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 :

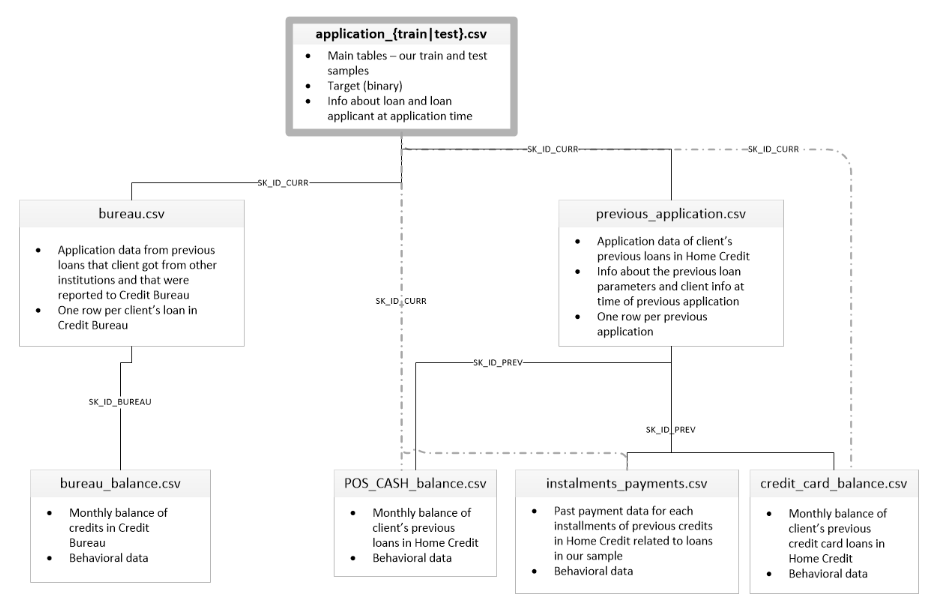


Figure 1

# 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

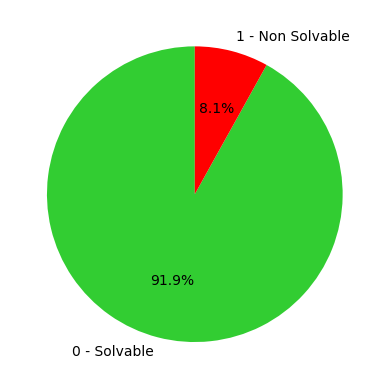
**Ré-Agrégations par valeurs Accepted et Rejected et puis ID**

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

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

## 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 :

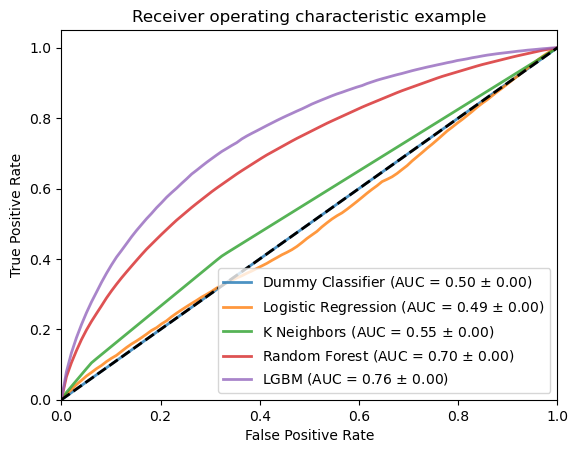
smote\_pipeline=imbPipeline([  
 ('imputer', SimpleImputer(strategy='median')),  
 ('sampling', SMOTE()),  
 ('classification', <classifier> )])

## 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

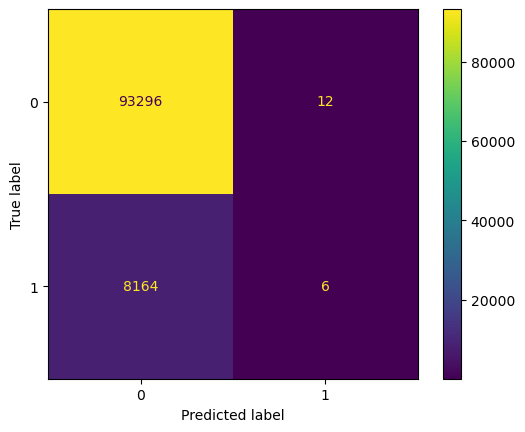
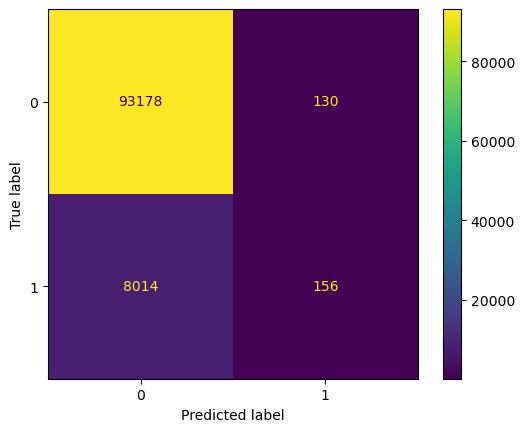
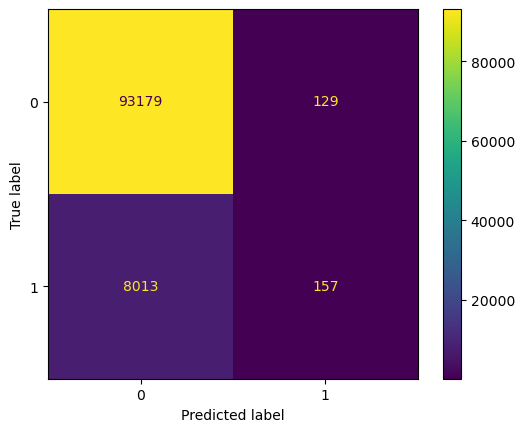
Pour la recherche des meilleurs hyperparamètres j'utilise HalvingGridSearch[[4]](#endnote-4) sur le hyperparamètres utilisés dans les différentes versions du kernel choisi (qui avaient été sélectionnés par Bayesian optimisation). Et si les hyperparamètres par défaut sont très différents, j'ajoute la 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 des sousensembles 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.

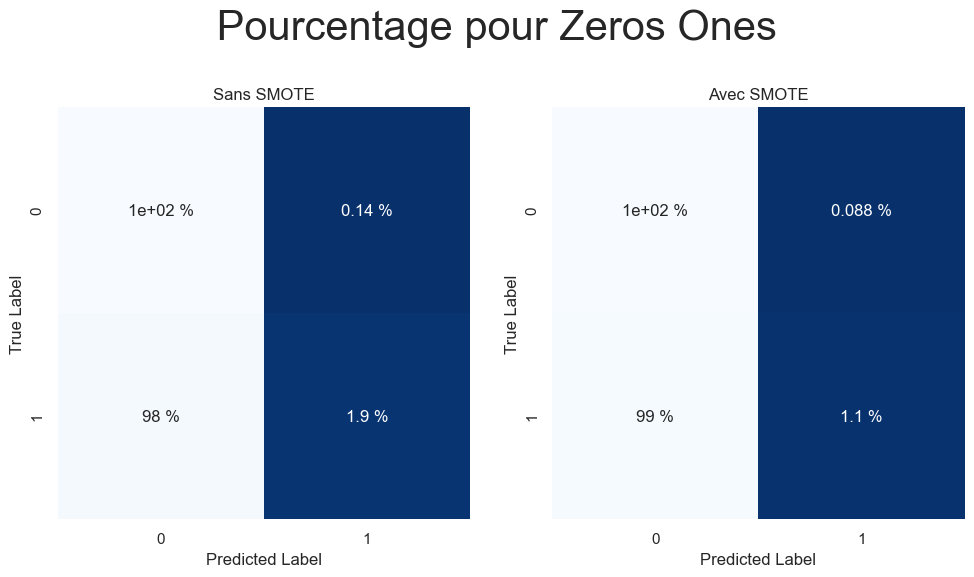
## La matrice de confusion avec AUC

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

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

# L’interprétabilité globale et locale DU MODÈLE

# LES LIMITES ET LES AMELIORATION POSSIBLES

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

1. Un features engineering plus poussé : le kernel de aguiars avait un auc plus élevé sur le classifier sans tuning et ancore plus élevé après tuning. 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. J’étais limitée, entre autres, par les limites du déploiement du dashboard sur une plateforme gratuite.
2. Le feedback d’un expert pour améliorer les KPI.
3. 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.
4. Mes tentatives avec SMOTE n’ont pas donné des résultats intéressants. Je n’ai pas tenté autres techniques pour les données déséquilibrées.
5. Ma métrique métier n’est pas validée par un expert.

Nous avons pu constater tout au long de ce document que les performances du modèle ne sont pas bonnes. Pour résumé : Au mieux, notre modèle peut trouver 40% des classes 1, et lorsqu’il en prédit une, il a raison à 36%.

# 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) 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> [↑](#endnote-ref-4)