

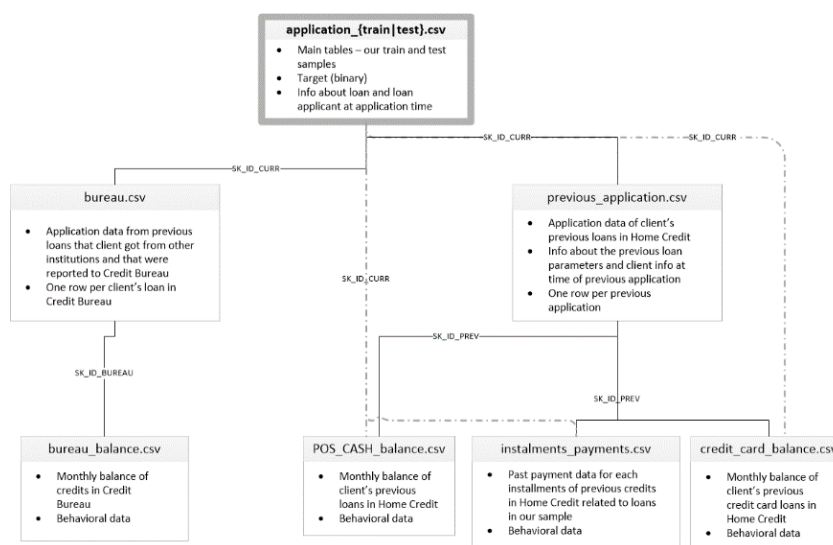
# Note méthodologique

## I. CONTEXTE

Cette note méthodologique constitue l'un des livrables du projet 7 « Implémenter un modèle de *scoring* » proposé dans le cadre de la formation de *Data Scientist* chez OpenClassRooms (OCR). Cette note a pour objectif de décrire le processus de modélisation et d'interprétabilité du modèle développé dans le cadre de ce projet. Pour ce projet, OCR nous propose de nous glisser dans la peau d'un Data Scientist travaillant pour une société financière, « Prêt à dépenser », qui propose des crédits à la consommation pour des personnes ayant peu ou pas du tout d'historique de prêt.

La société souhaite que nous développions un modèle de « *scoring* crédit » pour calculer la probabilité qu'un client rembourse son crédit, puis que notre modèle classe la demande en crédit accordé ou refusé. Elle souhaite donc développer un algorithme de classification en s'appuyant sur des sources de données variées : âge, sexe, emploi, logement, revenus, informations relatives au crédit, notation externe, etc...

Les données sont disponibles sur le site de Kaggle à l'adresse suivante : <https://www.kaggle.com/c/home-credit-default-risk/data>. Les données sont séparées en plusieurs sous fichiers « .csv » et reliés entre eux de la manière suivante (Fig. 1) :



**Fig. 1 : Description des données « Home credit default risk » (Kaggle)**

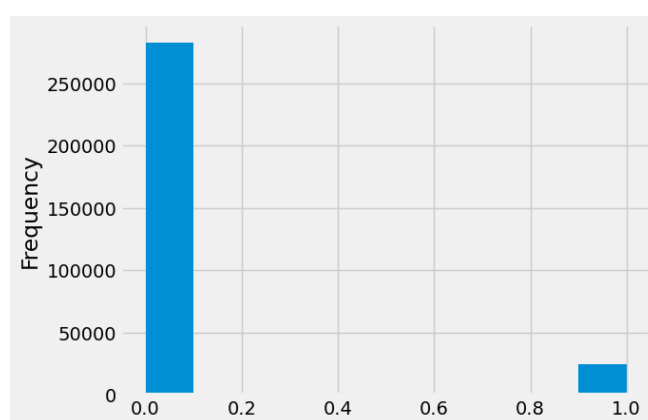
## II. PRE-PROCESSING

Pour faciliter notre travail et ne pas trop s'attarder sur les étapes d'analyse exploratoire, de préparation des données et de *feature engineering*, il nous était possible de consulter différents kernels Kaggle traitant le sujet. Je me suis pour ma part grandement inspiré des *notebooks* de Will Koehrsen disponibles à l'adresse suivante : <https://www.kaggle.com/willkoehrsen>.

## A. ANALYSE EXPLORATOIRE (EDA)

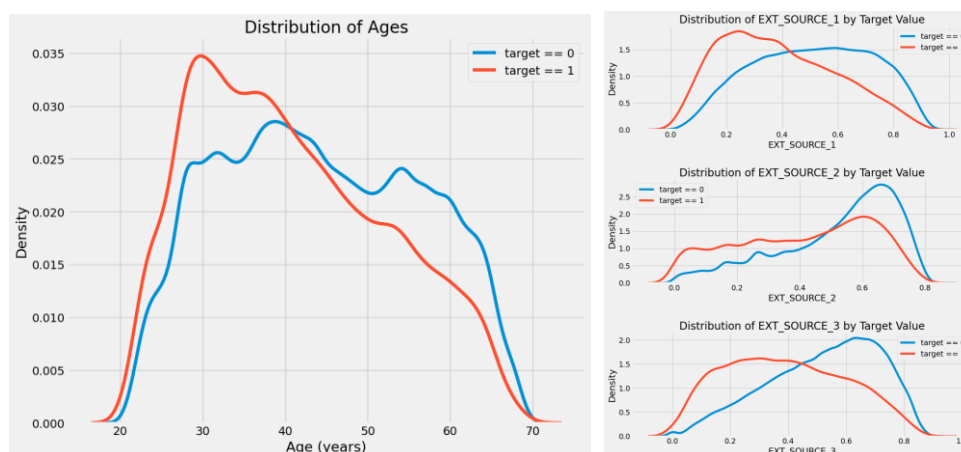
Les données d'entraînements et de tests nous ont été fournis en 2 fichiers « .csv », afin de faciliter leur manipulation : « application\_train.csv » et « application\_test.csv ». Le fichier d'entraînement initial comporte 307511 numéros de clients uniques et 122 *features*, dont une colonne *TARGET* qui correspond à une variable catégorielle : « 0 » pour « solvable » (le client est en capacité de rembourser son prêt) et « 1 » pour « non solvable » (le client n'est pas en mesure de rembourser son prêt). Le fichier de test comporte quant à lui 48744 numéros de clients uniques et 121 *features* (pas de colonne *TARGET*). Le jeu de données « test » représente donc 13,7 % des données totales.

Très vite, on s'aperçoit que les données du jeu d'entraînement est déséquilibré en ce qui concerne les catégories « 0 » et « 1 » de la *feature TARGET* (Fig. 2) : 8% de clients en défaut de paiement contre 92% de clients sans défaut de paiement. Cette problématique (déséquilibre de classes) a été considérée et son traitement est décrit plus loin dans le rapport.

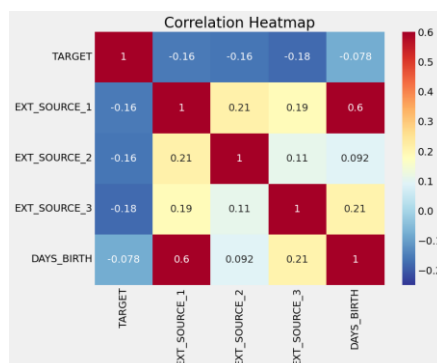


**Fig. 2 : Fréquence des données « 0 » et « 1 » (*feature TARGET*)**

Des variables intéressantes semblent se distinguer telles que « DAYS\_BIRTH » (« âge », en jours) (Fig. 3) ou les sources de revenus : « EXT\_SOURCE\_1 », « EXT\_SOURCE\_2 » et « EXT\_SOURCE\_3 ». Une légère anti-corrélation avec la *feature TARGET* est ainsi observée pour ces variables (Fig. 4). Néanmoins, les 4 variables observées ne corrélaient ou n'anti-corrélaient pas fortement avec la *feature* cible (« TARGET ») : il n'y a pas *a priori* de *data leakage* pour ces données (aucune variable ne corréla à plus de  $|0,2|$  avec la *feature TARGET*).



**Fig. 3 : Distributions suivant l'âge (à gauche) et suivant les ressources externes (à droites)**



**Fig. 3 : Corrélation Heatmap de certaines variables**

## B. PREPARATION DES DONNEES

Afin de réaliser un modèle de *machine learning*, il a été nécessaire de joindre les différentes tables. Un travail d'encodage (*label encoding* ou *one hot encoding*) a été réalisé sur certaines *features* catégorielles (ex : « MALE » => 2 colonnes « MALE » et « FEMALE » avec des valeurs 0 ou 1).

## C. FEATURES ENGINEERING

Le *feature engineering* a eu pour but :

- de corriger certaines données (écarter les valeurs aberrantes ou *outliers*) ;
- de catégoriser certaines variables (« AGE » => valeurs/catégories de 1 à 5) ;
- de créer de nouvelles variables (ex : « EXT\_SOURCES\_WEIGHTED » = « EXT\_SOURCE\_1 » \* 2 + « EXT\_SOURCE\_2 » + « EXT\_SOURCE\_3 » \* 3) ou de supprimer des variables trop vides, notamment.

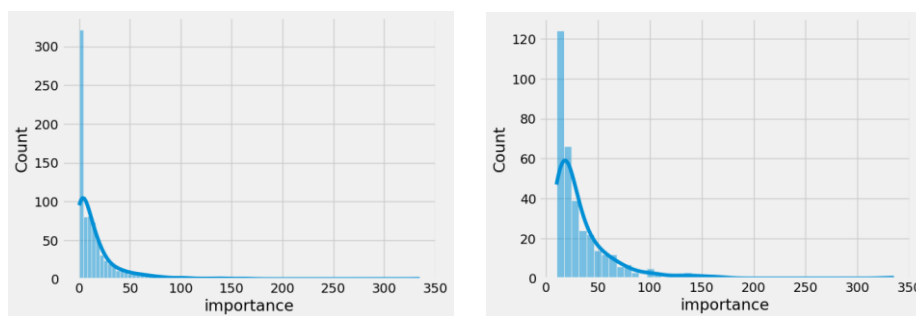
J'ai ainsi obtenu un jeu de données avec 1244 *features* (dont la *feature* TARGET pour le jeu de données d'entraînement).

Concernant un possible *data leakage*, l'hypothèse peut être écartée : il n'y a aucune *features* qui ne corrélaient ou n'anti-corrélaient avec un facteur supérieur à |0,24| avec la variable « TARGET ». Les 5 variables qui sont les plus liées à la variable TARGET sont : « EXT\_SOURCES\_WEIGHTED », « EXT\_SOURCES\_MEAN » et « EXT\_SOURCES\_NANMEDIAN », « APP\_EXT\_SOURCE\_2 \* EXT\_SOURCE\_3 » et « EXT\_SOURCES\_MAX » (anti-corrélation pour les 5).

# III. ENTRAINEMENT DU MODELE

## A. 1<sup>er</sup> ESSAI DU MODELE

L'algorithme utilisé pour ce projet est *LGBMClassifier* (« *Light Gradient Boosting Machine Classifier* »). Un premier essai de création de modèle (très long, bien que seulement les 800 variables les plus corrélées et/ou anti-corrélées à la *feature* TARGET aient été utilisées) a été réalisé et nous a permis de récupérer les *feature importances* du modèle. Une filtration sur les *features* ayant le plus d'importance nous a permis de réduire fortement le nombre de variables, pour passer à 347 *features* (Fig. 4).



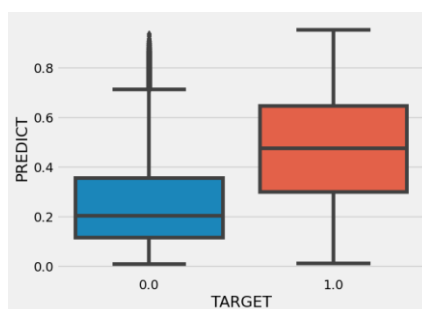
**Fig. 4 : Feature importances avant filtration (à gauche) et après filtration (à droite : seuil > 10)**

## B. GESTION DU DESEQUILIBRE DES CLIENTS 0 OU 1 (TARGET)

La gestion du déséquilibre des données concernant les clients non solvables (classés « 1 » dans la *feature* TARGET) est une étape clé dans le projet. J'ai testé deux approches.

### 1. Méthode 1 : *class\_weight*

Dans un premier temps, j'ai testé la pondération des deux groupes, en optimisant notamment un des paramètres de l'algorithme de *LGBMClassifier* : *class\_weight* = {0: 0.15, 1: 0.85}. La distribution des valeurs prédites (entre 0 et 1) pour les 2 groupes par le nouveau modèle (*model\_met1*) est ainsi représentées (Fig. 5).



**Fig. 5 : Distributions des valeurs prédites avec le *model\_met1***

Pour la méthode 1, les scores des métriques (5 permutations par cross-validation) nous donne les résultats suivants (Fig. 6) :

```

Training Data Shape: (387506, 347)
Testing Data Shape: (48744, 347)
[200] train's auc: 0.793118 train's binary_logloss: 0.510724 valid's auc: 0.778918 valid's binary_logloss: 0.382436
[400] train's auc: 0.810427 train's binary_logloss: 0.494575 valid's auc: 0.783694 valid's binary_logloss: 0.376615
[200] train's auc: 0.792651 train's binary_logloss: 0.51087 valid's auc: 0.780798 valid's binary_logloss: 0.382277
[400] train's auc: 0.810128 train's binary_logloss: 0.494652 valid's auc: 0.786338 valid's binary_logloss: 0.375976
[200] train's auc: 0.79353 train's binary_logloss: 0.510011 valid's auc: 0.776155 valid's binary_logloss: 0.383366
[400] train's auc: 0.811011 train's binary_logloss: 0.493808 valid's auc: 0.780939 valid's binary_logloss: 0.37723
[200] train's auc: 0.792373 train's binary_logloss: 0.510994 valid's auc: 0.784502 valid's binary_logloss: 0.382648
[400] train's auc: 0.809737 train's binary_logloss: 0.494975 valid's auc: 0.788989 valid's binary_logloss: 0.376539
[200] train's auc: 0.793713 train's binary_logloss: 0.509457 valid's auc: 0.775138 valid's binary_logloss: 0.384034
[400] train's auc: 0.811353 train's binary_logloss: 0.493129 valid's auc: 0.780399 valid's binary_logloss: 0.377709

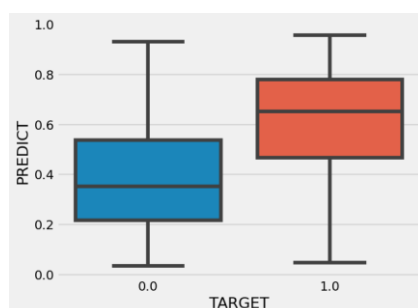
```

**Fig. 6 : Résultats des métriques pour chaque permutation (*n\_folds* = 10)**

En moyenne, les scores des métriques ont été les suivants : *train\_auc* = « 0,80 » ; *train\_binary\_logloss* = « 0,50 » ; *valid\_auc* = « 0,78 » et *valid\_binary\_logloss* = « 0,38 ». Le temps de calcul était d'environ 19 min (*n\_folds* = 10). Les valeurs d'AUC des données *train* et *valid* sont semblables pour les 10 itérations : nous pouvons en déduire qu'il ne semble pas y avoir d'*overfitting*.

## 2. Méthode 2 : *undersampling*

Dans un deuxième temps, j'ai testé la méthode dite de « *undersampling* » (réduction d'échantillons du groupe 0 pour atteindre le même nombre que ceux du groupe 1). Après optimisation du nouveau modèle créé (model\_met2), j'ai obtenu la distribution suivante des valeurs prédites pour les deux groupes d'individus (Fig. 7) :

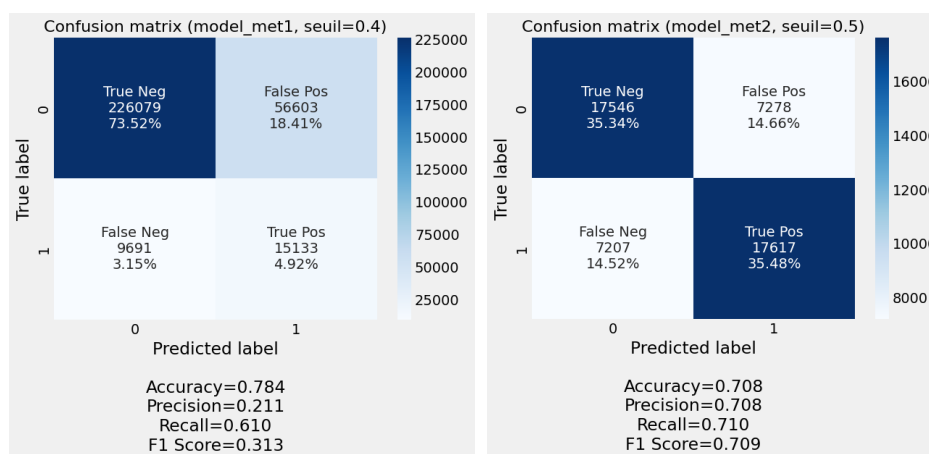


**Fig. 7 : Distributions des valeurs prédites avec le model\_met2**

En moyenne, les scores des métriques ont été les suivants : train\_auc = « 0,82 » ; train\_binary\_logloss = « 0,53 » ; valid\_auc = « 0,78 » et valid\_binary\_logloss = « 0,56 ». Le temps de calcul était d'environ 2 min.

## 3. Comparaison des 2 méthodes

La comparaison peut se faire en comparant les résultats des 2 matrices de confusion, à des seuils respectifs donnés (« 0,4 » pour le model\_meth1 et « 0,5 » pour le model\_meth2) (Fig. 8) :



**Fig. 8 : Matrices de confusion pour les prédictions obtenues avec le model\_meth1 (à gauche) et avec le model\_meth2 (à droite)**

Si je me fie à l'exactitude ( $Accuracy = (TN+TP)/(TN+TP+FN+FP)$ ), j'ai obtenu un meilleur pouvoir prédictif avec la méthode 1 (« *class\_weight* ») avec une *accuracy* de 0,784 (contre 0,708 avec l'autre méthode 2). De plus, la méthode 1 nous permet de minimiser les FN (3,15% contre 14,5%) : groupe que l'on souhaite minimiser au maximum (clients classés comme solvables alors qu'ils ne le sont pas en réalité). J'ai donc retenu le model\_met1 pour la suite du projet, bien que le temps de calcul soit beaucoup plus long (19 min pour  $n\_folds = 10$  et contre 2 min pour  $n\_folds = 5$ ).

Concernant le pouvoir prédictif du `model_met1` choisi, il est bien entendu inférieur à un modèle simple de référence (*dummyClassifier*) qui consisterait à classer tous les individus en 0 (solvable). Dans ce cas simple, cela donnerait un score prédictif de 0,92 : soit la proportion d'individus « 0 » dans le jeu de données initial (voir Fig. 1).

#### IV. GESTION DU COUT METIER ENTRE FN ET FP

Nous avons vu précédemment que suivant le seuil que l'on fixe (seuil de probabilité qui détermine si le client est classé en 0 ou en 1), les résultats de classification (et *d'accuracy* en particulier) changent. Il est donc nécessaire de créer une fonction afin d'évaluer un *scorer* qui nous permet de minimiser la prédiction des FN et des FP.

Un *scorer* contenant la fonction coût métier a ainsi été mis en place. Les étapes suivantes ont été répétées, après entraînement du modèle `model_met1` (métrique locale : AUC) :

- Etape 1 : test des seuils de probabilité de défaut par pas de 0.01 sur le modèle LGBM entraîné ;
- Etape 2 : obtention d'une matrice de confusion pour chaque seuil ;
- Etape 3 : recherche de la matrice de confusion (parmi celles obtenues à l'étape 2) qui minimise la fonction de coût.

Les hypothèses retenues pour les candidats aux crédits sont les suivantes (nous cherchons à minimiser au maximum les FN) :

Objet	Coût par client (unité arbitraire)	Classe
Octroi de crédit à un client qui fait défaut	100	FN (False Negative)
Octroi de crédit à un client qui ne fait pas défaut	-10	TN (True Negative)
Refus de crédit à un client qui aurait fait défaut	0	TP (True Positive)
Refus de crédit à un client qui n'aurait pas fait défaut	0	FP (False Positive)
Frais généraux pour chaque client	1	-

La fonction de coût, normalisée à un client, est la suivante :

$$\text{Coût} = \frac{100 * FN - 10 * TN + 1 * (TP + TN + FP + FN)}{TP + TN + FP + FN}$$

Le seuil optimal déterminé est donc de « 0,35 », comme le montre la courbe de coût normalisé suivante (Fig. 9) :

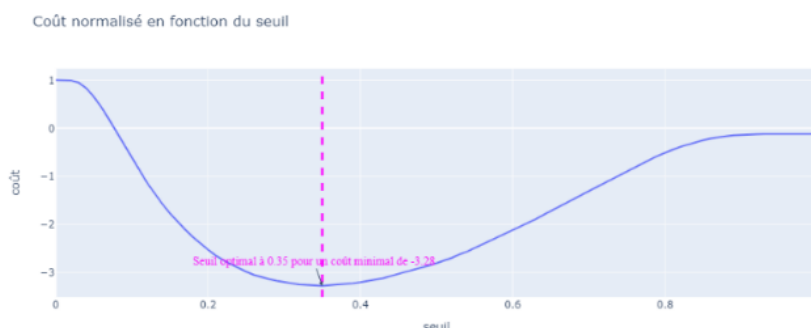
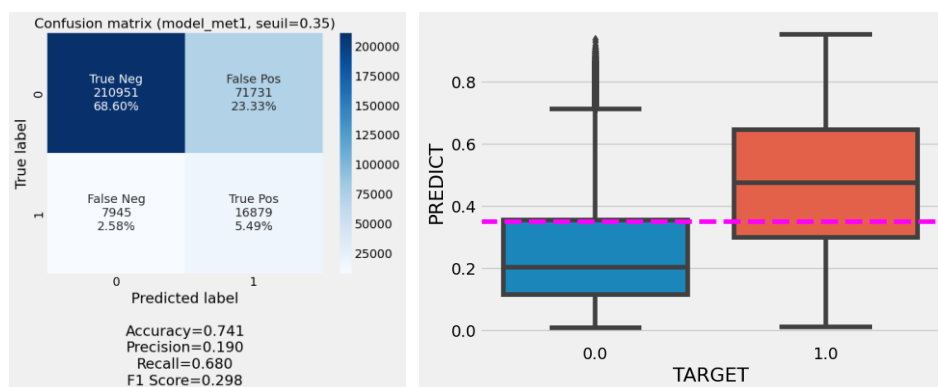


Fig. 9 : courbe de coût métier normalisé (`model_met1`)

Concernant la matrice de confusion et les distributions des valeurs prédites cela donne les résultats suivants (Fig. 10), avec les FN minimisées (2,6%) :

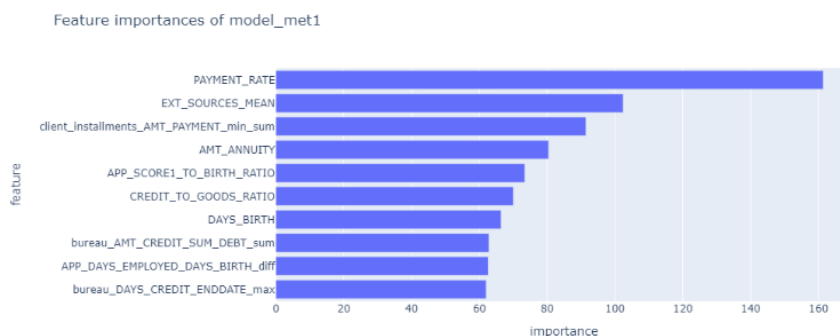


**Fig. 10 : Matrice de confusion avec le seuil optimisé (à gauche) et distribution des valeurs prédites (à droite)**

## V. INTERPRETABILITE GLOBALE ET LOCALE DU MODELE CHOISI

### A. INTERPRETABILITE GLOBALE : FEATURE IMPORTANCES

J'ai tracé les *feature importances* du modèle, autrement dit les variables qui ont le plus participé à la création du modèle (Fig. 11) :



**Fig. 11 : Feature importances du model\_met1**

Je me suis focalisé sur les 5 premières *features*. Assez majoritairement, la *feature* « PAYMENT\_RATE » joue un grand rôle dans la construction du model\_met1. Cette variable est un *ratio* ( $PAYMENT\_RATE = AMT\_ANNUITY/AMT\_CREDIT$ ) et a été généré par *feature engineering*. Cela correspond à la rente d'emprunt (montant total de remboursement étalé sur plusieurs mois) divisé par le montant du crédit, pour chaque emprunteur. La *feature* « EXT\_SOURCES\_MEAN » correspond à la moyenne des 3 *features* « EXT\_SOURCES » (scores normalisés des 3 différentes sources de revenus). La *feature* « client\_installments\_AMT\_PAYMENT\_min\_sum » est issue de la *feature* « AMT\_PAYMENT » qui correspond à ce que le client a réellement payé sur le crédit précédent pour ce versement. Enfin, la *feature* « APP\_SCORE1\_TO\_BIRTH\_RATIO » correspond au *ratio* suivant : « EXT\_SOURCE\_1'/DAYS\_BIRTH/365.25 ». Il traduit l'apport de « EXT\_SOURCE\_1 » rationalisé par l'âge de l'emprunteur (en années).

### A. INTERPRETABILITE LOCALE : SHAP VALUES

La technique SHAP locale explique la prédiction pour chaque client en calculant la contribution de chaque *feature* à la prédiction. Par exemple pour l'individu n°208550 (jeu de test), les contributions des dix *features* qui impactent le plus sa probabilité d'appartenir à la classe « 0 » (solvable) ou « 1 » (non solvable) sont les suivantes (Fig. 12) :

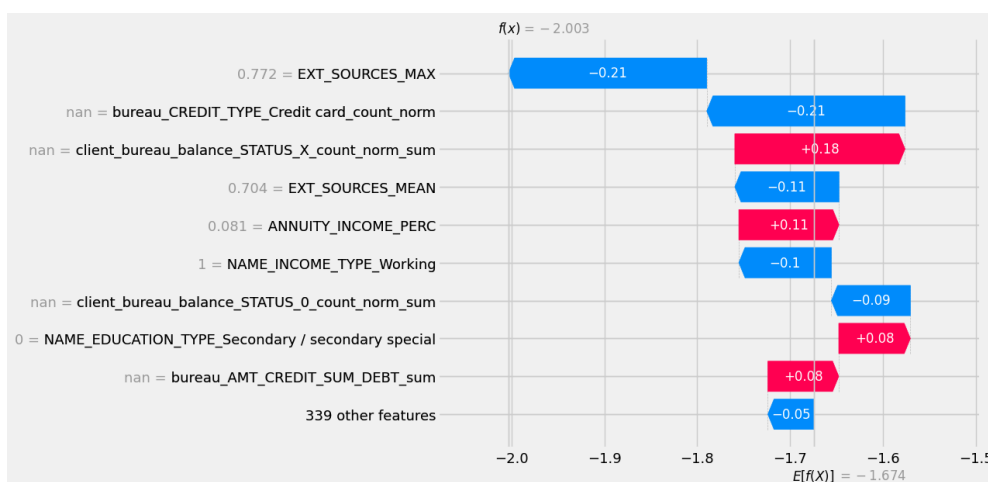


Fig. 12 : SHAP values de l'individu n°208550

Le score de prédiction de l'individu n°208550 est de « 0,08 » : autrement dit, il a 92% de chance d'appartenir au groupe solvable (« 0 »). La *feature* qui contribue le plus à ce score est « EXT\_SOURCE\_MAX », qui correspond à l'« EXT\_SOURCE » la plus élevée parmi les 3 *features* « EXT\_SOURCE ». Concernant cet individu, la valeur de cette *feature* est de « 0,772 » (qui correspond concrètement à la *feature* « EXT\_SOURCE\_2 » dans son cas). Une valeur élevée de cette variable tend à faire diminuer le score de prédiction (*feature* en bleue dans le graphique pour « diminution »), donc à ce que les individus tendent vers le groupe « 0 ». De même, la *feature* « EXT\_SOURCES\_MEAN » (moyenne des 3 *features* « EXT\_SOURCE\_ ») est assez élevée chez cet individu (en bleue aussi dans le graphique) : ce qui semble suggérer qu'il a des apports conséquents pour être solvable (classé dans le groupe « 0 »). Une majorité d'autres *features* ont des valeurs absentes (« NaN ») pour cet individu. Enfin, le fait que l'individu ait un travail (*feature* « NAME\_INCOME\_TYPE\_Working » = 1) tend à faire diminuer le score de prédiction : les banques prêtent plus facilement aux personnes avec un emploi que sans !

## VI. LIMITES ET AMELIORATIONS POSSIBLES DU MODELE

Le modèle a été entraîné en suivant des métriques tels que le ROC AUC (à maximiser) et le *logloss* (à minimiser) lors de la descente de gradient. Dans notre cas, l'axe d'amélioration principal serait de déterminer plus finement (pour ne pas dire plus intelligemment) une fonction coût métier plus proche de la réalité : en concertation avec des spécialistes du métier (le prêt bancaire).

Dans la même idée, il serait plus pertinent de bien connaître la signification des *features* traitées, notamment lors de l'étape de *feature engineering* : une expertise métier de ces *features* serait fortement appréciée. A titre d'exemple, la *feature* « APP\_EXT\_SOURCE\_2 \* EXT\_SOURCE\_3 \*

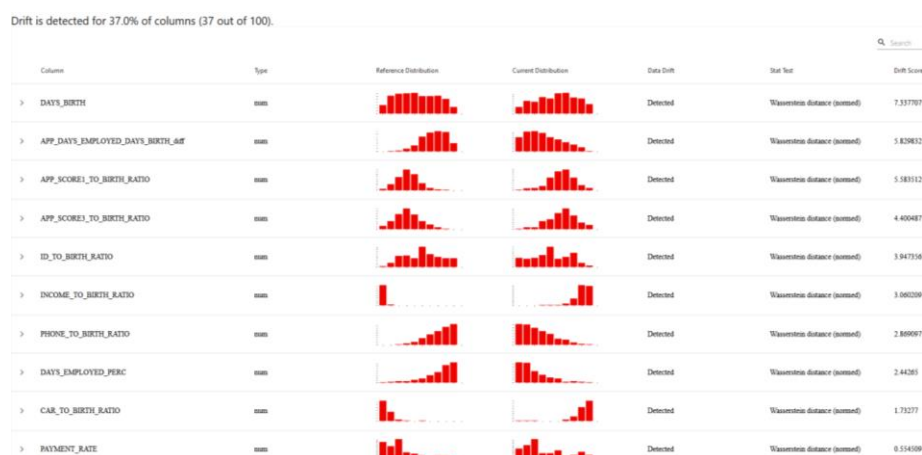


DAYS\_BIRTH » (qui est égale à : « EXT\_SOURCE\_1 » \* « EXT\_SOURCE\_2 » \* « DAYS\_BIRTH ») a-t-elle un sens ? Est-ce une donnée qui parle aux personnes travaillant dans les prêts bancaires ?

Enfin, pour que le modèle puisse gagner en exactitude (« *Accuracy* »), il faudrait beaucoup plus de données clients non solvables (classe « 1 »). En effet, le modèle est entraîné sur un jeu de données beaucoup trop déséquilibré (92% d'individus « 0 »). Le traitement de déséquilibre des classes a bien été effectué (les méthodes *class\_weight* et *undersampling* ont été testés ici ; j'aurais pu aussi tester de l'*oversampling* tel que la méthode SMOTE), mais le modèle finalement développé ne prédit pas mieux qu'un modèle naïf qui consisterait à tout classer dans le groupe des « 0 » : 92% d'exactitude, contre 78% avec le modèle *model\_met1* développé.

## VII. ANALYSE DU DATA DRIFT

J'ai utilisé la librairie *evidently* afin de notamment évaluer un possible *data drift* sur les 10 premières *features* ayant le plus d'importance dans la construction du *model\_met1*. J'ai utilisé un jeu de données d'entraînement et de test réduit ( $n = 5000$  et  $n = 500$ , respectivement) afin de gagner en temps de calcul. A noter que ce jeu de données réduits a été utilisé pour la suite du projet (pour la construction du *dashboard* notamment). L'analyse montre 6/7 tests sont validés. Seul le test du *data drift* n'est pas passé (37% des *features* testées présentes du *data drift*) (Fig. 13) :



**Fig. 13 : Capture d'écran illustrant les 10 features ayant le plus de data drift (model\_met1)**

Beaucoup de ces *features* sont liées à l'âge de l'individu (« DAYS\_BIRTH » exprimée en jours). Cela semble suggérer que les individus du jeu de données test sont globalement plus âgés que ceux du jeu de données d'entraînement (qui a permis de développer le modèle). Cela pose un souci dans le sens où le facteur « âge » est un facteur important dans l'obtention d'un prêt (voir Fig. 3).

De même, la *feature* « PAYMENT\_RATE » possède du *data drift*, alors qu'elle est la *feature* qui a le plus d'importance dans la construction du *model\_met1*. Les valeurs de cette variable semblent en effet plus importantes dans le jeu de données test que dans le jeu de données d'entraînement. Nous avons observé qu'une valeur plus élevée de cette variable favorise l'obtention d'un prêt (classement dans le groupe « 0 »).

En observant ces résultats, nous pourrions suggérer un réentraînement du modèle avec de nouvelles données.