Note Méthodologique P7

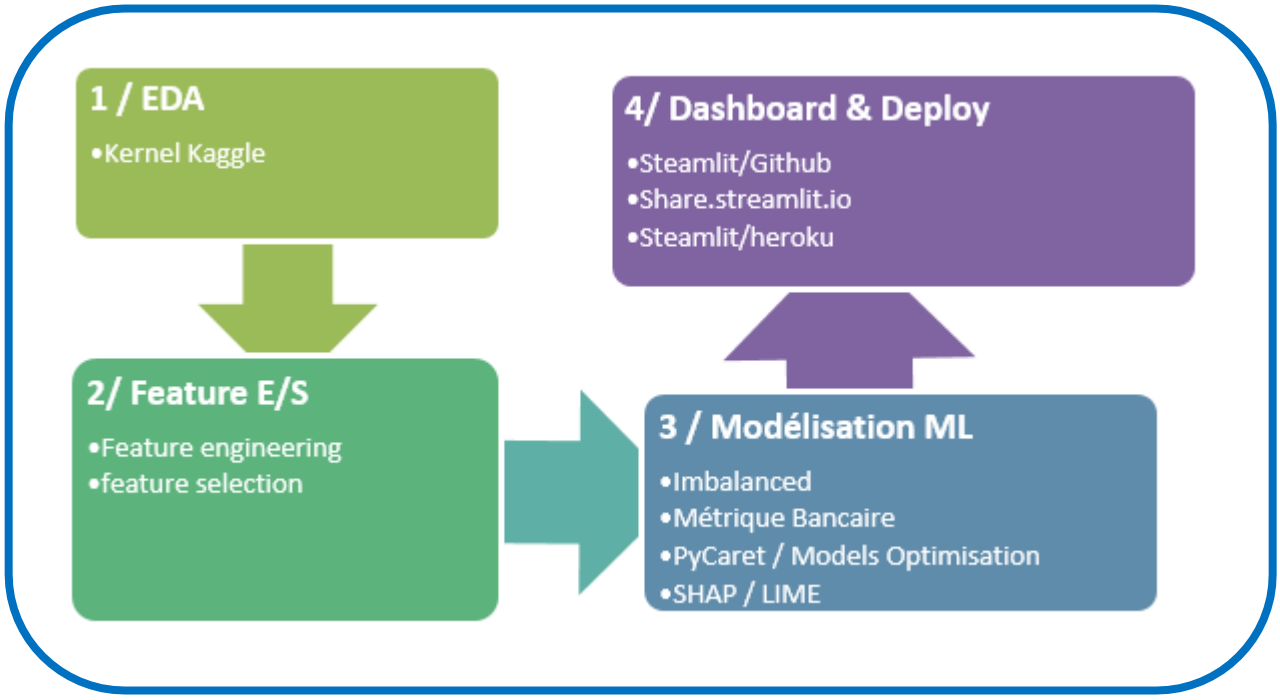
Thierry Vergé-Dépré
Data Scientist

OPENCLASSROOMS
CENTRAL SUPELEC

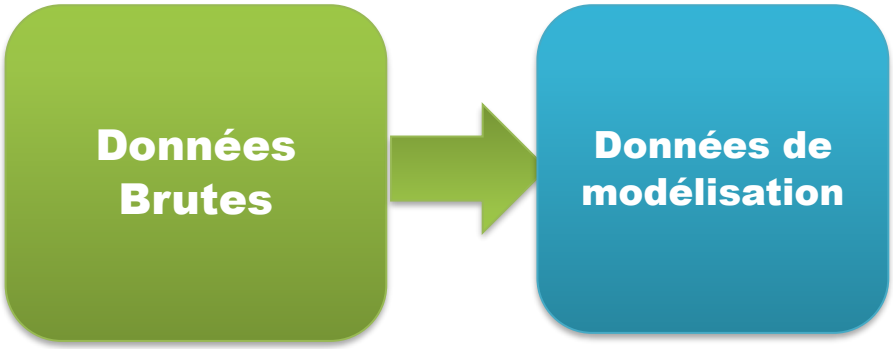
2020 - 2021

Table des matières

1 Adopter une démarche constructive



2 Construire le jeu de données « fiable » avant modélisation



3 Analyser les données

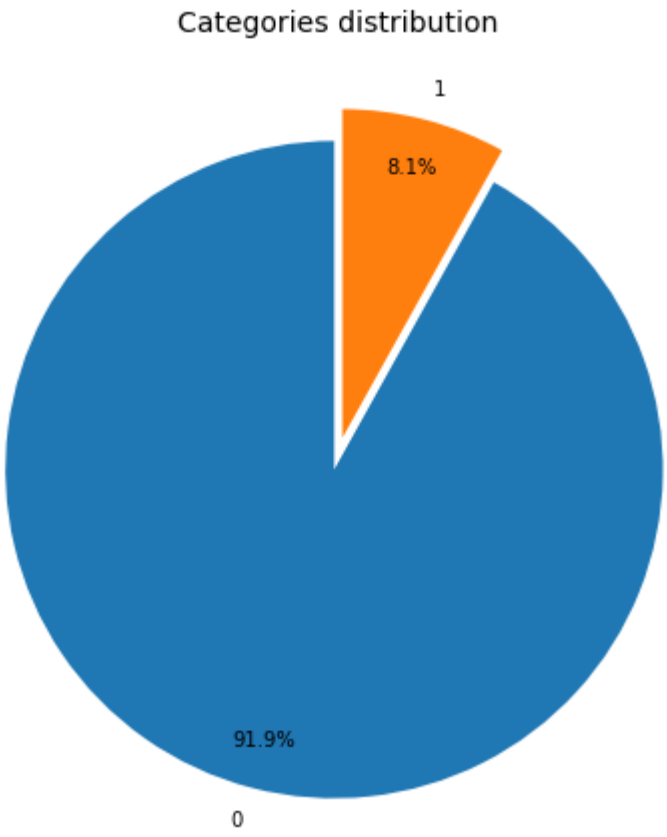
3.1 Exploration du jeu de données

a. Améliorer le niveau de connaissance du domaine métier
b. Développement d'un kernel EDA
c. Permettre la détection d'anomalies
    i. Comme : le ***déséquilibrage*** du jeu de données

Problématique principale des données :

Très large déséquilibre entre les classes :

- Clients sans défaut (classe 0) : **92%** du dataset
- Clients avec défaut (classe 1) : **8%** du dataset



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).

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 risquer à qui on refuse un prêt.

4 Ingénierie des features

4.1 Pré-processing du jeu de données

On obtient finalement 240 features pour notre jeu de données après pré-processing :

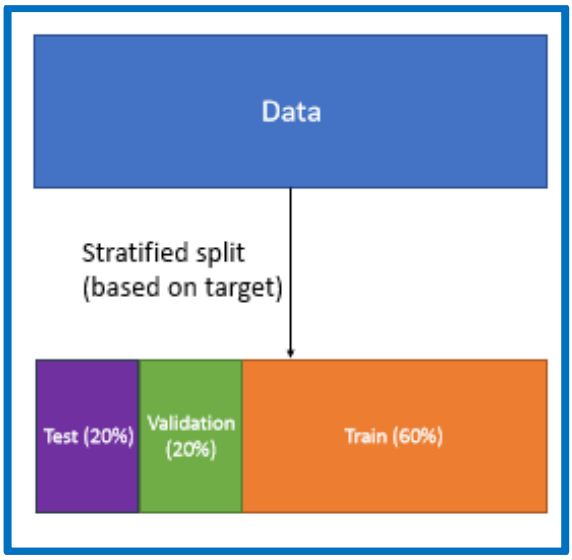
d. Cleaning (doublons, …)
e. Détection d'outliers
f. Création de variables orientées métier :
    i. Durée du crédit
    ii. Ratio du montant du crédit par rapport au revenu
    iii. Ratio des annuités par rapport au revenu
g. Fusion de toutes les tables et constitution d'une base étude finale
h. Pour coder des variables catégorielles en numérique
    i. Encodage des variables Catégorielles : LabelEncoder (si le nombre de catégories est <=2)
    ii. Encodage des variables Numériques : OneHotEncoder (sklearn.preprocessing)
i. Dans le cadre d'un modèle en pipeline :
    i. Imputation par la médiane : SimpleImputer (strategy='median') ; (sklearn.impute)
    ii. Standardisation des variables numériques : StandardScaler (sklearn.preprocessing)
    iii. Stabilité des paramètres : GridSearchCV (sklearn.model_selection)
j. Faire un tri de données via les Features (colonnes caractéristiques) pertinentes pour le modèle
k. Identifier et gérer les problématiques de données déséquilibrées

5 Modélisation

5.1 Première modélisation

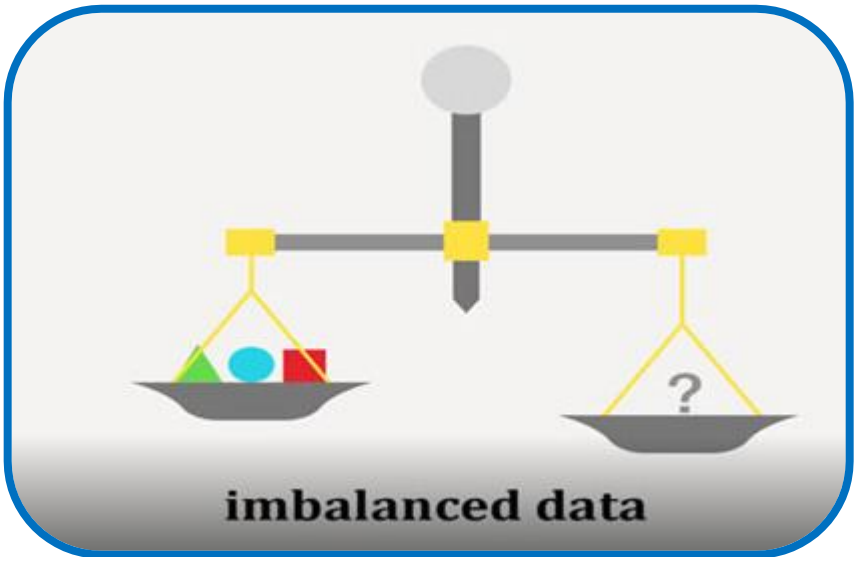
l. Entraînement et sélection de modèle :
    i. Découpage du jeu de données en train/validation/test afin d'entraîner un modèle pour chaque solution exposée dans la catégorie précédente. Le processus de sélection est le suivant :
        1. Entraîner un modèle sur le jeu de train pour chaque combinaison d'une grille d'hyperparamètres préalablement établie.
        2. Evaluer le Fbeta score de chaque modèle sur le jeu de validation et retenir celui avec le meilleur score.
        3. Répéter ces opérations pour chaque méthode de gestion du déséquilibre pour établir celle fonctionnant le mieux sur notre problématique.
        4. On pourra utiliser le jeu de test afin de tester le modèle retenu sur des valeurs qu'il n'aura jamais rencontrées.

Pour ce projet P7, on verra que le modèle ***RandomForestClassifier*** de Scikit-Learn obtient l'un des meilleurs scores.



Découpage du jeu de données initial

5.2 Traiter l'équilibrage des données déséquilibrées



https://machinelearningmastery.com/threshold-moving-for-imbalanced-classification/

m. Différentes méthodes pour gérer le déséquilibre des données ont été exploré :
   i. Gestion du déséquilibre en amont :
      1. ***Undersampling*** : réduit le nombre d'observations de la classe majoritaire au même nombre que la classe minoritaire.

En somme, ***Undersampling*** consiste à réduire la quantité de clients sans défaut.

Retenons que 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).*

2. ***SMOTE (Oversampling)*** : crée des données synthétiques pour ramener le nombre d'observations de la classe minoritaire au niveau de la classe majoritaire.

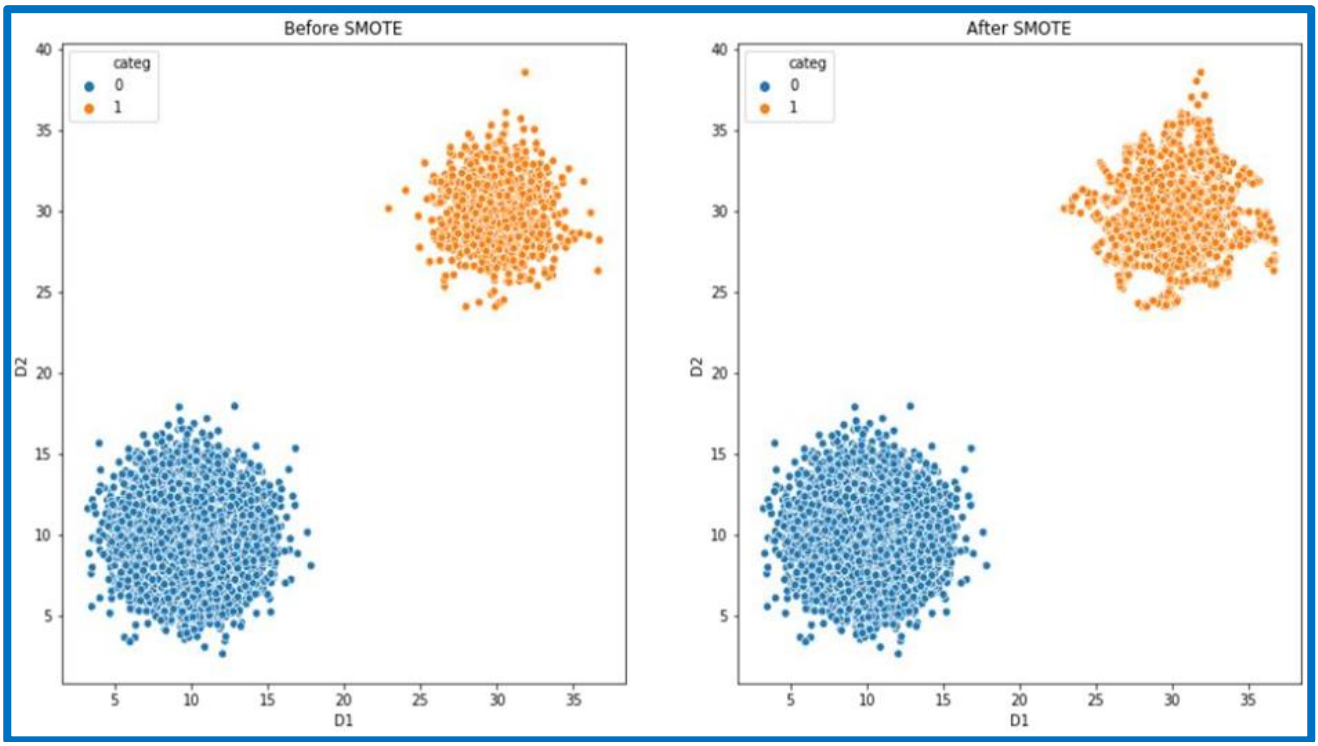
En définitif, ***OverSampling*** consiste à augmenter la quantité de clients avec défaut.

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 :

- 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).



Observation de distribution avant et après SMOTE

ii. Gestion du déséquilibre pendant l'entraînement du modèle :
    1. ***Class Weight*** : appliquer une pénalité plus importante à la fonction de perte lors d'une mauvaise classification de la classe minoritaire.

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.

iii. Gestion du déséquilibre en aval de l'entraînement :
    1. Gestion du seuil de probabilité : faire varier le seuil de probabilité à partir duquel on classifie un individu dans la classe 1.

Cette 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).

On va chercher 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.

5.3 Construire et comparaison de différents modèles :

n. Algorithmes d'apprentissage automatique **NON sensibles aux données déséquilibrées**
    i. Random Model
    ii. Dominant Class
    iii. XGBoost Classifier
    iv. XGBoost Classifier – avec réduction de Features fixé
    v. LightGBM Classifier
    vi. StackingClassifier

o. Algorithmes d'apprentissage automatique **sensibles aux données déséquilibrées**
    i. SGD LogisticRegression L2
    ii. SGD Linear SVM
    iii. Random Forest Classifier
    iv. ExtraTrees Classifier

5.4 Construire la métrique de comparaison des modèles :

p. Métriques : ROC_AUC :
    i. Construire une métrique Monétaire / Bancaire en pénalisant le défaut (avec un gros poids correspondant aux potentiel pertes en capital) et bonifier un remboursement (faible poids correspondant aux intérêts générés).
    ii. Une métrique 'custom' a été créée pour ce modèle. En effet, l'objectif métier étant la minimisation du risque de défaut, nous avons considéré qu'il était préférable de minimiser le nombre de « **faux négatifs** ». Nous avons donc attribué 10 points à chaque faux négatif.

	Prédit sans défaut (0)	Prédit en défaut (1)
Réel sans défaut (0)	Vrai négatif	Faux positif
Réel en défaut (1)	Faux négatif	Vrai positif

Comme indiqué précédemment, 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 » :

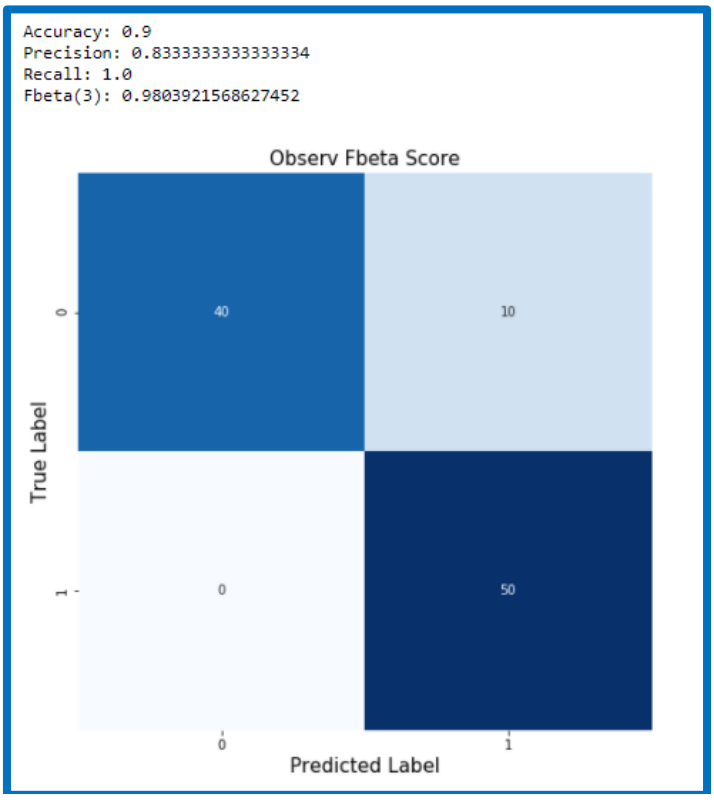
Equations du Fbeta Score - Source Wikipedia

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

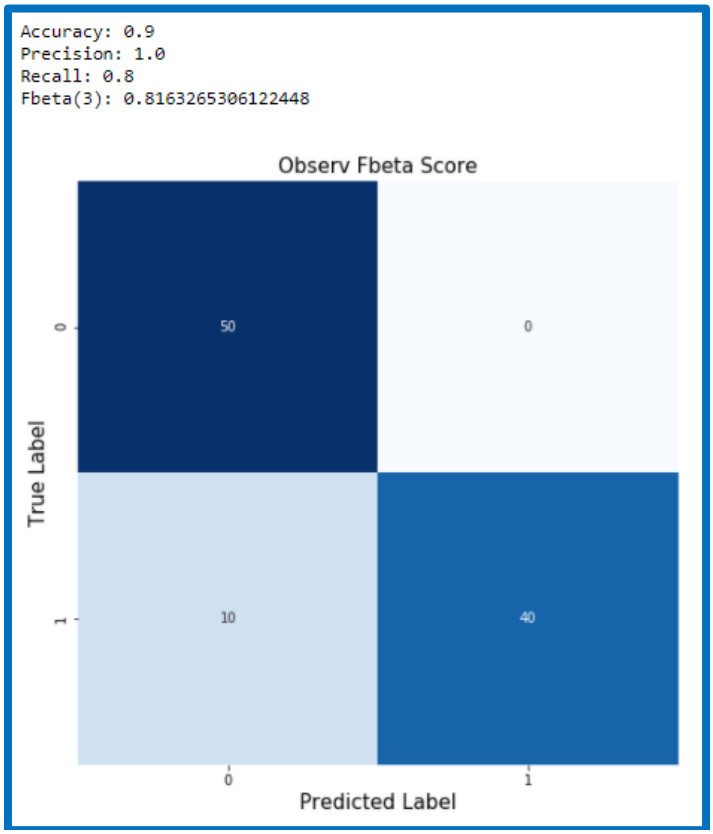
In terms of Type I and type II errors this becomes:

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

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 :



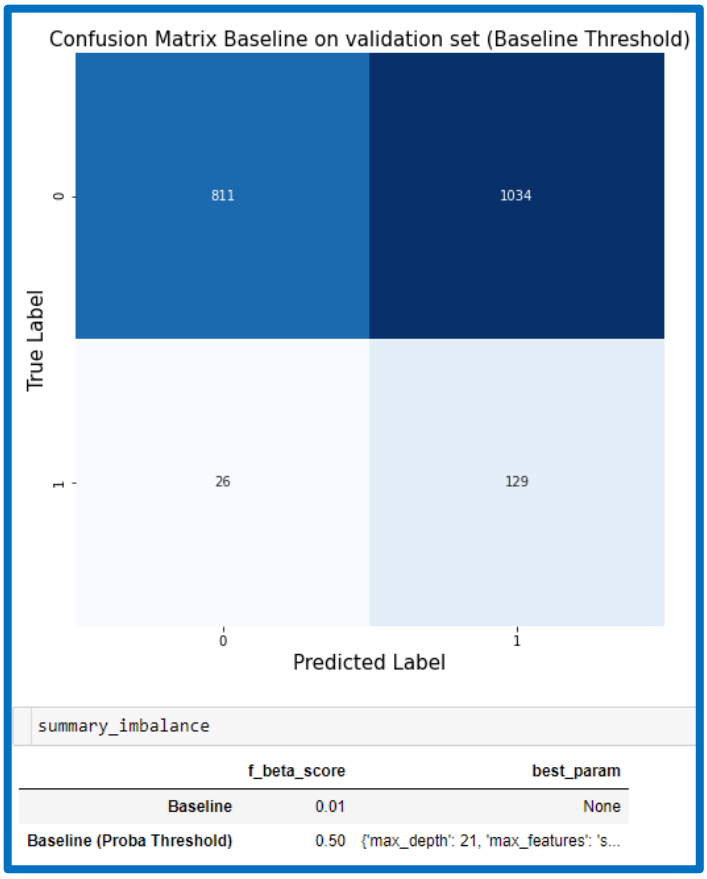
Observation du Fbeta score avec précision réduite



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 avecses prédictions. Ce qui nous donne des scores « faibles » :



	f_beta_score	best_param
Baseline	0.01	None
Baseline (Proba Threshold)	0.50	{'max_depth': 21, 'max_features': 's...

q. Plusieurs métriques ont été utilisées :
   i. ROC-AUC,
   ii. Recall,
   iii. Accuracy,
   iv. Precision,
   v. FBETA (Compromis entre ***Recall*** et ***Precision***)

r. Si la GridSearchCV utilisait ROC-AUC comme méthode de scoring, toutes ces métriques ont été utilisées pour l'observation des performances de nos différents tests de modèle.

5.5 Tests des modèles de Classification

s. Les tests peuvent se faire avec PyCaret (RF, LightGBM, AdaBoost, etc..)

4 Comparer tous les modèles

```
#compare_models()
#best_model = compare_models(fold=10)
compare_models(fold=10)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	0.9210	0.6052	0.0000	0.0000	0.0000	0.0000	0.0000	5.5260
nb	Naive Bayes	0.9210	0.6258	0.0000	0.0000	0.0000	0.0000	0.0000	0.0180
ridge	Ridge Classifier	0.9210	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.3430
rf	Random Forest Classifier	0.9209	0.7142	0.0000	0.0000	0.0000	-0.0003	-0.0011	0.6310
et	Extra Trees Classifier	0.9207	0.7051	0.0000	0.0000	0.0000	-0.0006	-0.0023	0.3910
lda	Linear Discriminant Analysis	0.9185	0.7501	0.0209	0.4576	0.0391	0.0288	0.0755	0.0900
gbc	Gradient Boosting Classifier	0.9168	0.7456	0.0189	0.1893	0.0335	0.0210	0.0364	0.8930
lightgbm	Light Gradient Boosting Machine	0.9165	0.7160	0.0304	0.2131	0.0531	0.0374	0.0562	0.2550
xgboost	Extreme Gradient Boosting	0.9164	0.6995	0.0438	0.2941	0.0741	0.0557	0.0839	1.9370
knn	K Neighbors Classifier	0.9156	0.5350	0.0171	0.1777	0.0308	0.0166	0.0307	0.0920
ada	Ada Boost Classifier	0.9153	0.7280	0.0398	0.2386	0.0668	0.0476	0.0685	0.2800
svm	SVM - Linear Kernel	0.8766	0.0000	0.0613	0.0487	0.0332	0.0080	0.0086	0.3440
dt	Decision Tree Classifier	0.8559	0.5473	0.1809	0.1529	0.1651	0.0872	0.0877	0.1990
qda	Quadratic Discriminant Analysis	0.7063	0.6934	0.3353	0.1430	0.1615	0.0724	0.0642	0.4300

Exemple tiré du notebook du Projet P7 :
« 4--P7-Pret_Credit_Conso_03bis_ PyCaret___Binary Classification___Modelling »
Source d'explication : *https://www.pycaret.org/tutorials/html/CLF101.html*

5.6 Optimisation possible

t. Optimisation du seuil décision

6.3 Random Forest Classifier

```
tuned_rf = tune_model(rf)
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.9218	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000
1	0.9218	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.9218	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000
3	0.9218	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000
4	0.9203	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000
5	0.9203	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000
6	0.9203	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000
7	0.9203	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000
8	0.9203	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000
9	0.9217	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000
Mean	0.9210	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000
SD	0.0007	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Exemple tiré du notebook du Projet P7 :
« 4--P7-Pret_Credit_Conso_03bis_ PyCaret___Binary Classification___Modelling »

u. Optimised LightGBM / Best Modèle : Optimisation Bayésienne

10 LightGBM

10.1 Bayesian Optimization

```
def lgbm_evaluation(num_leaves, max_depth, min_split_gain, min_child_weight,⇔
```

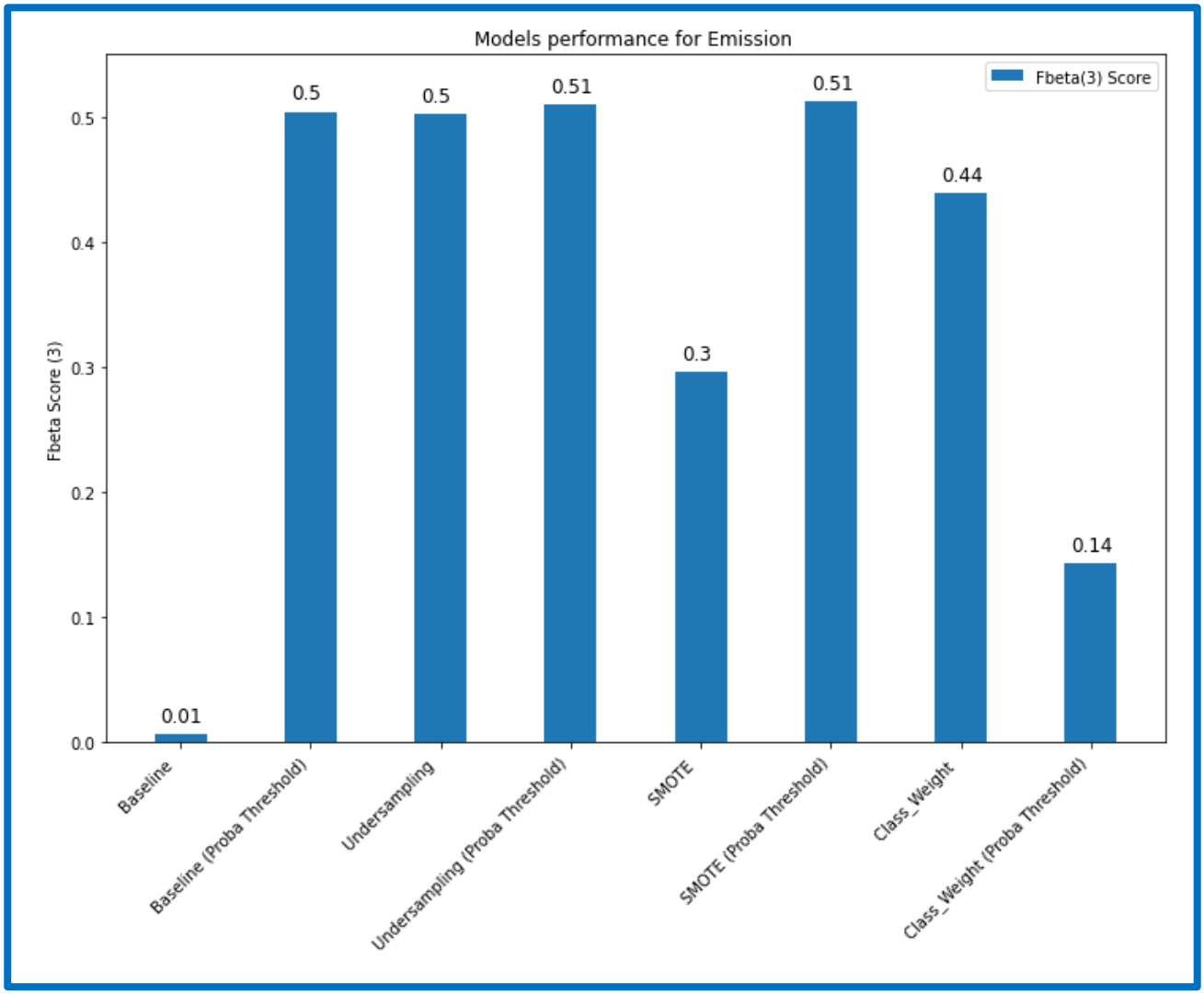
```
bopt_lgbm = BayesianOptimization(lgbm_evaluation, {'num_leaves' : (25,50),
                                                   'max_depth' : (6,11),
                                                   'min_split_gain' : (0, 0.1),
                                                   'min_child_weight' : (5,80),
                                                   'min_child_samples' : (5,80),
                                                   'subsample' : (0.5,1),
                                                   'colsample_bytree' : (0.5,1),
                                                   'reg_alpha' : (0.001, 0.3),
                                                   'reg_lambda' : (0.001, 0.3)},
                                 random_state = 4976)

bayesian_optimization = bopt_lgbm.maximize(n_iter = 6, init_points = 4)
```

```
|   iter    |  target   | colsam... | max_depth | min_ch... | min_ch... | min_sp... | num_le... | reg_alpha | reg_la... | s
ubsample |
-------------------------------------------------------------------------------------------------------------------------
----------
|  1        |  0.7743   |  0.9839   |  9.624    |  65.58    |  60.98    |  0.08223  |  39.55    |  0.1133   |  0.2049   |
0.6677   |
|  2        |  0.7789   |  0.5453   |  10.99    |  36.09    |  42.7     |  0.02383  |  43.12    |  0.1206   |  0.1951   |
0.8343   |
|  3        |  0.7761   |  0.7313   |  9.478    |  47.08    |  53.08    |  0.0249   |  36.94    |  0.2417   |  0.1072   |
0.5916   |
|  4        |  0.7823   |  0.5671   |  7.674    |  26.69    |  8.717    |  0.004937 |  31.48    |  0.118    |  0.09472  |
0.9706   |
|  5        |  0.7835   |  0.843    |  10.6     |  46.36    |  16.37    |  0.0809   |  31.43    |  0.03722  |  0.2425   |
0.8722   |
|  6        |  0.7831   |  0.6559   |  6.891    |  26.95    |  8.266    |  0.01307  |  33.15    |  0.02741  |  0.26     |
0.5547   |
|  7        |  0.7836   |  1.0      |  6.0      |  38.99    |  5.0      |  0.1      |  49.51    |  0.001    |  0.3      |
0.5      |
|  8        |  0.7837   |  1.0      |  6.0      |  13.29    |  5.0      |  0.07941  |  50.0     |  0.001    |  0.3      |
0.5      |
|  9        |  0.7846   |  1.0      |  6.0      |  70.57    |  5.0      |  0.1      |  26.87    |  0.001    |  0.3      |
0.5      |
|  10       |  0.7837   |  1.0      |  11.0     |  78.91    |  5.0      |  0.1      |  50.0     |  0.001    |  0.3      |
0.5      |
=========================================================================================================================
```

Exemple tiré du notebook du Projet P7 :
« 3--P7-Pret_Credit_Conso_03_ Data Cleaning _ Feature Engineering _ Feature Selection _ Modelling »
Source d'explication 1 : *https://distill.pub/2020/bayesian-optimization/*
Source d'explication 2 : *https://machinelearningmastery.com/what-is-bayesian-optimization/*

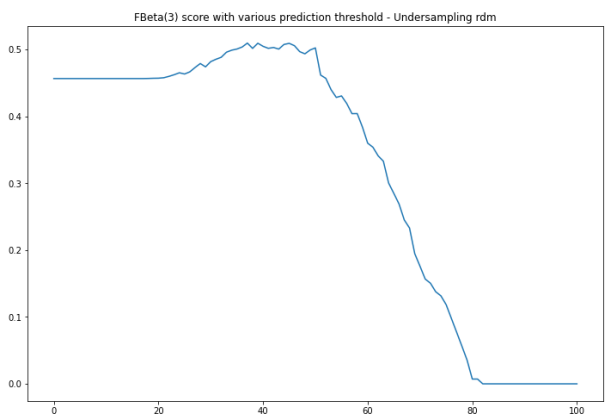
5.7 Résultats des Optimisations réalisées

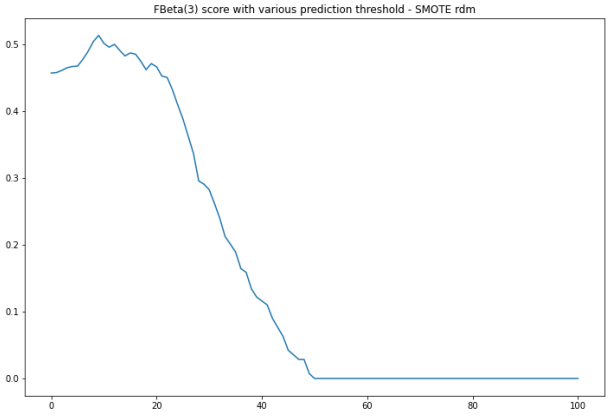


En termes de résultats, on remarque des scores presque équivalents pour 2 méthodes :

- Undersampling avec seuil de probabilité
- Oversampling avec seuil de probabilté

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 :





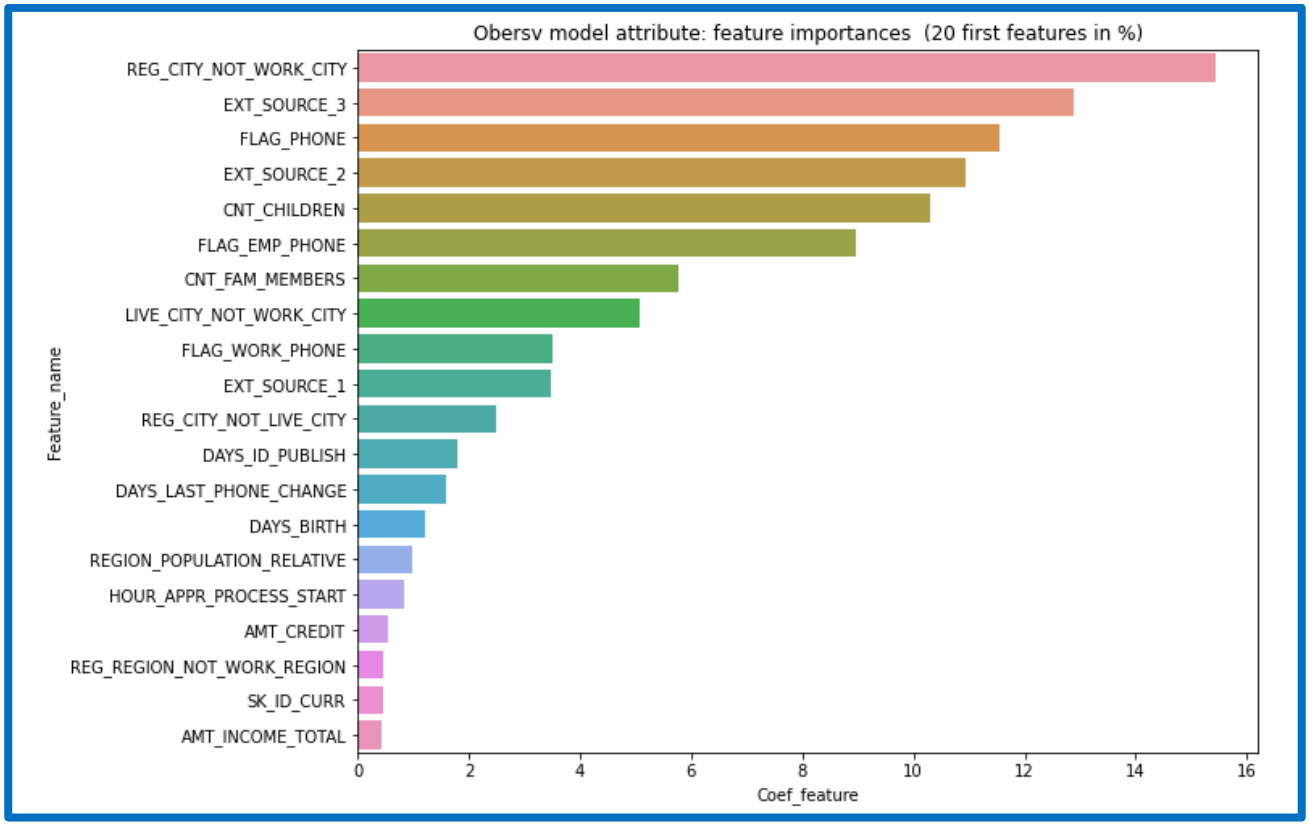
Méthode	*Undersampling avec seuil de probabilité*	*Oversampling avec seuil de probabilté*
Score	0.51	0.51
Seuil	0.37	0.09

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

6 Interprétation du(des) Modèle(s)

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 ou 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 :

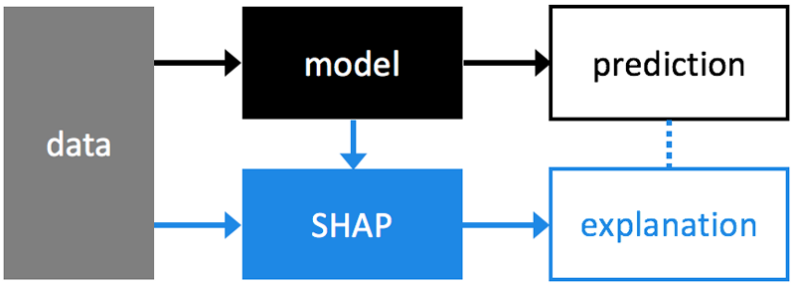


Cet attribut, « gratuit en temps de calcul », a cependant certaines limites :

- 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.

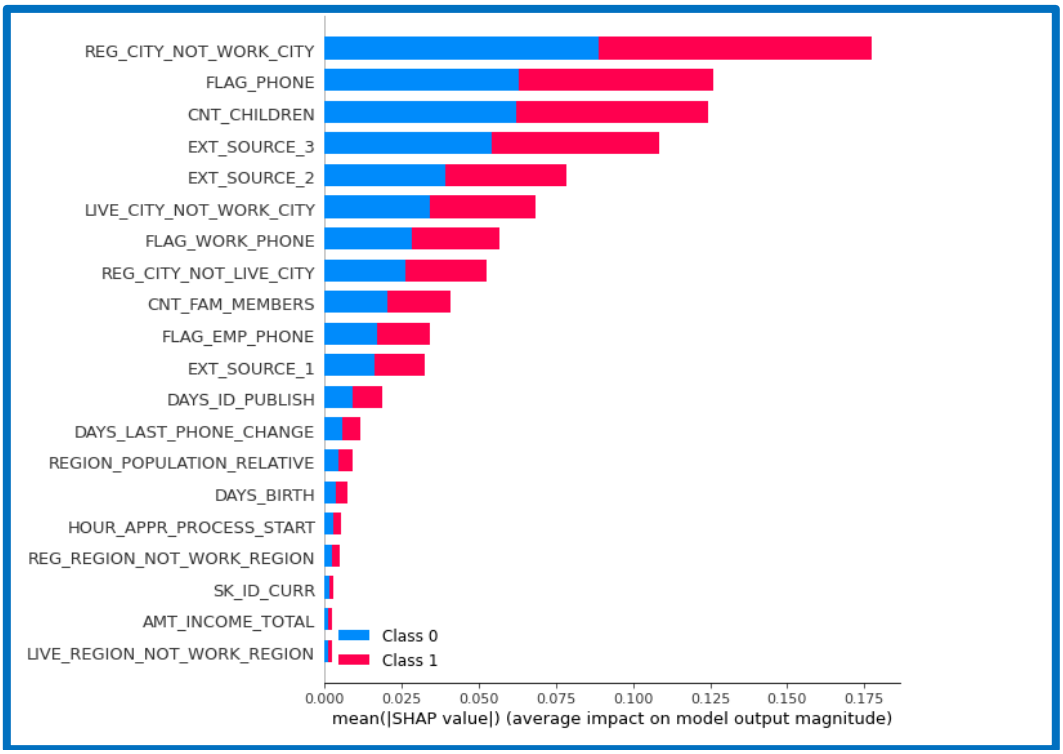
6.1 SHAP / LIME (feature importance / explanations/ interprétabilité)



Source explicative :
http://www.xavierdupre.fr/app/ensae_teaching_cs/helpsphinx/ml2a/td2a_mlplus_interpretabilite_des_modeles.html

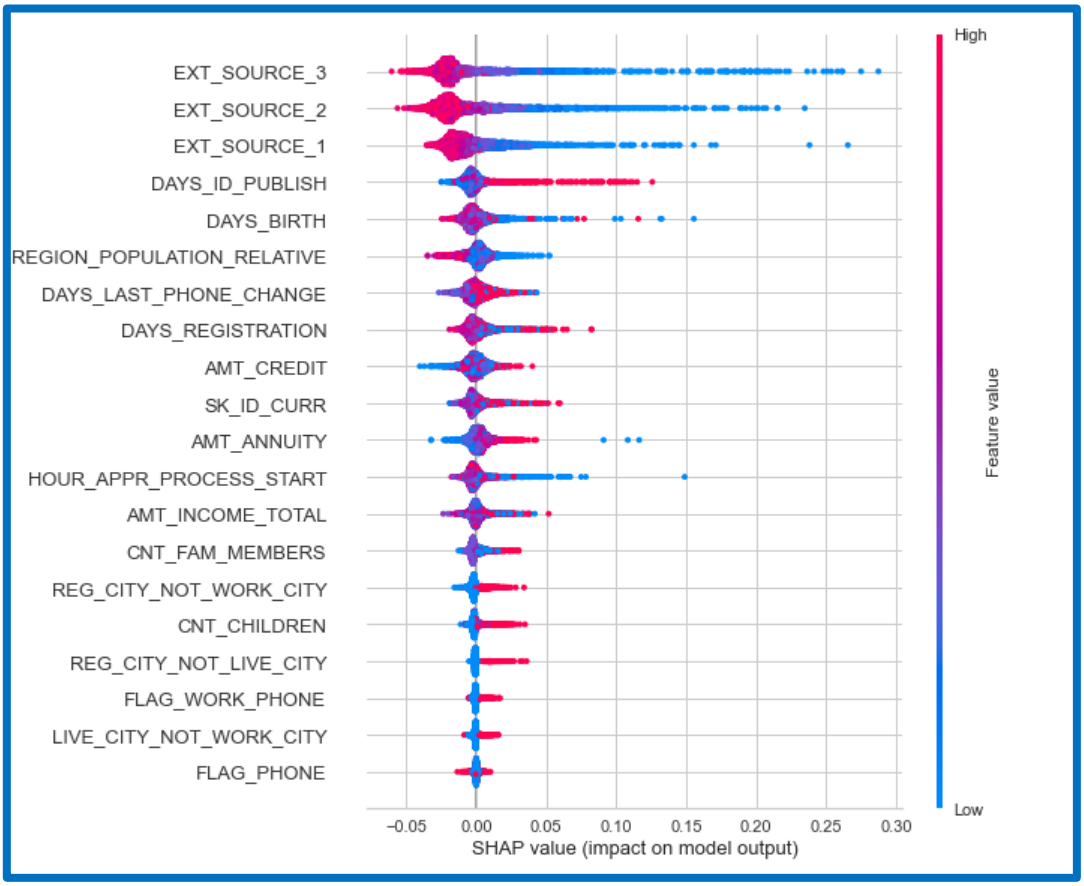
Autre source, un livre complet sur l'interprétabilité :
https://christophm.github.io/interpretable-ml-book/

v. Exemple ici avec le l'interprétation du modèle ***RandomForestClassifier*** :



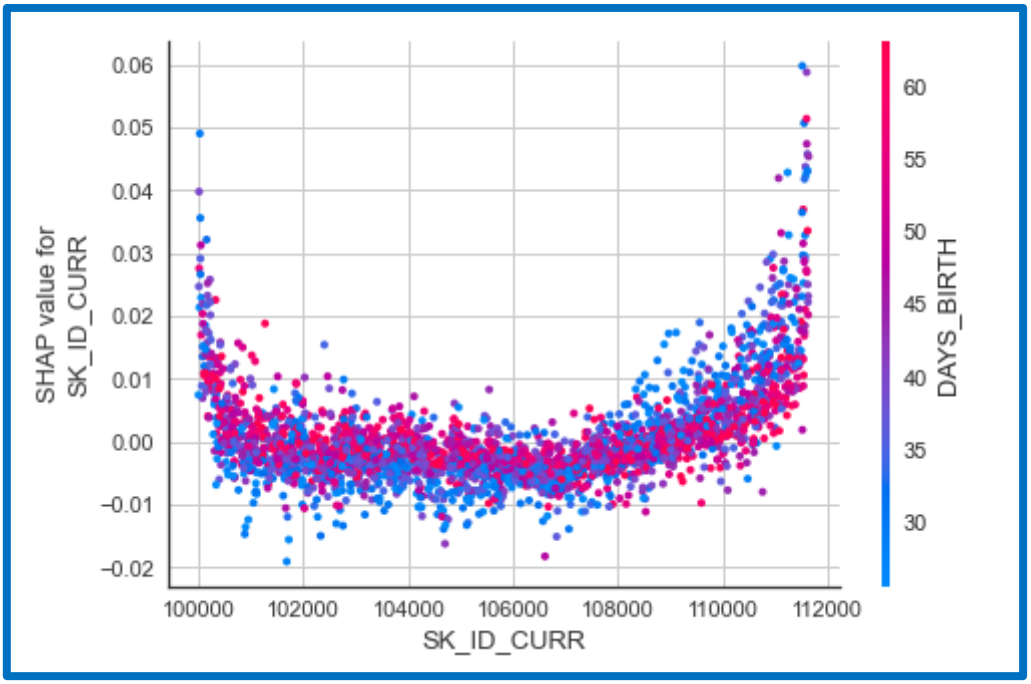
Exemple tiré du notebook du Projet P7 :
« 5--P7-Pret_Credit_Conso_04_Scoring »

w. Autre présentation de l'interpréteur SHAP (modèle ***RandomForestClassifier***) : « **L'Impact** »



Le modèle repose donc fortement sur le scoring déjà utilisé par l'organisme de crédit, et retenu dans les variables EXT_SOURCE.

x. Autre présentation de l'interpréteur SHAP (modèle ***RandomForestClassifier***) : « **Corrélation** »



y. Autre présentation de l'interpréteur SHAP (modèle ***RandomForestClassifier***) : « **Raison** »

7 Dashboard

- Déploiement de l'Application : Streamlit Heroku / StreamLit Github ou Share.streamlit.io

https://towardsdatascience.com/how-to-use-streamlit-to-deploy-your-ml-models-wrapped-in-beautiful-web-apps-66e07c3dd525

7.1 Git/GItHub

Références Github :
https://github.com/slundberg/shap

Individual prediction (pour dashboard par exemple)
https://www.kaggle.com/dansbecker/shap-values

Mon GitHUb :
https://github.com/TVD78370

8 Plan d'Amélioration

z. Plusieurs pistes d'améliorations ont été identifiées :
    i. Feature Engineering complémentaire, avec plusieurs pistes :
        1. Utilisation des datasets de complément proposé dans le jeu de données initial
        2. Création de variables supplémentaires
        3. Optimisation des existantes (suppression, réduction de dimension, augmentation à l'aide des méthodes polynomiales par exemple)
    ii. Ajout de compléments de comparaison sur le dashboard

iii. Construction d'une API qui héberge le modèle et score les clients en temps réel, sur la base d'informations fournies par le chargé de clientèle.
iv. Affiner la métrique (fbeta score) en collaboration avec les équipes métier en se basant sur les pertes / coûts d'opportunité réels du secteur
v. Augmenter le nombre d'individus de la catégorie 1 de notre dataset
vi. Mise en place d'un modèle d'ensemble (Stacking) avec une partie dédiée à l'apprentissage des profils avec défaut de paiement
vii. Revue et amélioration du pre-processing obtenu via un Kernel Kaggle.