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 classifie 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) :

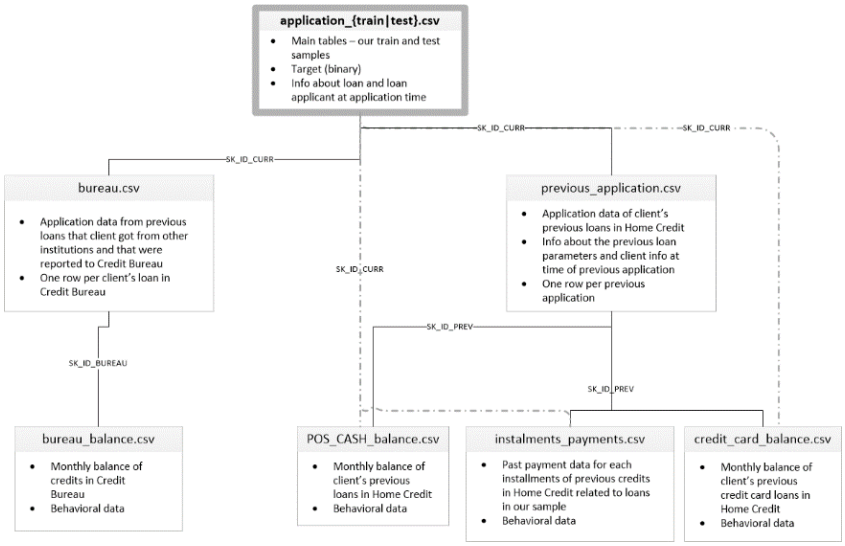

Fig. 1 : Description des données « Home credit default risk » (Kaggle)

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'entrainements 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'entrainement 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'entrainement 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.

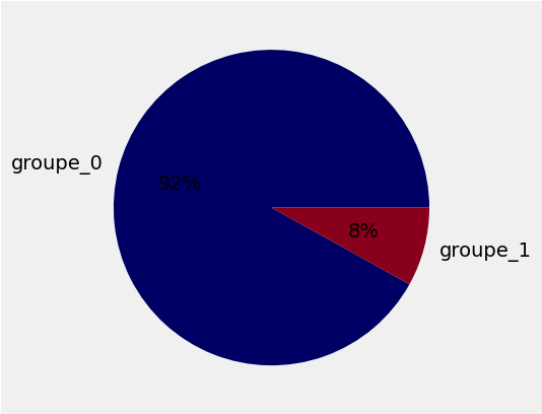

Fig. 2 : Proportions des données « 0 » et « 1 » (feature TARGET)

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_2 ». Une légère anti-corrélation avec la *feature* « TARGET » est ainsi observée pour ces variables (Fig. 4). Néanmoins, les 4 variables observées ne corrèlent ou n'anti-corrèlent pas fortement avec la *feature* cible (« TARGET ») : il n'y a pas *a priori* de *data leakage* pour ces données (aucune variable ne corrèle à plus de |0,2| avec la *feature* TARGET.

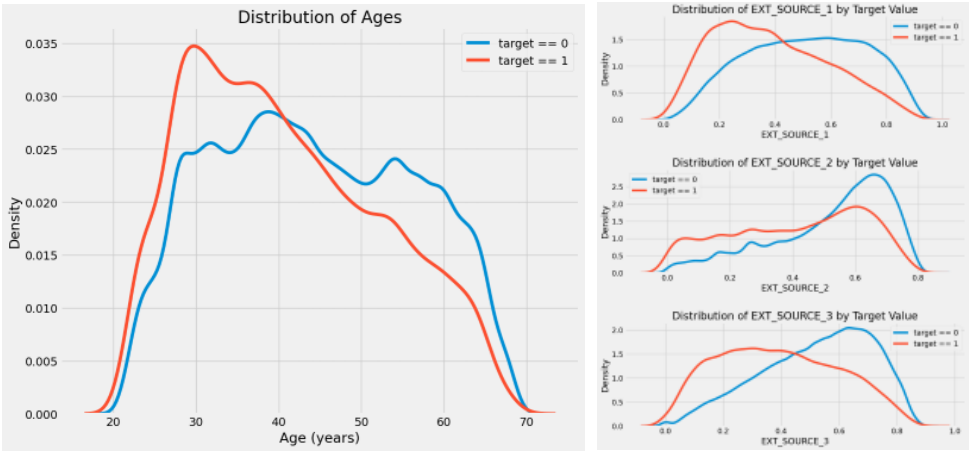

Fig. 3 : Distributions suivant l'âge (à gauche) et suivant les ressources externes (à droites)

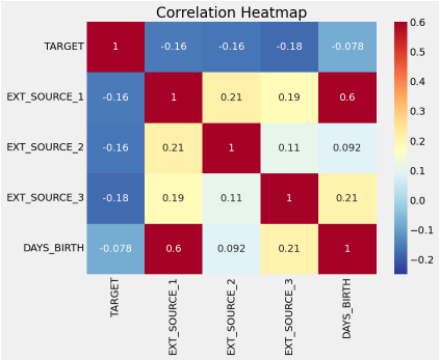

Fig. 3 : Corrélation Heamap de certaines variables

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'entrainement).

Concernant un possible *data leakage*, l'hypothèse peut être écartée : il n'y a aucune features qui ne corrèlent ou n'anti-corrèlent avec un facteur supérieur à |0,69| 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_PROD » (anti-corrélation pour les 5).

III. ENTRAINEMENT DU MODELE

A. 1er 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).

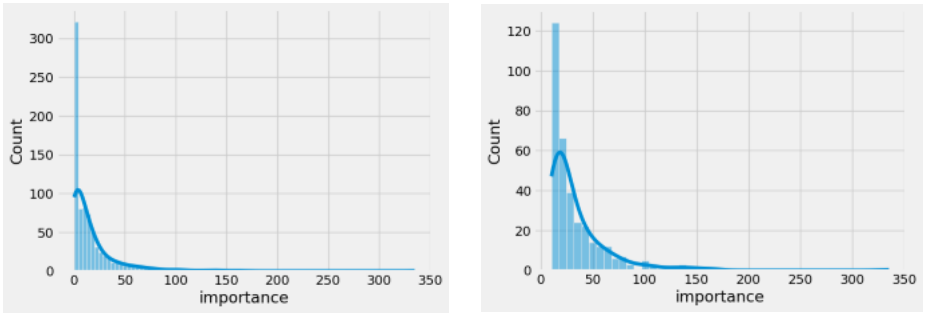

Fig. 4 : Feature importances avant filtration (à gauche) et après filtration (à droite : seuil > 10)

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).

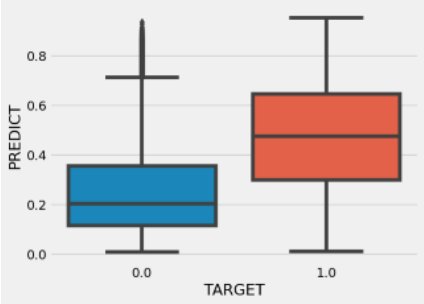

Fig. 5 : Distributions des valeurs prédites avec le model_met1

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

```
Training Data Shape:  (307506, 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 crée (model_met2), j'ai obtenu la distribution suivante des valeurs prédites pour les deux groupes d'individus (Fig. 7) :

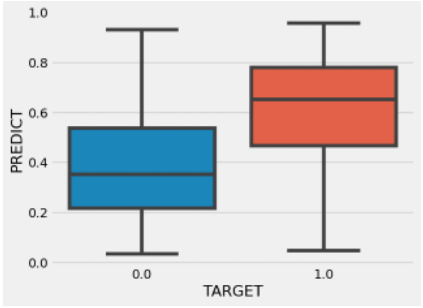

Fig. 7 : Distributions des valeurs prédites avec le model_met2

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) :

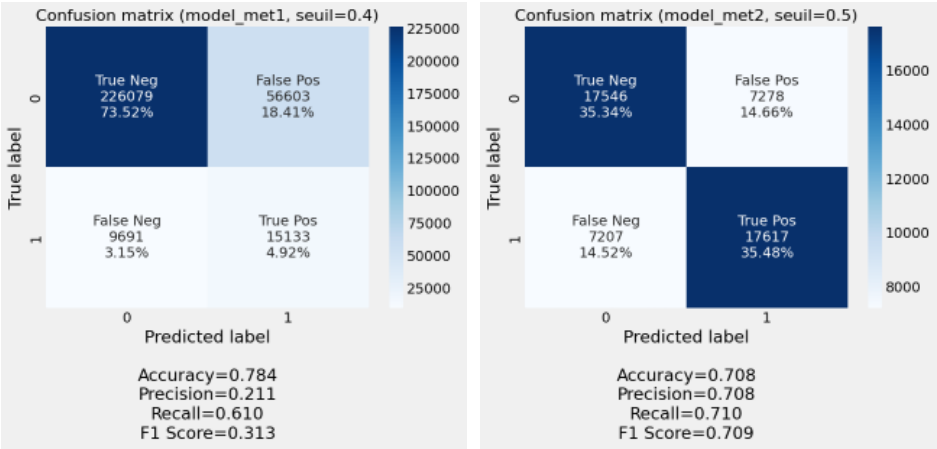

Fig. 8 : Matrices de confusion pour les prédictions obtenues avec le model_meth1 (à gauche) et avec le model_meth2 (à droite)

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 model simple de référence (*dummyClassifier*) qui consisterait à classer tous les individus en 0 (solvables). 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 entrainement 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 entrainé ;
- 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 : $Coût = \frac{100*FN-10*TN+1*(TP+TN+FP+FN)}{TP+TN+FP+FN}$. Le seuil optimal déterminé est donc de « 0,35 », comme le montre la courbe de coût normalisé suivante (Fig. 9) :

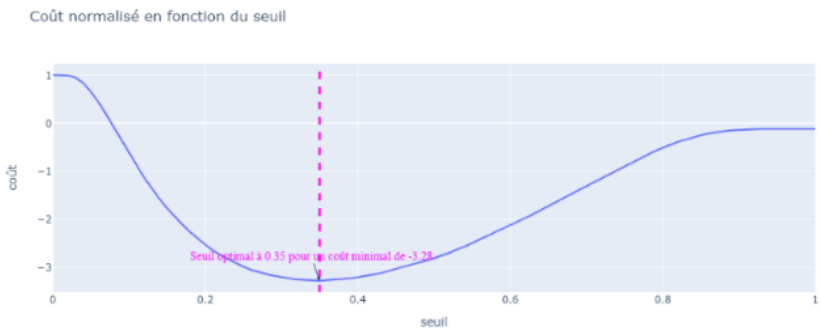

Fig. 9 : courbe de coût métier normalisé (model_met1)

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

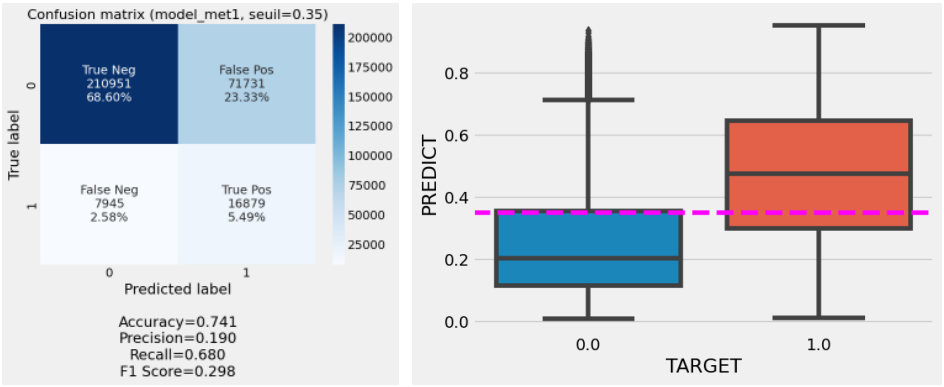

Fig. 10 : Matrice de confusion avec le seuil optimisé (à gauche) et distribution des valeurs prédites (à droite)

V. Réentrainement du modèle choisi et *pipeline* continu

Afin d'assurer une intégration continue, une distribution continue et un déploiement continue (approche CI/CD) de la future API hébergent le modèle de prédiction, il a été nécessaire de ré-entrainer le modèle choisi (i.e. utilisation de *LGBM Classifier* avec une pondération des classes) en intégrant l'optimisation des hyperparamètres (dont le 'class_weight') et le seuil optimal dans un *pipeline* continu : les résultats ont été sauvegardés par l'outil *MLFlow*. J'ai ainsi rajouté dans mon nouveau *Notebook* intitulé « Desoubzdanne_Denis_2_notebook_122023.ipynb » le chapitre XVII « Pipeline de déploiement continu ». Le nouveau modèle obtenu nous donne ainsi comme résultats la matrice de confusion et les distributions des valeurs prédites suivantes, pour un seuil à 0,4038 et un *class_weight* = {0: 0.08, 1: 0.92} (Fig. 11) :

Fig. 11 : Matrice de confusion avec le seuil optimisé (à gauche), ROC_AUC (au milieu) et distribution des valeurs prédites (à droite) du modèle ré-entrainer et intégrer dans un pipeline continu

Les résultats sont sensiblement les mêmes (les scores d'« accuracy » sont les mêmes), avec un seuil un peu plus élevés cette fois-ci à 0,4038 (au lieu de 0.35 précédemment). C'est ce modèle nouvellement entrainé qui a été utilisé par la suite (et notamment avec l'API développée). Ce modèle est appelé tout simplement « model_ref ».

VI. INTERPRETABILITE GLOBALE ET LOCALE DU MODELE CHOISI

A. INTERPRETABILITE GLOBALE : *SHAP VALUES*

L'interprétabilité globale du « model_réf » a été effectuée en traçant les *Shap values* du modèle en question (Fig. 12) :

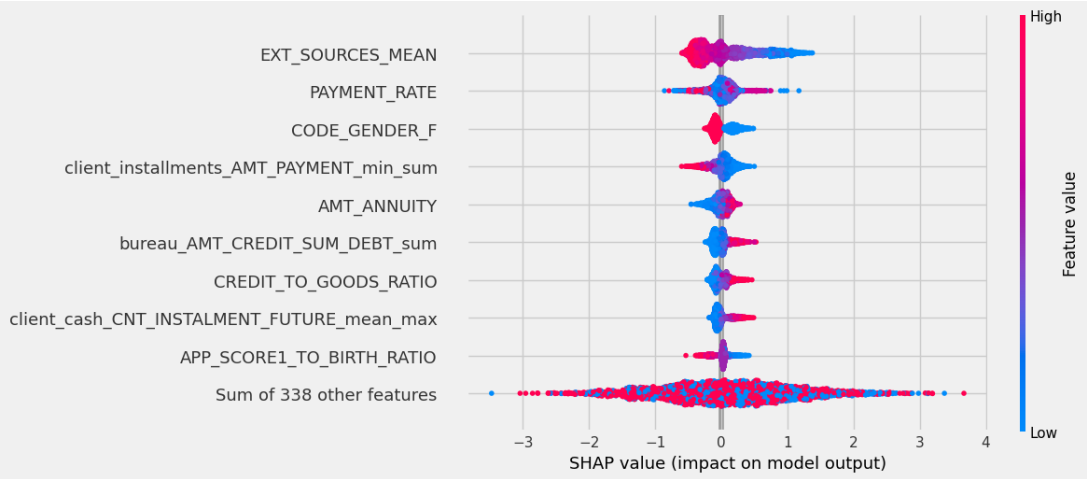

Fig. 12 : Shap values (beeswarm) du model_ref

Je me suis focalisé sur les 5 premières *features* qui contribuent le plus à la création du modèle. Assez majoritairement, la *feature* « EXT_SOURCES_MEAN » apparaît comme étant la variable qui participe le plus à la construction du modèle. Cette variable est la moyenne de 3 autres variables : « EXT_SOURCE_1, _2 & _3 » (générée par *feature engineering*). Elles correspondent aux 3 sources de revenus possibles d'un client. Plus la valeur de la variable « EXT_SOURCES_MEAN » est élevée (en rouge), plus le modèle tend à prédire des valeurs proches de 0 (sur une échelle de 0 à 1). La feature « PAYMENT_RATE » joue aussi un grand rôle dans la construction du modèle. Cette variable est un *ratio* (*PAYMENT_RATE = AMT_ANNUITY/AMT_CREDIT*) et a aussi 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* « CODE_GENDER_F » correspond au genre du client (si oui ou non il est une femme) : une valeur positive (client féminin) tendra à prédire un score proche de 0 (clients solvables). La *feature* « client_installments_AMT_PAYMENT_min_sum » est issue de la *feature* « AMT_PAYMENT » : cette dernière correspond à ce que le client a réellement payé sur le crédit précédent pour ce versement. Une valeur élevée tendra à prédire une valeur proche de 1 (clients non solvables).

B. 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° 351831 (*feature* « SK_ID_CURR » du jeu de test restreint : « test_samp.csv »), 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. 13) :

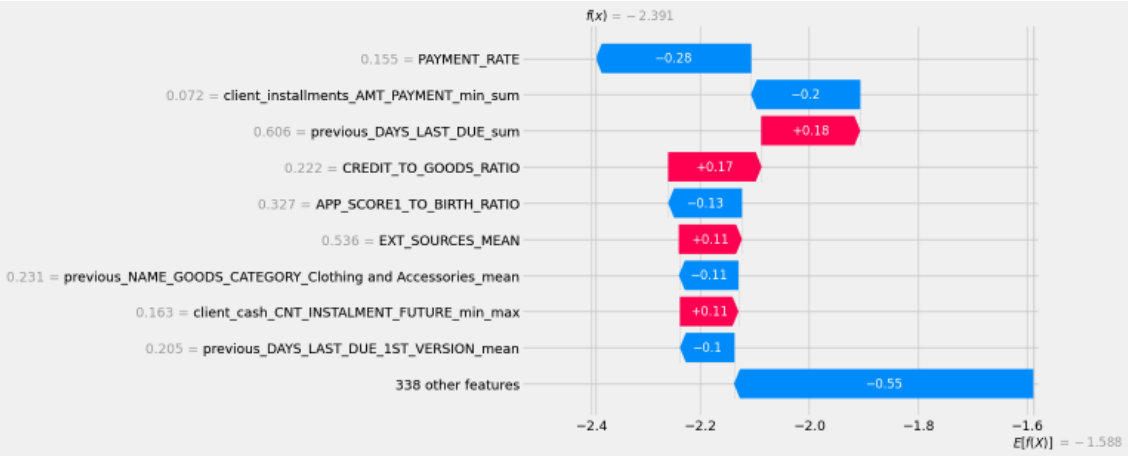

Fig. 13 : SHAP values (waterfall) de l'individu n°351831

Le score de prédiction de l'individu n°351831 est de « 0,09 » : autrement dit, il a 91% de chance d'appartenir au groupe solvable (« 0 »). La *feature* qui contribue le plus à ce score est le « PAYMENT_RATE » (voir ci-dessus). Cette valeur est assez faible pour cet individu. Une valeur relativement faible 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 ». Il en est de même pour la *feature* « client_installments_AMT_PAYMENT_min_sum ». En revanche, les *features* « previous_DAYS_LAST_DUE_sum » et « CREDIT_TO_GOODS_RATIO » ont des valeurs assez élevées et tendent à ce que l'individu en question ait un score de prédiction plus proche de 1 (non solvable). Enfin, la *feature* « EXT_SOURCES_MEAN » est assez défavorable pour cet individu : ce qui semble suggérer que ces revenus soient relativement modestes (en-dessous de la médiane des individus solvables), bien qu'il soit classé dans le groupe 0.

VII. LIMITES ET AMELIORATIONS POSSIBLES DU MODELE

Le modèle a été entrainé 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 entrainé 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 74% avec le model_réf développé.

VII. ANALYSE DU *DATA DRIFT*

J'ai utilisé la librairie *evidently* afin de notamment évaluer un possible *data drift* sur l'ensemble des *features* ayant permis la construction du model_réf. J'ai utilisé un jeu de donné d'entrainement et de test réduit (n = 3076 et n = 488, 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 que 6/7 des tests sont validés. Seul le test du *data drift* n'est pas passé (35% des *features* testées présentes du data drift) (Fig. 14) :

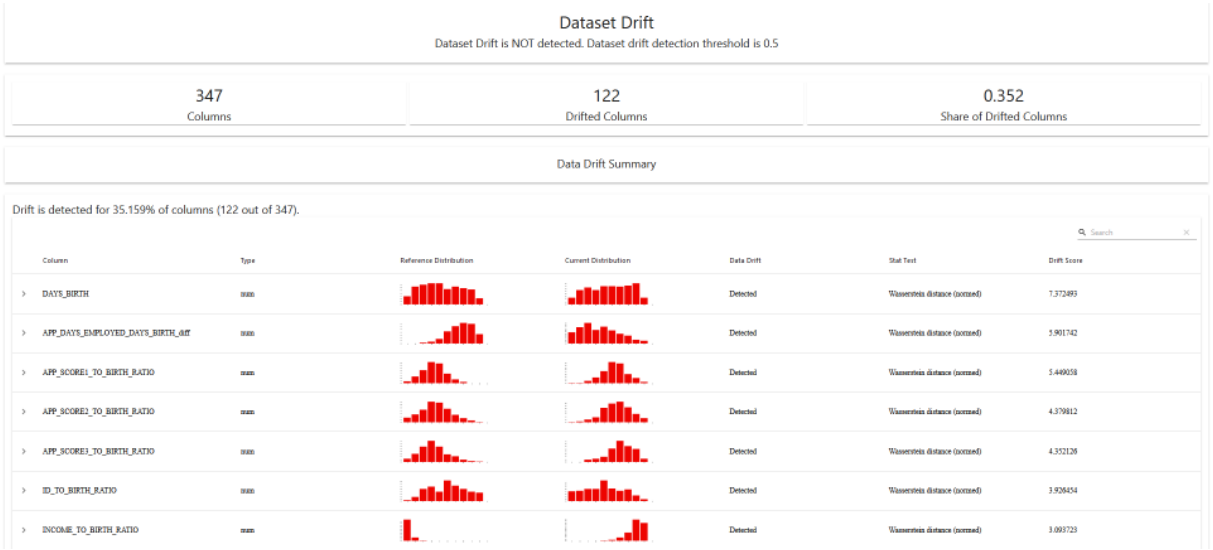

Fig. 14 : Capture d'écran illustrant les 7 features ayant le plus de data drift (model_ref)

Beaucoup de ces *features* sont liées à l'âge de l'individu (*feature* « 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'entrainement (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_ref (voir rapport associé : « ddrift_report.html »). Les valeurs de cette variable semblent en effet plus importantes dans le jeu de données test que dans le jeu de données d'entrainement. Nous avons observé qu'une valeur plus élevée de cette variable favorisée l'obtention d'un prêt (classement dans le groupe « 0 »).

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