|  |
| --- |
| Modèle de Scoring : Prédiction sur la probabilité de faillite d'un client de façon automatique |

|  |
| --- |
| **RESUME**  Mise en place d’un modèle de classification sur un jeu de données fortement équilibré. Ce document détaille la construction de ce modèle, l’optimisation, l’analyse des résultats et les métriques utilisées. |

Table des matières

[INTRODUCTION 3](#_Toc28939057)

[REFERENCES 3](#_Toc28939058)

[DONNEES D’ENTREES 4](#_Toc28939059)

[FEATURE ENGINEERING 5](#_Toc28939060)

[1. Kernel kaggle 5](#_Toc28939061)

[2. RESAMPLING 6](#_Toc28939062)

[3. PREPROCESSING 7](#_Toc28939063)

[CONSTRUCTION MACHINE LEARNING 8](#_Toc28939064)

[1. Modèle XGBOost 8](#_Toc28939065)

[ACCURACY, PRECISION, RECALL, F1 SCORE 8](#_Toc28939066)

[LA MATRICE DE CONFUSION 8](#_Toc28939067)

[COURBE ROC ET SCORE AUC 9](#_Toc28939068)

[FEATURES IMPORTANCE 10](#_Toc28939069)

[2. OPTIMISATION XGBOOST 11](#_Toc28939070)

[HYPERPARAMETRES 11](#_Toc28939071)

[MESURES DE PERFORMANCES 12](#_Toc28939072)

[ACCURACY, PRECISION, RECALL, F1 SCORE 12](#_Toc28939073)

[AXES D’AMELIORATIONS A aPPORTER 15](#_Toc28939074)

# INTRODUCTION

L’entreprise souhaite **développer un modèle de Scoring de la probabilité de défaut de paiement du client** pour étayer la décision d'accorder ou non un prêt à un client potentiel en s’appuyant sur des sources de données variées (données comportementales, données provenant d'autres institutions financières, etc.).

A partir d’un kernel Kaggle existant, qui a permis de faciliter la préparation des données nécessaires, nous avons procédé à l’élaboration du modèle de Scoring.

# REFERENCES

[1] Lien détail du projet 7 OPENCLASSROOMS : <https://openclassrooms.com/fr/projects/632/assignment>

[2] Lien Kernel Kaggle : <https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction>

[3] Lien téléchargement des données : <https://www.kaggle.com/c/home-credit-default-risk/data>

[4] Librairie Python « Imblearn » pour équilibrage des données : <https://imbalanced-learn.readthedocs.io/en/stable/user_guide.html>

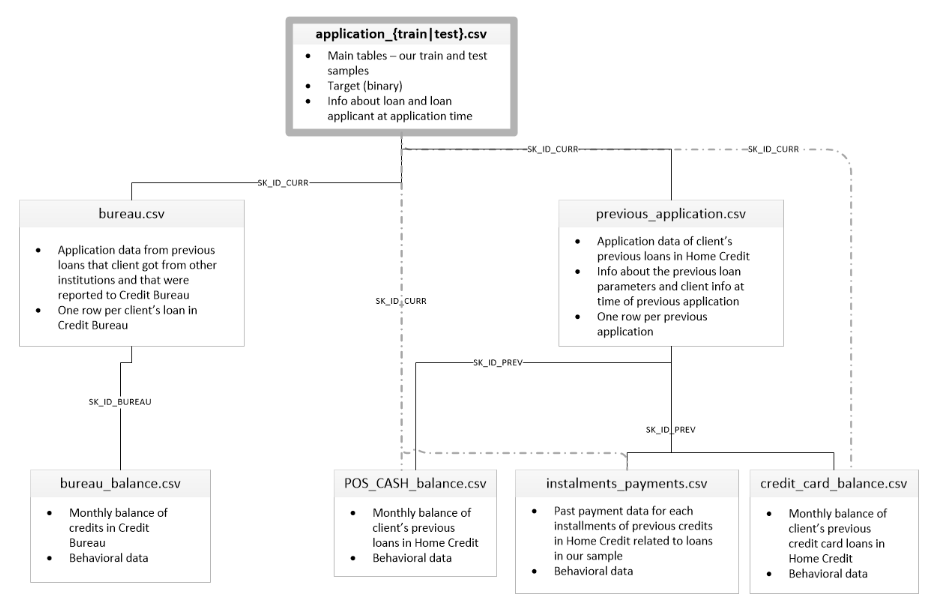
# DONNEES D’ENTREES

L’entreprise met à disposition 7 fichiers CSV contenant des données spécifiques à certains paramètres.

Nous disposons d’une base de données nommée « train », qui nous a servi à entraîner notre modèle. Cette base contient une variable « cible ».

Nous avons aussi à disposition une base de données « test ». Cette ne nous a pas été utile à la construction et à l’entraînement du modèle. Son utilisation a été extérieure au sujet de la présente note technique.

**Description des liens des différentes base de données :**



*Figure 1*

# FEATURE ENGINEERING

Dans cette partie, nous présentons brièvement le Kernel Kaggle que nous avons récupéré, afin de comprendre la base qui a été réalisé et sur laquelle repose notre modèle.

## Kernel kaggle

Avant toute chose, il est bon de savoir que les travaux réalisés dans ce Kernel ont été fait uniquement sur la même base de données, à savoir « train.csv ».

## RESAMPLING

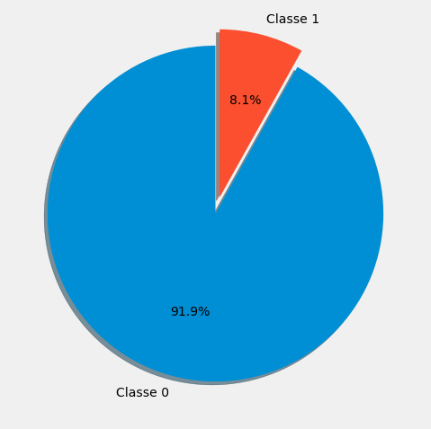
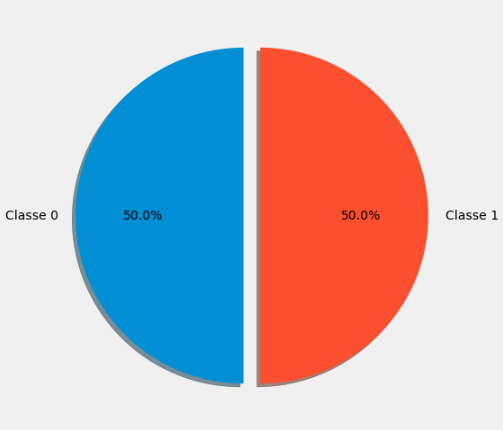
Le principal problème dans notre cas de figure se situe dans le déséquilibre des *targets*. Ici, nous avons à faire à une classification binaire dans laquelle la classe 0 représente les personnes qui ont payé leur crédit, et la classe 1, les personnes ayant rencontré des problèmes pour rembourser leur prêt.

Afin d’éviter de construire un modèle ne prédisant que la classe majoritaire, il convient d’entraîner ce modèle sur un jeu de données équilibré. Pour cela nous avons utilisé une librairie Python nommée *Imblearn [4]*.

Technique utilisée dans la librairie *Imblearn* [4] : **Random Under Sampler**

Il existe plusieurs techniques de sampling de données. Parmi les plus efficaces, on trouve le Tomek links pour l’under-sampling et le SMOTE pour l’over-sampling. Dans notre cas, dans un souci d’optimisation de temps de calculs, nous avons utilisé le Random Under Sampler qui s’avère le plus rapide.

Sur la figure 2 ci-dessous, on constate le déséquilibre initial entre les deux classes. Tandis que sur la figure 3, nous avons rééquilibré parfaitement les 2 classes.

*Figure 2*

*Figure 3*

**Nombre d’individus dans chaque classe avant et après rééquilibrage des données :**

|  |  |  |
| --- | --- | --- |
|  | **Classe 0** | **Classe 1** |
| **Avant Rééquilibrage** | 226038 | 19970 |
| **Après Rééquilibrage** | 24825 | 24825 |

## PREPROCESSING

Pour rappel, l’entraînement du modèle se fait uniquement sur le fichier « train.csv ». Le preprocessing est constitué de deux traitements :

* Une transformation des données comprenant :
  + Une imputation des valeurs manquantes créées lors tu traitement des outliers.
  + Une application d’un MinMaxScaler sur le dataset.
* Un échantillonnage du dataset. Cette partie a été réalisée avec la méthode « Train\_Test\_Split » de Scikit-learn. Avec un ratio de 80% pour les données d’entraînement et 20% pour les données de tests.

*Figure 4*

# CONSTRUCTION MACHINE LEARNING

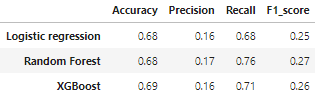
## Modèle XGBOost

Après étude de plusieurs modèles de classification, notre choix s’est porté sur le XGBoost. Voici les premiers résultats obtenus pour la première tentative de prédiction.

### ACCURACY, PRECISION, RECALL, F1 SCORE

**PRINCIPE :** Les deux métriques qui nous intéressent ici sont la *Précision*, et le *Recall* :

* La Précision : Ce coefficient détermine que, quand le classifieur déclare que la prédiction est un 1, il a raison à X%.
* Le Recall : Ce coefficient détermine le pourcentage de détection des 1 du classifieur.

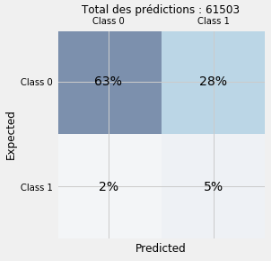


*Figure 5*

**ANALYSE :** Les 3 modèles présentent des résultats similaires, et ceux-ci ne sont pas bons. On pourrait traduire ces résultats en disant que notre modèle arrive à trouver 75% des classes 1, mais que, quand il prédit une classe 1, il n’a raison que dans 15% des cas.

### LA MATRICE DE CONFUSION

**PRINCIPE :** La matrice de confusion consiste à compter le nombre de fois où des observations de la classe 0 ont été rangées dans la classe 1. Par exemple, si nous voulons connaître le nombre de fois où le classifieur à bien réussi à classer une classe 1, on examinera la cellule à l’intersection de la ligne 1 et de la colonne 1.



*Figure 6*

**39040**

**17608**

**1563**

**3292**

**TN**

**FP**

**FN**

**TP**

**Total de classes 0 réelles**

**56648**

**Total de classes 1 réelles**

**4855**

**ANALYSE :** Nous distinguons quatre catégories dans la matrice de confusion :

1. Les True Negatives (TN) : L’intersection de la ligne 0 avec la colonne 0. Ce sont des individus représentant un gain pour l’entreprise.
2. Les False Negatives (FN) : L’intersection de la ligne 1 avec la colonne 0. Ce sont des individus ayant comme valeur réelle 1 mais que le modèle à prédit en 0. Ces individus sont potentiellement un risque pour l’entreprise.
3. Les False Positives (FP) : L’intersection de la ligne 0 avec la colonne 1. Ce sont des individus ayant comme valeur réelle 0 mais que le modèle à prédit en 1. Ces individus pourraient être un risque pour l’entreprise.
4. Les True Positives (TP) : L’intersection de la ligne 1 avec la colonne 1. Ce sont des individus étant à risque pour l’entreprise.

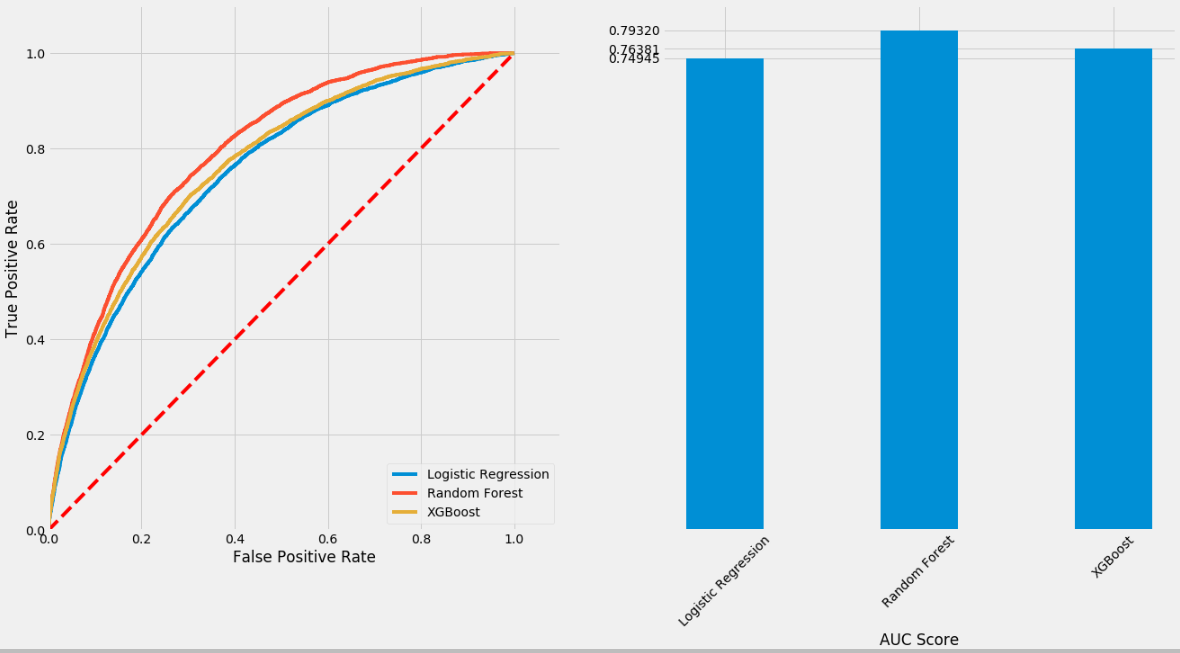
Notre modèle idéal serait de retrouver 100% de TP, car ce sont les individus qui ne remboursent pas leur prêt. C’est aussi la catégorie la plus difficile à prédire car étant minoritaire par rapport à la catégorie *0 (Cf. FEATURE ENGINEERING – Partie 2).*

### COURBE ROC ET SCORE AUC

**PRINCIPE :** La courbe ROC (Receiver Operating Characteristic) est un outil communément utilisé avec les classifieurs binaires. Elle croise le taux de TP avec le taux de FP.

Sur la figure 7, la ligne en pointillée représente la courbe ROC d’un classifieur purement aléatoire. Un bon classifieur s’en écarte autant que possible (vers le coin supérieur gauche).

Une autre façon de comparer des classifieurs consiste à mesurer l’aire sous la courbe (Area Under the Curve ou AUC). Un classifieur parfait aurait un score AUC égal à 1, tandis qu’un classifieur purement aléatoire aurait un score AUC de 0.5.



*Figure 7*

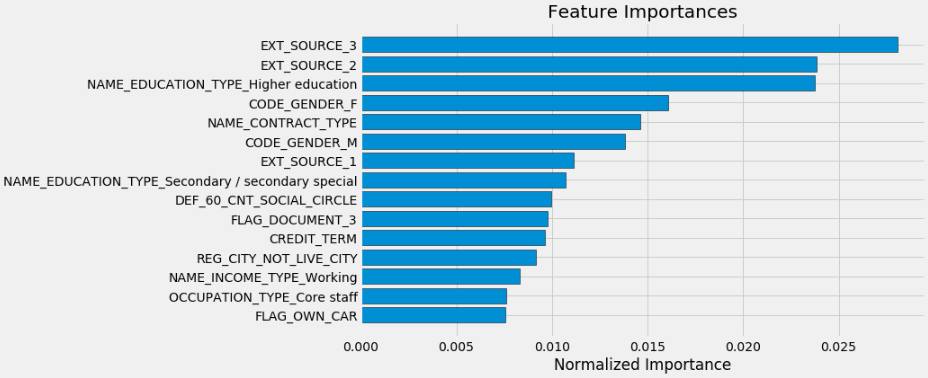
**ANALYSE :** On constate que les performances des modèles ne sont pas bonnes. Pour rappel, nous cherchons une courbe ROC se rapprochant le plus du coin supérieur gauche du graphique.

De plus, nos trois modèles testés ici présentent des résultats plus ou moins similaires.

### FEATURES IMPORTANCE

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

Sur la figure 8, l’importance des variables se lit en pourcentage (En multipliant les valeurs des abscisses par 100).



*Figure 8*

## OPTIMISATION XGBOOST

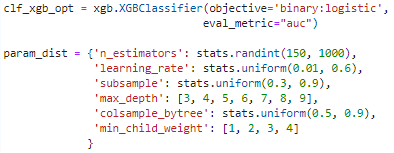
Notre première étude de prédictions nous montre que nous avons des résultats mauvais. Cependant nous pouvons essayer d’optimiser le modèle en jouant sur ses hyperparamètres. C’est ce que nous verrons dans cette partie.

### HYPERPARAMETRES

Le modèle XGBoost possède trois familles d’hyperparamètres :

1. General Parameters : Pour régler les préférences du modèle
2. Booster Parameters : Pour régler le modèle à chaque étape de prédiction (Arbre / Régression)
3. Learning Task Parameters : Pour régler le type de prédiction que l’on souhaite

Il existe un grand nombre d’hyperparamètres pour ce modèle, et, à ce stade, nous n’en avons utilisé que quelques-uns :



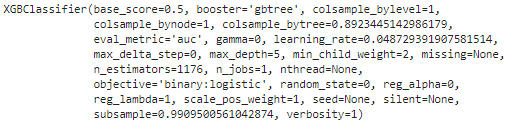
*Figure 9*

La figure 9 représente les hyperparamètres utilisés pour optimiser notre modèle :

1. N\_estimators.
2. Learning\_rate.
3. Subsample : Indique la fraction des observations devant être des échantillons aléatoires pour chaque arbre. Des valeurs faibles rendent l'algorithme plus conservateur et empêchent le sur-ajustement, mais des valeurs trop petites peuvent conduire à un sous-ajustement.
4. Max\_depth : Profondeur maximale d'un arbre. Utilisé pour contrôler le sur-ajustement car une profondeur plus élevée permettra au modèle d'apprendre des relations très spécifiques à un échantillon particulier.
5. Colsample\_bytree.
6. Min\_child\_weight : Définit la somme minimale des poids de toutes les observations requises chez un enfant. Utilisé pour contrôler le sur-ajustement. Des valeurs plus élevées empêchent un modèle d'apprendre des relations qui peuvent être très spécifiques à l'échantillon particulier sélectionné pour un arbre. Des valeurs trop élevées peuvent conduire à un sous-ajustement, par conséquent, il doit être réglé à l'aide de CV.

Nous avons utilisé la fonction RandomizedSearchCV pour trouver la meilleure itération possible pour notre modèle.

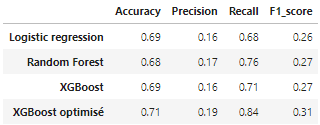
Meilleurs paramètres trouvés :



*Figure 10*

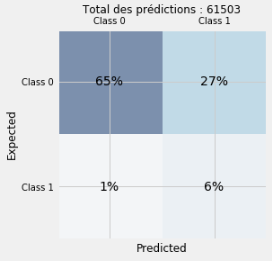
### MESURES DE PERFORMANCES

### ACCURACY, PRECISION, RECALL, F1 SCORE



*Figure 11*

#### MATRICE DE CONFUSION



**40032**

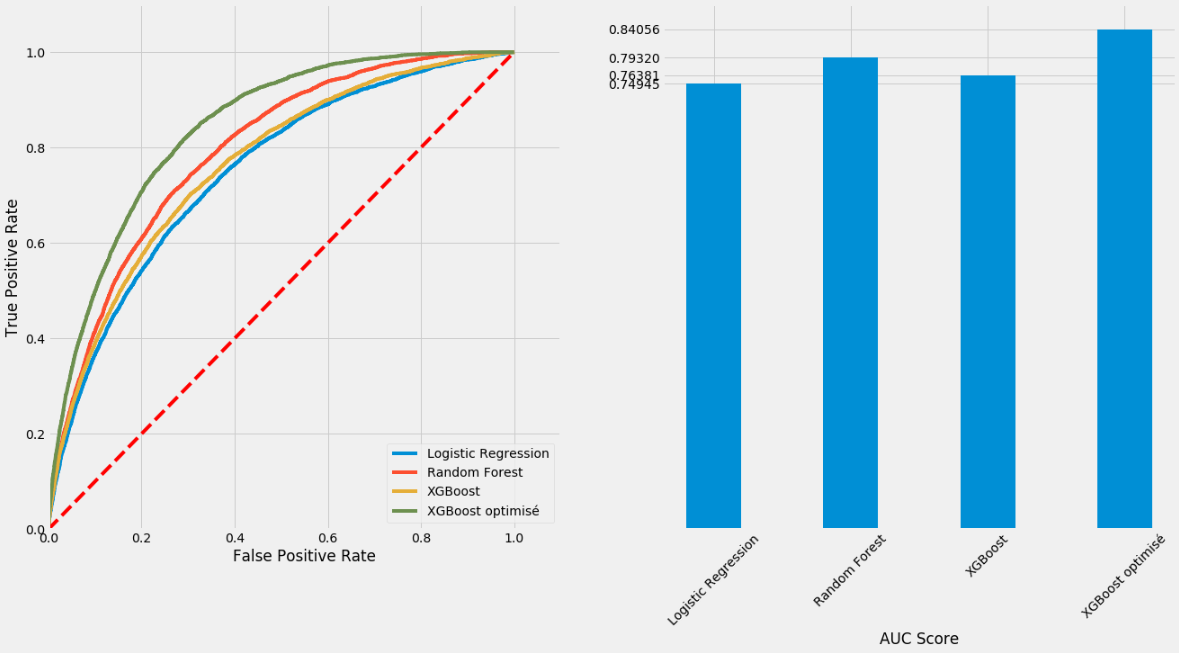
**16616**

**877**

**3978**

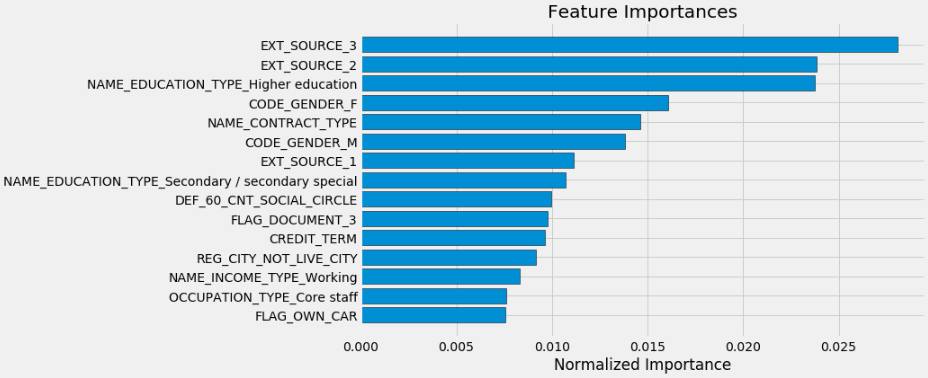
*Figure 12*

#### COURBE ROC ET SCORE AUC



*Figure 13*

#### FEATURES IMPORTANCE



*Figure 14*

#### COURBE PRECISION/RAPPEL

**PRINCIPE :** La courbe de Précision/Rappel nous permet de visualiser le meilleur compromis que l’on puisse avoir avec notre modèle. Le compromis Précision/Rappel se définit grâce au *threshold* (seuil de décision).

Explication du treshold :

*Le seuil de décision est une valeur que nous fixons et qui va limiter qu’une valeur appartient à la classe 0 ou à la classe 1.*

0

1

0.65

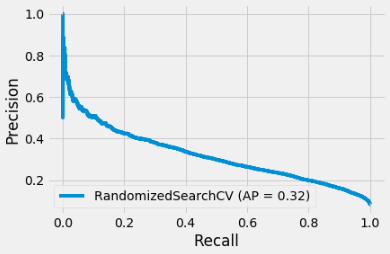
Threshold = 0.5

Threshold = 0.75

*Figure 15*

*Sur l’exemple de la figure 15, notre modèle fait une prédiction qu’un individu a une probabilité de 65% d’appartenir à la classe 1. Si nous fixons la valeur du threshold à 0.5, notre individu sera dans la classe 1. Par contre si nous fixons la valeur du threshold à 0.75, notre individu sera dans la classe 0.*

Sur la courbe de Précision/Rappel (Figure 16), nous pouvons voir que le meilleur compromis Précision/Rappel nous permettra d’obtenir un Rappel de 0.4 pour une précision de 0.36 environ. *Dans une future version de ce document, nous afficherons les valeurs de seuils correspondants sur ce même graphique.*



*Figure 16*

# AXES D’AMELIORATIONS A aPPORTER

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%.

Nous pouvons l’expliquer par le features engineering qui est à améliorer. En effet, le Kernel choisi est plutôt pauvre sur le traitement des données. Il ne se focalise que sur une seule table et ne crée pas beaucoup de variable qui peuvent être utiles à un modèle de classification comme des moyennes, des médianes, des écarts-types, et ça pour plusieurs features. Il existe peut-être un Kernel plus abouti qui permettra une meilleure performance prédictive au modèle. Sinon, prendre le temps de réaliser nous même notre feature engineering, ce qui nous permettra de bien comprendre nos données et ainsi construire un feature engineering adapté à notre besoin.