# Note méthodologique sur l’entraînement du modèle pour « prêt à dépenser »

## Table des matières

# Entraînement d’un modèle prédisant la capacité de remboursement d’un client.

## 1 Contexte

Des données bancaires et annexes ont été collectées sur de nombreux clients et sont disponibles ici.

L’objectif est de les utiliser pour établir un modèle supervisé permettant de prédire la capacité de remboursement d’un nouveau client venant faire une demande de crédit chez « prêt à dépenser ».

Notons que pour accélérer le projet, la phase de nettoyage et de génération de variables pertinentes à partir des données brutes a été réalisée en utilisant les pipelines présents sur ce répertoire GitHub.

Dans cette note méthodologique, je fais synthèse :

- de l’exploration des données et du problème inhérent de déséquilibre des classes ;
- de mes réflexion sur la construction d’une fonction coût métier adaptée au problème, et des choix des métriques plus classiques que j’ai décidé de monitorer ;
- de la méthode suivie pour opter pour le modèle le plus convainquant et des résultats obtenues aux différentes étapes ;
- de l’interprétabilité du modèle retenu ;
- de l’analyse du data-drift ;
- et enfin, des limites et améliorations possibles concernant le modèle retenu.

## 2 Exploration des données

Cf. *quick_EDA_of_Kaggle_Aguiar_feature_engineering.ipynb*

### A - Prédicteurs et explication des pré-traitements et options de pré-traitements disponibles via la fonction *load_split_clip_scale_and_impute_data()*

À la sortie du nettoyage et du feature engineering, nous sommes en présence de 735 variables dont :

- 42 sont intégralement renseignées ;
- 482 présentent au moins 10 % de valeurs manquantes ;
- 262 présentent au moins 50 % de valeurs manquantes ;

Du côté des 356 250 individus, on observe la distribution du nombre de valeurs manquantes suivantes :

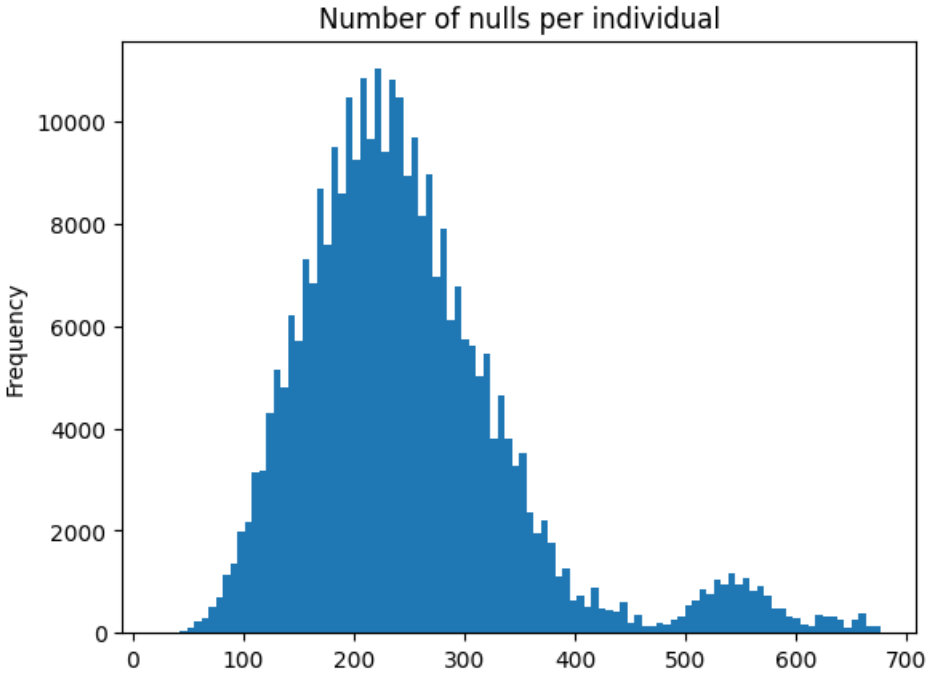



*Figure 1: Distribution du nombre de valeurs manquantes par client*

Les valeurs manquantes devant être gérées et imputées pour faire tourner certains modèles, la fonction écarte les clients présentant plus de 400 valeurs manquantes et ceux ne présentant pas d'information sur la variable ciblée. Il reste alors 286 952 individus.

Les variables catégorielles y sont imputées grâce au plus proche voisin car un moyennage n'est pas envisageable. Pour les variables numériques, plusieurs options sont disponibles (imputation par médiane, zéro, knn).

Par ailleurs, on découvre la présence de valeurs infinies (positives et négatives) dans les données. 53 variables et plus de 285 000 individus en sont dotées. Il est donc possible de borner optionnellement les données à partir des valeurs extrêmes rencontrées dans le jeu d'entraînement.

Différentes options de scaling sont également accessibles et la fonction renvoie, en plus des jeux de tests et d'entraînements, les différents objets permettant le pré-traitement afin de les intégrer si besoin dans le modèle final.

## B - Déséquilibre des classes au sein de la variable cible

On observe un déséquilibre des classes à prédire avec un ratio d'environ 11,5. Il faut savoir que cela complique la tâche d'apprentissage des modèles qui ont généralement une propension à prédire la classe dominante. Pour éviter cet écueil, 3 méthodes ont été envisagées.

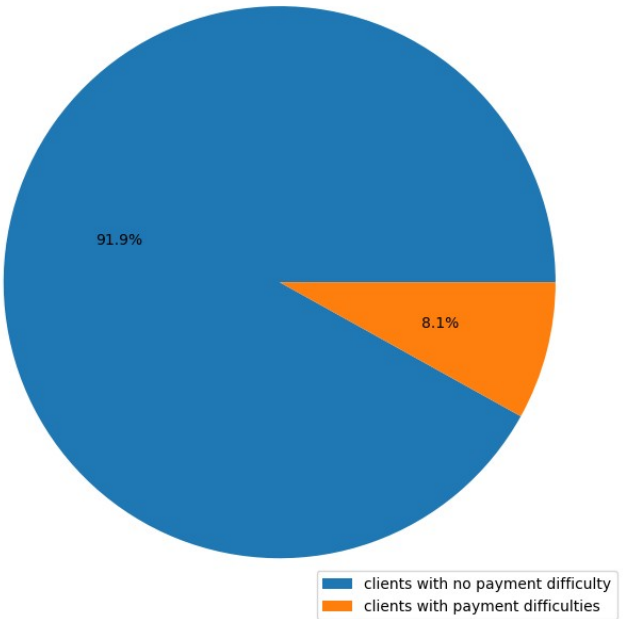



*Figure 2: Proportion des classes dans la variable cible.*

- utilisation d’une option dans les hyper-paramètres des modèles permettant de tenir compte de pondérations des classes lors de l’évaluation de celui-ci afin d’orienter l’apprentissage des paramètres du modèle.

- pré-traitement des données d’apprentissage afin d’équilibrer artificiellement les effectifs des classes en sur-échantillonnant la classe minoritaire (ou en générant des échantillons plausibles artificiels avec la méthode SMOTE) et/ou en sous-échantillonnant la classe dominante afin de retrouver une situation plus équilibrée. Dans ce cas, attention de ne pas modifier la population qui sert d’évaluation au modèle.

- ajustement du seuillage des probabilités pour décider de la classe (threshold moving).

Remarque : expérimentations ou informations supplémentaires disponibles dans *building_classifiers.ipynb* .

# 3 Élaboration de la fonction métier et choix de métriques classiques à monitorer.

## A - Considérations générales

Le modèle prédit 2 classes :

- 0 ; classe dite négative ; le client n’a pas de problème a remboursé son crédit.
- 1 ; classe dite positive ; le client possède des difficultés à rembourser son crédit.

Économiquement, on comprend dès lors que prédire un faux-positif induit un manque à gagner puisque l’on évince un bon client en le prenant pour un mauvais payeur, tandis que prédire un faux-négatif provoque une perte d’argent puisque le client n’est pas en mesure de rembourser alors qu’on l’en pensait capable.

Au bilan, on veut donc limiter les mauvaises prédictions, les faux-négatifs davantage.

## B - Fonction métier

Sous l’hypothèse qu’un faux-négatif engendre des complications et des frais en moyenne 5 fois supérieurs aux pertes liées à la prédiction d’un faux-positif, la fonction métier imaginée, nommée *LoI* pour ‘loss of income’ est décrite ainsi :

$$LoI = \frac{5FN + FP}{N_{indiv}}$$

où *FN* est le nombre de faux-négatifs, *FP* celui de faux-positifs et *N_{indiv}* est le nombre total d’individu appartenant à l’échantillon sur lequel on évalue le modèle.

Elle permet d’évaluer un risque moyen financier ramené à un individu dans une population donnée.

On souhaitera donc minimiser cette quantité.

### C - Métriques monitorées

Avec les mêmes considérations, la fonction f-beta pénalise davantage les faux-négatifs lorsque beta est supérieur à 1.

$$F_\beta = \frac{(1+\beta) \times TP}{(1+\beta) \times TP + \beta \times FN + FP}$$

Notons cependant qu'augmenter beta diminue conjointement l'impact des mauvaises prédictions (quelles qu'elles soient) relativement aux vrai-positifs. Il est donc illusoire de traduire l'hypothèse précédente en choisissant trivialement beta=5.

Par compromis sur les 2 effets cités ci-dessus, la **fonction f-2** a été retenue pour le monitoring. On voudra la maximiser.

Pour une raison différente, on monitorera également l'**AUC**. Elle reflète la capacité du classifieur à attribuer des probabilités supérieures aux éléments de la classe positive relativement aux éléments de la classe négative. Autrement dit, elle donne une indication de la capacité du modèle à ordonner correctement les éléments en fonction de leur classe. De plus, par connaissance à priori du jeu de données, elle permettra de surveiller d'éventuels effets de sur-apprentissage.

# 4 Méthodologie suivie pour sélectionner le meilleur type de modèle.

cf répertoire *hyperparameter_tuning.*

Pour trouver le meilleur modèle sans être confronté à des coûts computationnels débordants, j'ai choisi d'organiser une recherche des meilleurs hyper-paramètres pour plusieurs modèles sur un jeu de données restreint (2 000 individus) divisé en un jeu d'entraînement (1600) et un jeu de test (400).

Cette recherche des meilleurs hyper-paramètres a été effectuée grâce à la librairie hyperopt en tablant sur 50 itérations par modèle et en cherchant à minimiser la fonction métier *LoI* précédemment définie. Les scores obtenus ont été sauvegardés en utilisant Mlflow.

Pour l'évaluation des modèles, un schéma de cross-validation avec 5 folds a été mis en place sur le jeu d'entraînement. Chaque modèle a donc été entraîné 5 fois sur des combinaisons de 4 folds donnant lieu à une prédiction sur le fold restant et à une prédiction sur le jeu de test. Les prédictions sur les folds restants ont été concaténées pour donner une prédiction sans data leakage sur le jeu d'entraînement, et elles ont été moyennées sur le jeu de test. Cela permet à la fois une évaluation robuste (cross-validation), et de juger d'un éventuel sur-apprentissage si les scores obtenus par cross-validation sont significativement plus élevés que ceux du jeu de test.

**Remarque importante :** Plutôt que d'évaluer la prédiction des classes avec un seuillage par défaut, une classe Scorer, disponible dans le répertoire *project_tools* a été implémentée, capable de recevoir les probabilités prédites par un modèle et de chercher le seuil optimisant la métrique pour laquelle le Scorer a été instancié. Ainsi, le processus de « threshold moving » a été intégré à la recherche des meilleurs paramètres.

S’il aurait été intéressant de faire varier également les pré-traitements dans cette recherche du meilleur modèle, la configuration a été figée :

- Variables catégorielles imputées au plus proche voisin et one-hot-encoded.
- Variables numériques bornées (pour gestion des valeurs infinies).
- Valeur ramenées dans l’intervalle [0, 1] pour les variables numériques.
- Imputation des variables numériques par knn, k=2

Les types de modèles candidats furent :

- Regression logistique avec régularisation Lasso
- Regression logistique avec régularisation Ridge
- RandomForest classifier
- SVC radial basis function kernel
- SVC polynomial kernel
- lightgbm classifier
- xgboost classifier

# 5 Résultats

En annexes sont présentés pour les 3 métriques, le top-10 des 250 modèles évalués, ainsi qu’une comparaison graphiques des meilleurs scores obtenus par chaque type de modèle, vainqueur affiché à gauche.

- On y voit que les SVC obtiennent de mauvais scores et peinent à généraliser sur le jeu de test.
- 2 modèles obtiennent des résultats consistants sur les 3 métrics : le RandomForest et lightgbm.
- Pour la métrique loss of income :

best tuning for each model type:

| parent_run_name | run_id | metrics.CV_loss_of_income | metrics.test_loss_of_income | fit_time_s |
|---|---|---|---|---|
| Lasso-type Logistic Regression | ea8d66f46e0f43709e647163f1ac73aa | 0.294375 | 0.3675 | 4.123 |
| LightGBM Classifier | 293f72ac0fbb44a98e55950c3c7bd52c | 0.304375 | 0.3475 | 7.896 |
| Random Forest | d8e0a324bfb94fc3bbfedf9c4d4360be | 0.290625 | 0.3200 | 24.896 |
| Rigde-type Logistic Regression | 131da8d8245b42ef87fe890a9728bf1d | 0.301875 | 0.3575 | 5.145 |
| SVC_poly | ff73ce5216cb452f8d47bde5d3b051cd | 0.342500 | 0.3750 | 14.738 |
| SVC_rbf | 12b4802eebb84382ac5d6db9ab694b6e | 0.330625 | 0.3675 | 24.900 |
| XGBoost Classifier | 705f72b891ea41e6aaf6fcd5d369ae6c | 0.302500 | 0.3450 | 360.156 |

*Figure 3: Résultats pour les meilleures configurations de chaque type de modèle testés en optimisant la fonction métier.*

Le Random Forest donne les meilleurs résultats. On peut en profiter au passage pour avoir une indication du temps d’entraînement sur la cross-validation pour chaque type de modèle

# 6 Approfondissement des réglages pour le random forest.

## A - SMOTE ou non ?

*Cf. random_forest/random_forest_smote.py pour le code et explore_mlflow_tracking.ipynb pour les résultats*

Pour le moment seules les techniques de mouvement de seuil pour décider de l'affectation d'une classe ainsi que l'utilisation des pondérations pour l'évaluation ont été intégrées dans la recherche des hyper-paramètres.

J'ai donc décidé d'effectuer une seconde recherche d'hyper-paramètres plus spécifique au random forest en intégrant ou non des étapes de pré-traitements comprenant parfois des hyper-paramètres (notamment les étapes de sur et sous échantillonnage).

Ces étapes sont les ajouts optionnels en pré-traitement du random forest :

- d 'un premier bloc que l'on nommera SMOTE constitué :
    - d'un SMOTENC qui permet la création d'individus artificiels à partir des individus de la classe minoritaire, tout en assurant une génération plausible des variables catégorielles qui ne peuvent correspondre à un moyenne pondérée risquant de sortir de leur ensemble de définition.
    - d'un sous-échantillonneur de la classe dominante.

    À noter que les proportions de ré-échantillonnage ont été envisagées comme des hyper-paramètres du modèle.
- d'un second bloc permettant de one-hot-encoder les variables catégorielles (déjà label encoded).

J'ai donc obtenu 3 pipelines pour lesquels ont été cherchés au cours de 100 itérations hyperopt sur un jeu de données restreint (1500 individus, imputation knn, scaling minmax), les meilleurs hyper-paramètres. Les meilleurs configurations de ces pipelines donnent les scores suivants :

| parent_run_name | metrics.CV_AUC | metrics.CV_loss_of_income | metrics.CV_f2 | r |
|---|---|---|---|---|
| SMOTE pipeline with Random Forest | 0.723628 | 0.300000 | 0.466667 | |
| SMOTE pipeline, ohe categorical, with Random Forest | 0.727545 | 0.299167 | 0.482456 | |
| ohe categorical with Random Forest | 0.736479 | 0.283333 | 0.494327 | |

*Figure 4: Scores des 3 pipelines de pré-traitement dans leur meilleure configuration.*

Pour les 3 métriques, l'introduction de SMOTE dégrade les résultats et la présence du one-hot encoder les améliore. Il semblerait que parfois, le jeu sur le seuil de décision des classes soit le plus pertinent pour faire face à un problème de déséquilibre des classes.

## B - Scaling et imputation

*cf. random_forest/random_forest_imputation_and_scaling_variation.py*

Toujours sur un jeu de taille similaire, j'ai évalué les impacts de ces 2 étapes de pré-traitements, sans SMOTE et avec one-hot encoding, en faisant une recherche des meilleurs hyper-paramètres du random forest connecté en sortie, sur 100 itérations. Les meilleurs scores obtenus sont :

| | | metrics.CV_AUC | metrics.CV_loss_of_income | metrics.CV_f2 |
|---|---|---|---|---|
| scaling_method | imputation_method | | | |
| minmax | knn | 0.751201 | 0.278333 | 0.523349 |
| | median | 0.747347 | 0.264167 | 0.525862 |
| | zero | 0.732510 | 0.258333 | 0.508475 |
| standard | knn | 0.752525 | 0.281667 | 0.513866 |
| | median | 0.735382 | 0.262500 | 0.510949 |
| | zero | 0.730186 | 0.263333 | 0.502793 |

*Figure 5: Résultats pour les meilleures configurations du classifieur Random Forest suivant diverses méthodes de scaling et d'imputation. Jeu de données restreint.*

Ici et pour la première fois, on constate une dissonance entre la fonction métier et les métriques plus standards. Rappelons que loss_of_income doit être minimisée là où f2 et AUC doivent être maximisées. On a donc un arbitrage plus compliqué qu'auparavant pour choisir les modèles et des questions surgissent :

- Les résultats sont-ils significatifs au vue de la taille de l'échantillon choisi ?
- La métrique métier est-elle contestable sur certains points ?

## C - Scores des meilleurs modèles random forest

Parce qu'une imputation par knn est très coûteuse et parce que notre rôle est d'optimiser le modèle pour minimiser les pertes de l'entreprise, j'ai choisi de poursuivre la recherche des hyper-paramètres sur l'ensemble des individus (~285 000) en appliquant un pré-traitement minmax et imputation par valeur zéro comme trouvé précédemment. Sur 100 itérations hyperopt, on obtient pour les meilleures configurations:

| | metrics.CV_AUC | metrics.CV_loss_of_income | metrics.CV_f2 | metrics.test_AUC |
|---|---|---|---|---|
| run_id | | | | |
| d8af249a9012411199de19d33f40a783 | 0.698053 | 0.332687 | 0.431447 | 0.704462 |
| f35499ce02624c28b2acd8ff480adf7f | 0.700987 | 0.332822 | 0.435012 | 0.706140 |
| 9f987f2d9b424619b7dc6f705b6eecbf | 0.699141 | 0.332874 | 0.433245 | 0.706612 |
| 9a1e10a281b9487eb9464b394ebb0749 | 0.698443 | 0.332966 | 0.431926 | 0.705587 |

*Figure 6: Résultats pour les meilleures configurations du random forest entrainé sur l'intégralité du jeu de données.*

Cela m'est apparu comme décevant et j'ai donc également recherché les meilleurs hyper-paramètres pour le lightgbm qui était aussi très bien classé au regard des 3 métriques. En l'entraînant sur l'ensemble des individus, sans pré-traitement, puisque le modèle peut recevoir des valeurs manquantes ou infinies, et est aussi en capacité de gérer les variables catégorielles encodées par des entiers.

# 7 Résultats pour le lightgbm classifier

| run_id | metrics.CV_AUC | metrics.CV_loss_of_income | metrics.CV_f2 |
|---|---|---|---|
| b28c67e1b2e44dc58afdd82fcc002e5f | 0.721945 | 0.314635 | 0.460757 |
| edec9d8321fc4fc4ba61cc51be0d7d58 | 0.700061 | 0.328125 | 0.436897 |
| 568ca738102747a794001419482a6dfe | 0.697434 | 0.329542 | 0.434046 |
| 519314198d6a408ba4be680f39105604 | 0.699174 | 0.329708 | 0.435880 |
| f89ee8ac919647eba28ad16516ad3cf3 | 0.700384 | 0.329875 | 0.437427 |
| 092dae9fae454af6983d0332bc7b3e12 | 0.699498 | 0.330000 | 0.436155 |
| 9a3318ffe9814495b30e94065addb12d | 0.699520 | 0.330333 | 0.434699 |

*Figure 7: Résultats des meilleurs configurations du lightgbm classifier sur l'ensemble du jeu de données.*

Pour des raisons de temps, j'ai arrêté la modélisation ici et sélectionné le meilleur modèle lightgbm qui obtient de meilleures performances que le random forest.

# 8 Interprétabilité du modèle

## A - Globalement

Les variables les plus déterminantes ont été détectées à l'aide de 2 méthodes :

- moyenne des valeurs absolues des shap values par variable.
- utilisation de l'attribut feature_importances_ du classifieur.

Environ 2/3 des variables sont communes aux 2 méthodes. Le rapport sur le data drift dont je fais état plus loin a été mené sur l'union des top-20 récupérés par chaque méthode.

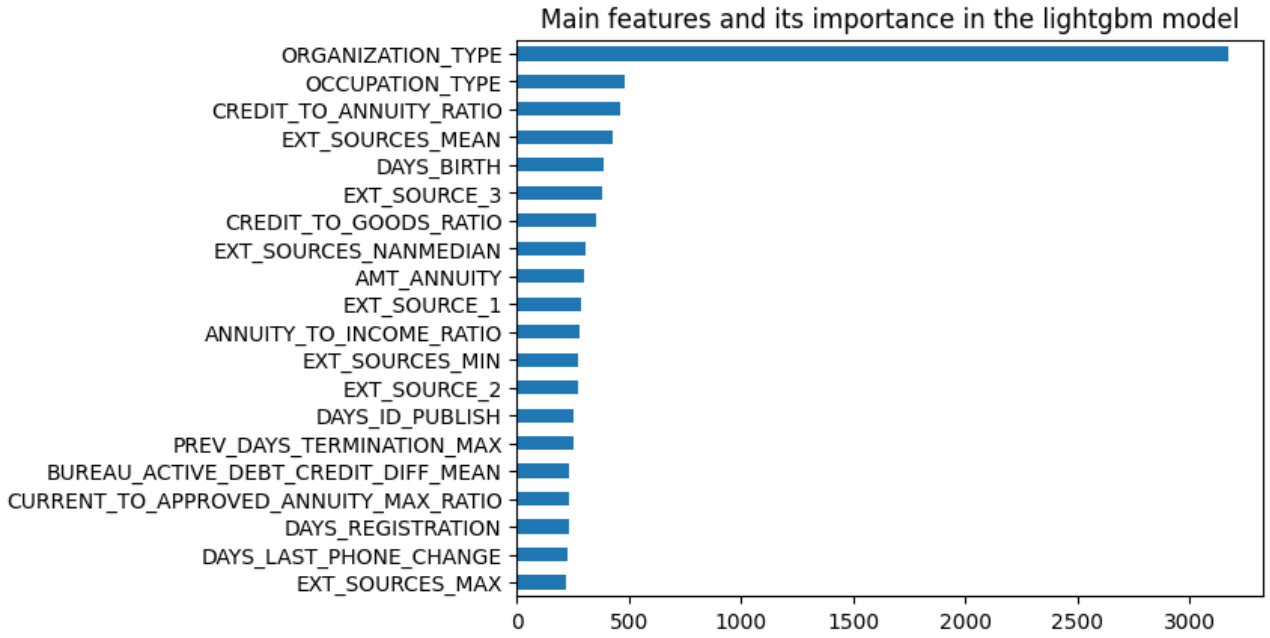



*Figure 8: Variables les plus importantes selon l'attribut feature_importances_ de lightgbm*

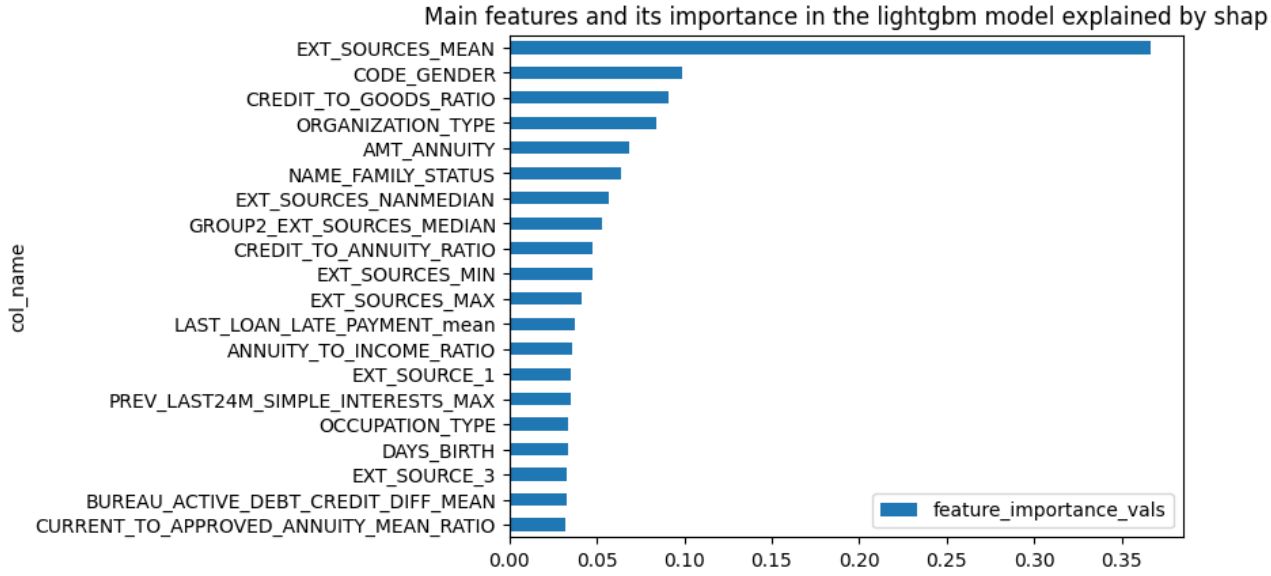



*Figure 9: Variables les plus importantes par moyenne des valeurs absolues des shap values.*

En résumé, les variables importantes sont :

- de nature bancaire :
  - ext_source_N
  - dette actuelle
  - des ratios entre les revenus, les annuités et le crédit.
- statutaires :
  - situation familiale
  - type de la structure dans laquelle on travaille
  - type de métier pratiqué
  - age
  - genre
- Comportementales :
  - achat du dernier téléphone
  - dernier paiement effectué en retard

On peut remarquer au passage que selon shap, le modèle a un comportement discriminatoire sur le genre de l'individu. Peut-être souhaiterions-nous retirer cet aspect de l'algorithme pour éviter des retours clients incendiaires.

## B - Localement

On peut regarder les variables ayant les plus grandes shap values en valeur absolue pour un client donné. Le graphique waterfall de shap permet d'expliquer le rôle de ces variables prédominantes. Poussant la prédiction vers la gauche (en bleu), elle rassure sur les capacités de remboursement de l'individu. Vers la droite (en rose), la valeur de la variable est mal perçue par le modèle et pénalise le client dans l'obtention de son crédit.

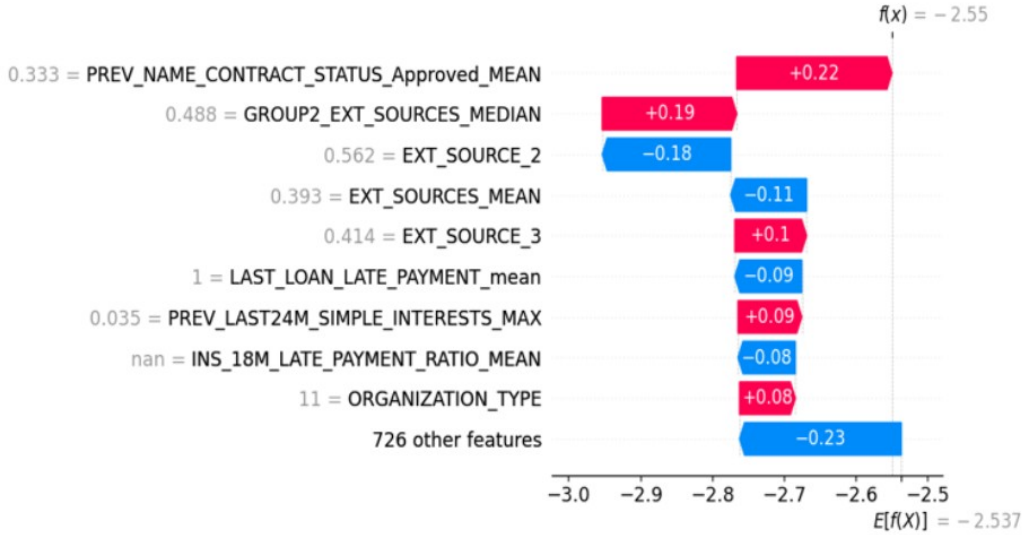



*Figure 10: Exemple d'un waterfall de la librairie shap pour le client 100 079*

# 9 Étude du data drift avec Evidently

*Cf répertoire data_drift*

Evidently permet de comparer rapidement les distributions de mêmes variables au sein de 2 jeux de données distincts afin de percevoir un éventuel data drift (évolution des données au fil du temps relativement aux données sur lesquelles le modèle a été entraîné).

Cette comparaison est possible graphiquement (distributions juxtaposées ou exploration interactive au sein d'un rapport html fourni dans le répertoire *data_drift*) et une mesure de similarité des distributions est effectuée en fonction du type de la variable.

Grâce à un seuillage sur ces mesures, le rapport effectue un test de la proportion des variables ayant dérivé par rapport au jeu d'entraînement utilisé pour bâtir le modèle et permet donc de juger d'une éventuelle obsolescence du modèle. Une utilisation pour vérification avant déploiement du modèle dans le pipeline CI/CD serait une bonne pratique.

Au regard des variables les plus importantes, aucune dérive n'est constaté entre le jeu d'entraînement et le jeu de test. Le modèle est donc légitimement applicable sur le jeu de test.

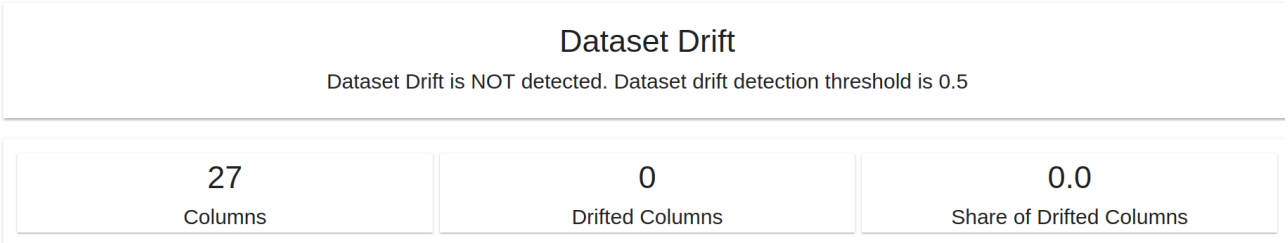



*Figure 11: Résumé du rapport d'evidently sur le data drift*

# 10 Conclusion : limites et axes d’améliorations

La routine pour rechercher le meilleur modèle s’effectue encore en local et par étapes dissociées. Il me semble avoir couvert les essentiels pour le random forest mais pas pour d’autres modèles qui ont peut-être été délaissés trop vite et trop injustement. L’échantillon choisi ayant peut-être manqué d’un peu de représentativité.

Néanmoins, ces choix ont été faits pour permettre une première implémentation du code suffisamment rapide et obtenir des premiers modèles raisonnables pour construire le reste du projet (API et dashboard).

Il semble facile de passer à l’étape supérieure en lançant la recherche d’hyper-paramètres sur un cluster (hyperopt et spark étant compatibles) et aussi de manière plus systématique (avec plus de variation niveau pré-traitements, l’adjonction de SMOTE, pour tous les modèles).

Avant cela, toutefois, il faut rappeler que le nombre de variables (735) reste élevé et qu’aucun retour n’a été fait sur le feature engineering qui comporte sans doute des erreurs.

La recherche des hyper-paramètres étant un gouffre énergétique et financier, il est très fort probable que passer du temps sur la sélection des principales variables pour chaque type de modèle et explorer ces dernières afin d’en améliorer la qualité ait un effet mélioratif sans commune mesure avec le réglage fin des hyper-paramètres.

En continuant l’exploration des modèles sur des ressources similaires, tester les modèles sur des jeux de données au nombre de variables réduit et compenser les éventuels défauts de ces variables dominantes apparaît donc comme la priorité.

La métrique métier a donné des résultats plutôt cohérent avec des métriques bien connues mais un entretien avec des spécialistes du domaine permettrait sans doute d’en créer une encore plus pertinente. L’inclusion du nombre des vrais-positifs afin de ne pas focaliser uniquement sur les pertes mais plutôt sur un gain algébrique moyen est aussi une piste à envisager ?

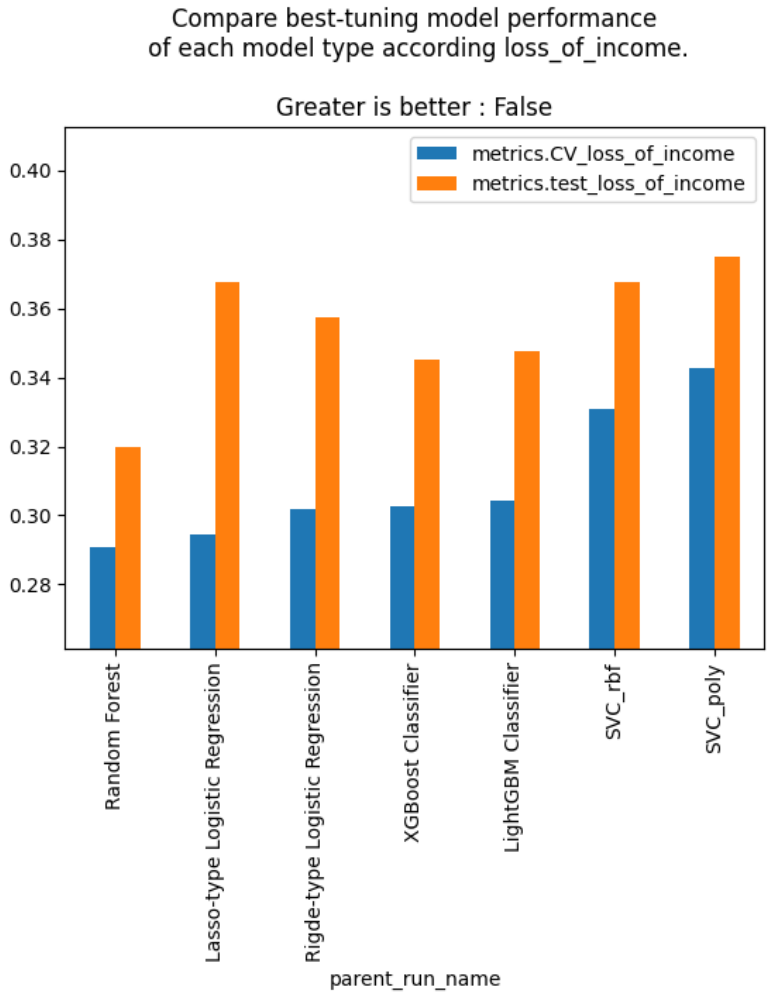
Enfin, reste le souci de la discrimination par genre, une décision est à prendre. Doit-on enlever cette variable du modèle ? Comment peut-on l’expliquer et l’assumer ?

# 11 ANNEXES

Résultats pour la fonction métier loss of income :

loss_of_income
top-15:

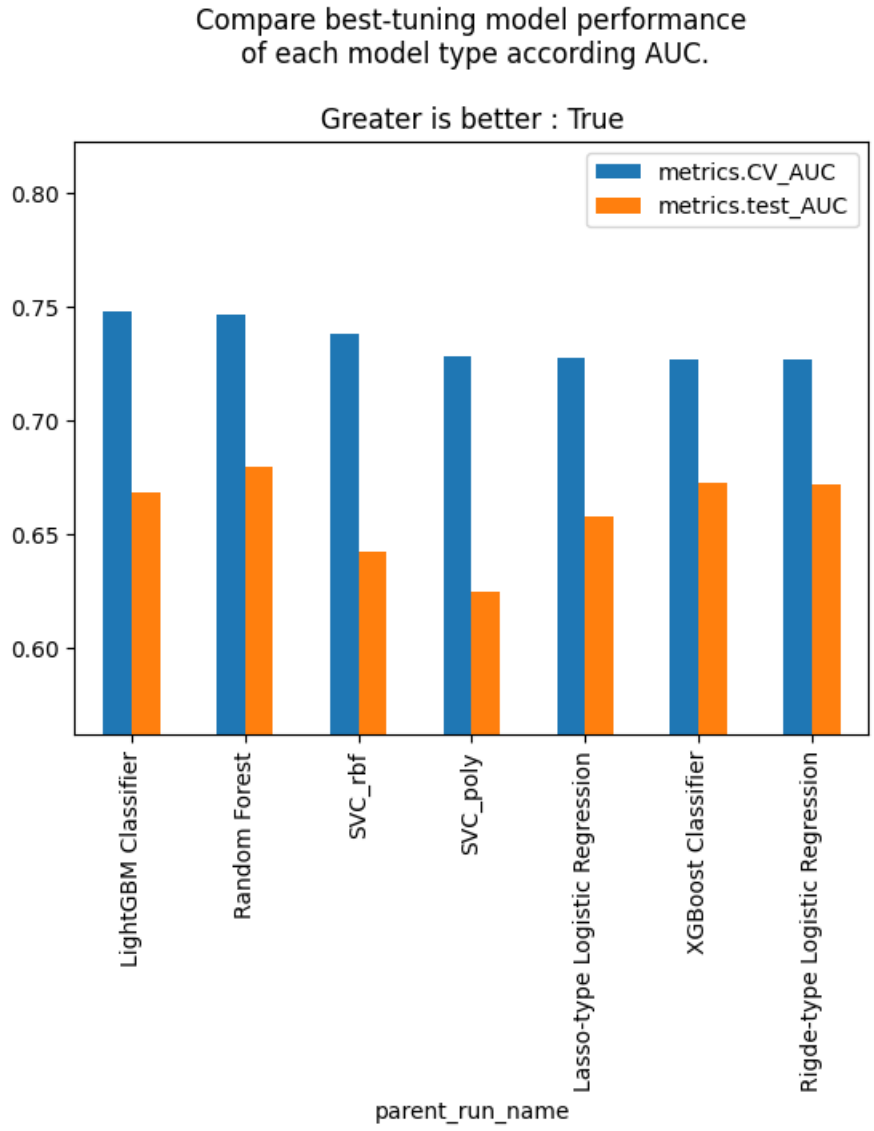
| | run_id | metrics.CV_loss_of_income | metrics.test_loss_of_income | fit_time_s | parent_run_name |
|---|---|---|---|---|---|
| 0 | d8e0a324bfb94fc3bbfedf9c4d4360be | 0.290625 | 0.3200 | 24.896 | Random Forest |
| 1 | 8732eae2be37434bb47447f0525dc63b | 0.291250 | 0.3150 | 22.811 | Random Forest |
| 2 | ea8d66f46e0f43709e647163f1ac73aa | 0.294375 | 0.3675 | 4.123 | Lasso-type Logistic Regression |
| 3 | 126a09ec6b1c42c0b6f5484361feca6b | 0.295625 | 0.3700 | 4.127 | Lasso-type Logistic Regression |
| 4 | 7ea45a4e57df49ffa98356522929c323 | 0.295625 | 0.3675 | 3.634 | Lasso-type Logistic Regression |
| 5 | 4572bea7814645cdaac2d46659cb0570 | 0.295625 | 0.3750 | 3.714 | Lasso-type Logistic Regression |
| 6 | 73584fdb25744214bf5a5a8dcc2789db | 0.296250 | 0.3675 | 4.340 | Lasso-type Logistic Regression |
| 7 | ae1caf1d42d84762b80e478a60983a86 | 0.296250 | 0.3675 | 4.336 | Lasso-type Logistic Regression |
| 8 | e43dd70c864c4698ae87fe6838fb6e81 | 0.296250 | 0.3650 | 4.104 | Lasso-type Logistic Regression |
| 10 | e921ec606fbc41caaed819b2ec8561fa | 0.296875 | 0.3600 | 4.451 | Lasso-type Logistic Regression |



Pour l'AUC :

AUC
top-15:

| | run_id | metrics.CV_AUC | metrics.test_AUC | fit_time_s | parent_run_name |
|---|---|---|---|---|---|
| 75 | 4090ba0636d64443a10f1b27ab8fbf0e | 0.747704 | 0.667988 | 8.044 | LightGBM Classifier |
| 102 | 583c45e0c3d0433eb32cc6c50beff236 | 0.746457 | 0.679052 | 38.040 | Random Forest |
| 81 | 501752272e5a4ebdb5b91a1d5e97caee | 0.743891 | 0.684502 | 37.901 | Random Forest |
| 0 | d8e0a324bfb94fc3bbfedf9c4d4360be | 0.741748 | 0.670217 | 24.896 | Random Forest |
| 76 | 5a421afb372b4f26919e0f0b1c5b7f47 | 0.740180 | 0.689291 | 22.100 | Random Forest |
| 1 | 8732eae2be37434bb47447f0525dc63b | 0.739778 | 0.682933 | 22.811 | Random Forest |
| 64 | 97f8ceb4f7a740e5afab268c1f61ea69 | 0.739320 | 0.677855 | 8.257 | LightGBM Classifier |
| 112 | e5225a720d2e4c1cb99c4405423b16fe | 0.739239 | 0.672529 | 15.087 | Random Forest |
| 194 | 44a9bab20b94401583213e2c9bd6a7eb | 0.737951 | 0.641937 | 26.418 | SVC_rbf |
| 39 | 544e7082658f4c5ea17f34f5a4b36ba6 | 0.736607 | 0.673768 | 23.998 | Random Forest |



Pour f-2 :

f2
top-15:

| | run_id | metrics.CV_f2 | metrics.test_f2 | fit_time_s | parent_run_name |
|---|---|---|---|---|---|
| 0 | d8e0a324bfb94fc3bbfedf9c4d4360be | 0.502994 | 0.398010 | 24.896 | Random Forest |
| 75 | 4090ba0636d64443a10f1b27ab8fbf0e | 0.502092 | 0.397490 | 8.044 | LightGBM Classifier |
| 64 | 97f8ceb4f7a740e5afab268c1f61ea69 | 0.496098 | 0.410156 | 8.257 | LightGBM Classifier |
| 81 | 501752272e5a4ebdb5b91a1d5e97caee | 0.493519 | 0.418410 | 37.901 | Random Forest |
| 38 | 293f72ac0fbb44a98e55950c3c7bd52c | 0.492468 | 0.401460 | 7.896 | LightGBM Classifier |
| 112 | e5225a720d2e4c1cb99c4405423b16fe | 0.492063 | 0.406504 | 15.087 | Random Forest |
| 104 | e8459dda04ff4e748fbee62f386fedd5 | 0.490731 | 0.384615 | 8.430 | LightGBM Classifier |
| 102 | 583c45e0c3d0433eb32cc6c50beff236 | 0.490376 | 0.411523 | 38.040 | Random Forest |
| 76 | 5a421afb372b4f26919e0f0b1c5b7f47 | 0.489899 | 0.427807 | 22.100 | Random Forest |
| 97 | 84ed62e9b6d0461195bf5fb2705ced81 | 0.487541 | 0.413223 | 10.732 | LightGBM Classifier |