NOTE MÉTHODOLOGIQUE

# Table de matières

[**La méthodologie d'entraînement du modèle 1**](#_iqgiheiwg5w2)

[**Le traitement du déséquilibre des classes 3**](#_oq1n84kajbk7)

[**La fonction coût métier, l'algorithme d'optimisation et la métrique d'évaluation 4**](#_n3sjswe1ig7)

[**Un tableau de synthèse des résultats 4**](#_az6yf9uw5ewb)

[**L’interprétabilité globale et locale du modèle 6**](#_i3acuxbnxll2)

[**Les limites et les améliorations possibles 7**](#_oom1h0ezduo1)

[**L’analyse du Data Drift 7**](#_9urq2p3nn4d)

# La méthodologie d'entraînement du modèle

L’objectif du projet est de développer un modèle capable de prédire si un client sera solvable ou non. Pour développer ce dit modèle, nous avons accès aux informations bancaires de plusieurs centaines de milliers de clients dont on connaît leur statut de solvabilité. Le modèle doit donc prendre en compte les variables fournies et en identifier les plus pertinentes. Le modèle ainsi développé doit être en mesure de réduire le pourcentage d’erreur estimé à 8.1 %, c’est-à-dire, qu’en prenant un client dans la liste au hasard, nous estimons à 8.1 % le pourcentage que le client soit insolvable. Notre problème est donc un problème de classification binaire et les modèles testés devront y répondre.

Ne sera pas traitée la préparation des données, provenant principalement du kaggle ([LightGBM with Simple Features | Kaggle](https://www.kaggle.com/code/jsaguiar/lightgbm-with-simple-features/script)).

Pour toute préparation au développement d’un modèle de machine learning, il faut réaliser la séparation des données en données d’entraînement et de test. Cela a pour objectif de séparer ce que le modèle voit lors de l’entraînement et de pouvoir valider les résultats du modèle sur des données non rencontrées. Cela évite le data leakage et de biaiser les résultats obtenus. Par la même occasion, cela nous permet de vérifier si le modèle overfit ou underfit et ainsi identifier si le modèle généralise conformément en rencontrant de nouvelles données.

Dans notre cas, les données ont été séparées en données d’entraînements, de test et de validation, ce qui a été rendu nécessaire par l’utilisation d’un modèle de réseau de neurones, qui nécessite ce type de séparation. Les données de tests permettront de comparer les performances des modèles entre eux. Nous avons choisi une séparation 80 % / 10 % / 10 %, avec une fourchette haute pour le pourcentage de données d’entraînements, qui nous est possible de par la quantité importante de données disponibles pour l’entraînement.

Le développement du modèle a été réalisé en deux étapes :

* Une première étape de sélection d’un modèle à optimiser parmis un choix varié de différents algorithmes pouvant nous aider à répondre à notre question.
* Une deuxième étape d’optimisation des hyperparamètres du modèle sélectionné

Nous avons testé plusieurs algorithmes variés (arbres de décision, régression, k voisins..), afin d’en identifier un ou plusieurs qui sortent du lot et qui seraient les plus appropriés à notre problématique:

* DummyClassifier
* KNeighborsClassifier
* RandomForestClassifier
* Naives Bayes Classifier
* LogisticRegression
* SGDClassifier
* GradientBoostingClassifier
* XGBClassifier
* LightGBM
* Neural Network (Keras)

Lors de la première étape, nous avons réduit le nombre de variables à utiliser lors de la recherche du meilleur algorithme (k = 100, par sélection ANOVA des 100 features les plus significatives) et réduit le nombre d’hyperparamètres à tester. En effet, certains algorithmes peuvent être très long à réaliser leur calcul sur un nombre important de données. Afin d’obtenir des résultats fiables, nous avons réalisé un GridSearch pour tester les différents hyperparamètres, avec 3 splits.

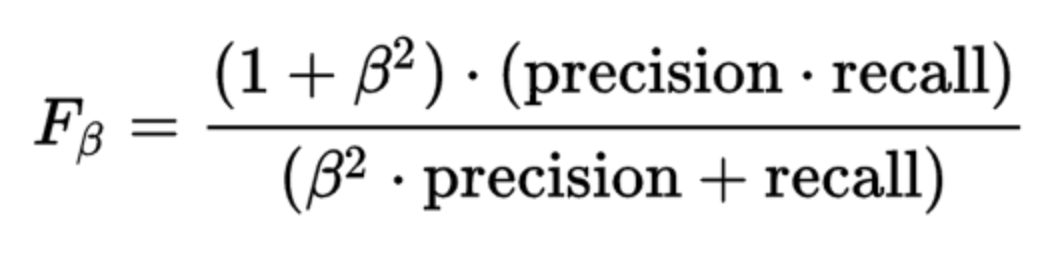
Avant de tester chaque modèle et leur hyper paramètres, le jeu de données a été transformé :

Une imputation des valeurs manquantes a été réalisée et qui été nécessaire dû faite que certains algorithmes ne tolèrent pas la présence de valeurs manquantes. Pour remplir ces valeurs manquantes, ils ont été rempli par la médiane (choix après des tests préliminaires). Alternatives qui auraient pu être possibles : par la moyenne, par 0, ou encore éliminer les lignes ou features avec valeurs manquantes. Les features présentant 0 de variance après imputation ont été filtrés ensuite.

Pour réduire les temps de calcul, une sélection de features additionnelles a été réalisée, avec soit 40 ou 60 features choisis à l’aide d’un test ANOVA.

Enfin, les features ont été scalées en utilisant StandardScaler (choix après tests préliminaires), étant donné que nous avons des valeurs qui peuvent être très grandes ou très petites.

Il faut séparer les métriques utilisées pour comparer les modèles entre eux et la métrique utilisée pour le choix des meilleurs hyperparamètres. En effet, lors du gridsearch, la métrique f\_beta avec un bêta = 2 a été utilisée. Elle s’est montrée comme la métrique la plus appropriée à ce problème de classification. La métrique permet de mettre l’accent soit sur la précision ou le recall et avec un bêta = 2, on se focalise plus sur le recall, c'est-à-dire que nous voulons identifier plus de prédictions positives correctes sur l’ensemble des positifs (on veut diminuer le nombre d’erreurs).



Pour comparer les modèles, on va s'intéresser à plusieurs métriques :

**- Temps d'entraînement et pour la prédiction**

Pour l'optimisation plus fine du modèle sélectionné, on voudrait tester plus d'hyper paramètres, on veut donc éviter de gonfler le temps de recherche des paramètres. Également, pour la prédiction, on veut éviter qu'il soit trop long.

**- ROC auc (train/test)**

On veut des scores assez proches entre les deux (pas d'overfitting/underfitting) et un score élevé nous permet d'estimer une bonne prédiction.

**- Recall moyen**

On veut un recall moyen le plus haut possible, métrique nous donnant une idée de la bonne affiliation des données par le modèle.

Nous suivrons également les résultats obtenus sur la matrice de confusion (Vrais positifs et négatifs).

Lors de la seconde étape, après sélection du meilleur algorithme, 200 features seront testées, avec plus d’hyperparamètres. Un gridsearch avec 3 splits sera également utilisé, et un score métier (présenté plus bas) sera utilisé pour développer le modèle final. Nous analyserons l’importance des features prise par le modèle, important pour comprendre et expliquer le modèle. A la suite de cela, nous analyserons le data drift, pour identifier si le modèle reste pertinent malgré l’apport de nouvelles données.

# Le traitement du déséquilibre des classes

Ce que nous avons pu observer avec notre jeu de données est qu’il est fortement déséquilibré. En effet, dans notre problème de classification binaire, le rapport entre les deux classes se situe à 0.08. C’est à dire qu’une classe représente 92 % des événements, les clients ayant obtenus un prêt, contre 8 % des cas pour un client ayant eu son prêt refusé.

Ce risque peut induire lors du développement d’un modèle de grandes erreurs de précision. En effet, selon la méthode de score utilisée, le modèle ne satisferait que la classe majoritaire. Pour remédier à ce problème, il a été développé plusieurs méthodes de rééchantillonnage, pour limiter les effets de déséquilibres. Dans notre cas, la classe imblearn en soutient plusieurs :

* NearMiss : Il va sous échantillonner la classe majoritaire
* SMOTE : Il va sur échantillonner la classe minoritaire
* SMOTEEN : Il va faire un mixte des deux méthodes précédentes

Chacune de ces méthodes ont leur mérite, cependant, dans notre cas, SMOTE et SMOTEEN sont les plus appropriées car ils permettent d’utiliser la majorité des données. NearMiss ferait descendre le nombre de la classe majoritaire à 8 %.

Lors du développement de notre modèle, SMOTE a été privilégié parmi les trois, car dû aux grands nombres de données, les temps de calculs sont considérablement augmentés et les résultats préliminaires ont montré que SMOTE était une méthode très adaptée au problème.

Une différente méthode est possible pour prendre en compte le déséquilibre, c’est d’apposer un poids aux classes. En effet, certains modèles prennent comme paramètre un dictionnaire de poids de classe, tel que le modèle de Régression logarithmique de la librairie Sklearn. Nous avons donc, dans certains cas, mis un poids 1:10 pour compenser le déséquilibre.

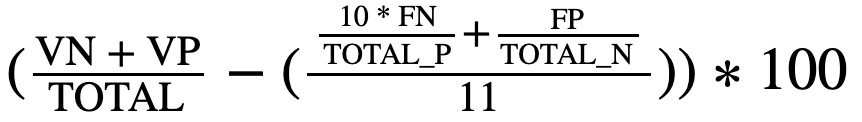
L’ensemble des résultats ont montré qu’utiliser une méthode de rééchantillonnage ou poids de classe améliore considérablement les résultats obtenus. Dans le modèle final, une de ces méthodes est donc présente.

# La fonction coût métier, l'algorithme d'optimisation et la métrique d'évaluation

Pour le développement de notre modèle de classification des clients, en plus de suivre les différentes métriques d’évaluations, nous avons développé une métrique propre au problème, un “score métier”. Ce score prend en compte la problématique que donner un prêt à un client insolvable puisse entraîner un coût à l’entreprise.

**- Objectif : Limiter les pertes d'argents**

On veut donc maximiser les bonnes prédictions et minimiser les erreurs. Certaines erreurs sont plus importantes que d'autres. Un faux négatif (donné un prêt à une personne insolvable) est plus dommageable que ne peut pas donner un prêt. Pour matérialiser ce point, on va donner un coût 10 fois plus important aux faux-négatifs qu'aux faux-positifs. Ce qui donne cette formule :



VN : Vrais négatifs; VP : Vrais positifs; FN : Faux négatifs; FP : Faux positifs;

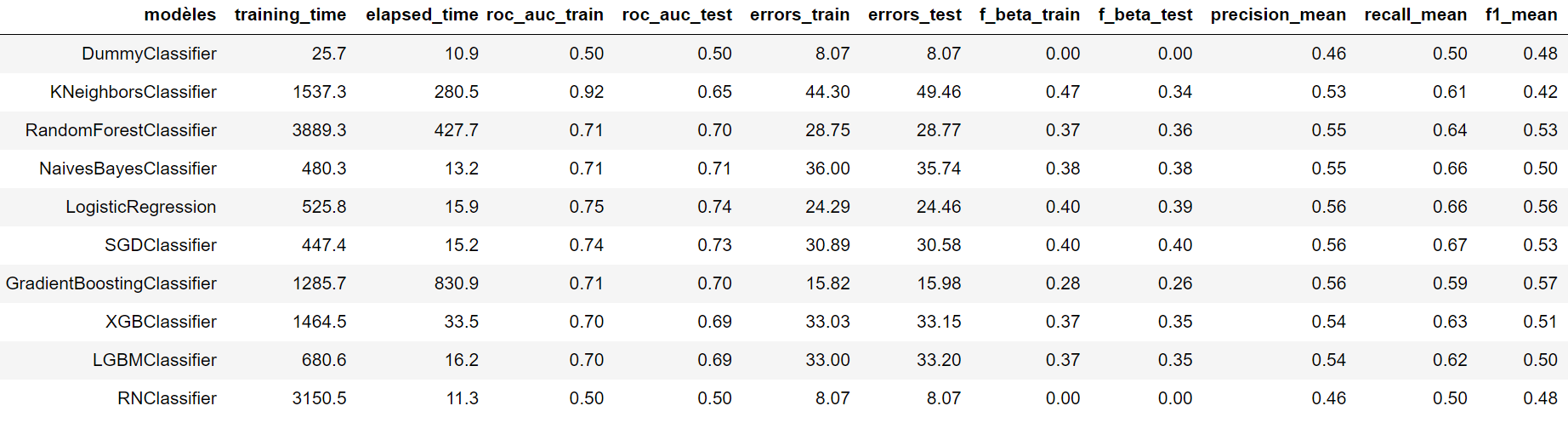
TOTAL\_P : Vrais positifs + Faux négatifs; TOTAL\_N : Vrais négatifs + Faux positifs

Le score métier allant de 0 à 100, avec un modèle parfait qui retournerait 100.

Pour améliorer les résultats du score métier, il est également possible d’optimiser le seuil de prédiction. En effet, quand un modèle de classification réalise une prédiction, il donne une probabilité pour chaque classe et de façon non défini, il utilise un seuil de 0.5 pour l’attribution de la classe. Pour cette étape d’optimisation du score métier, nous allons tester une fenêtre élargie de seuils, entre 0 et 1, afin d’identifier le plus pertinent au modèle identifié.

Une comparaison entre l’avant et l’après optimisation du seuil sera réalisée, avec en particulier la nouvelle répartition des VP, VN, FN et FP.

# Un tableau de synthèse des résultats



Si on analyse certaines des métriques d’intérêts :

**Le temps de recherche et d'exécution :**

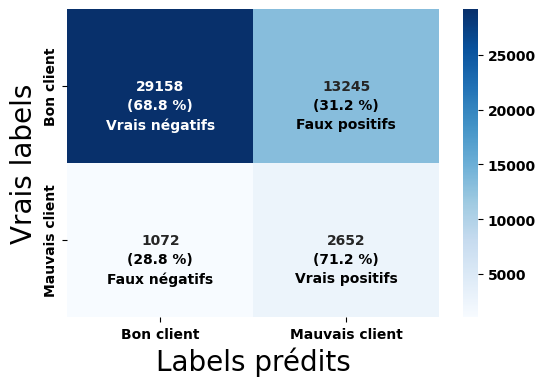
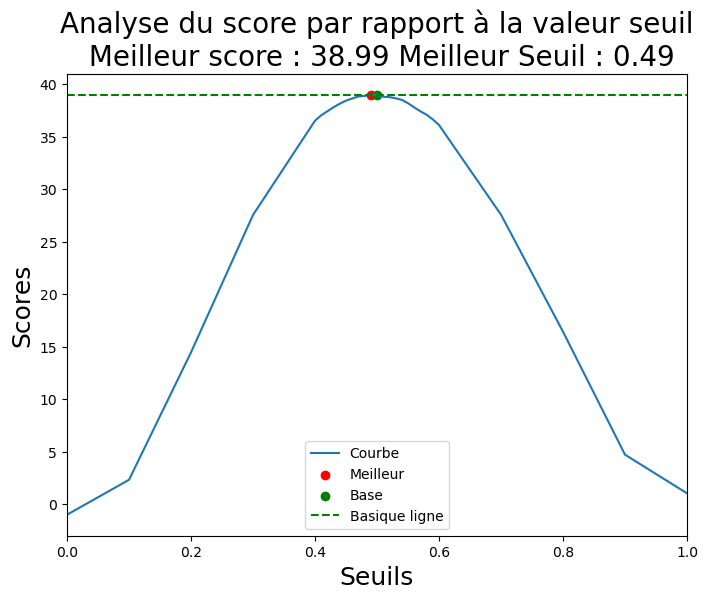
Trois modèles prennent un temps long dans leur exécution lors de leur prédiction tel que les Kneighbords, RandomForest et GradientBoosting (mais également dans l'entraînement). Le modèle de réseau neuronal prend également du temps dans sa recherche d'hyperparamètres, mais est assez rapide pour la prédiction. On a ensuite des modèles très rapides à entraîner et dont la prédiction est tout autant rapide : NaiveBayesClassifier, LogisticRegression, SGDClassifier et LGBMClassifier. XGBclassifier est en intermédiaire en termes de temps de recherche des hyperparamètres et la prédiction

**ROC AUC :**

On retrouve bien une valeur de 0.5 pour DummyClassifier. A part pour kneighbords, il n'y a pas d'overfitting ou d'underfitting sur les modèles. Étonnamment le modèle RN n'a pas bien performé. Des tests préliminaires l'avaient très bien placé, il faudra analyser la possible raison d’une contre-performance. On va donc le mettre de côté pour l'instant. Les autres modèles se situent entre 0.69 et 0.75 sur les données d'entraînements et de test, ce qui représentent de bons scores. On observe que les modèles LogisticRegression et SGDClassifier développés ont les meilleurs résultats.

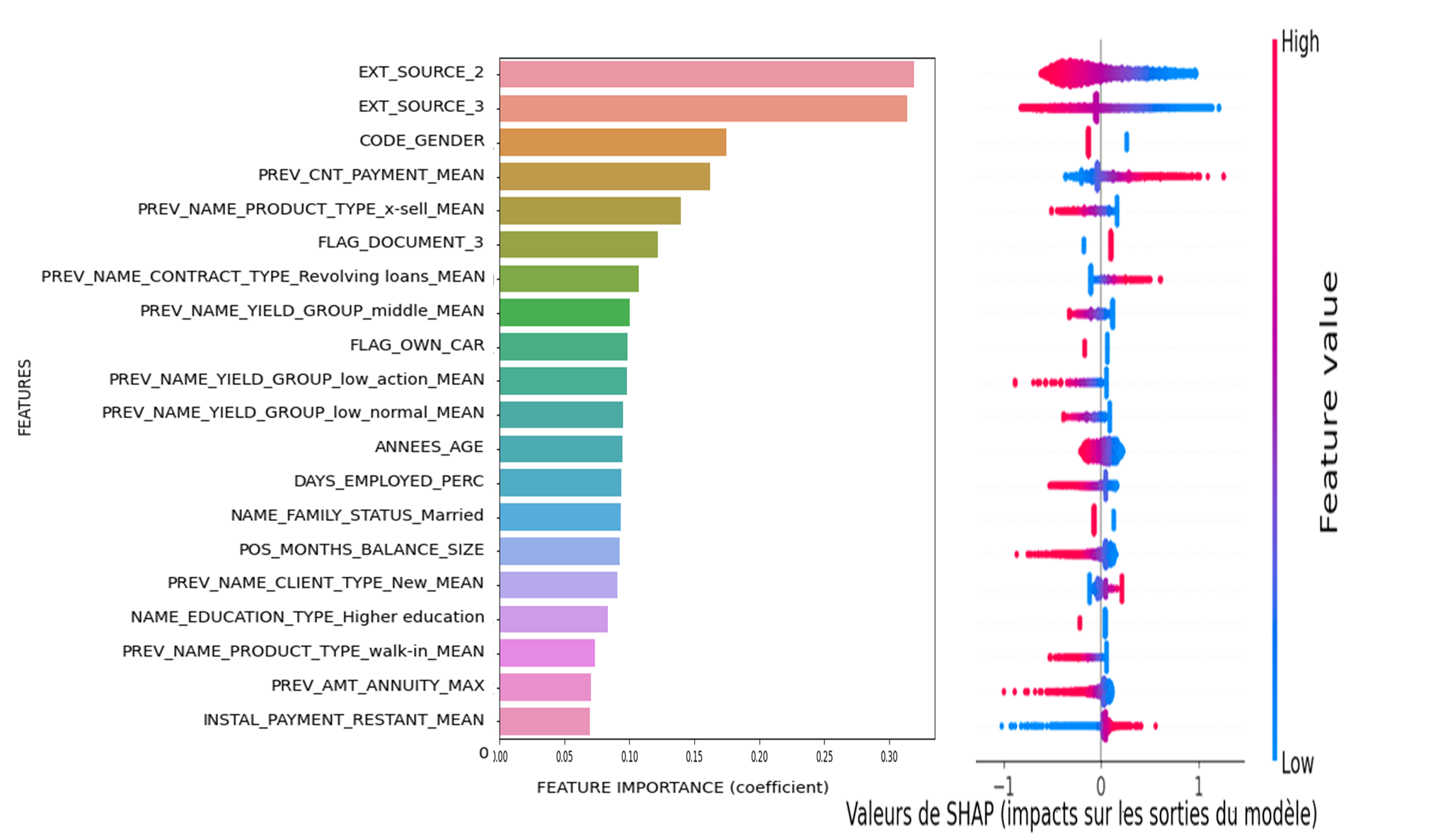
**Recall :**

On retrouve bien une valeur de 0.5 pour le DummyClassifier. Sur les autres modèles, les tendances de résultats sont assez similaires à ceux obtenus avec ROC\_AUC avec des scores se situant entre 0.59 et 0.67. Le meilleur score attendu serait de 1. Donc on a des scores montrant que 59 à 67 % pour chaque classe est bien affiliés. LogisticRegression, SGDClassifier et NaivesBayes sont les meilleurs modèles sur cette analyse.

A la suite de ça, nous avons sélectionné LogisticRegression, qui possède les meilleurs résultats. Nous avons optimisé les hyperparamètres afin de développer le meilleur modèle, en utilisant le score métier comme métrique lors du GridSearch. Pour le développement du modèle final, plus de 200 variables ont été utilisées, avec une sélection réduite à 200 par ANOVA. Une amélioration de la valeur seuil a également été réalisée. Il a été montré qu’en utilisant un seuil de prédiction de 0.49 pour la classe 1 (0.51 pour la classe 0), on optimise au mieux le score métier, entraînant une identification de 70 % des vrais négatifs et vrais positifs avec 30 % d’erreurs de Faux positifs et Faux négatifs. On passe d’un pourcentage d'erreurs de 8.1 % sans modèle à 3.5 % avec le modèle. Le ROC AUC est quant à lui de 0.75-0.76, ce qui est un bon score.

# L’interprétabilité globale et locale du modèle

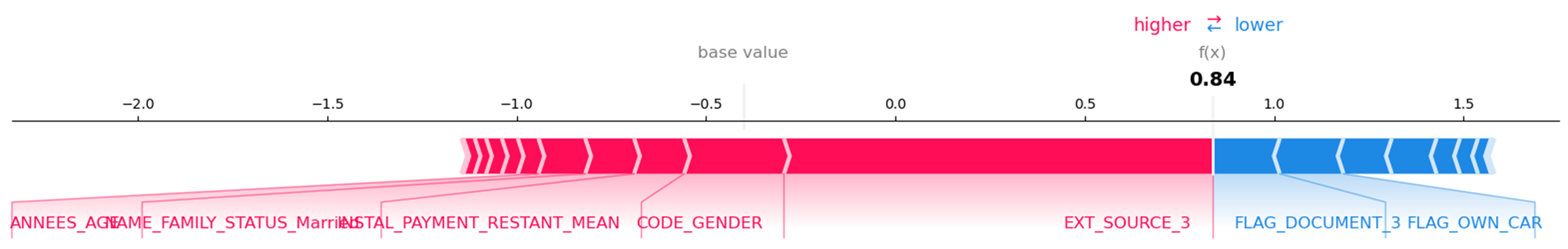
Pour analyser et mieux comprendre le modèle, nous avons analysé l’importance globale et locale des features, de façon indépendante via le module SHAP.

* **Importance globale (via SHAP)** : 

La figure de gauche nous montre quelles features contribuent (positivement ou négativement) au modèle. EXT\_SOURCE\_2 et 3 sont les features qui contribuent le plus au modèle. Le fichier “shap\_values\_model.csv” permet de retrouver la feature importance pour l’ensemble des features utilisées dans le modèle. De façon notoire, le modèle estime que CODE\_GENDER, FLAG\_OWN\_CAR, ANNEES\_AGE et DAYS\_EMPLOYED\_PERC contribuent fortement au modèle. Sur la figure de droite, on observe comment chaque valeur de chaque feature contribue à la note finale. Par exemple, plus EXT\_SOURCE 2 et 3 sont haut, plus ils vont contribuer à une prédiction de solvabilité (0). Alors que pour une valeur ANNEES\_AGE petite contribuera à une prédiction d’insolvabilité. Donc l’ensemble de ces données vont nous permettre d’expliquer au mieux comment le modèle prend en compte les données.

* **Importance locale (via SHAP)** :

Un autre point important à la compréhension du modèle et à son acceptabilité est d’expliquer au mieux les résultats obtenus. SHAP nous permet de comprendre pourquoi le modèle prédit une valeur. Par exemple, ce client est insolvable selon le modèle principalement par ses valeurs EXT\_SOURCE\_3 (faibles ressources) et CODE\_GENDER (c’est un homme).



# Les limites et les améliorations possibles

Un aspect qui s’est révélé crucial et limitant dans le développement d’un modèle pertinent est la taille du dataset. En effet, plusieurs centaines de milliers de données sont disponibles ce qui est très intéressant dans le développement d’un modèle robuste et pertinent, cependant cela nous limite dans l’optimisation du modèle et dans la recherche du meilleur modèle.

En effet, le développement d’un modèle sur ces conditions nécessite un nombre d’heures de calcul très important, sans forcément y voir un gain substantiel en performance. Réduire la taille du dataset, pour ensuite tester un plus grand nombre d’hyper paramètres pourrait être une solution, mais certaines données pourraient être perdues pour l’entraînement, ce qui nécessiterait la mise en place de vérifications. Une solution serait de passer par une segmentation des clients (par agglomération par exemple) et ensuite utiliser des clients représentatifs de chaque cluster.

Afin d’augmenter la confiance qu’on peut avoir dans le modèle et d’améliorer ses résultats, il faudrait augmenter la part du nombre de clients n’ayant pas obtenu un prêt avec de nouvelles données. Pour pallier au déséquilibre de classes, nous avons utilisé une méthode qui a artificiellement augmenté les données de la classe sous représentée, mais cette méthode n’apporte aucune nouvelle variabilité dans les données de cette classe, donc au final, on se retrouve avec des données multipliées.

J’identifie trois axes pour améliorer le modèle développé :

* Comme les modèles ensemblistes, développer plusieurs sous-modèles qui seront combinés en un seul modèle. Avec en objectif chaque sous modèle qui se spécialisent, pour que le modèle final combine les forces de chaque afin d’obtenir un modèle général plus puissant.
* Un aspect majeur dans le développement d’un modèle et particulièrement dans ce domaine sont les features utilisées. Dans ce cas, nous avons suivi les recommandations d’autres utilisateurs pour la création de nos features étant donné notre manque d’expérience dans le domaine bancaire. Plusieurs voies s’offrent à nous pour développer cette partie et améliorer nos résultats. Dans un premier cas, nous pourrions consulter des experts du domaine afin d'identifier des variables importantes dans l’obtention d’un prêt et réaliser du feature engineering approprié sur ces variables. Dans un second cas, nous pourrions nous focaliser sur les features les plus importantes du modèle et faire du feature engineering sur ces features afin de les améliorer.
* Dernièrement, on pourrait un peu plus travailler sur les features. En effet, la recherche d’outliers ou d’erreurs d’entrées n’ont pas été très poussées par exemple. En ce qui concerne les valeurs manquantes, nous avons utilisé un imputer, remplissant par la médiane. Une autre approche aurait pu être utilisée, tel que les KNeighbors.

En travaillant sur ces différents points, nous pourrons améliorer le modèle développé.

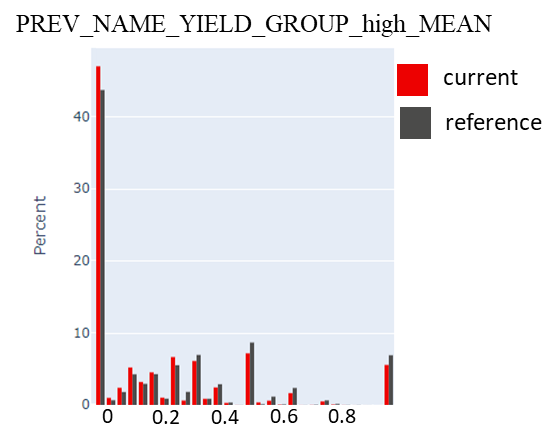
# L’analyse du Data Drift

Lors du développement d’un modèle, sa durée de vie est un élément crucial à vérifier. Il faut donc vérifier si le modèle reste pertinent et à jour dans le futur et si la distribution des données des variables utilisées lors de l’entraînement suivront la même distribution. Afin d’étudier le data drift, nous utilisons la librairie Evidently, qui permet de mesurer aisément la distribution variable à variable entre le dataset d’origine (utilisé lors de l’apprentissage) et de nouvelles données non vues par le modèle. Dans notre cas, nous avons utilisé les données du fichier application\_test.

En étudiant les 200 variables utilisées sur le modèle, du Data Drift n’a été que très rarement observé.

Le résultat est disponible dans ce fichier : “”. On va des variables les plus importantes pour le modèle aux moins importantes, donc notre intérêt est principalement focalisé sur les premières pages.

Un data drift a été retrouvé sur 13 variables sur 200. Ce qui concerne 5-10 % des variables seulement. Mais si on compare à l'importance des variables, cela ne concerne que 2 variables sur les 40 plus importantes. CODE\_GENDER : Qui a un Drift score de 0.23 selon la méthode statistique Jensen-Shannon distance. PREV\_NAME\_YIELD\_GROUP\_high\_MEAN : Qui a un Drift score de 0.11 selon la méthode statistique Wasserstein distance (normed). Exemple ci-bas :

On peut donc estimer que le drift est assez faible. Comme observation on peut noter que sur CODE\_GENDER, le data drift correspond à une inversion du nombre d'hommes par rapport au nombre de femmes.

Donc, le modèle développé va rester assez pertinent avec les nouvelles données qui pourraient lui être données lors de son déploiement.