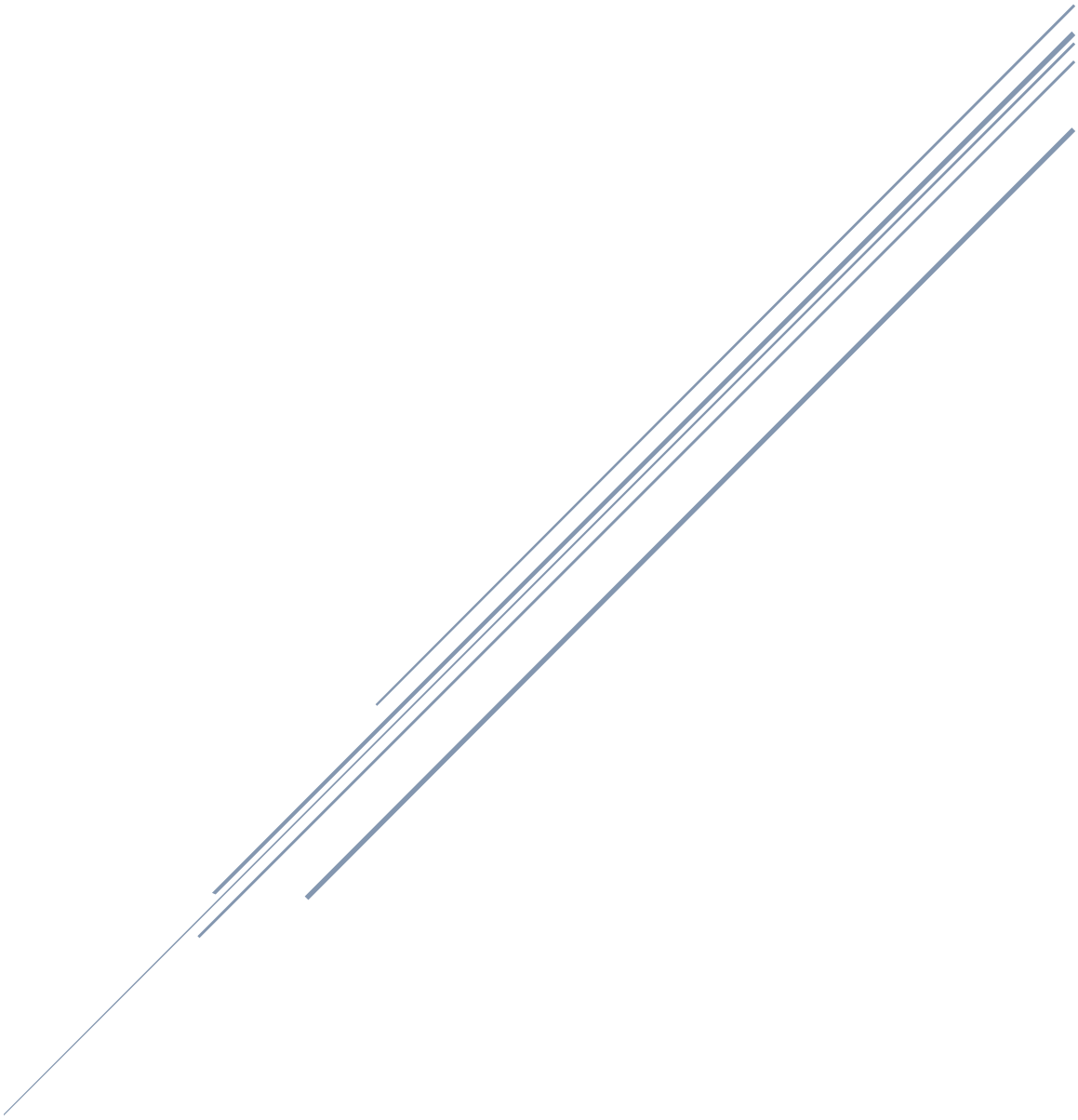
NOTE METHODOLOGIQUE

Projet 7 : « Implémentez un modèle de scoring »



Quentin Stepniewski
Novembre 2020

Sommaire

I. Contexte du projet

1. Description de la mission

Ce compte-rendu présente la méthodologie employée pour mener à bien le projet 7 du parcours Data Scientist d'Openclassrooms : « Implémentez un modèle de scoring ».

Le projet consiste à développer pour la société « Prêt à Dépenser » (une société qui propose des crédits à la consommation pour des personnes ayant peu ou pas du tout d'historique de prêt), un modèle de prédiction de la probabilité de défaut de paiement d'un client avec pas ou peu d'historique de prêt.

Pour ce projet, on utilisera une base de données de 307 000 clients comportant 121 features (âge, sexe, emploi, logement, revenus, informations relatives au crédit, notation externe, etc.)

2. Problématique

La difficulté principale de ce projet réside dans le déséquilibre du jeu de données à notre disposition. En effet, on constate que la proportion de **clients sans défaut** de paiement (représentés par la catégorie 0 dans le jeu de données) correspond à près de 92% du dataset contre **8%** pour les **clients avec défaut** (représentés par la catégorie 1)

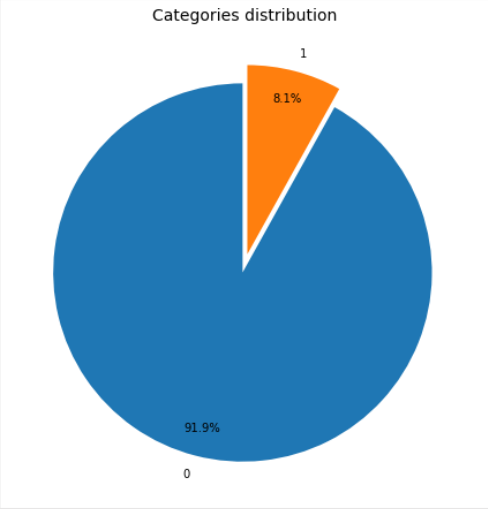


Figure 1 : Distribution clients avec/sans défaut de paiement

Cette situation de déséquilibre va rendre l'apprentissage de nos modèles délicat. En effet, ici, un algorithme classique aura tendance à prédire systématiquement la classe « 0 », obtenant ainsi une précision (accuracy_score) de 92%. Or, d'un point de vue métier, le fait d'accorder un crédit à un client qui ne le remboursera pas est bien plus grave que de ne pas accorder un prêt à un client qui va le rembourser. On ne peut donc pas se contenter de cette méthode, ce sera donc tout l'enjeu de notre projet.

Cette étude présente également un autre déséquilibre : l'importance finale (d'un point de vue métier) des deux classes. En effet, un client à risque non détecté est beaucoup plus impactant pour la société qu'un client sans risqué à qui on refuse un prêt.

3. Pre-processing via Kernel Kaggle

Ce projet se base sur les données obtenues via le preprocessing du notebook suivant :

Home Credit Default Risk - Start Here: A gentle introduction

Dans ce notebook, l'auteur met en place un certain nombre d'étapes :

- Encoding des variables catégorielles
- Détection d'outliers, anomalies
- Création de variables orientées métier :
    - Durée du crédit
    - Ratio du montant du crédit par rapport au revenu
    - Ratio des annuités par rapport au revenu
    - …

On obtient au final 240 features pour notre jeu de données.

II. Méthodologie de gestion d'un jeu de données déséquilibré

1. Objectif et choix de score

Comme présenté dans la section précédente, le projet ne pourra pas être mené à bien uniquement en se basant sur une méthode de scoring habituelle (puisque que la classe des clients avec défaut nécessite une plus grande attention que la classe de clients sans défaut).

Notre catégorisation étant sous forme de deux classes opposées (avec ou sans défaut de paiement), on peut représenter le problème via la matrice de confusion suivante :

	Clients prédits sans défaut (0)	Clients prédits en défaut (1)
Clients réellement sans défaut (0)	Vrais Négatifs	Faux Positifs
Client réellement en défaut (1)	**Faux Négatifs**	Vrais Positifs

Si on se place d'un point de vue métier, on va chercher à minimiser les faux négatifs (clients avec forts risques de défaut non détectés) qui vont engendrer des coûts bien plus importants.

En termes de métrique, on peut donc se dire qu'on cherche à maximiser le Recall afin d'avoir le moins de faux négatifs possibles :

$$Recall = \frac{vrais\ positifs}{vrais\ positifs + faux\ négatifs}$$

Cependant, si on cherche à maximiser uniquement cette métrique, on risque de se retrouver avec énormément (voire uniquement) de clients sans défaut classifiés en défaut (faux positifs). On va donc également (mais à moindre mesure) chercher à maximiser la métrique suivante :

$$Précision = \frac{vrais\ positifs}{vrais\ positifs + faux\ positifs}$$

Ici, on cherche donc une fonction pour optimiser ces deux métriques en favorisant le Recall pour respecter les contraintes métier.

La métrique qui permet de réaliser ceci est le « Fbeta score » :

$$F_\beta = (1+\beta^2)\cdot\frac{\mathrm{precision}\cdot\mathrm{recall}}{(\beta^2\cdot\mathrm{precision})+\mathrm{recall}}.$$

In terms of Type I and type II errors this becomes:

$$F_\beta = \frac{(1+\beta^2)\cdot\mathrm{true\ positive}}{(1+\beta^2)\cdot\mathrm{true\ positive}+\beta^2\cdot\mathrm{false\ negative}+\mathrm{false\ positive}}.$$

Figure 2 : Equations du Fbeta Score - Source Wikipedia

Dans cette métrique, β correspond à un coefficient d'importance relative du Recall par rapport à la précision.

La valeur à donner à cette métrique nécessite d'être déterminée (ou au moins validée) par les experts métiers.

Ici, on a choisi de fixer β= 3 :

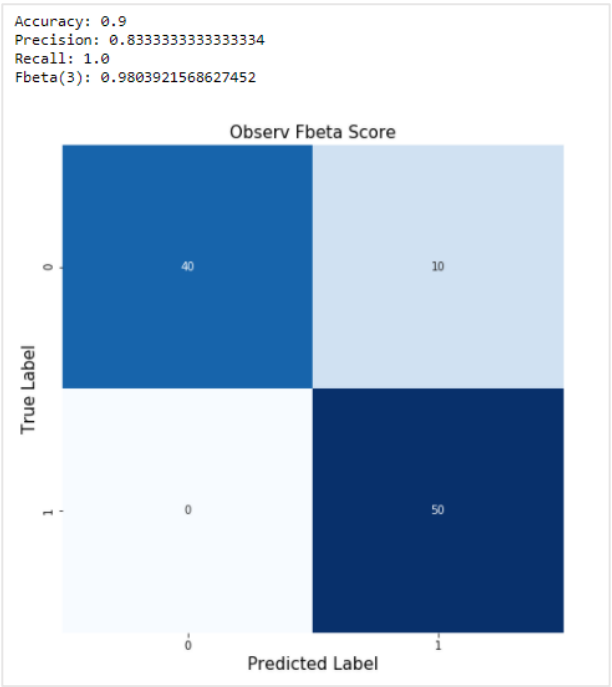


Figure 3 : Observation du Fbeta score avec précision réduite

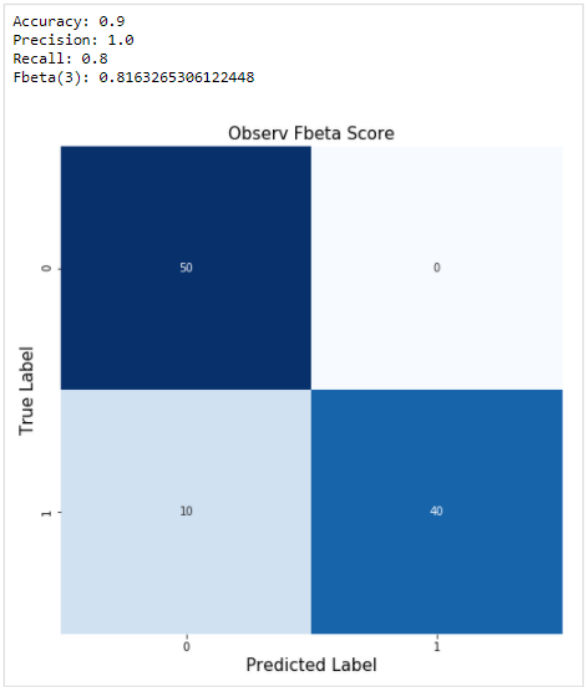


Figure 4 : Observation du Fbeta score avec Recall réduit

Ainsi, on obtient un rapport faux négatifs / faux positifs aux alentours de 10.

On va donc chercher à entrainer un modèle ayant pour but de maximiser cette métrique avec ses prédictions.

2. Gestion du déséquilibre

La première partie a permis de gérer l'objectif de notre modèle (le score qu'on va chercher à optimiser). Cependant, la base de données utilisée présente toujours un problème significatif, à savoir le déséquilibre des deux classes (92% sans défauts, 8% avec défaut). Ce déséquilibre va poser un sérieux problème pour l'apprentissage de l'algorithme.

Il est nécessaire de mettre en place des actions afin de réduire l'impact de ce déséquilibre. Plusieurs solutions sont envisageables :

- Gérer ce déséquilibre en amont :
    - Réduire la quantité de clients sans défaut (Undersampling)
    - Augmenter la quantité de clients avec défaut (OverSampling)
- Gérer ce déséquilibre pendant l'entraînement du modèle (associer un poids différent à chaque classe)
- Gérer ce déséquilibre en aval (jouer sur le seuil de prédiction du modèle).

a. Undersampling

L'Undersampling est une méthodologie qui consiste à ramener la classe surreprésentée au même volume que la classe sous-représentée.

Cette méthodologie fait face à des limitations (principalement le fait de *se passer de données potentiellement importantes pour notre étude).*

b. Oversampling (SMOTE)

La méthode SMOTE consiste à créer des données synthétiques en se basant sur les données de la classe sous-représentée afin de la ramener au même volume que la classe surreprésentée.

Cette méthode a cependant un inconvénient : se basant sur les données existantes, elle aura tendance à créer des données très proches de celles-ci de manière locale sans chercher à correspondre à une distribution globale comme le montre le graphique suivant :

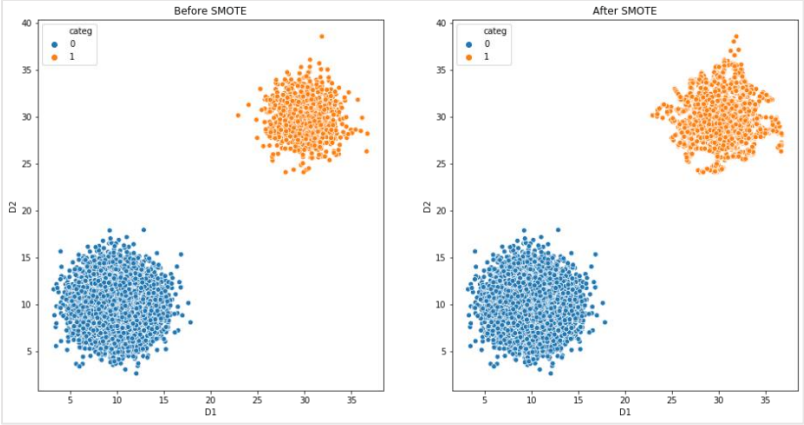


Figure 5 : Observation de distribution avant et après SMOTE

Sur ce graphique, on a représenté deux groupes de données avec le même écart-type mais avec une moyenne différente (dans un souci d'observabilité) :

- Le jeu de données bleu comporte 10 000 individus.
- Le jeu de données orange comporte initialement 1 000 individus et est ramené à 10 000 via SMOTE.

On observe bien que SMOTE a conservé la distribution au niveau local et se rapproche difficilement de la distribution attendue (observable sur le groupe bleu).

c. Class_Weight

La méthode « Class_Weight » permet de modifier le poids associé aux observations. Ainsi, au moment du calcul de la fonction de perte lors de l'entraînement du modèle, une observation mal classée et provenant de la classe minoritaire va pénaliser davantage la fonction de perte de l'algorithme qu'une observation mal classée provenant de la classe majoritaire

d. Seuil de prédiction

Cette dernière méthode s'effectue une fois l'algorithme entraîné. Ici, on va chercher à trouver le seuil de probabilité pour lequel on maximise le score Fbeta (toujours dans l'idée d'essayer de porter une attention plus grande à la classe des clients à risque) :

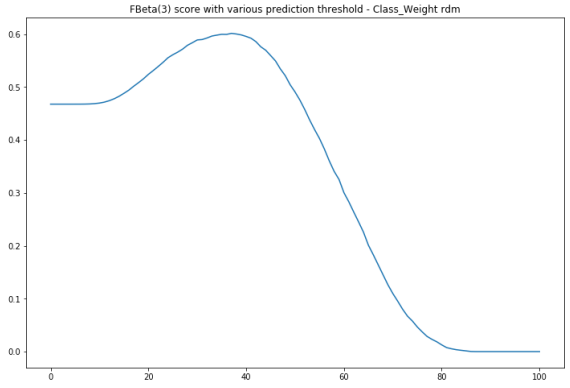


Figure 6 : Impact du seuil de probabilité sur le score Fbeta

On cherche à trouver le seuil de probabilité minimum pour classer un client 'avec défaut'. Par exemple, si on fixe ce seuil à 0.4, un client dont les probabilités estimées par l'algorithme sont :

- 0.58 d'appartenir à la classe 0
- 0.42 d'appartenir à la classe 1

sera tout de même classifié 1.

3. Entraînement et sélection de modèle

Découpage du jeu de données en train/ validation/test afin d'entraîner un modèle pour chaque solution exposée dans la catégorie précédente. Le processus de sélection est le suivant :

- Entraîner un modèle sur le **jeu de train** pour chaque combinaison d'une grille d'hyperparamètres préalablement établie.
- Evaluer le Fbeta score de chaque modèle sur le **jeu de validation** et retenir celui avec le meilleur score
- Répéter ces opérations pour chaque méthode de gestion du déséquilibre pour établir celle fonctionnant le mieux sur notre problématique
- On pourra utiliser le **jeu de test** afin de tester le modèle retenu sur des valeurs qu'il n'aura jamais rencontrées.

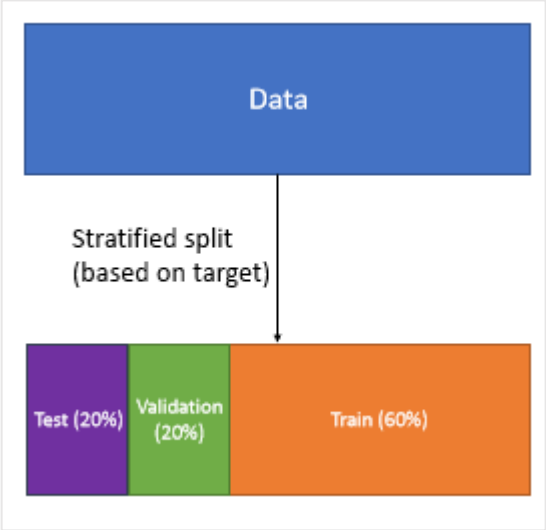


Figure 7 : Découpage du jeu de données initial

Pour ce projet, on utilisera uniquement *RandomForestClassifier* de *Scikit-Learn.*

Nb : Ici il faut bien comprendre que nous cherchons à optimiser un score et non la fonction de perte propre à l'algorithme de classification.

En termes de résultats, on remarque des scores presque équivalents pour 2 méthodes :

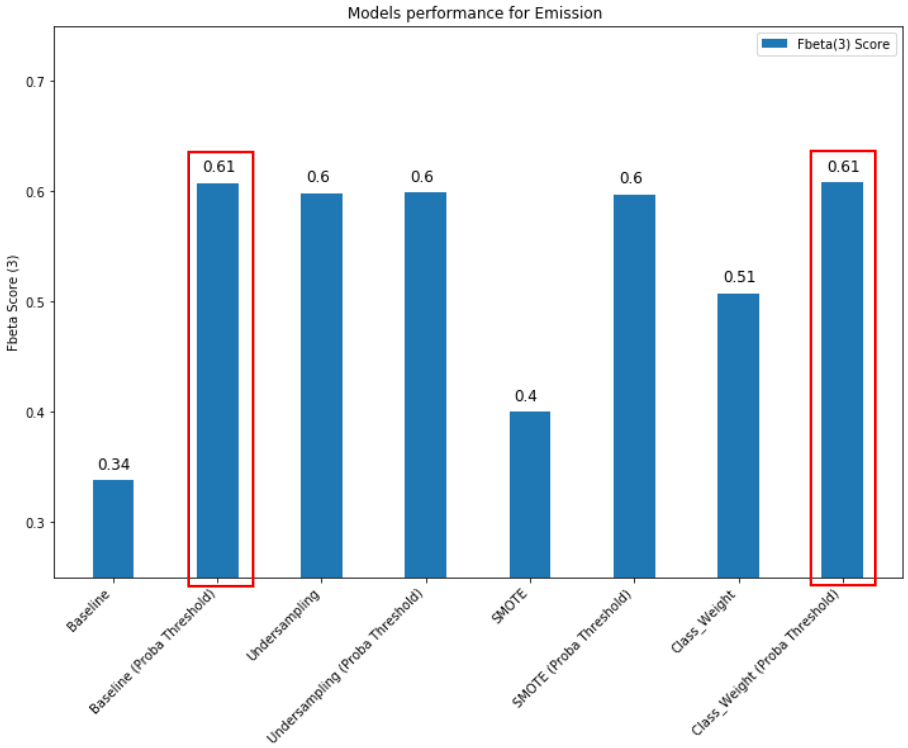


Figure 8 : Résultats des différentes méthodes

Pour choisir entre ces deux méthodes qui utilisent la modification du seuil de probabilité pour la prédiction, on choisira d'observer les deux courbes de score en fonction du seuil de probabilité afin de sélectionner la plus stable des deux :

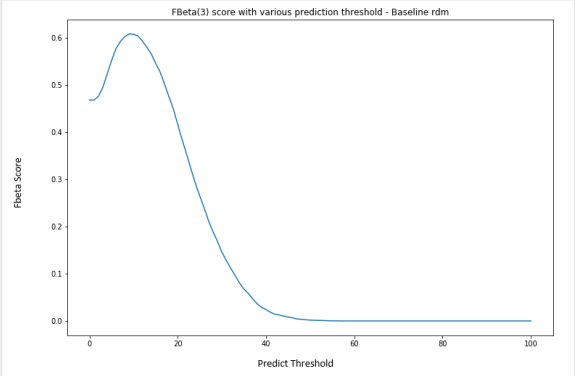


Figure 9 : Dataset de base + Seuil de proba

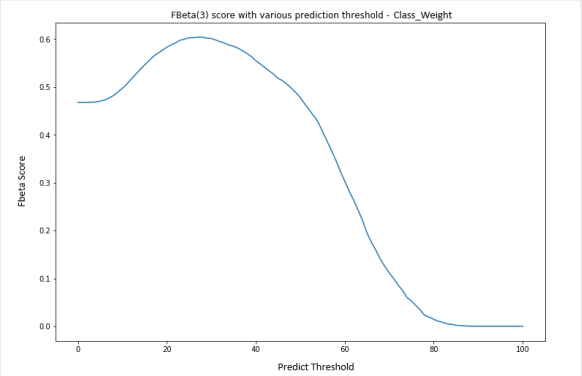


Figure 10 : Class_Weight Method + Seuil de proba

Ici, on retiendra donc la méthode de Class_Weight associé à un seuil de probabilité plus bas pour la prédiction de la classe 1 (aux alentours de 0.3), car la méthode semble avoir un score plus stable autour de son seuil présentant le score maximum.

III. Interprétabilité

L'intérêt de ce projet est de pouvoir fournir notre outil de prédiction aux équipes opérationnelles. Celles-ci devront être en mesure de justifier les décisions établies par l'algorithme. De ce fait, il semble nécessaire d'ajouter un module d'interprétabilité à cet outil, permettant d'avoir un certain nombre de renseignements expliquant le refus/ l'acceptation d'un prêt à un client.

Utilisant un algorithme de RandomForest, on pourrait naturellement se tourner vers l'attribut 'feature_importances' qui donne un aperçu, en une ligne de code, des features les plus importantes dans la construction du modèle :

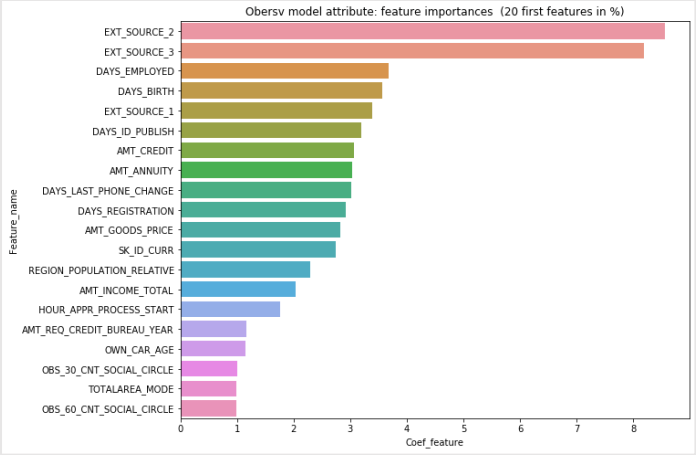


Figure 11 : Feature Importances du modèle final sélectionné

Cet attribut, « gratuit en temps de calcul », a cependant certaines limites :

- Il a du mal à prendre en compte correctement les features de type One Hot Encoding.
- Il correspond plus ou moins à la fréquence d'utilisation d'une feature dans la construction des arbres, ce qui ne représente pas parfaitement l'importance réelle de la feature pour le modèle (une feature peut être utilisée une seule fois dans un arbre de décision et avoir une importance très significative de séparation de groupes de données).

Ici, on utilisera une autre option : **SHAP (*SHapley Additive exPlanation*)**. Il s'agit d'une méthode se basant sur la théorie des jeux : l'idée est de moyenner l'impact qu'une variable a pour toutes les combinaisons de variables possibles.

Cette méthode est assez coûteuse en temps de calcul, mais se rapproche vraiment de l'état de l'art dans le domaine :

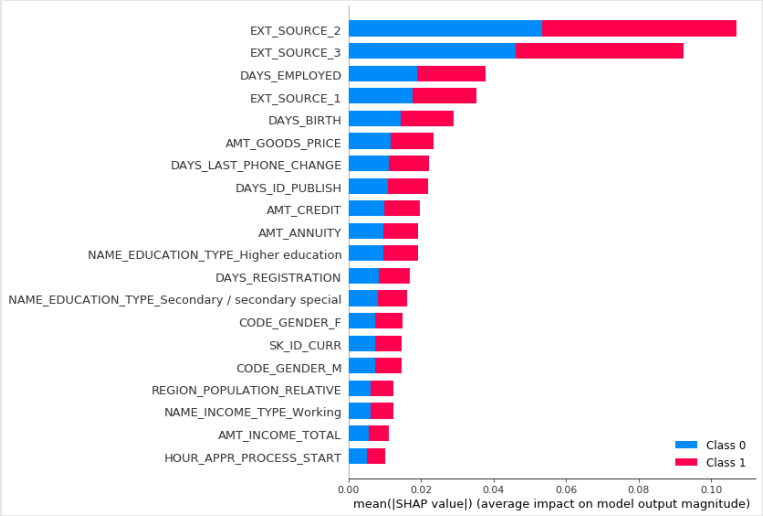


Figure 12 : Feature Importances via méthode SHAP

IV. Conclusion et améliorations possibles

On a donc ici des résultats plutôt intéressants quant à la gestion du déséquilibre de nos données :

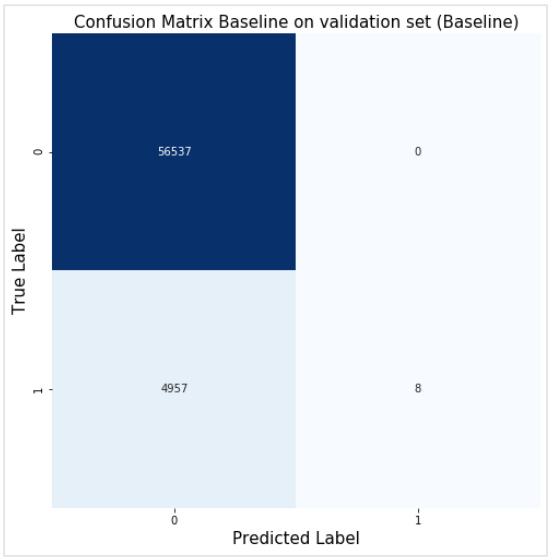


Figure 13 : Prédiction via dataset initial

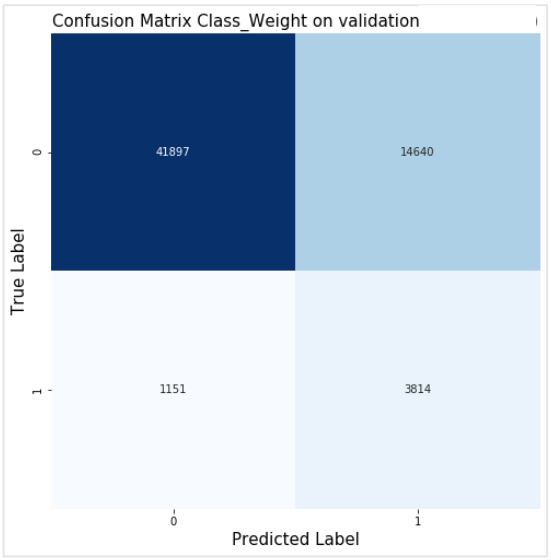


Figure 14 : Prédiction après méthodes de gestion de déséquilibre

On pourrait chercher à améliorer les performances obtenues de plusieurs manières :

- Affiner la métrique (fbeta score) en collaboration avec les équipes métier en se basant sur les pertes / coûts d'opportunité réels du secteur
- Augmenter le nombre d'individus de la catégorie 1 de notre dataset
- Mise en place d'un modèle d'ensemble (Stacking) avec une partie dédiée à l'apprentissage des profils avec défaut de paiement
- Revue et amélioration du pre-processing obtenu via un Kernel Kaggle.