Note Explicative

PROJET 7 : IMPLEMENTEZ UN MODELE DE SCORING

Prédiction sur la probabilité de faillite d'un client

RESUME

Mise en place d'un modèle de classification sur un jeu de données équilibré. Ce document détaille la construction de ce modèle, l'optimisation et l'analyse des résultats.

TABLE DES MATIERES

REFERENCES

[1] Lien projet 7 : https://s3.eu-west-1.amazonaws.com/course.oc-static.com/projects/Data_Scientist_P7/P7_DS_Projet.pdf

[2] Lien Kernel Kaggle : https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction

[3] Lien téléchargement des données : https://www.kaggle.com/c/home-credit-default-risk/data

[4] Librairie Python « Imblearn » pour équilibrage des données : https://kobia.fr/imbalanced-data-et-machine-learning/

[5] Interprétabilité des modèles de machine learning : https://www.nexialog.com/wp-content/uploads/2022/03/Interpretabilite-du-ML-Nexialog-Consulting.pdf

https://www.aquiladata.fr/insights/interpretabilite-des-modeles-de-machine-learning/

INTRODUCTION

L'entreprise souhaite **mettre en œuvre un outil de "scoring crédit" pour calculer la probabilité** qu'un client rembourse son crédit, puis 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 (données comportementales, données provenant d'autres institutions financières, etc.).

A partir d'un kernel Kaggle existant, qui a permis de faciliter la préparation des données nécessaires, nous avons procédé à l'élaboration du modèle de Scoring.

DONNEES D'ENTREES

Nous avons utilisé la base de données «application_{train/test}.csv », qui nous a servi à entraîner notre modèle. Cette base contient la variable « TARGET » (variable cible).

Ce diagramme montre comment toutes les données sont liées :

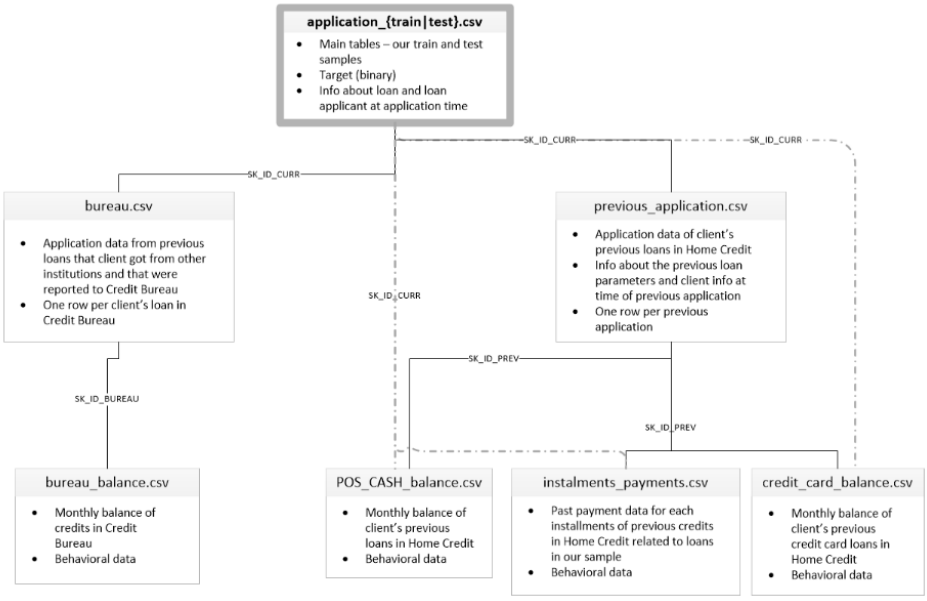


Figure 1

FEATURE ENGINEERING

Dans cette partie, nous présentons brièvement le Kernel Kaggle que nous avons récupéré, afin de comprendre la base qui a été réalisé et sur laquelle repose notre modèle.

1. KERNEL KAGGLE

Les tâches réalisées dans ce Kernel ont été réalisés principalement sur le dataset « application_train.csv ».

Data train/test

* "application_test.csv" est le dataset que nous utilisons pour simuler un nouveau client dans la base.
* Les tâches réalisées dans ce Kernel ont été réalisés principalement sur le dataset « application_train.csv ».

Valeurs manquantes

* Imputation des valeurs manquantes avec SimpleImputer en utilisant 'mode' pour les caractéristiques catégorielles et en utilisant 'médian' pour les caractéristiques numériques

Encodage variables

* One Hot Encoding pour les variables catégorielles avec get_dummies.

Alignement datasets

* Alignement des datasets "train" et "test" pour conserver des structures identiques.

Création de features

2. RESAMPLING

La principale problématique se situe dans le déséquilibre de la variable « TARGET ». Nous avons une classification binaire dans laquelle la **classe 0** représente les personnes qui ne rencontre pas de difficultés pour payer leur crédit, et la **classe 1**, représente les personnes qui rencontrent des difficultés pour rembourser leur crédit. On peut parler de données déséquilibrées dès lors que les deux classes ne sont pas présentes avec la même fréquence dans les données (le ratio n'est pas 50%/50%).

Afin d'éviter de construire un modèle ne prédisant que la classe majoritaire, il convient d'entraîner ce modèle sur un jeu de données équilibré. Pour cela nous avons utilisé une librairie Python nommée *Imblearn [4]* : nous avons utilisé plusieurs techniques dans la librairie *Imblearn* : **Random Under Sampler, Random Over Sampler et SMOTE**

Dans notre cas, dans un souci d'optimisation de temps de calculs, nous avons utilisé le « Random Under Sampler » qui s'avère le plus rapide. Sur la figure 2 ci-dessous, on constate le déséquilibre initial entre les deux classes. Tandis que sur la figure 3, nous avons rééquilibré parfaitement les 2 classes.

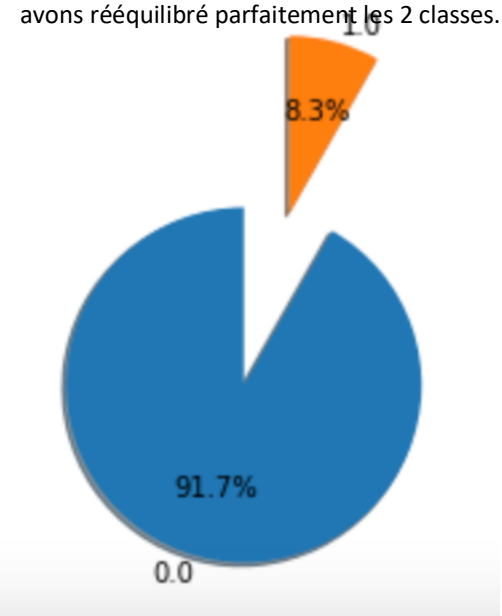


Figure 2

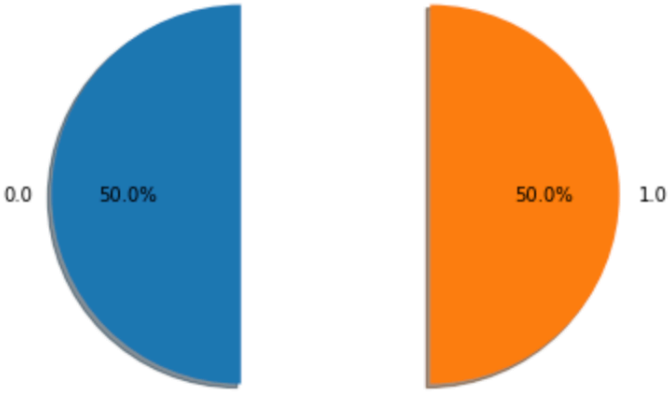


Figure 3

Nombre d'individus dans chaque classe avant et après sampling (rééquilibrage des données) :

RandomUnderSampler	**Classe 0**	**Classe 1**
Avant Sampling (*Figure2)*	96326	8671
Après Sampling (*Figure3)*	8671	8671

3. PREPROCESSING

L'entraînement du modèle se fait sur le fichier « application_train.csv ». Le preprocessing est constitué de deux traitements :

- Une transformation des données comprenant :
  - Une imputation des valeurs manquantes créées lors du traitement avec SimpleImputer
  - One Hot Encoding pour les variables catégorielles avec get_dummies.
- Un échantillonnage du dataset. Cette partie a été réalisée avec la méthode « train_test_split» de Scikit-learn. Avec un ratio de 70% pour les données d'entraînement et 30% pour les données de tests.

Figure 4

ALGORITHME D'OPTIMISATION / METRIQUE

1. MODELE RANDOMFORESTCLASSIFIER

Après entraînement de deux modèles de classification (LGBMClassifier et RandomForestClassifier), notre choix s'est porté sur le RandomForestClassifier. Voici les premiers résultats obtenus pour la première tentative de prédiction.

PRINCIPE : Les deux métriques qui nous intéressent ici sont la *Précision*, et le *Recall* :

- La Précision : Ce coefficient détermine que, quand le classifieur déclare que la prédiction est un 1, il a raison à X%.
- Le Recall : Ce coefficient détermine le pourcentage de détection des 1 du classifieur.

	Model	AUC	Accuracy	Precision	Recall	F1	Score_matrix	Time
0	LGBMClassifier	0.731003	0.672911	0.149934	0.665829	0.244753	0.671043	321.847345
1	RandomForestClassifier	0.7185	0.667956	0.147599	0.664154	0.241523	0.666953	2.463298

Figure 5

ANALYSE : Les 2 modèles présentent des résultats équivalents. On pourrait constater que notre modèle arrive à trouver 67% des classes 1, mais que, quand il prédit une classe 1, il n'a raison que dans 15% des cas.

LA MATRICE DE CONFUSION

PRINCIPE : La matrice de confusion consiste à compter le nombre de fois où des observations de la classe 0 ont été rangées dans la classe 1. Par exemple, si nous voulons connaître le nombre de fois où le classifieur à bien réussi à classer une classe 1, on examinera la cellule à l'intersection de la ligne 1 et de la colonne 1.

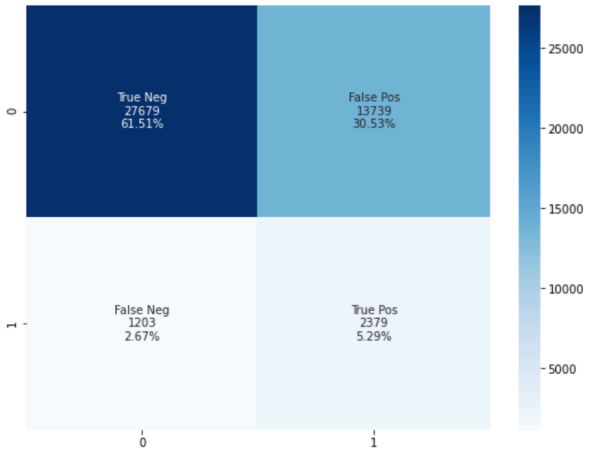


Figure 6

ANALYSE : Nous distinguons quatre catégories dans la matrice de confusion :

1. Les True Negatives (TN) : L'intersection de la ligne 0 avec la colonne 0. Ce sont des individus représentant un gain pour l'entreprise.
2. Les False Negatives (FN) : L'intersection de la ligne 1 avec la colonne 0. Ce sont des individus ayant comme valeur réelle 1 mais que le modèle à prédit en 0. Ces individus sont potentiellement un risque pour l'entreprise.
3. Les False Positives (FP) : L'intersection de la ligne 0 avec la colonne 1. Ce sont des individus ayant comme valeur réelle 0 mais que le modèle à prédit en 1. Ces individus pourraient être un risque pour l'entreprise.
4. Les True Positives (TP) : L'intersection de la ligne 1 avec la colonne 1. Ce sont des individus étant à risque pour l'entreprise.

Notre modèle idéal serait de retrouver 100% de TP, car ce sont les individus qui ne remboursent pas leur prêt. C'est aussi la catégorie la plus difficile à prédire car étant minoritaire par rapport à la catégorie *0.*

COURBE ROC ET SCORE AUC

PRINCIPE : La courbe ROC (Receviez Operating Characteristic) est un outil communément utilisé avec les classifieurs binaires. Elle croise le taux de TP avec le taux de FP. Sur la figure 7, la ligne en pointillée représente la courbe ROC d'un classifieur purement aléatoire. Un bon classifieur s'en écarte autant que possible (vers le coin supérieur gauche). Une autre façon de comparer des classifieurs consiste à mesurer l'aire sous la courbe (Area Under the Curve ou AUC). Un classifieur parfait aurait un score AUC égal à 1, tandis qu'un classifieur purement aléatoire aurait un score AUC de 0.5.

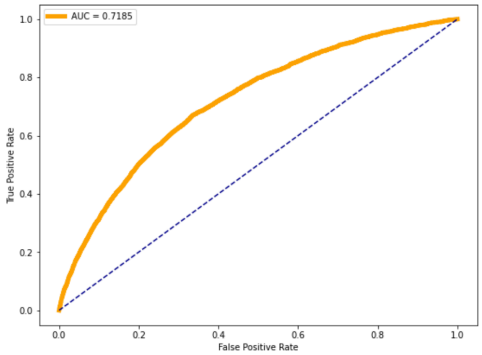


Figure 7

ANALYSE : On constate que la performances (AUC) du modèle n'est pas bonne. Pour information, nous cherchons une courbe ROC se rapprochant le plus du coin supérieur gauche du graphique.

FEATURES IMPORTANCE

PRINCIPE : L'analyse de l'importance des variables nous permet de visualiser sur quelles variables s'appuie le modèle pour effectuer ses prédictions. Sur la figure 8, l'importance des variables se lit en pourcentage (En multipliant les valeurs des abscisses par 100).

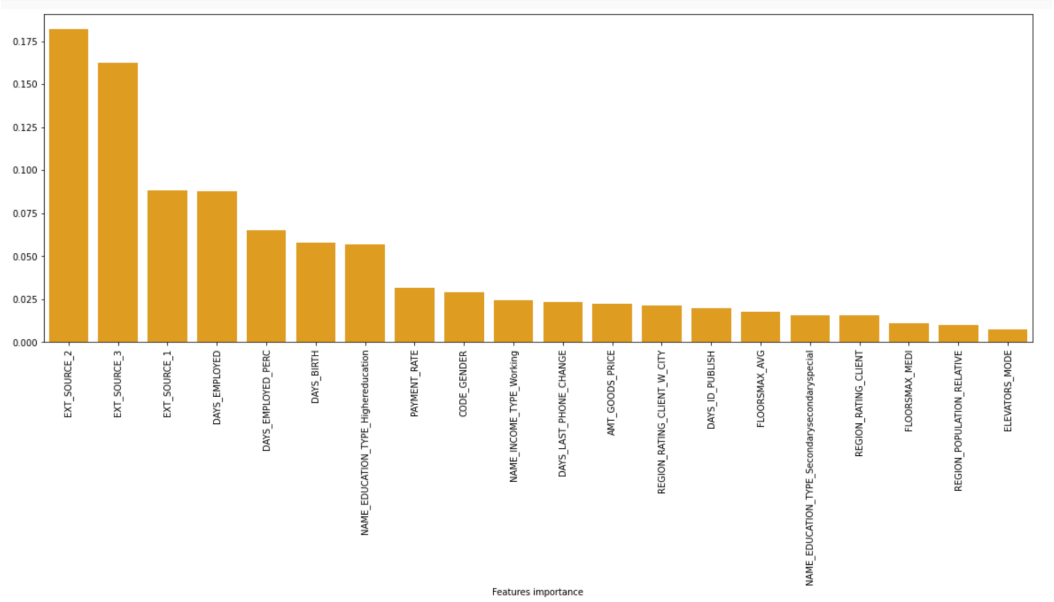


Figure 8

2. OPTIMISATION RANDOMFORESTCLASSIFIER

Notre première étude de prédictions nous montre que nous avons des résultats moyens. Cependant nous pouvons essayer d'optimiser le modèle en jouant sur ses hyper-paramètres.

HYPERPARAMETRES

Il existe un grand nombre d'hyperparamètres pour ce modèle, ici, nous n'en avons utilisé que quelques-uns :

```
1 best_model = RandomForestClassifier()
```

```
1 # Optimisation
2 param_grid = {
3     'n_estimators': [10,15,20],
4     'max_depth' : [2,4,6],
5     'random_state': [0,42],
6     'max_samples' : [.15,.3,.45]
7 }
```

Figure 9

La figure 9 représente les hyperparamètres utilisés pour optimiser notre modèle :

1. n_estimators : Le nombre d'arbres dans la forêt ;
2. max_depth : La profondeur maximale de l'arbre. Si aucun, les nœuds sont développés jusqu'à ce que toutes les feuilles soient pures ou jusqu'à ce que toutes les feuilles contiennent moins de min_samples_split échantillons ;
3. random_state a été fixé;
4. max_samples : Le nombre d'échantillons à tirer de X pour former chaque estimateur de base.

Nous avons utilisé la fonction GridSearchCV pour trouver la meilleure itération possible pour notre modèle.

Meilleurs paramètres trouvés :

```
1 grid.best_params_
```

```
{'max_depth': 2, 'max_samples': 0.15, 'n_estimators': 10, 'random_state': 0}
```

Figure 10

MESURES DE PERFORMANCES

	Model	AUC	Accuracy	Precision	Recall	F1	Score_matrix	Time
6	DecisionTreeClassifier	0.639593	0.684267	0.130734	0.525126	0.209349	0.642296	0.294771
3	DecisionTreeClassifier	0.63954	0.446933	0.104154	0.782524	0.183839	0.535439	0.310873
4	DecisionTreeClassifier	0.632241	0.528311	0.109611	0.691513	0.189228	0.571353	0.326244
9	DecisionTreeClassifier	0.621955	0.6444	0.124274	0.573423	0.204276	0.625681	0.270908
5	DecisionTreeClassifier	0.614754	0.638822	0.123447	0.579844	0.203558	0.623268	0.328777
1	DecisionTreeClassifier	0.605656	0.711422	0.12783	0.450865	0.199186	0.642705	0.306393
8	DecisionTreeClassifier	0.594465	0.463244	0.102058	0.73646	0.179273	0.5353	0.264036
0	DecisionTreeClassifier	0.590464	0.487178	0.101215	0.690676	0.176557	0.540847	0.840611
2	DecisionTreeClassifier	0.577975	0.853178	0.170551	0.218593	0.191607	0.685818	0.38586
7	DecisionTreeClassifier	0.538344	0.239911	0.084775	0.872697	0.154538	0.406797	0.257115

Figure 11

MATRICE DE CONFUSION

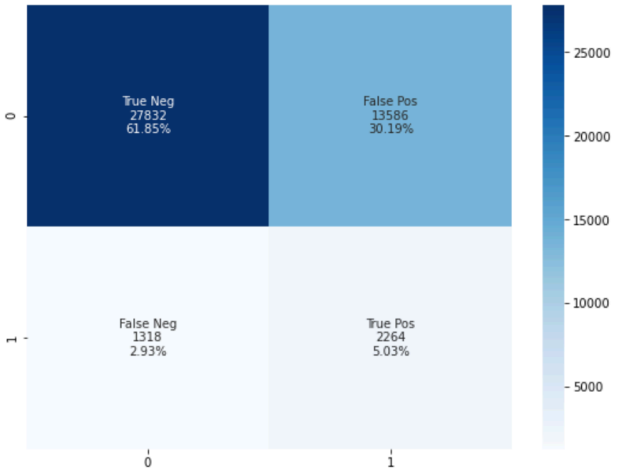


Figure 12

COURBE ROC ET SCORE AUC

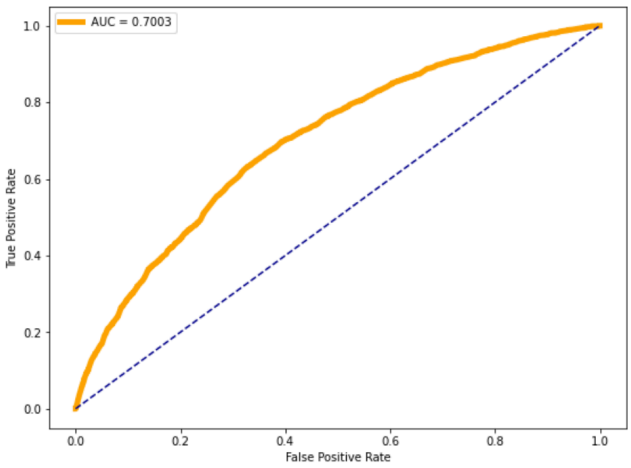
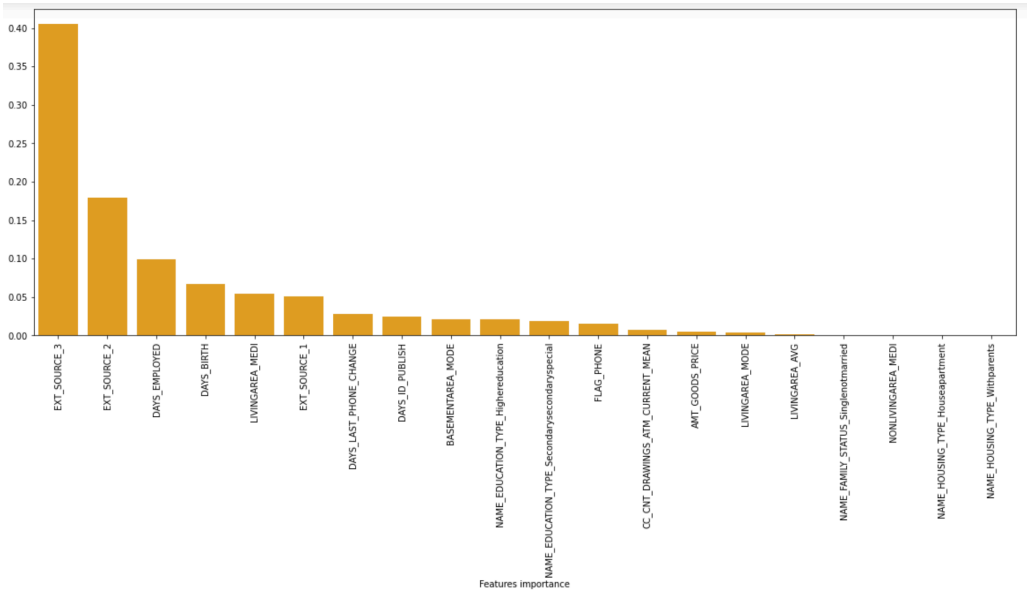


Figure 13

FEATURES IMPORTANCE

Figure 14

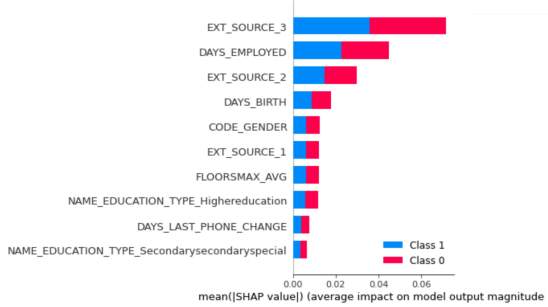


INTERPRETABILITE DU MODELE

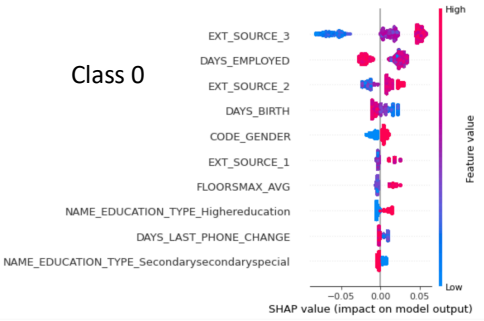
L'Interprétabilité est la mesure dans laquelle un humain peut comprendre les causes d'une décision (d'une prédiction) d'un modèle. Les différents calculs d'importance de variables (features importance), en particulier pour le modèle de forêt aléatoire (Random Forest) pour lequel la librairie Python Sklearn propose de base plusieurs calculs. Nous montrerons ensuite différents graphiques de la librairie SHAP permettant d'analyser les prédictions d'un modèle.

Global :

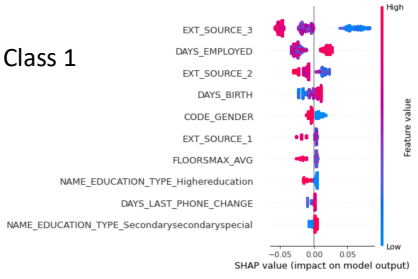
SHAP permet de calculer, pour chaque donnée, l'importance de chaque variable. Il suffit de faire la moyenne des importances afin d'avoir des importances « globales » pour chaque variable.



Class 0



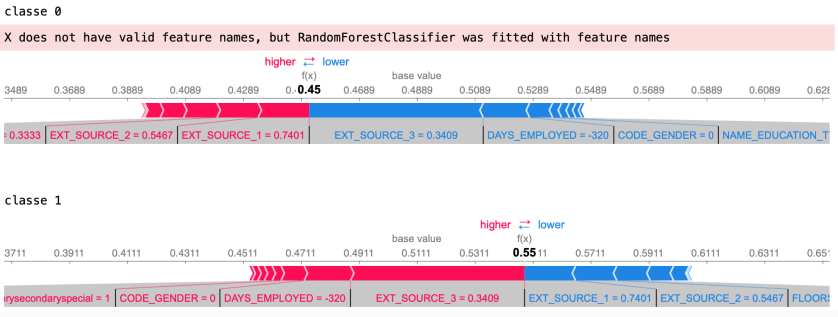
Class 1

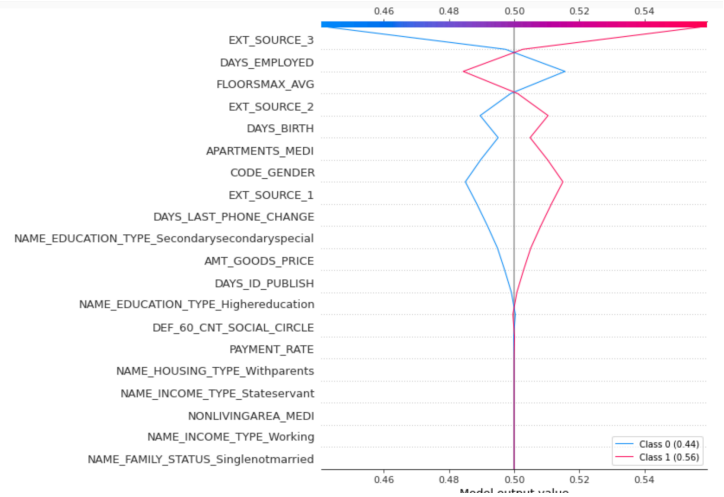


Ici, chaque point correspond à une donnée (de test). Les valeurs SHAP (abscisse) sont représentées pour chaque variable (ordonnée), pour chaque donnée. La couleur représente la valeur de la variable. Des « épaisseurs » plus importantes sur l'axe des ordonnées (par exemple pour la variable DAYS_BIRTH en abscisse 0) permettent de représenter une densité de points plus importante. Sur ce graphique qui correspond à la **classe 0**, on comprend donc qu'un EXT_SOURCE_3 élevée rend la possibilité qu'une personne ne rencontre pas de difficultés pour payer son prêt (car une valeur SHAP élevée, donc une appartenance plus forte à la classe 0), alors qu'être passé au collège ("NAME_EDUCATION_TYPE_Secondarysecondaryspecial") ne rend pas plus possible q'une personne ne rencontre pas de difficulté à payer son prêt.

Local :

La valeur de base (base value) est la valeur moyenne obtenue comme sortie pour cette classe, alors que la valeur de sortie (model output value) est la valeur prédite par le modèle. Les valeurs SHAP de chaque variable, proportionnelles aux tailles des flèches, « poussent » la prédiction depuis la valeur de base jusqu'à la valeur prédite. Ainsi, dans le cas de la classe 0, la valeur de la variable EXT_SOURCES_3 est déterminante pour dire que la donnée appartient à la classe 0, alors que la valeur d'EXT_SOURCES_3 qui permet de dire que la donnée n'appartient pas à la classes 1.





La model output value est cette fois la valeur moyenne des output values pour les différentes classes, et où les valeurs SHAP sont décalées afin de correctement reproduire le score de sortie du modèle. On peut ainsi facilement comparer les effets des variables sur les prédictions des différentes classes.

Synthèse

L'Interprétabilité est un aspect métier important dans le processus de construction d'un modèle de Machine Learning dans des domaines assez réglementés. Il a différentes méthodes simples (permutation feature importance) et complexes (basées sur la valeur de Shapley). Cette dernière a été utilisée dans ce Notebook.

AXES D'AMELIORATIONS A APPORTER

Nous avons pu constater tout au long de ce document que les performances du modèle ne sont pas bonnes. Pour résumé : Au mieux, notre modèle peut trouver 52% des classes 1, et lorsqu'il en prédit une, il a raison à 15%.

Nous pouvons l'expliquer par le features engineering qui est à améliorer. En effet, le Kernel choisi est plutôt pauvre sur le traitement des données. Il ne se focalise que sur une seule table et ne crée pas beaucoup de variable qui peuvent être utiles à un modèle de classification comme des moyennes, des médianes, des écarts-types, et ça pour plusieurs features. Il existe peut-être un Kernel plus abouti qui permettra une meilleure performance prédictive au modèle. Sinon, prendre le temps de réaliser nous-même notre feature engineering, ce qui nous permettra de bien comprendre nos données et ainsi construire un feature engineering adapté à notre besoin.