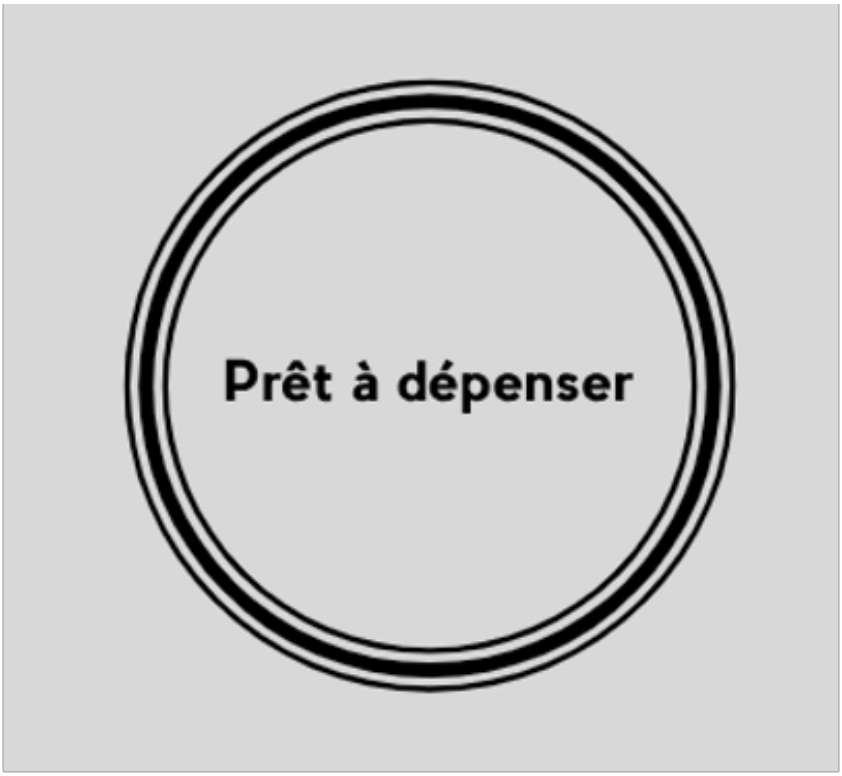
Prêt-a-dépenser

Note méthodologique



I) Méthodologie d'entraînement du modèle

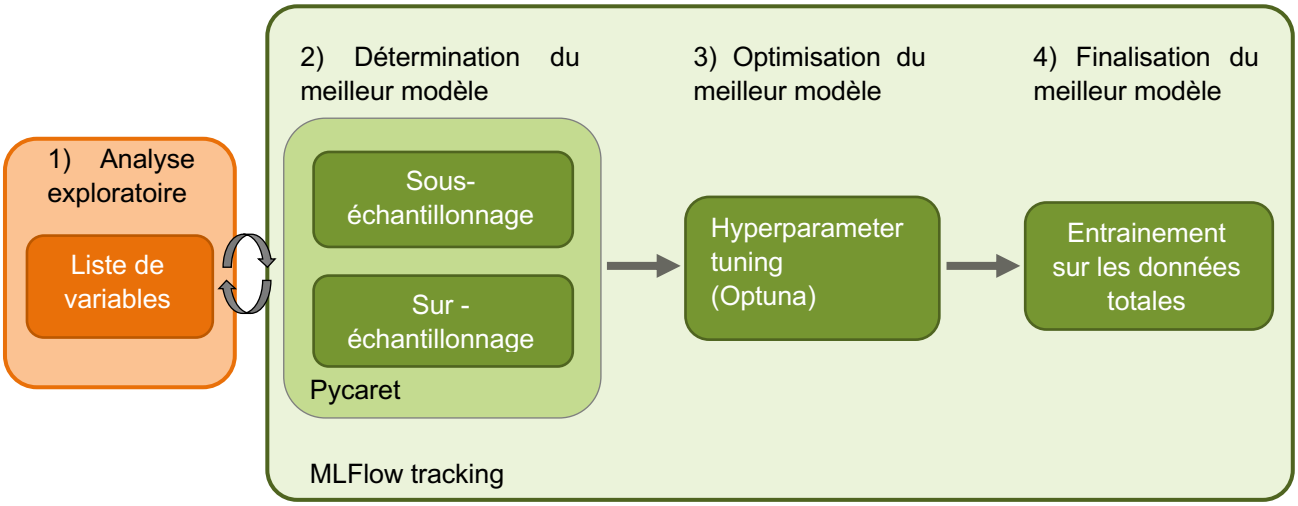


Figure 1: Schéma récapitulatif de la méthodologie de modélisation.

1) Analyse exploratoire

La première étape est de réaliser une analyse exploratoire des données afin de préparer la modélisation. Différents processus cruciaux sont établis durant cette étape : gestion des valeurs nulles, aperçu de la répartition des effectifs entre les classes de notre variable cible (ici '**TARGET'**) (Figure 2), **ingénierie de variables** (création de variables explicatives) et **sélection des variables** les plus impactantes.

La création de variable a été effectuée selon le code Kaggle de J. Aguiar[1] permettant de générer 768 variables explicatives. Afin de lutter contre des problèmes de sur-apprentissage, de gestion du temps et de performance, une liste de variables importantes a été établie.

La sélection de variable a d'abord été effectuée via une classification linéaire SVC comprenant une pénalité L1. Cette pénalité attribue un coefficient à chaque variable en fonction de son effet sur la réponse. Ensuite, toutes les variables ayant ce coefficient différent de 0 sont gardées pour la première étape de la modélisation. Une liste de 172 variables a été retenue par cette méthode.

[1] https://www.kaggle.com/code/jsaguiar/lightgbm-with-simple-features/script

2) Détermination du meilleur modèle

Afin de déterminer le modèle le plus performant sur nos données, de nombreux modèles ont été comparés via la bibliothèque Pycaret[2].

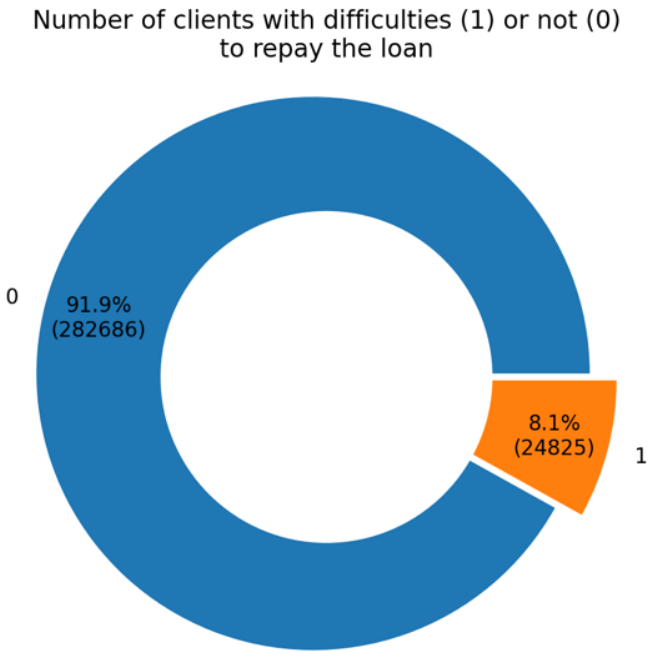


Figure 2 : Nombre de clients avec des difficultés (1) ou non (0) à rembourser le prêt.

Deux approches ont été utilisées afin de pallier aux déséquilibres de la variable cible (Figure 2) : Une approche de sous-échantillonnage et une approche de sur-échantillonnage.

Méthode de sous-échantillonnage : un échantillon du jeu de données comportant autant de clients 0 que de clients 1 a été composé (24825 clients par classe).

Méthode de sur-échantillonnage : la méthode SMOTE[3] est utilisée par défaut sur Pycaret. Cette méthode crée des données de manière synthétique en combinant un algorithme des *k plus proches voisins* plus du bruit de fond (noise).

Dans les deux cas, l'initialisation de Pycaret est la suivante :

- Variable cible = 'TARGET'
- Autres variables = numériques
- Imputation données manquantes = remplacement par 0
- Gestion du déséquilibre des données : Non (sous-ech.) / SMOTE (sur-ech.)
- Validation croisée = 4 fold
- Enregistrement des logs via MLFLOW tracking

Les deux méthodes indiquent que le modèle **LightGBM**[4] est le plus performant et l'un des plus rapide (voir les tableaux partie III). Ce modèle utilise la méthode de *gradient boosting* basée sur des arbres de décisions. Par la suite, la liste des variables les plus importantes a été déterminée par l'importance des variables enregistrée durant l'entrainement des données.

[2] pycaret.org. PyCaret, April 2020. URL https://pycaret.org/about
[3] N. V. Chawla, K. W. Bowyer, L. O.Hall, W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," Journal of artificial intelligence research, 321-357, 2002.
[4] Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, et al. Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems. 2017;30:3146–54.

Au final, les 62 variables affectant le plus la prédiction (60% de la somme cumulée de l'importance des variables) ont été sélectionnées pour la suite de la modélisation et pour le Dashboard.

3) Optimisation du meilleur modèle

L'optimisation des hyperparamètres du modèle permet d'avoir de meilleurs résultats sur nos métriques et un modèle plus performant. L'approche utilisée est l'optimisation par Optuna[5], une bibliothèque python spécialisé dans l'optimisation d'hyperparamètre. Cette bibliothèque permet de gérer efficacement le choix d'hyperparamètres et d'annuler des essais non concluants. L'algorithme de choix d'hyperparamètres dans cette étude est le Tree-structured Parzen Estimator (TPEsampler), basée sur une approche bayésienne.

L'optimisation suit l'algorithme suivant :

- Préparation des données : séparation des données **d'entraînement** et en données de **test**, respectant le déséquilibre de classe (stratifiée en y).
- Création du modèle avec les paramètres de bases (pour données déséquilibrées), modifié pour prédire une classe selon un seuil de décision personnalisée (0,4).
- Optimisation par **Optuna** : A chaque run, un set d'hyperparamètre est sélectionné, le modèle est entraîné par validation croisée (3 folds) puis les métriques sont calculées et enregistrées via MLFlow. La métrique à optimiser est **l'indice de profit**, obtenue par validation croisée (moyenne des 3 folds). Ensuite, le modèle est testé sur les données tests. L'indice de profit, la courbe ROC et la matrice de confusion sur les données tests sont ensuite enregistrées via le tracking MLFlow.

4) Finalisation

La dernière étape de la modélisation est la finalisation du modèle, c'est-à-dire l'entraînement du meilleur modèle, optimisé, sur l'ensemble du jeu de données. Le modèle ainsi entraîné et le graphique d'Importance des variables ont été enregistrés via le tracking MLFlow.

[5] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In KDD (arXiv).

II) Fonction coût métier et Métriques d'évaluation

a

Valeurs vraies		Valeurs prédites: 0	Valeurs prédites: 1
	0	Vrais négatifs (TN)	Faux positifs (FP)
	1	Faux négatifs (FN)	Vrais positifs (TP)

b

Métrique	Description
AUC	Aire sous la courbe ROC construite avec le couple « taux de vrai positif (TPR = Recall) » et « taux de faux positifs (FPR) » à différents seuils de décision. $TPR = \frac{TP}{TP+FN}$ $FPR = \frac{FP}{FP+TN}$
Accuracy	Prédictions correctement identifiées $\frac{TP+TN}{TP+TN+FP+FN}$
Recall	Proportion de vrais positifs identifiés correctement $\frac{TP}{TP+FN}$
Precision	Proportion de prédiction positives étant correcte : $\frac{TP}{TP+FP}$
F1	Combinaison de la Precision et du Recall : $2 \times \left(\frac{Precision * Recall}{Precision + Recall}\right)$
Indice de Profit	Fonction de métier donnant du poids aux prédictions. $\frac{(TP \times 0) + (TN \times 3) + (FP \times -1) + (FN \times -10)}{TP + TN + FP + FN}$

Tableau 1 : Présentation de la matrice de confusion et des métriques d'évaluation de modèle de classification. a) Matrice de confusion. b) Métriques d'évaluation et brève description.

Les algorithmes de classification sont majoritairement évalués à partir de métriques calculées via une matrice de confusion (Tableau 1). Ce tableau de contingence confond les valeurs vraies et les valeurs prédites par le modèle.

La fonction coût-métier est appelée **indice de profit**, sur laquelle le modèle a été optimisée. Elle pénalise les Faux Négatifs (clients avec difficultés de remboursement, prédit sans difficulté) fortement par une valeur de -10 (perte de 10 000 € pour l'entreprise par exemple). Elle pénalise aussi les Faux Positifs (clients sans difficulté, prédits avec difficultés) par -1, symbolisant une perte virtuelle (minime) pour l'entreprise. La fonction favorise les Vrai Négatifs (clients sans diff., prédits sans diff.) symbolisant un gain pour l'entreprise. Enfin, les Vrai Positifs (client avec diff., prédit avec diff.) sont ont un poids de 0, ne symbolisant ni coût ni perte pour l'entreprise. Enfin, cet indice est rapporté au nombre de client.

Comme précisé précédemment, la sélection du meilleur modèle et son optimisation ont été effectuées sur la fonction coût-métier (indice de Profit).

III) Tableaux de synthèse des résultats

Synthèse des résultats issus de Pycaret sous condition de sous-échantillonnage :

Model	Accuracy	AUC	Recall	Prec.	F1	Profit	TT(Sec)
Light Gradient Boosting Machine	0.7055	0.7747	0.7039	0.7062	0.705	-0.566	0.9025
Gradient Boosting Classifier	0.6998	0.7677	0.7064	0.6971	0.7017	-0.5817	15.58
Ada Boost Classifier	0.6899	0.7539	0.6925	0.6889	0.6907	-0.6628	3.8025
Random Forest Classifier	0.6927	0.7556	0.6776	0.6987	0.688	-0.6961	8.125
Linear Discriminant Analysis	0.6822	0.7438	0.6821	0.6823	0.6821	-0.7248	0.435
Ridge Classifier	0.6818	0.0	0.6816	0.6819	0.6817	-0.7278	0.1575
Extra Trees Classifier	0.6872	0.7508	0.6677	0.6948	0.681	-0.7478	3.095
Naive Bayes	0.5608	0.6217	0.807	0.541	0.6473	-0.8359	0.1975
Quadratic Discriminant Analysis	0.6356	0.6775	0.5746	0.6592	0.6099	-1.2337	0.23
Logistic Regression	0.6062	0.646	0.5742	0.6136	0.5932	-1.3523	9.9525
Decision Tree Classifier	0.5908	0.5908	0.5887	0.5911	0.5899	-1.3707	1.8825
SVM - Linear Kernel	0.5113	0.0	0.6627	0.5309	0.5542	-1.4667	0.7525
K Neighbors Classifier	0.5495	0.5661	0.5592	0.5485	0.5538	-1.6242	4.38
Dummy Classifier	0.5	0.5	0.0	0.0	0.0	-3.4998	0.105

Synthèse des résultats issus de Pycaret sous condition de sur-échantillonnage (SMOTE) :

Model	Accuracy	AUC	Recall	Prec.	F1	Profit	TT(Sec)
Extra Trees Classifier	0.9145	0.7397	0.0769	0.3616	0.1269	1.9688	40.6367
Light Gradient Boosting Machine	0.9154	0.7558	0.0684	0.3706	0.1154	1.9683	7.5267
Random Forest Classifier	0.9125	0.7275	0.0684	0.3093	0.112	1.9565	106.4533
Dummy Classifier	0.9193	0.5	0.0	0.0	0.0	1.9506	0.7033
Gradient Boosting Classifier	0.9049	0.7225	0.1118	0.2787	0.1596	1.9473	256.9267
Ada Boost Classifier	0.8765	0.711	0.2039	0.2175	0.2105	1.8783	64.93
Decision Tree Classifier	0.8246	0.549	0.2202	0.1365	0.1685	1.6786	12.4633
SVM - Linear Kernel	0.8076	0.0	0.2303	0.1525	0.1341	1.6156	3.2533
Linear Discriminant Analysis	0.6872	0.7446	0.6785	0.1603	0.2594	1.351	3.7567
Ridge Classifier	0.6872	0.0	0.6785	0.1603	0.2594	1.3507	1.0167
K Neighbors Classifier	0.6804	0.5539	0.3711	0.1002	0.1578	1.1747	220.4467
Logistic Regression	0.6361	0.6462	0.5706	0.1228	0.202	1.0943	30.1133
Quadratic Discriminant Analysis	0.4526	0.6485	0.747	0.1029	0.1808	0.4457	5.1333
Naive Bayes	0.3018	0.6141	0.8418	0.0902	0.1629	-0.1116	1.22

Résultats du meilleur run d'optimisation par Optuna (moyenne sur 3 folds de validation croisée):

Model	Accuracy	AUC	Recall	Prec.	F1	Profit
Light Gradient Boosting Machine	0.869	0.764	0.366	0.271	0.311	1.928

IV) Interprétabilité globale et locale du modèle

Afin de faire preuve de transparence et de pouvoir justifier une décision (comme accorder un crédit), il est nécessaire de pouvoir interpréter les prédictions du modèle.

Avec des modèles linéaires, le coefficient de chaque variable sur la prédiction est connu et même testé statistiquement. Cependant, certains modèles de Machine Learning ne sont pas interprétables directement et sont considérés comme des boîtes-noires. C'est le cas du LightGBM (basé sur un algorithme de gradient boosting) utilisé dans cette analyse.

On distingue deux niveaux d'interprétabilité. La première est le niveau **global**, correspondant à la contribution d'une variable sur l'ensemble des données. La seconde est le niveau **local**, correspondant à la contribution d'une variable sur une prédiction précise (ici sur la capacité d'un client spécifique à rembourser son prêt).

1) Interprétabilité globale :

Afin de déterminer l'interprétabilité globale du modèle, différentes solutions ont été mises en place. Le modèle LightGBM est basé sur des arbres de décisions : il est possible de calculer le nombre de fois où la variable est utilisé dans l'ensemble des arbres. Cette valeur est calculée durant l'entraînement du modèle et est disponible sous le nom de **Feature Importance** (Figure 3).

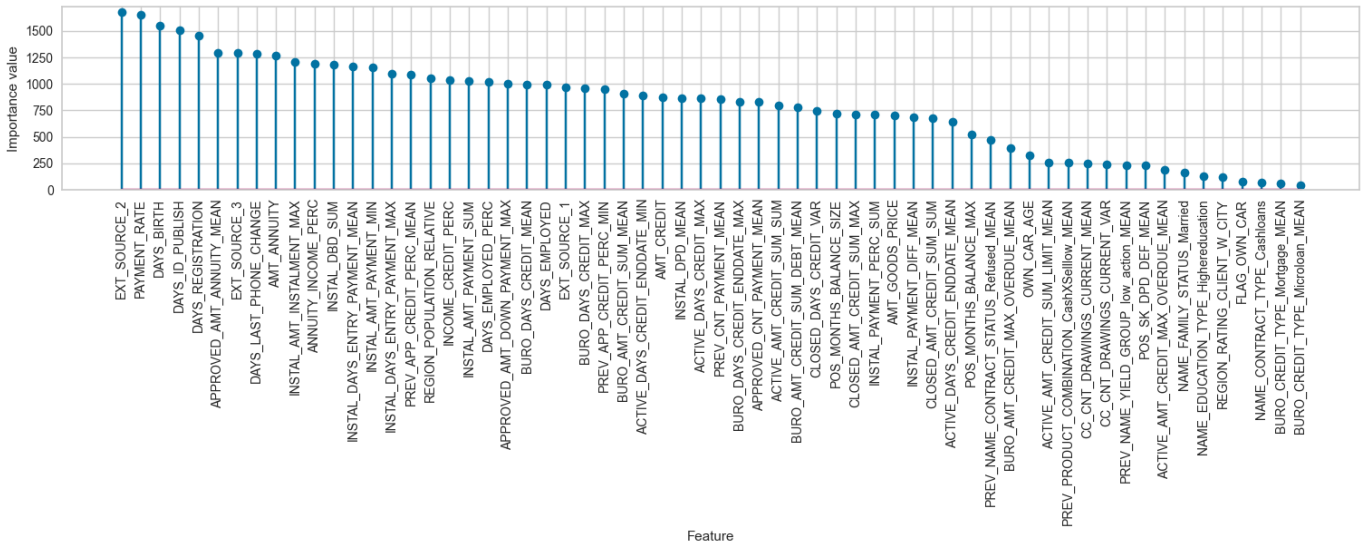


Figure 3 : Importance des variables du modèle LightGBM montrant l'interprétabilité globale. La valeur de l'importance correspond au nombre de fois où la variable a été utilisé durant l'entrainement.

La seconde approche est d'utiliser la bibliothèque SHAP développée par Lundberg, permettant de calculer les valeur SHapley Additive exPlanations [6]. Cet algorithme se base sur la théorie des jeux coopératifs, formulée par Lloyd Shapley en 1953 [7]. Les valeurs SHAP permettent de quantifier l'influence de chaque variable sur la prédiction **au niveau local** (pour chaque client). Ainsi, la moyenne des valeurs SHAP de tous les individus (clients) permettent de calculer l'effet de chaque variable sur la réponse, **au niveau global** (Figure 4).

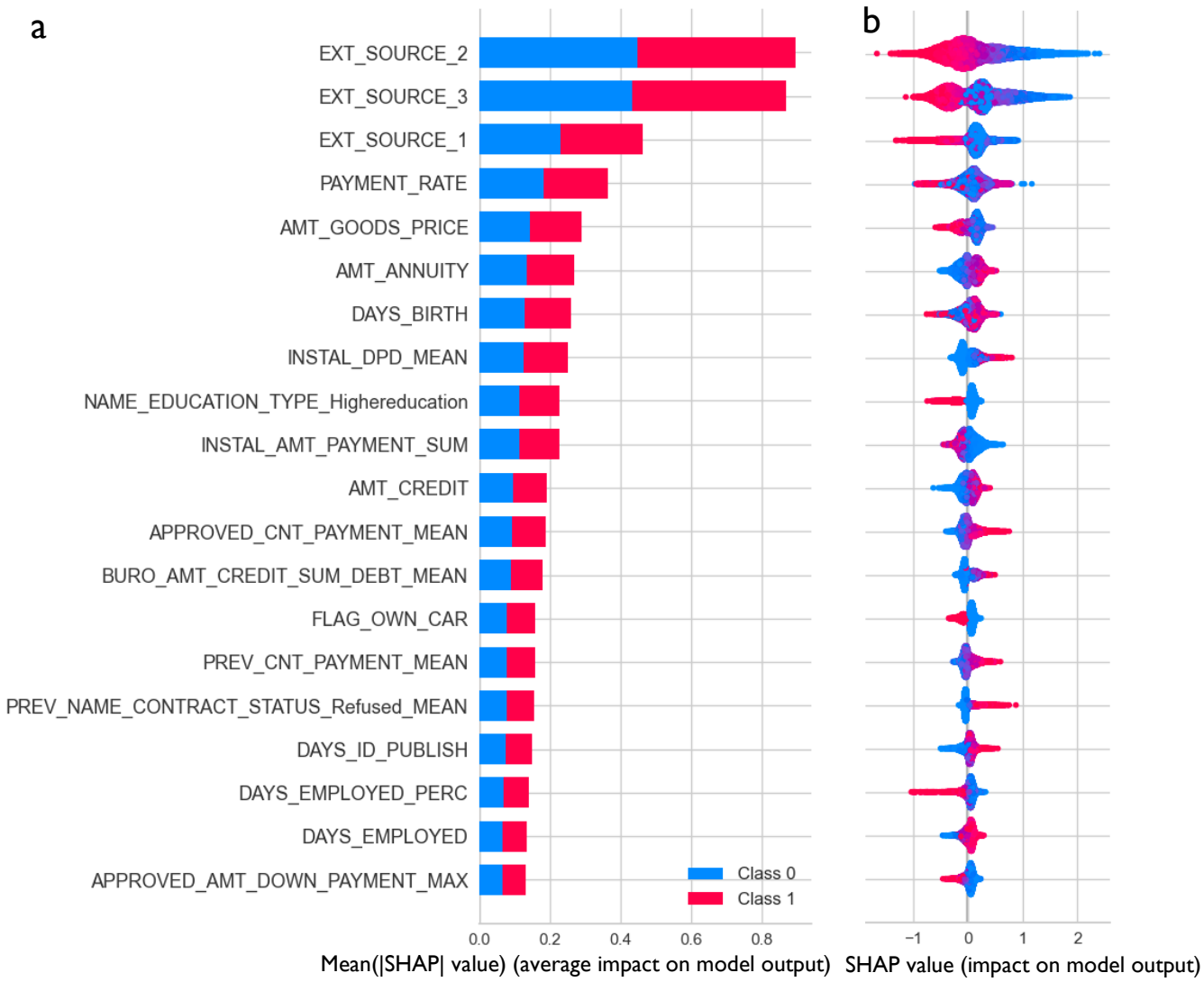


Figure 4 : Interprétabilité globale du modèle par valeur SHAP. a) Graphique en barre de la moyenne des valeurs SHAP des variables calculées sur un ensemble de clients. b) Graphique en essaim des valeurs SHAP de chaque variable. Chaque point représente un client, coloré en fonction de sa classe. Ce graphique permet de déterminer le sens de l'effet de chaque variable.

[6] Lundberg & Lee. A Unified Approach to Interpreting Model Predictions. *NIPS, 2017*.
[7] L. S. Shapley, « A Value for n-Person Games », in H. Kuhn and A. Tucker (Eds.), *Contribution to the Theory of Games*, vol. II, Princeton, 1953, p. 303-317

2) Interprétabilité locale :

Au niveau local, les valeurs SHAP peuvent être calculées pour chaque client, quantifiant ainsi l'effet de chaque variable sur la prédiction (Figure 5). Les valeurs positives favorisent l'appartenance à la classe 1 (difficultés à rembourser), les valeurs négatives favorisent l'appartenance à la classe 0 (pas de difficultés).

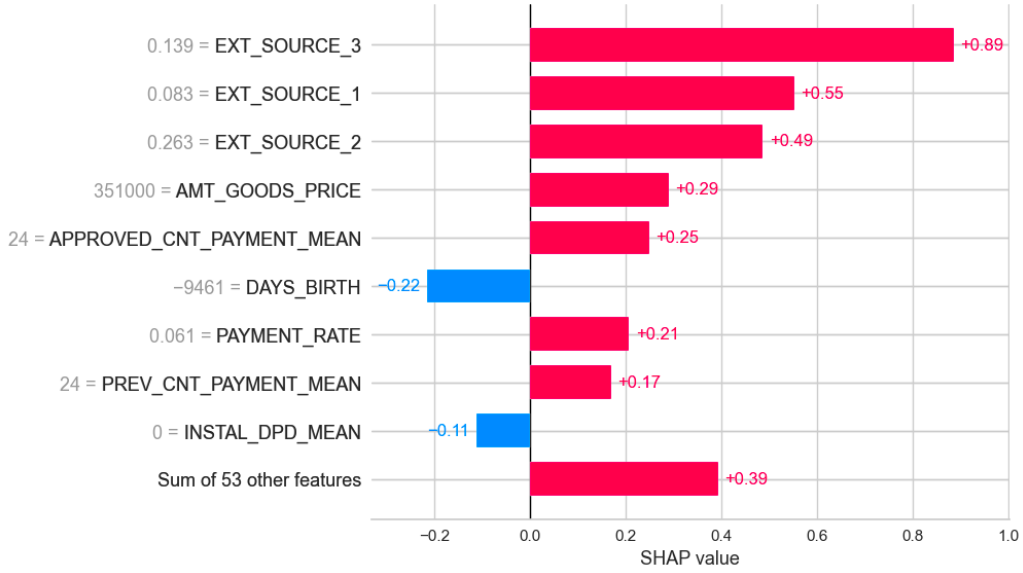


Figure 5 : Graphique en barre des valeurs SHAP d'un client particulier (ID : 100002)

V) Les limites et les améliorations possibles

De nombreuses améliorations peuvent être apportées au modèle afin d'améliorer ses performances de prédiction. Cependant, ces améliorations peuvent demander plus de temps d'entrainement et d'optimisation, ainsi que plus de ressources matérielles, pouvant engendrer des coûts supplémentaires.

La liste ci-dessous détaille des pistes d'améliorations de la modélisation sur les différentes étapes :

- Création de variables : Prise en compte d'interactions entre les variables
- Sélection de variables : Algorithmes performants et robustes (Boruta)
- Modification de la fonction de « l'indice de Profit » avec des valeurs réelles
- Optimisation : Augmenter le nombre d'itérations d'Optuna (optimisation des hyperparamètres)

VI) Analyse de la dérive de données

La dérive des données correspond à des changements de distributions de nos variables avec l'ajout ou modification de clients. Or, notre modèle a été spécifiquement entraîné sur ces variables à un instant *t*, qui peuvent ne plus correspondre à la réalité de l'instant *t+1*. La dérive des données peut donc entraîner une dégradation des performances du modèle au fil du temps.

Il existe différents outils permettant de tester la dérive des données, généralement basés sur la comparaison de distribution des variables entre un jeu de données de référence et un jeu de données cible (contenant de nouvelles données). La bibliothèque Evidently[8] est l'un de ces outils, présentant la dérive des données à l'échelle de chaque variable et à l'échelle globale. Le jeu de données de référence est celui utilisé pour entraîner le modèle ; le jeu de données cible est celui dont on ne connait pas la difficulté de remboursement des clients (*application_test.csv*)

1) Dérive à l'échelle de chaque variable

Le rapport Evidently informe de la présence de dérive de données sur 7 variables (Figure 4). En effet, on observe une différence significative entre les distributions de ces variables entre les données de référence et les nouvelles données (cible).

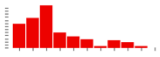
	Column	Type	Reference Distribution	Current Distribution	Data Drift	Stat Test	Drift Score
›	PAYMENT_RATE	num			Detected	Wasserstein distance (normed)	0.582777
›	INCOME_CREDIT_PERC	num			Detected	Wasserstein distance (normed)	0.271927
›	AMT_GOODS_PRICE	num			Detected	Wasserstein distance (normed)	0.210475
›	AMT_CREDIT	num			Detected	Wasserstein distance (normed)	0.206039
›	AMT_ANNUITY	num			Detected	Wasserstein distance (normed)	0.172495
›	NAME_CONTRACT_TYPE_Cashloans	num			Detected	Jensen-Shannon distance	0.147861
›	DAYS_LAST_PHONE_CHANGE	num			Detected	Wasserstein distance (normed)	0.138886

Figure 2: Rapport Evidently des variables présentant de la dérive. La différence statistique entre les deux distributions est calculée par un test de Wasserstein ou Jensen-Shannon.

[8] https://github.com/evidentlyai/evidently

2) Dérive à l'échelle du jeu de donnée totale

La bibliothèque Evidently considère qu'il y a une dérive de données à l'échelle globale uniquement si plus de 50% des variables (ou un tiers des plus importantes) dérivent. Ce n'est pas le cas avec nos données, il n'y a donc pas de dérive de données globale. En revanche, il faudra tout de même être prudent avec les variables PAYMENT_RATE, AMT_GOODS_PRICE, AMT_CREDIT et AMT_ANNUITY. Ces quatre variables sont importantes pour le modèle et présente de la dérive.

Voici la conduite à tenir en cas de dérive des données :

- Tester le modèle après *n* nouveaux clients et mesurer les différentes métriques
- Établir une alerte basée sur un seuil à ne pas dépasser (par exemple un AUC de 0,75)
- En cas d'alerte, réentraîner et réoptimiser le modèle
- Redéployer le modèle