Construire un modèle de scoring qui donnera une prédiction sur la probabilité de faillite d'un client de façon automatique.

|  |
| --- |
| **RESUME**  Bla bla bla |

Table des matières

[INTRODUCTION 2](#_Toc117792233)

[DONNÉES 2](#_Toc117792234)

[LA MÉTHODOLOGIE D'ENTRAINEMENT DU MODÈLE 3](#_Toc117792235)

[1. Kernel Kaggle 3](#_Toc117792236)

[2. Feature Engineering 3](#_Toc117792237)

[3. Resampling 4](#_Toc117792238)

[4. Preprocessing 4](#_Toc117792239)

[5. Modèle LGBM 4](#_Toc117792240)

[6. Calibration des hyperparamètres pour le score auc 5](#_Toc117792241)

[7. La matrice de confusion avec AUC 5](#_Toc117792242)

[La fonction coût métier, l'algorithme d'optimisation et la métrique d'évaluation (1 page maximum) 6](#_Toc117792243)

[LA MATRICE DE CONFUSION 6](#_Toc117792244)

[COURBE ROC ET SCORE AUC 7](#_Toc117792245)

[FEATURES IMPORTANCE 7](#_Toc117792246)

[8. OPTIMISATION XGBOOST 8](#_Toc117792247)

[HYPERPARAMETRES 8](#_Toc117792248)

[L’interprétabilité globale et locale DU MODÈLE (1 page maximum) 8](#_Toc117792249)

[LES LIMITES ET LES AMELIORATION POSSIBLES 10](#_Toc117792250)

[BIBLIOGRAPHIE ET NOTES 10](#_Toc117792251)

# INTRODUCTION

L’entreprise souhaite **mettre en œuvre un outil de “scoring crédit” pour calculer la probabilité**qu’un client rembourse son crédit, puis classifier la demande en crédit accordé ou refusé. L’**algorithme de classification** s’appuie sur des sources de données variées (données comportementales, données provenant d'autres institutions financières, etc.).

# DONNÉES

8 fichiers CSV contiennent les données clients.

Les 2 fichiers de base, application train et test, contiennent les mêmes features et peuvent être soumis à jointure pour être élaborées au même temps. Le fichier train contient une variable target (cible), qui est absente dans le fichier test. Avec la partie test on simulera un nouveau client qui demande un prêt. Le fichier train sera la base de l’entraînement du modèle.

Les autres 5 fichiers contiennent des informations bancaires et de prêts précédents.

Description des 8 fichers et des jointures :

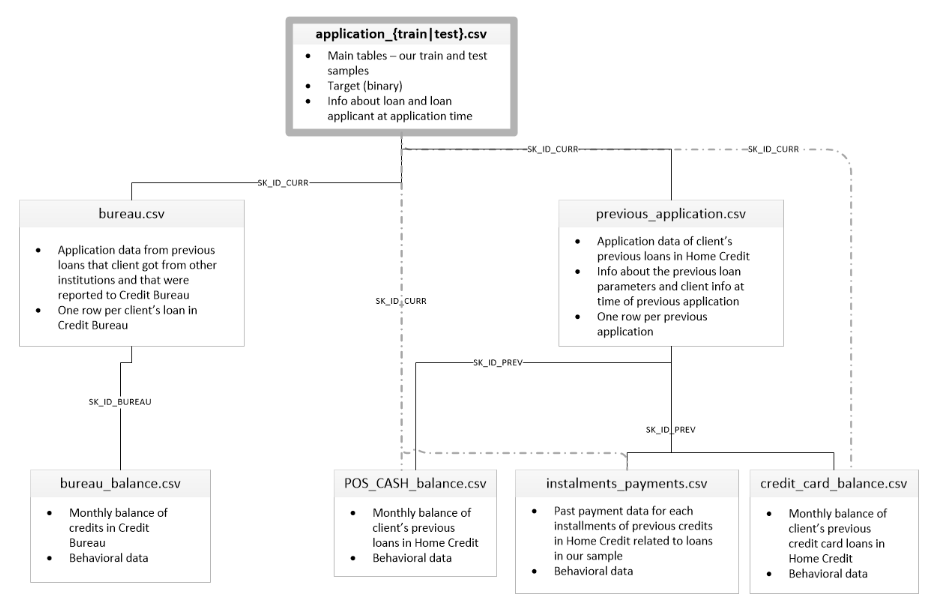


Figure 1

# LA MÉTHODOLOGIE D'ENTRAINEMENT DU MODÈLE

La première étape de l’élaboration du modèle de scoring a été le choix d’un kernel Kaggle existant pour rendre la préparation des données plus rapide.

## Kernel Kaggle

Le kernel conseillé par Openclassroom, celui de aguiars[[1]](#endnote-1), est très élégant et a servi de base au 7ème placé dans le concours Kaggle. Mais il est assez pauvre en commentaires et trop riche en features. Il a été le point de départ de mon data engineering. Pour l’adapter à des ressources informatiques plus limitées, j’ai réduit le nombre de features crées par agrégation et supprimé les features très corrélés[[2]](#endnote-2).

## Feature Engineering

Les techniques de feature engineering appliquées aux données ont été :

**Outliers** : Transformation des valeurs aberrantes en NaN.

Features catégorielles :

**NaN ad hoc** : la valeur XNA, qui correspond à NaN pour les features catégorielles, est transformé en Nan.

**Features binaires** : les features à 2 valeurs sont transformées en 0 et 1.

**Features quantifiables** : les features quantifiables, par exemple le niveau d’études, qui peut être réinterprété comme années d’études, sont transformé en numériques.

**Features à valeur dominante** : les features qui ont un nombre négligeable di valeurs minoritaires, sont supprimées.

**Hyponymes :** Réduction du nombre de valeur en les substituant par un hyperonyme commun (par exemple, ‘loan’ à la place de 'Car loan' et 'Microloan').

**One hot encoding**

**Composition et dérivation de features et drop :** par exemple df['INCOME\_CREDIT\_RATIO'] = df['AMT\_INCOME\_TOTAL'] / df['AMT\_CREDIT']

**Agrégations par ID** : plusieurs lignes avec le même ID sont agrégées en calculent la moyenne ou la somme pour les features, ou, moins souvent, les valeurs uniques, le nombre, le max ou le min.

**Ré-Agrégations par valeurs Accepted et Rejected et puis ID.**

**Drop des features similaires et avec une corrélation forte :** en s’appuyant à un heatmap.

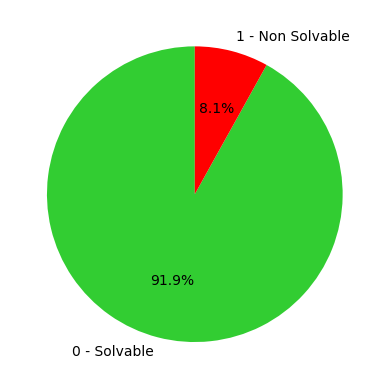
**Permutation feature importance** : drop des features moins importants.

**Drop des colonnes avec plus de 75% de nans**

Le dataframe final contient 114 features et 307507 individus.

## Resampling

La distribution des TARGET 0 (pour « solvable ») et 1 (pour « à risque de défaut ») est très déséquilibrée :

Figure 2

Plusieurs tests de resampling ont été fait tout au long du en utilisant la librerie Imblearn et la technique SMOTE pour rééquilibrer les deux classes en créant des nouveaux individus de classe 1, mais on n’a pas obtenu des améliorations significatives. L’utilisation de SMOTE demande un dataframe sans NaNs, donc à chaque fois, la pipeline est :

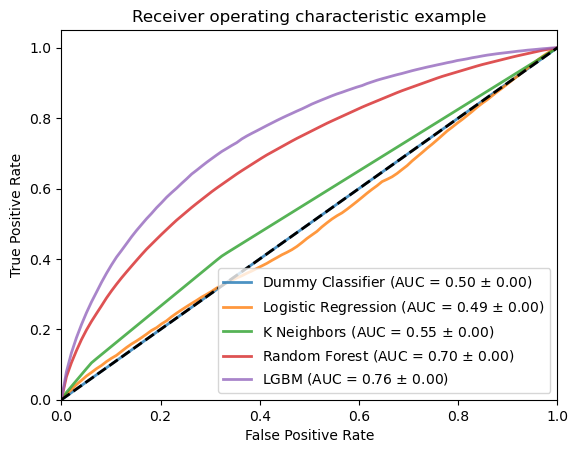
smote\_pipeline=imbPipeline([  
 ('imputer', SimpleImputer(strategy='median')),  
 ('sampling', SMOTE()),  
 ('classification', <classifier> )])

## Preprocessing

Pour préparer l’étape successive de choix du classifier, j’ai dû imputer les valeurs manquantes par la valeur médiane du feature et utiliser le scaler RobustScaler. Ce preprocessing n’est pas nécessaire pour le classifieur choisi.

## Modèle LGBM

Après étude de plusieurs modèles de classification sur la métrique roc auc[[3]](#endnote-3), avec 6 kfolds sur un split de   
données train-validation de 67%-33%, mon choix s’est porté sur le LGBM, tout comme dans le kernel choisi.

 Figure 3

## Calibration des hyperparamètres pour le score auc

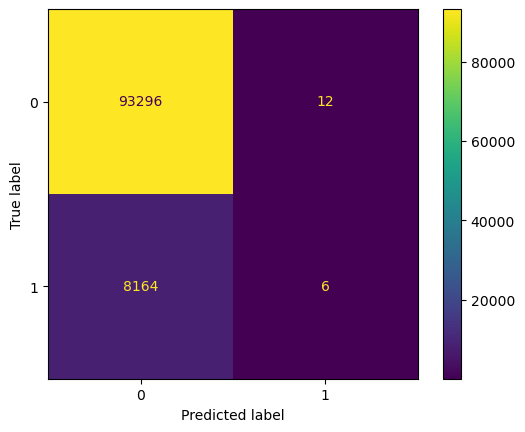
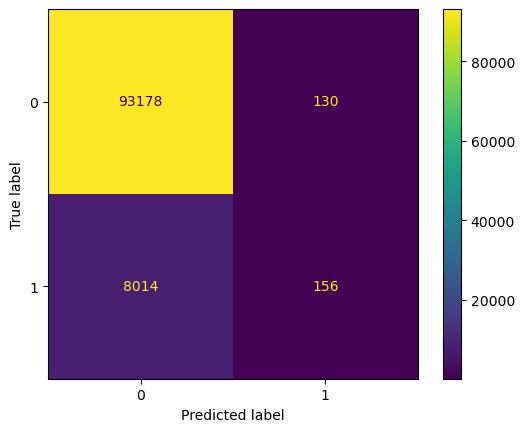
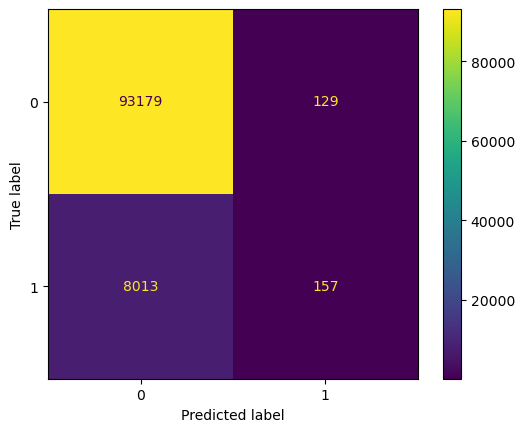
Pour la recherche des meilleurs hyperparamètres j'utilise HalvingGridSearch[[4]](#endnote-4) sur le hyperparamètres utilisés dans les différentes versions du kernel choisi (qui avaient été sélectionnés par Bayesian optimisation). Et si les hyperparamètres par défaut sont très différents, j'ajoute la valeur par défaut à la gridsearch. Un choix dicté par le temps et les ressources informatiques, qui évidemment limite la qualité des résultats.

En plus, le dataframe complet est inutilisable avec les ressources dont je dispose : même dans Google Colab avec GPU, après 12 heure de calcul la Gridsearch est arrêtée. J'utilise pourtant un sousensemble du dataframe. La stratégie de validation par défaut est le stratified Kfolds, sur 3 folds. Avec un petit sousensemble du dataframe je finis par avoir des folds de test sans classe 1. Donc, j’ai dû utiliser des sousensembles un peu plus importants.

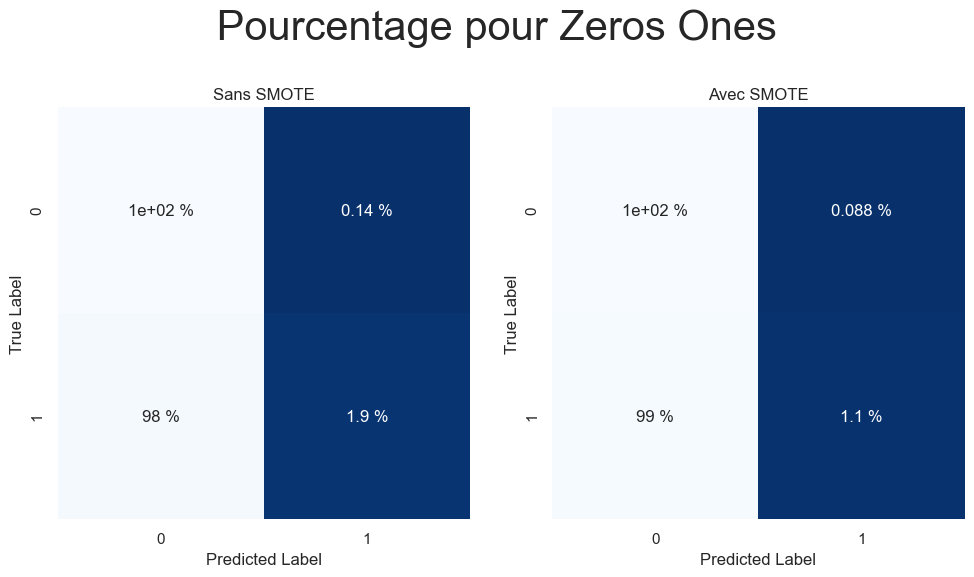
## La matrice de confusion avec AUC

La matrice de confusion montre une amélioration (modeste) de la classification avec le tuning des hyperparamètres. SMOTE ne donne pas de résultats vraiment meilleurs.

Optimisé Baseline SMOTE



Le modèle n’arrive à bien prédire que le 2% des clients non solvables.



# LA FONCTION COÛT MÉTIER, L'ALGORITHME D'OPTIMISATION ET LA MÉTRIQUE D'ÉVALUATION

La métriques auc n’est pas complètement satisfaisante pour notre classification business : il nous faut une classification qui donne des poids différents aux classes identifiées correctement ou pas.

## La fonction coût métier

Chaque résultat dans une matrice de confusion a un coût[[5]](#endnote-5) ou un bénéfice pour notre banque :

* TP: Clients non solvables identifiés comme non solvables. True one. Coût 0, bénéfice 0. La banque ni gagne ni risque rien.
* TN: Clients solvable identifiés comme solvables. True zero. Bénéfice 1. La banque gagne sur les intérêts payés par les clients.
* FP: Clients solvable identifiés comme NON solvables. False one. Coût 1. La banque perd un bon client.
* FN: Clients NON solvable identifiés comme solvables. False zero. Coût 10. La banque risque une perte d'argent. La perte d'argent est supérieure parce qu'il s'agit de toute la somme prêtée.

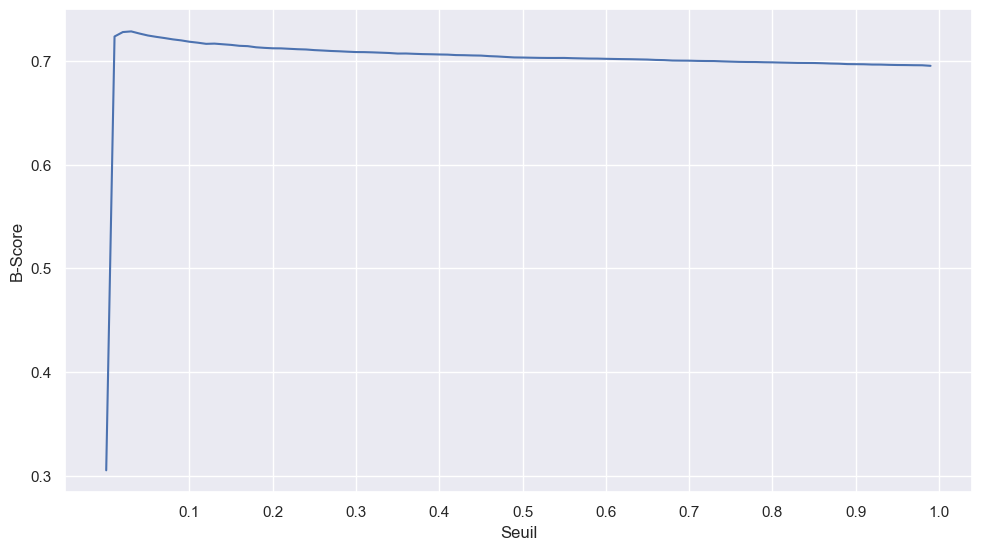
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | Predicted | | |
| 0 | 1 | |
| True | 0 | TN: 1 | | FP: -1 |
| 1 | FN: -10 | | TP: 0 |

L’optimisation des hyper-paramètres par HalvingGridsearch est relancée avec cette nouvelle métrique.

## Choix du seuil optimal

Dans un classifier, l’appartenance à une classe binaire est calculée sur une probabilité entre 0 et 1, avec un seuil à 0,5. Or, ce seuil peut ne pas résulter optimal pour des données très déséquilibrées.

Pour repérer le seuil optimal selon ma métrique métier, on calcule le coût global pour chaque seuil entre 0 et 1, avec des intervalles de 0,01 :



# L’interprétabilité globale et locale DU MODÈLE (1 page maximum)

## La feature importance

L’analyse de l’importance des variables nous permet de visualiser sur quelles variables s’appuie le modèle pour effectuer ses prédictions.

# LES LIMITES ET LES AMELIORATION POSSIBLES

Un certain nombre de pistes sont parcourables pour améliorer la performance du modèle :

1. Un features engineering plus poussé : le kernel Kaggle de aguiars, auquel je me suis inspirée pour le projet, avait un auc plus élevé sur le classifier sans tuning et encore plus élevé après tuning des hyperparamètres . Il a pu garder plus de features (avec du overfitting, surement). Sa version qui est arrivée 7e au concours Kaggle était encore plus riche de features. D’autre part, d’autres techniques[[6]](#endnote-6) peuvent être tentées pour réduire le nombre des features. Troisièmement, le feedback d’un expert est indispensable pour améliorer les KPI.
2. Plus de ressources pour le tuning des hyperparamètres. Une première tentative de tuning sur Colab avec GPU a dépassé les 12 heures et a été arrêtée par Google.
3. Tester l’optimisation bayesienne, comme dans le kernel d’aguiar.
4. Mes tentatives avec SMOTE n’ont pas donné des résultats intéressants. Le paramètre is\_imbalance de LightGBM, non plus. Je n’ai pas tenté d’autres techniques ou d’autres hyperparamètres pour les données déséquilibrées.
5. Ma métrique métier n’est pas validée par un expert. Elle pourrait être plus fine et quantifier le risque en fonction du prêt.

# BIBLIOGRAPHIE ET NOTES

1. https://www.kaggle.com/code/jsaguiar/lightgbm-with-simple-features?scriptVersionId=6025993 [↑](#endnote-ref-1)
2. https://www.kaggle.com/code/willkoehrsen/start-here-a-gentle-introduction [↑](#endnote-ref-2)
3. La courbe ROC (Receiver Operating Characteristic) croise le taux de TP avec le taux de FP. La courbe ROC de notre classifier LGBM comparé aux autres est présenté dans la figure 3. Le score mesure l’aire sous la courbe (Area Under the Curve ou AUC). Un classifieur parfait aurait un score AUC égal à 1. Un classifieur dummy a un auc de 0,5. [↑](#endnote-ref-3)
4. <https://neptune.ai/blog/hyperparameter-tuning-in-python-complete-guide>

   “Scikit-learn also provides the [**HalvingGridSearchCV**](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.HalvingGridSearchCV.html#sklearn.model_selection.HalvingGridSearchCV)  estimator that can be used to search a parameter space using successive halving. Successive halving (SH) is like a tournament among candidate parameter combinations. SH is an iterative selection process where all candidates (the parameter combinations) are evaluated with a small amount of resources at the first iteration. Only some of these candidates are selected for the next iteration, which will be allocated more resources. For parameter tuning, the resource is typically the number of training samples, but it can also be an arbitrary numeric parameter such as n\_estimators in a random forest. Only a subset of candidates ‘survive’ until the last iteration. These are the candidates that have consistently ranked among the top-scoring candidates across all iterations. Each iteration is allocated an increasing amount of resources per candidate.” https://scikit-learn.org/stable/modules/grid\_search.html [↑](#endnote-ref-4)
5. Les couts choisis sont arbitraires. Perdre un bon client peut être considéré une perte symbolique plus que réelle. [↑](#endnote-ref-5)
6. https://www.analyticsvidhya.com/blog/2020/10/feature-selection-techniques-in-machine-learning/ [↑](#endnote-ref-6)