# Note Méthodologique

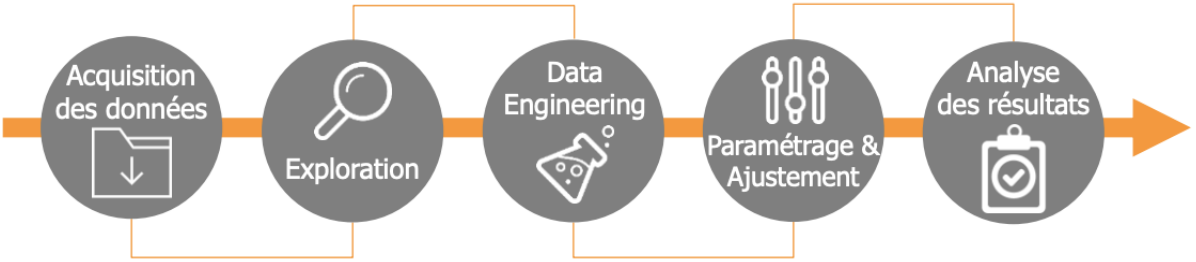
Projet 7 Parcours Data Scientist

Nicolas Pasero Mai 2021
*OpenClassrooms - Centrale Supélec*

# Synthèse

Les algorithmes de *Machine Learning* divergent dans leurs méthodes de modélisation mais se basent sur le même principe. Le projet a été traité à l'aide de 3 algorithmes de *gradient boosting* (LightGBM, XGBoost et CatBoost) utilisés en raison de leur simplicité d'interprétation pour certains mais aussi par expérience, face à leur performances passées. L'entraînement a été fait sur *Google Colab* avec utilisation d'un *GPU*.

Un projet de *Machine Learning* se construit en plusieurs étapes successives. La mise en oeuvre a été opérée selon une démarche fréquente qui se construit de la manière suivante :




La première étape consiste à récupérer des données cohérentes afin de former la base sur laquelle l'algorithme de *Machine Learning* va apprendre. Des données financières, comportementales, etc… relatives à différents clients sont récupérées sur Kaggle.

Avant d'utiliser le jeu de données à des fins de prédiction, le retraitement des données est une tâche importante qui intervient naturellement à la suite de la phase exploratoire. En effet, les données brutes sont souvent bruitées, peu fiables et incomplètes. Leur utilisation peut générer des résultats trompeurs d'où l'intérêt de procéder à certaines modifications ou à un enrichissement de la base.

Un kernel Kaggle est utilisé pour faciliter l'exploration des données nécessaires à l'élaboration du *modèle de scoring*. La mise en œuvre de statistiques descriptives découle ensuite et permet de comprendre et d'avoir une vue d'ensemble des données grâce à de la visualisation graphique.

L'objectif final d'un projet de *Machine Learning* est de construire un modèle de prédiction. C'est à partir de la base retraitée que les algorithmes abordés précédemment sont paramétrés puis implémentés de manière optimale.

Pour analyser les résultats et juger de l'efficacité d'un algorithme, donc du modèle qui en découle, il est nécessaire de calculer l'erreur des apprentissages effectués par une métrique d'évaluation, à savoir l'aire sous la courbe *ROC (AUC)*. Enfin, une fonction coût a été implémentée afin de pénaliser l'impact des erreurs sur la décision d'octroi de crédit.

# Table des matières

# 1 - Prétraitement des données

L'entraînement d'un modèle nécessite une étape transitoire entre le preprocessing et la modélisation. Ce modèle de prédiction peut être représenté par une fonction qui prend des données en entrée et une décision en sortie. Dans le cas d'*apprentissage supervisé* où nous souhaitons expliquer une variable de sortie, l'échantillon est classiquement subdivisé en 2 parties :

- L'échantillon d'apprentissage correspond à l'échantillon principal où sont appliquées les méthodes, sur lesquelles les algorithmes apprennent. Il sert à ajuster le modèle, et représente dans notre cas 80% des données.

- L'échantillon de test est utilisé pour évaluer le modèle optimal (au sens du résultat de la validation croisée). Il n'a donc pas été utilisé pour l'apprentissage, ce qui fait que le modèle sélectionné est indépendant de cet échantillon test. L'idée est de simuler la réception de nouvelles données (entrée) afin de prédire la variable à expliquer à partir du modèle final et de les comparer aux « vraies » valeurs de la variable à expliquer. Cet échantillon permet d'évaluer objectivement l'erreur réelle. L'échantillon test représente donc 20% des données.

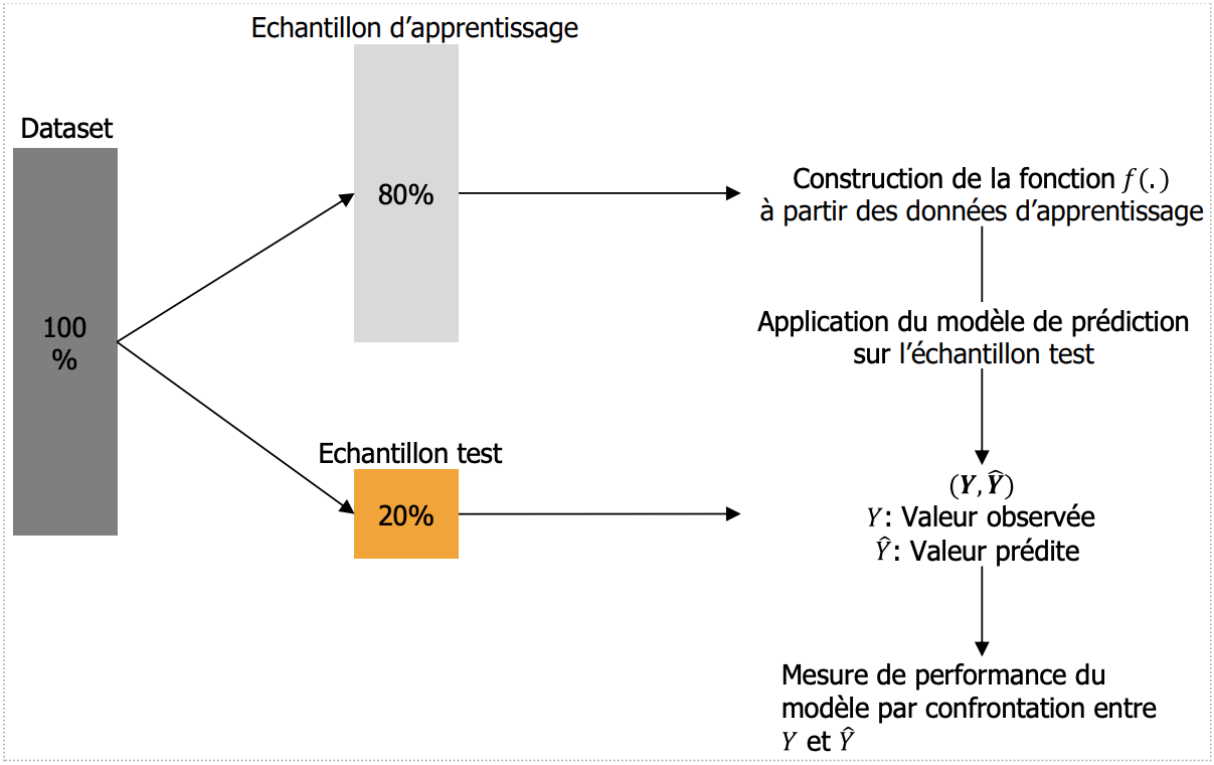



Figure 1 - Découpage des données Train/Test

Ce découpage de données dans un projet de *Machine Learning* est une étape très importante qu'il ne faut pas négliger, il existe en effet un risque de surévaluer le modèle (over-fitting) ou tout simplement le contraire (under-fitting). En effet, par nature un modèle va coller (mais pas trop) à ses données d'entraînement.

*train_test_split()* de Scikit-Learn permet d'obtenir facilement une répartition aléatoire des individus en base d'apprentissage et de test.

Le problème de *Machine Learning* à résoudre est un problème de *classification binaire*, il faut s'assurer que chaque côté contient une proportion raisonnable des classes 0 et 1.

L'Analyse Exploratoire des Données a permis d'identifier un fort déséquilibre entre la précision trouvée entre la classe 0 et la classe 1. L'échantillon de travail est déséquilibré, 92% en classe 0 vs 8% en classe 1. Le traitement *Oversampling* (ou suréchantillonnage en français) permet d'ajuster la distribution de classe de manière à avoir une répartition plus égalitaire.

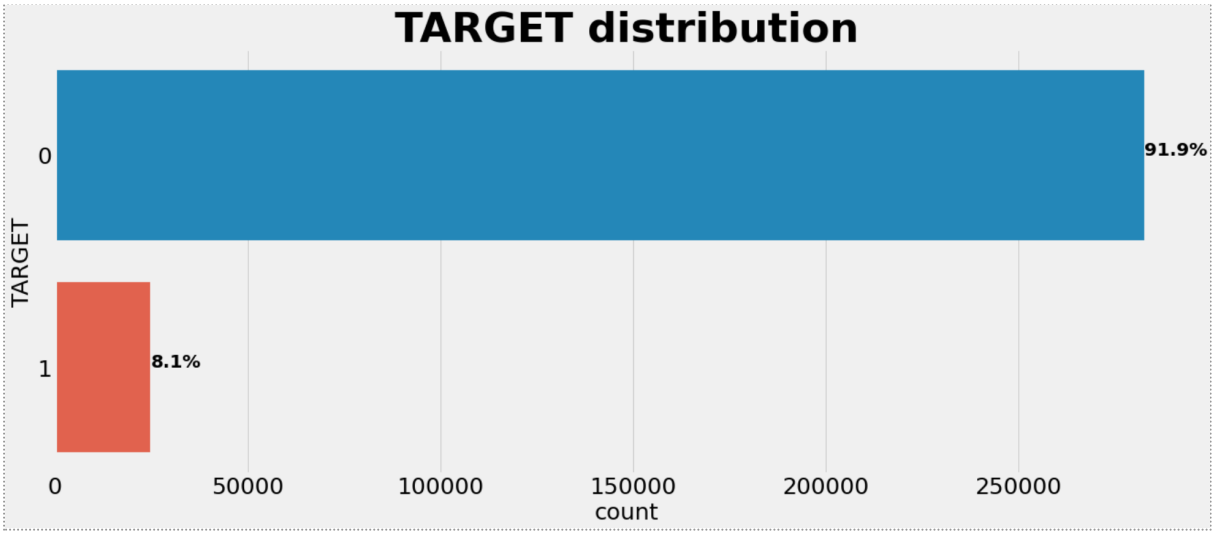



Figure 2 - Déséquilibre des classes

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 sur-échantillonnant avec remplacement. Utilisation de :
`from imblearn.over_sampling` (imbalanced-learn Python library SMOTE class)

```
Label 1, Before using SMOTE: 17412
Label 0, Before using SMOTE: 197845
```

```
Label 1, After using SMOTE: 197845
Label 0, After using SMOTE: 197845
```

# 2 - Modélisation de la probabilité de défaut de paiement du client

---

## 2.1 - Notions de base en Machine Learning

La modélisation du scoring qui donnera 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.

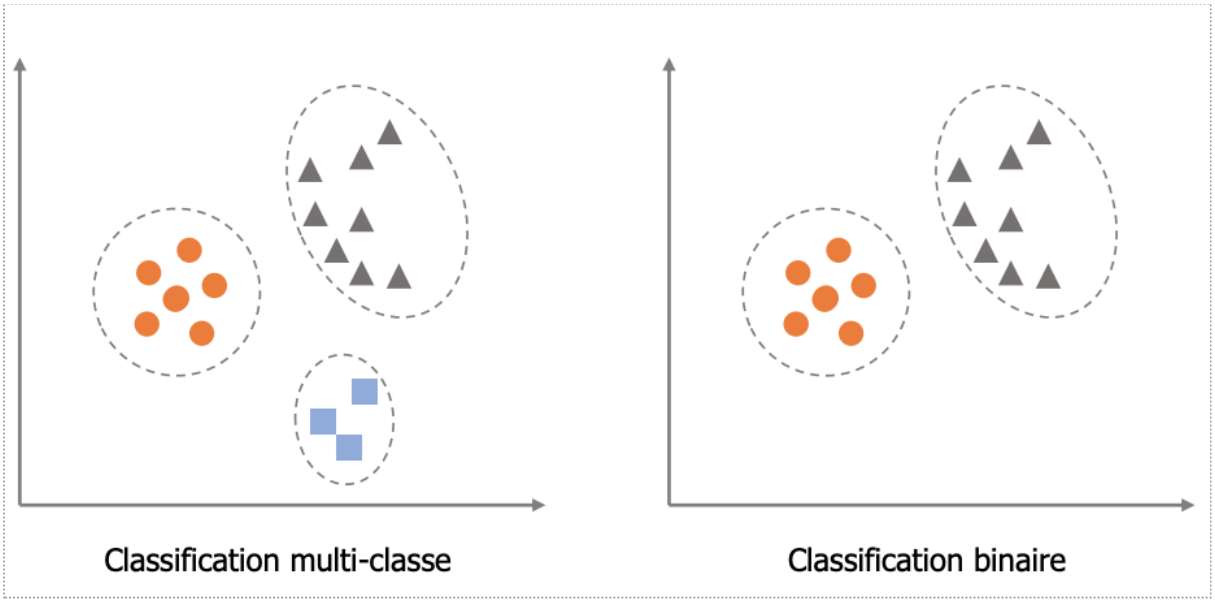



Figure 3 - Représentation des observations

Notons qu'il existe une autre forme de classification appelée *clustering*, dont le principe est de regrouper des données par similarité sans avoir d'information au préalable. La différence avec la classification est que les classes ne sont pas identifiées en amont mais émergent à partir de l'exploration de la structure des données. Le *clustering* est souvent utilisé dans des problématiques de réduction de variable et n'a pas été utilisé ici.

La méthode utilisée précédemment appartient à la famille des *algorithmes d'apprentissage supervisé*, dont le but est de généraliser à partir de ce qu'on connaît déjà, tandis que le *clustering* entre dans la catégorie de l'*apprentissage non-supervisé*, où le but est plutôt de chercher à faire émerger de l'information qui n'était pas présente au départ. Les décisions d'octroi de crédit sont prédéterminées, d'où le cadre de l'*apprentissage supervisé*.

## 2.2 - Algorithmes utilisés pour construire le modèle de scoring

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 le modèle de scoring. Contrairement aux modèles statistiques classiques qui doivent vérifier certaines hypothèses sur la distribution des données, le *Machine Learning* a de fortes capacités prédictives compte tenu de son approche non paramétrique et de la faculté de ses algorithmes à apprendre sans a priori, à partir des données. Au vu du large panel de possibilités, trois algorithmes de *Gradient Boosting* ont été testés. Une baseline a également été fixée par *Régression Logistique*, un modèle statistique classique.

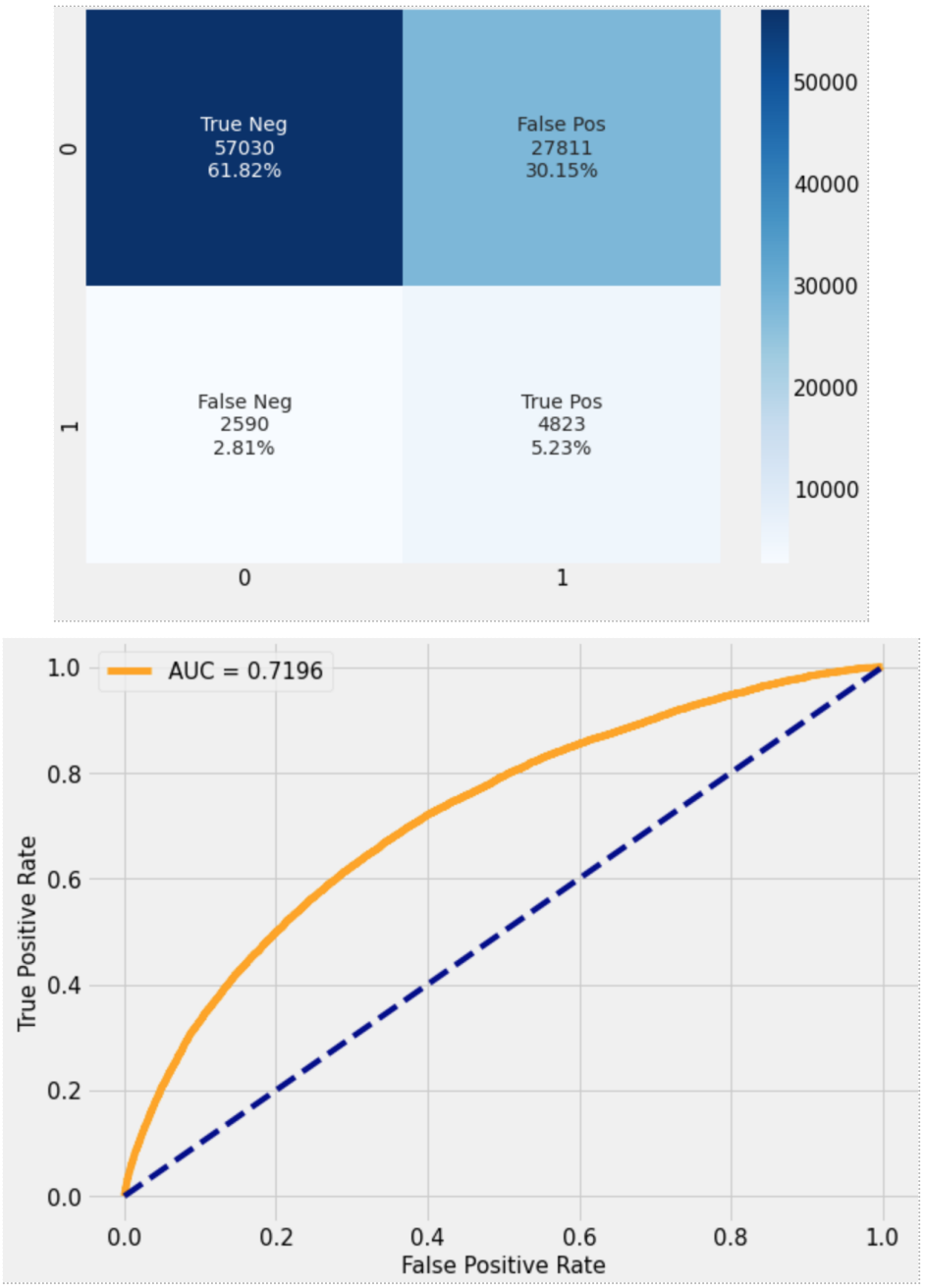



Figure 4 - Résultats Régression logistique : Matrice de confusion, AUC score

| | Model | AUC | Accuracy | Precision | Recall | F1 | Time |
|---|---|---|---|---|---|---|---|
| **1** | LGBMClassifier | 0.773782 | 0.920264 | 0.563758 | 0.0339943 | 0.0641221 | 10.7289 |
| **0** | CatBoostClassifier | 0.773559 | 0.920144 | 0.5625 | 0.0279239 | 0.0532065 | 133.957 |
| **2** | XGBClassifier | 0.764208 | 0.919841 | 0.59375 | 0.00768919 | 0.0151818 | 4.2202 |

## 2.3 - Sélection des variables pertinentes

Utilisation de la technique d'**élimination des caractéristiques récursives avec validation croisée ( RFECV** ). `from sklearn.feature_selection import RFECV`

Terminologie :

- *Récursif* : impliquant de faire ou de dire la même chose plusieurs fois afin de produire un résultat ou un effet particulier.
- *Caractéristique* : propriété individuelle mesurable ou caractéristique d'un phénomène observé, variable du jeu de données.
- *Validation croisée* : une technique pour évaluer les modèles ML en entraînant plusieurs modèles ML sur des sous-ensembles des données d'entrée disponibles et en les évaluant sur le sous-ensemble complémentaire des données.

Brève explication de la méthode de l'élimination des caractéristiques récursives (RFE). C'est essentiellement une sélection rétrograde des prédicteurs. Elle commence par construire un modèle sur l'ensemble des prédicteurs et calcule un score d'importance pour chaque prédicteur. Le ou les prédicteurs les moins importants sont ensuite supprimés, le modèle est reconstruit et les scores d'importance sont à nouveau calculés. En pratique, l'analyste spécifie le nombre de sous-ensembles de prédicteurs à évaluer ainsi que la taille de chaque sous-ensemble. Par conséquent, la taille du sous-ensemble est un paramètre de réglage pour RFE. La taille du sous-ensemble qui optimise les critères de performance est utilisée pour sélectionner les prédicteurs en fonction des classements d'importance. Le sous-ensemble optimal est ensuite utilisé pour entraîner le modèle final.

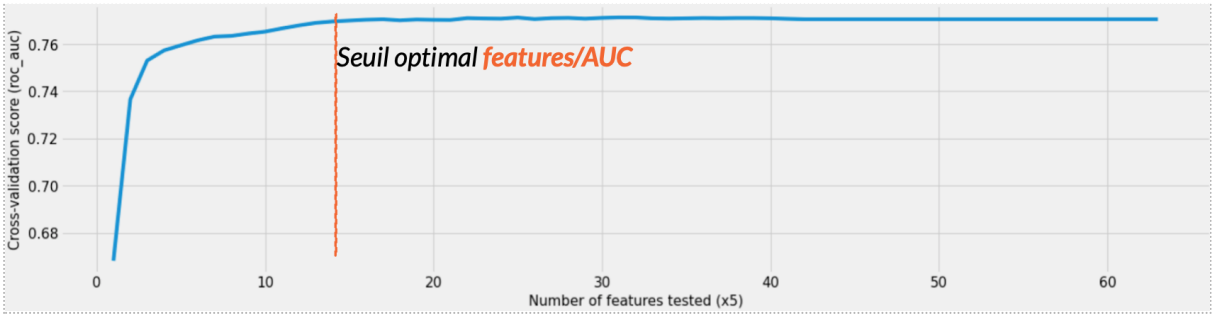



Figure 5 - Précision obtenue avec chaque nombre de features obtenues

## 2.4 - Fonction coût

L'entreprise nommée "Prêt à dépenser" lutte contre les défauts de paiement des clients, les pertes financières ne sont en effet pas souhaitables (frais de recouvrement, pertes,…).

Le modèle ne permettra pas d'éviter totalement ce risque, à titre d'exemple une erreur de prédiction aura pour conséquence soit un défaut de paiement du client, soit un refus de crédit à un client qui pourrait rembourser sa dette sans aucune défaillance. Les erreurs de prédiction doivent être minimisées, dans cette logique une *fonction coût* ayant pour objectif de pénaliser les *Faux Positifs* et les *Faux Négatifs* a été implémentée.

Terminologie :

- FP (*Faux Positifs*) : les cas où la prédiction est positive, mais où la valeur réelle est négative. Perte d'opportunité si le crédit client est refusé à tort, alors qu'il aurait été en mesure d'être remboursé.
- FN (*Faux Négatifs*) : les cas où la prédiction est négative, mais où la valeur réelle est positive. Perte réelle si le crédit client accepté se transforme en défaut de paiement.
- TP (*Vrais Positifs*) : les cas d'acceptation, le crédit client sera remboursé.
- TN (*Vrais Négatifs*) : les cas de refus, le crédit client ne pourra pas être remboursé.

Ainsi, les pertes d'un crédit en raison d'une mauvaise classification dépendent des probabilités *Faux Positifs* et *Faux Négatifs*. L'idée est d'éviter les clients avec un fort risque de défaut. Il est donc nécessaire de pénaliser les FP et FN cités précédemment. Pour réduire ce risque de perte financière, il faut maximiser deux critères *Recall* et *Precision*.

$$\text{Recall} = \frac{tp}{tp + fn} \qquad \text{Precision} = \frac{tp}{tp + fp}$$

Fonction d'optimisation *Recall* et *Precision* avec une importance plus forte pour le critère Precision:

$$\text{Fscore} = \frac{precision.\, recall}{\beta^2.\, precision + recall}$$

**L'application de cette métrique métier passe par la quantification de l'importance relative entre *Recall* et *Precision*, à savoir Beta (β).**

Cela revient à estimer le coût moyen d'un défaut, et le coût d'opportunité d'un client refusé par erreur. Cette connaissance métier n'est pas évoquée à ce stade du projet, nous allons donc l'estimer. Cette hypothèse pourra bien entendu être modifiée avec un interlocuteur métier.

- Défaut de paiement 30% du montant du crédit en pertes et autres recouvrements.
- 10% de chance d'obtenir un crédit pour un client lambda qui souhaite emprunter.

$$\text{Beta} = \frac{coef\,Recall}{coef\,Precision}$$

L'hypothèse fixée dans le projet est **Beta = 3**. Des tests fonctionnels ont été ajoutés pour estimer la cohérence des scores obtenus…

***Compréhension du score en pourcentage sur le risque de défaillance d'un client :***
**0%** *Risque faible de défaillance* ----------//---------- **100%** *Risque fort de défaillance*

## 2.5 - Algorithme d'optimisation HyperOpt

Une alternative pour l'optimisation des fonctions ainsi que des hyperparamètres des pipelines *Machine Learning*. Le réglage des hyperparamètres est basé sur l'**optimisation bayésienne**. `from hyperopt import fmin, tpe, hp, STATUS_OK, Trials`

La configuration optimale des hyperparamètres pour une fonction donnée ne doit pas être entièrement basée sur l'intuition ou l'expérience de certains. La recherche d'une telle configuration optimale doit s'appuyer sur des approches garantissant l'optimalité nécessaire. Parmi les approches les plus utilisées, on retrouve les méthodes basées sur des recherches exhaustives (par exemple *Grid Search* et *Random Search*), ou par *optimisation bayésienne.*

L'approche de l'*optimisation bayésienne* se concentre sur un modèle de probabilité P (score | configuration), qui est mis à jour par un processus itératif d'interrogation d'un historique (score, configuration) dont l'objectif est la maximisation du score donné pour une configuration. HyperOpt prend l'*optimisation bayésienne* comme prémisse en faisant quelques variations dans le processus d'échantillonnage, la définition et la réduction de l'espace de recherche et les algorithmes pour maximiser le modèle de probabilité

**HyperOpt** nécessite 4 paramètres pour une implémentation de base qui sont: la fonction à optimiser , l' espace de recherche , l' algorithme d'optimisation et le nombre d'itérations.
La fonction StratifiedKFold(5) a été utilisée pour trouver la meilleure itération.

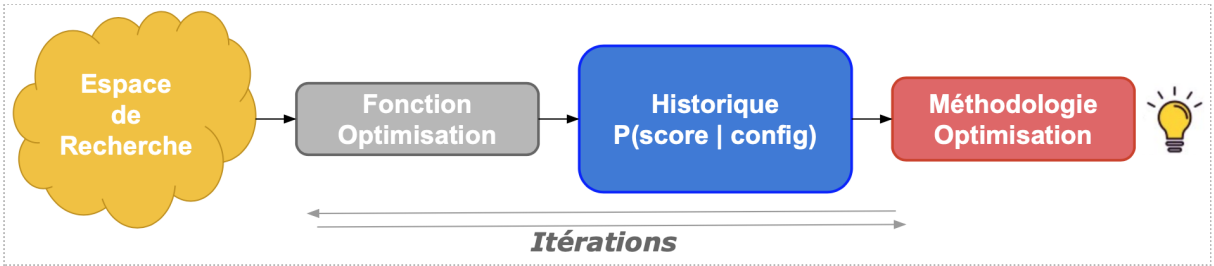



Figure 6 - Schéma représentatif de HyperOpt

La mise en œuvre de *LightGBM* est facile, la seule chose compliquée est le réglage des hyperparamètres. *LightGBM* couvre plus de 100 hyperparamètres. Dans le contexte du projet l'idée est de pouvoir optimiser quelques hyperparamètres via HyperOpt.

En amont, il est nécessaire d'identifier des hyperparamètres pouvant avoir un impact dans l'amélioration de la métrique d'évaluation.

- **`n_estimators`** : nombre d'arbres séquentiels.
- **`learning_rate`** : détermine l'impact de chaque arbre sur le résultat final.
- **`max_depth`** : profondeur maximale d'un arbre.
- **`subsample`** : fraction d'observations à sélectionner pour chaque arbre.
- **`colsample_bytree`** : fraction d'observations à sélectionner pour chaque arbre

```
#Parameter space
space = {
    'n_estimators': hp.quniform('n_estimators', 100, 600, 100),
    'learning_rate': hp.uniform('learning_rate', 0.001, 0.03),
    'max_depth': hp.quniform('max_depth', 3, 7, 1),
    'subsample': hp.uniform('subsample', 0.60, 0.95),
    'colsample_bytree': hp.uniform('colsample_bytree', 0.60, 0.95),
    'reg_lambda': hp.uniform('reg_lambda', 1, 20)
    }
```

```
%%time
best = fmin(fn=objective, space=space, max_evals=30, rstate=np.random.RandomState(1), algo=tpe.suggest)

100%|██████████| 30/30 [36:08<00:00, 72.28s/it, best loss: 0.9771273042187306]
CPU times: user 36min 4s, sys: 42.8 s, total: 36min 46s
Wall time: 36min 8s
```

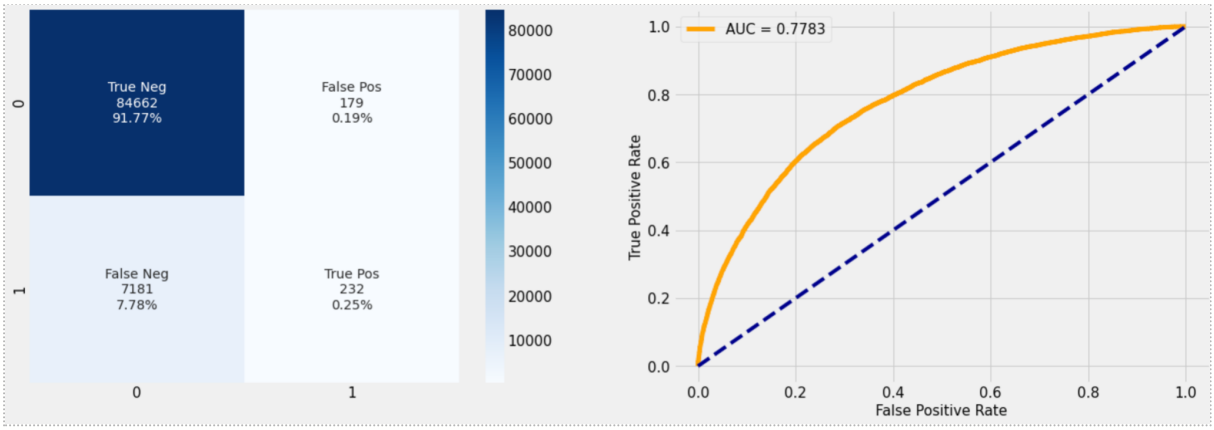



Figure 7 - Résultats obtenus avec le modèle optimisé

## 3 - Interprétabilité du modèle

---

La réponse au besoin d'interprétabilité est prépondérante, le contexte de prédiction n'est pas uniquement appliqué à des experts de la data science mais au contraire à des experts du crédit. Un chargé de clientèle doit pouvoir utiliser le modèle via l'application mise à disposition, en face à face avec son client, dans le but de lui expliquer le plus simplement possible la décision envisagée dans l'étude de son dossier.

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. La classe **lightgbm.LGBMClassifier** permet de mieux comprendre le choix et l'importance de des caractéristiques via un attribut *feature_importances_*

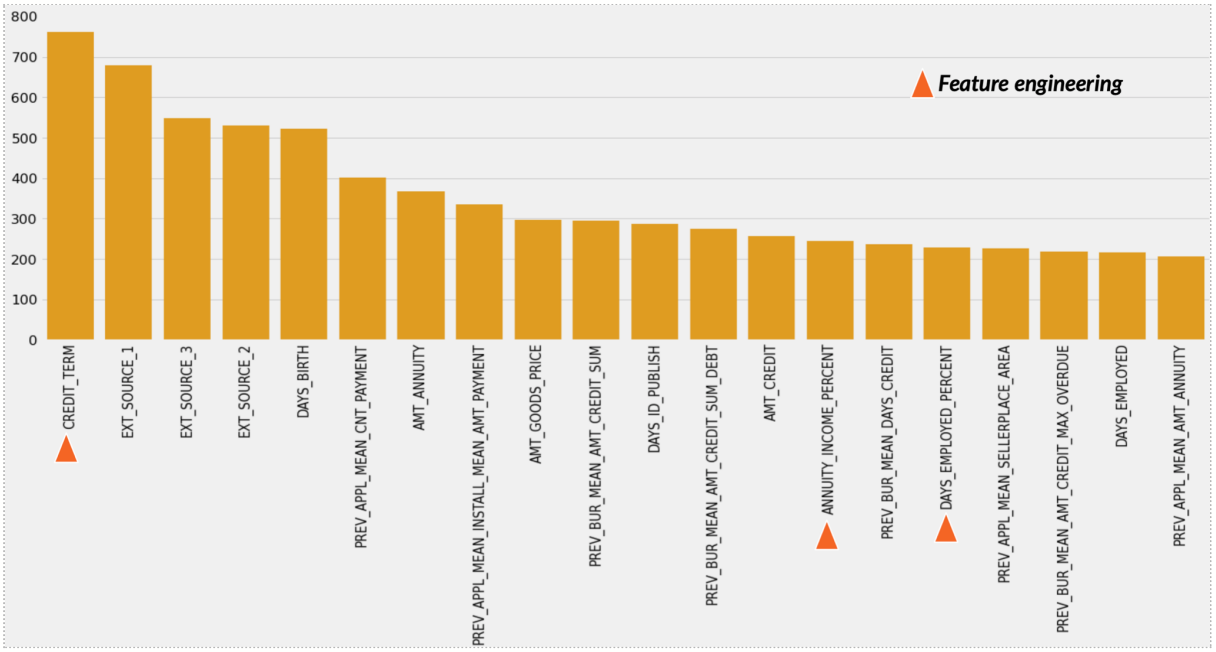



Figure 8 - Top 20 des features les plus importantes dans le modèle de scoring

## 4 - Limites et améliorations

---

Les résultats obtenus sont basés sur une hypothèse non confirmée par les équipes métier. Le coefficient Beta pourrait être affiné dans ce sens pour éventuellement améliorer la métrique d'évaluation.

L'espace de recherche HyperOpt permet de larges possibilités, le choix des hyperparamètres est évolutif, la question d'élargissement vers d'autres hyperparamètres peut également permettre d'augmenter les performances actuelles.