| Modèle de Scoring : Prédiction sur la probabilité de faillite d'un client de façon automatique  <https://github.com/IJBILI> |
| --- |

| **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](#_1fob9te)

[REFERENCES 3](#_3znysh7)

[DONNEES D’ENTREES 4](#_2et92p0)

[FEATURE ENGINEERING 5](#_tyjcwt)

[1.](#_3dy6vkm) Kernel kaggle 5

[2.](#_1t3h5sf) RESAMPLING 6

[3.](#_4d34og8) PREPROCESSING 7

[CONSTRUCTION MACHINE LEARNING 8](#_2s8eyo1)

[1. Modèles](#_17dp8vu)

[Average precession](#_3rdcrjn)

[LA MATRICE DE CONFUSION 8](#_26in1rg)

[COURBE ROC ET SCORE AUC 9](#_lnxbz9)

[FEATURES IMPORTANCE 10](#_35nkun2)

[2.](#_1ksv4uv) OPTIMISATION XGBOOST 11

[HYPERPARAMETRES 11](#_44sinio)

[MESURES DE PERFORMANCES 12](#_2jxsxqh)

[ACCURACY, PRECISION, RECALL, F1 SCORE 12](#_z337ya)

[AXES D'AMÉLIORATIONS À aPPORTER 15](#_3j2qqm3)

# 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>

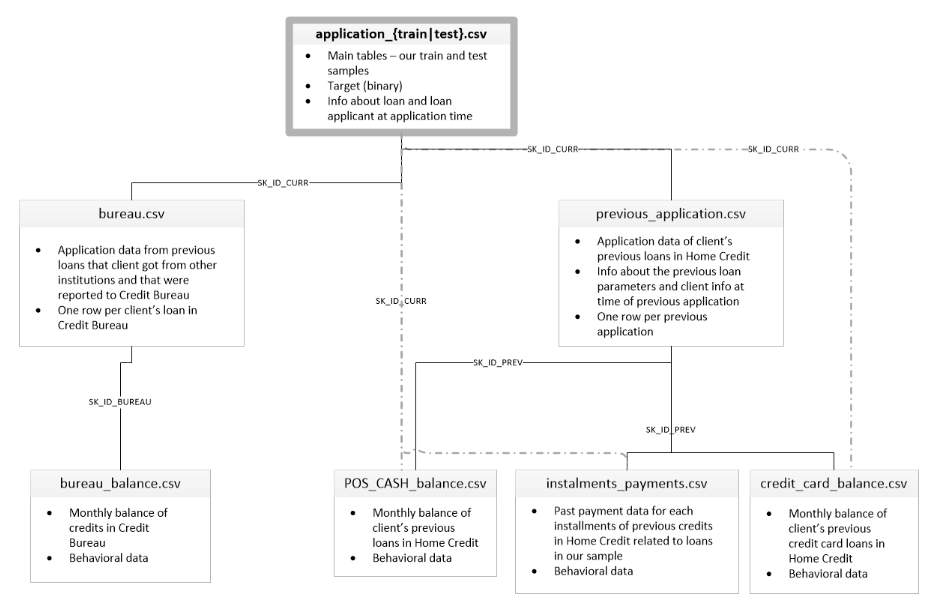
# DONNEES D’ENTREES

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

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

Nous avons aussi à disposition une base de données « application\_test ». Cela ne nous a pas été utile..

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



*Figure 1*

# FEATURE ENGINEERING

### 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.

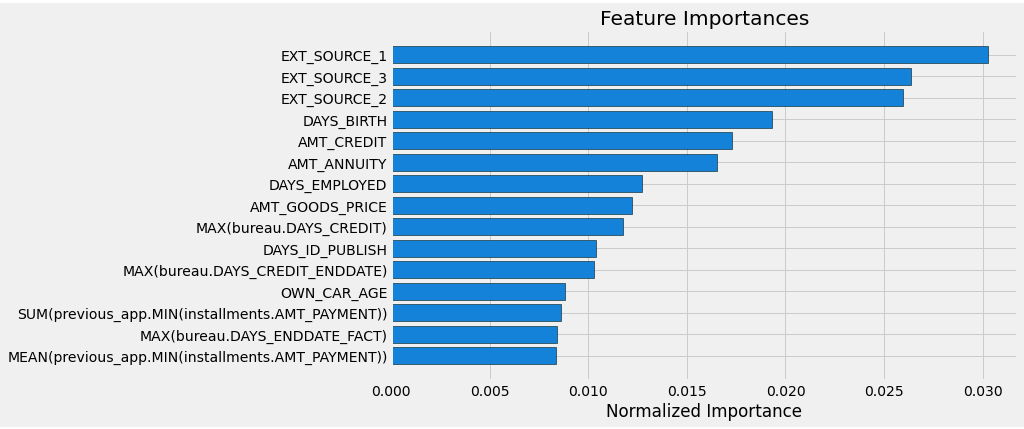
Features engineering est une étape très importante de l'apprentissage automatique. Features engineering fait

références au processus de conception des caractéristiques artificielles dans un algorithme.

Ces caractéristiques artificielles sont ensuite utilisées par cet algorithme afin d'améliorer ses performances ou

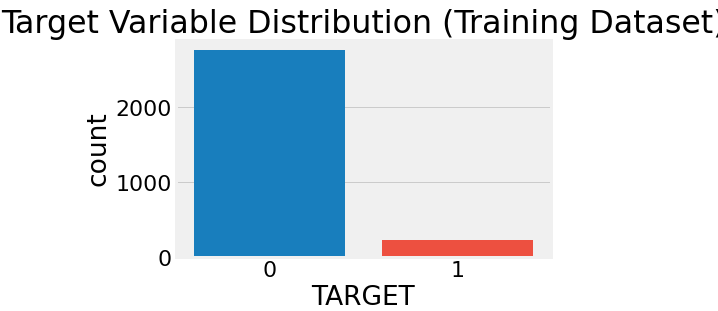
d'obtenir de meilleurs résultats.

Sur la figure, l’importance des variables se lit en pourcentage.



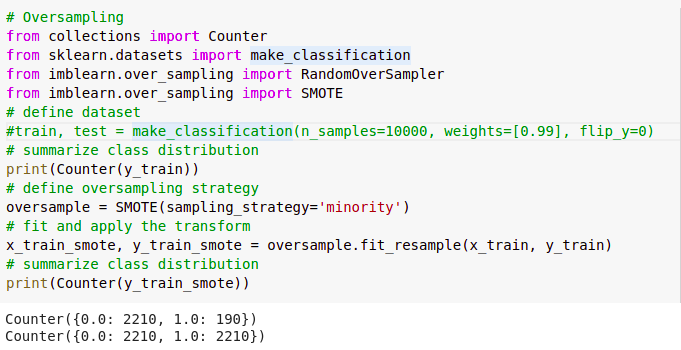
## RESAMPLING

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]*.

Pour les techniques de suréchantillonnage, SMOTE (Synthetic Minority Oversampling Technique) est considéré comme l'un des algorithmes d'échantillonnage de données les plus populaires et les plus influents dans le Machine Learning et l'exploration de données. Avec SMOTE, la classe minoritaire est sur-échantillonnée en créant des exemples synthétiques plutôt qu'en suréchantillonnage avec remplacement. Utilisation de : from imblearn.over\_sampling (imbalanced-learn Python library SMOTE class)



## PREPROCESSING

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

* Une transformation des données comprenant :
  + Une imputation des valeurs manquantes créées lors du traitement des outliers.
  + Une application d’un StandardScaler 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.





# CONSTRUCTION MACHINE LEARNING

## Algorithmes utilisés pour construire le modèle de scoring

La modélisation du scoring qui donne une prédiction sur la probabilité de faillite d'un client a été implémentée par classification. La classification consiste à identifier les classes d'appartenance de nouveaux objets à partir d'exemples antérieurs connus. Dans le contexte métier du projet, la classification est binaire représentée par une variable de sortie à deux classes, à savoir acceptation du crédit ou refus du crédit. En opposition à un problème multi-classes, c’est-à-dire une variable cible peut-être représentée par plusieurs classes.

### 

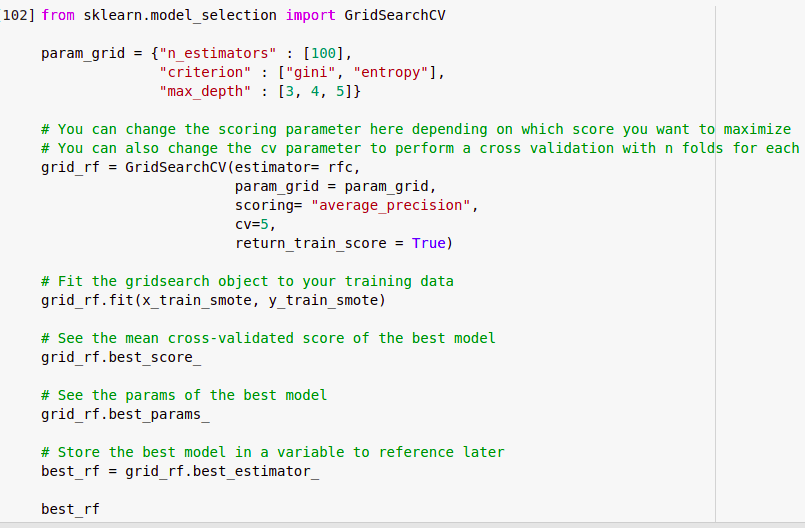
Toute la force du Machine Learning réside dans la diversité des approches utilisées. Plus le nombre de méthodes testées est élevé, plus il sera possible de trouver le meilleur algorithme permettant de répondre à la problématique, ici trois algorithmes ont été testés. Une baseline a également été fixée par Dummy classifier, un modèle classique.

Pour le choix du métrique, on a utilisé la métrique average precision.

**Optimisation des modèles :**

Le but de cette partie est de trouver l'ensemble d'arguments de configuration de modèle spécifiques qui entraînent les meilleures performances du modèle .

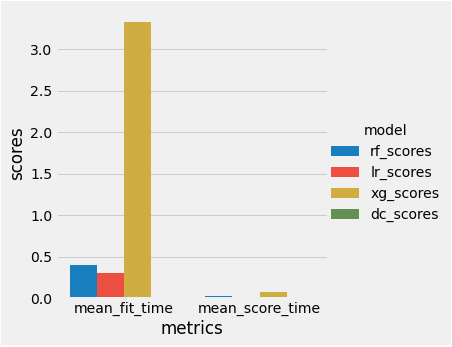
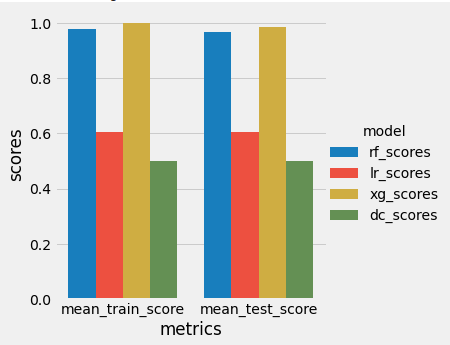
La méthode d'optimisation des hyperparamètres est basée sur GRID SEARCH CV.



Nous avons utilisé la fonction GridsearchCV pour trouver la meilleur itération possible pour le modèle Random Forest :

RandomForestClassifier(criterion='entropy', max\_depth=5, random\_state=42)

**Comparaison des modèles avec le métrique average precision :**

****

### **Conclusion :**

En comparant les performances de deux modèles de classification, on trouve que le random forest est plus performant suivant les différents scores.

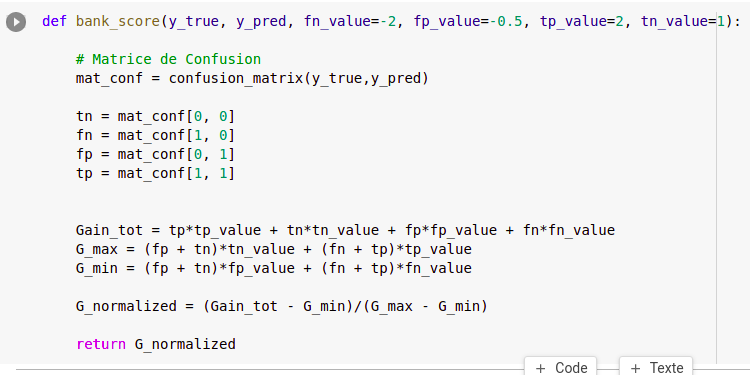
### 

### Métrique spécifique

La métrique Bancaire consiste à calculer le gain obtenu pour l'ensemble des individus du jeu de données. Pour cela, on va pénaliser le défaut (avec un gros poids correspondant aux potentiel pertes en capital) et bonifier un remboursement (faible poids correspondant aux intérêts générés). Les valeurs des poids sont les suivantes :

* fn\_value = -2
* tp\_value = 0
* tn\_value = 1
* fp\_value = -0.5

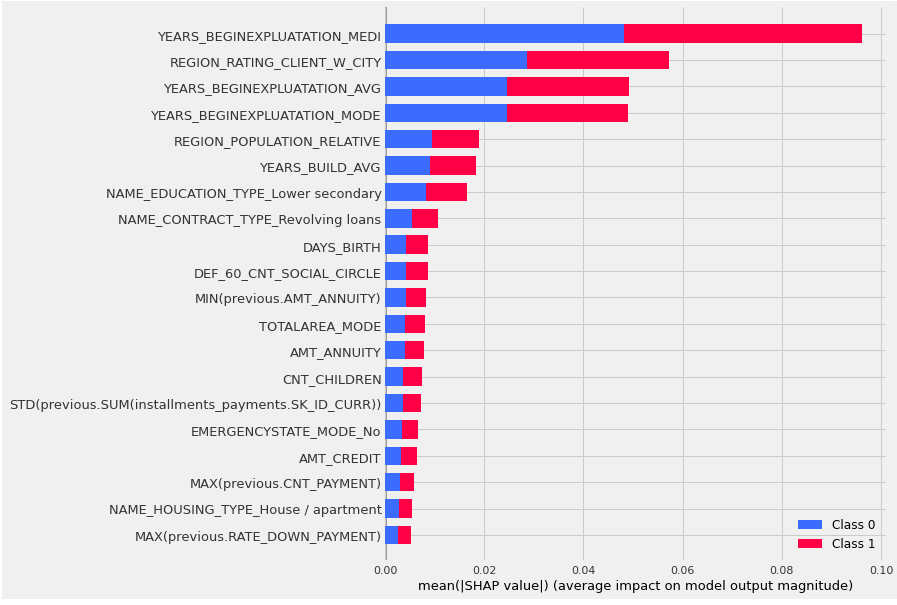
Les poids sont arbitraires et il est tout à fait possible de changer ces valeurs à la convenance de l'optique métier.

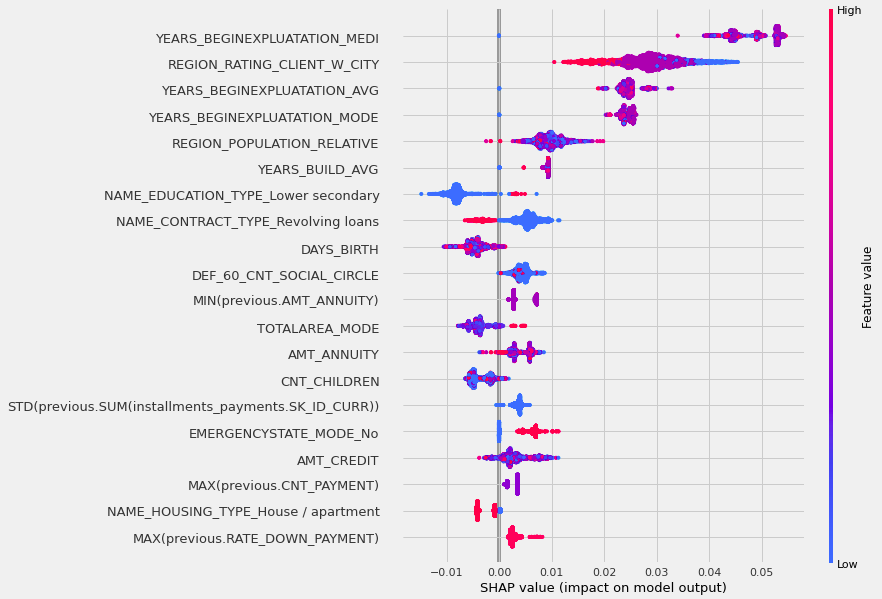


### 

### **L’interprétabilité globale du modèle avec shap :**

Importance globale des variables en utilisant les valeurs de Shap. Sur l’image de gauche, l’importance des variables est calculée en moyennant la valeur absolue des valeurs de Shap. Sur la droite, les valeurs de Shap sont représentées pour chaque variable dans leur ordre d’importance, chaque point représente une valeur de Shap (pour un exemple), les points rouges représentent des valeurs élevées de la variable et les points bleus des valeurs basses de la variable





### **L’interprétabilité locale du modèle avec lime :**

C’est une implémentation du concept de substitut, pour une instance unique.

On génère un échantillon de données à partir de l’instance qui nous intéresse en :

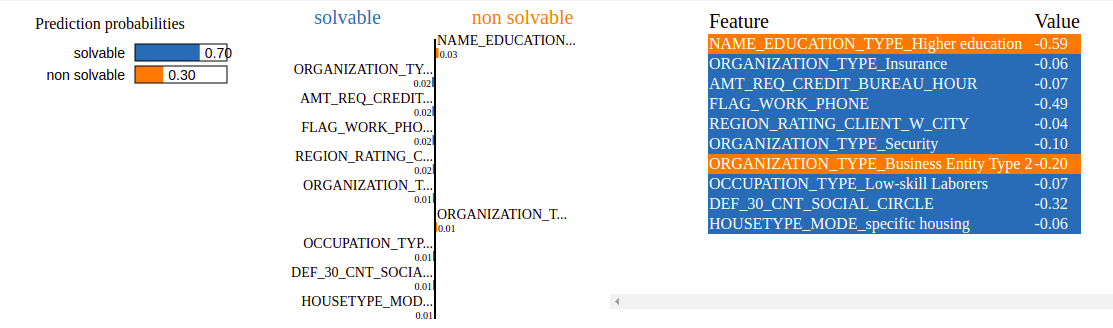
* enlevant des mots du texte d’origine, si on utilise des données textuelles
* remplaçant des blocs de pixels similaires par du gris, si on utilise des images
* permutant certaines données de l’échantillon d’entraînement du modèle original faisant partie du voisinage de l’instance étudiée, si on utilise des données tabulaires

Cet échantillon est ensuite passé à notre modèle *boîte noire* pour avoir des prédictions.

Ensuite, un modèle transparent linéaire est entraîné sur ce même échantillon, mais avec un poids pour chaque instance, basé sur la similarité de cette dernière avec l’instance étudiée.

Le poids que ce modèle donne à chaque *feature* représente l’importance de celle-ci dans l’explication donnée par LIME.

En d’autres termes, « l’interprétation » désigne l’évaluation globale du processus de prise de décision. Elle vise à représenter l’importance relative de chaque variable. L’idée est donc d’expliciter au mieux le score renvoyé par le modèle et permet de mieux comprendre le choix et l’importance de des caractéristiques via un attribut feature\_importances\_

**

### 

### 

# AXES D'AMÉLIORATIONS À APPORTER

**Améliorer le traitement de données :**

Nous pouvons l’expliquer par le features engineering qui est à améliorer. En effet, le Kernel choisi est plutôt pauvre en traitement des données. Il ne se focalise que sur une seule table et ne crée pas beaucoup de variables 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.