la classe 0 (bonne solvabilité).

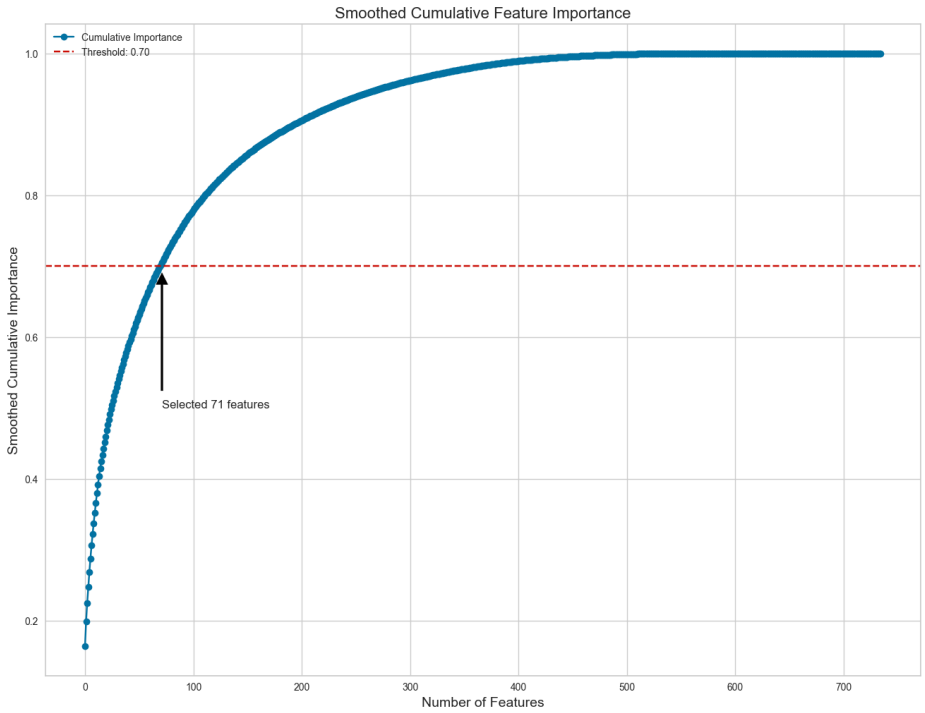


b. Importance cumulée des variables

L'analyse de l'importance des variables a montré qu'un degré d'importance élevé peut être maintenu tout en se limitant à un nombre de variables réduits. Afin de faciliter le travail d'analyse de l'équipe financière, nous pourrons donc envisager de définir une liste de variables que l'on pourra inclure dans la solution finale qui sera intégrée au dashboard.

Pour ce faire, nous avons représenté l'importance cumulée en fonction du nombre des variables.

Nous pourrons donc pour les raisons de simplification citées ci-dessus de définir un seuil d'importance cumulée à 70% ce qui revient à sélectionner les 71 variables.



Nous avons ensuite comparé les performances de ce nouveau modèle (71 variables) avec le modèle d'origine (741 variables). Le tableau ci-dessous montre que les scores sont sensiblement similaires. Ce qui justifie la possibilité de réduire le nombre de variables de 90% tout en restant sur un niveau de scores similaires.

Métrique	Résultats Initiaux (741 variabales)	Modèle Réduit (71 variables)
AUC	0.790	0.790
Score métier	0.486	0.495
Seuil de prédiction	0.486	0.480

5- Limitations et perspectives d'améliorations

Lors de notre analyse, nous avons relevé certaines limitations et pistes d'amélioration qui nécessiteraient d'être soulevées :

- Bien que l'élaboration du dashboard s'inscrit dans une démarche de transparence vis-à-vis de la clientèle, nous avons constaté que certaines variables ne sont pas assez documentées ! Exemple : EXT_SCOURCE_1, EXT_SCOURCE_2, EXT_SCOURCE_3.
- Une interaction avec les conseillers financiers pourrait être utiles pour prédéfinir les variables représentatives du point de vue métier. Inversement, savoir si certaines variables sont bien indispensables pour l'analyse financière.
- Du côté de la rentabilité financière, notre étude a démontré que les demandes de crédits de certains clients solvables pourraient être refusées ! D'autres non solvables, bien que minoritaires, peuvent obtenir un crédit. Par conséquence, cette étude purement prédictive devrait être complétée par une démarche d'analyse des risques.

Du point de vue technique nous pourrons également envisager les améliorations suivantes :

- Nous pouvons toujours améliorer les performances de la validation croisée des hyperparamètres en augmentant le nombre d'itérations, de folds et inclure d'autres paramètres qui ont été omis.
- Les poids affectés aux FP et FN pour le calcul de la fonction coût sont des estimations qui pourraient être soumis à la critique. On peut également envisager d'autres approches telles que les métriques F1 et F2 pondérées.
- Le choix de l'algorithme LightGBM est également discutable ! L'étude benchmarck a démontré que Gradient Boosting classifier (GBC) fournit un score AUC meilleur. Il n'a pas été retenu à cause de l'importance du temps de traitement.

6- Analyse de la dérive de données

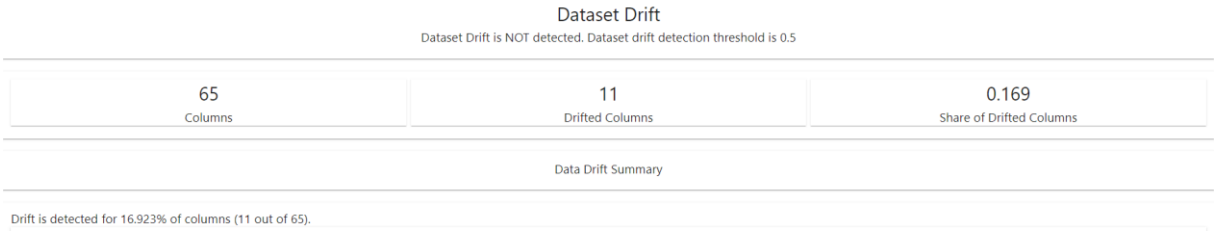
La dérive des données se produit lorsque les distributions des variables changent en raison de l'ajout ou de la modification des individus. Cependant, le modèle a été entrainé sur ces variables à un moment donné, qui peut ne plus refléter la réalité dans le futur, entraînant ainsi une détérioration des performances du modèle au fil du temps.

Pour évaluer la dérive des données, on peut comparer les distributions des variables entre un jeu de données de référence et un jeu de données cible contenant de nouvelles données en utilisant la bibliothèque Evidently. Nous pourrons ainsi identifier la dérive des données au niveau de chaque variable et au niveau globale. Le jeu de données de référence est celui utilisé pour former le modèle,

tandis que le jeu de données cible, tel que application_test.csv, représente des données dont la probabilité de remboursement des clients est inconnue.

a. Analyse de la dérive globale du jeu de données.

La bibliothèque Evidently identifie une dérive de données à l'échelle globale uniquement si plus de 50% des variables (ou un tiers des plus importantes) montrent une dérive. Cependant, dans nos données, cela ne se produit pas, indiquant l'absence de dérive de données globale.



b. Analyse de la dérive des variables

L'analyse Evidently permet de détecter la dérive de données sur 11 variables. En effet, les distributions des variables changent de manière significative entre les datasets d'entraînement et de test

Nous devrons être particulièrement vigilants vis-à-vis des variables PAYMENT_RATE, AMT_REQ_CREDIT_BUREAU_QRT, AMT_GOODS_PRICE et AMT_CREDIT, car elles présentent une dérive et sont significatives pour le modèle.

Il est recommandé de mettre en place un système de monitoring qui permet de s'assurer que l'évaluation du modèle fournira des métriques proches de celles du dataset d'origine. Nous devrions également définir des seuils des métriques à partir desquels il faudra réentrainer et redéployer le modèle.