|  |  |
| --- | --- |
|  | **2021** |
|  | Nadia Zabéti |

|  |
| --- |
| **[NOTE METHODOLOGIQUE]** |
| Projet de scoring pour Prêt à dépenser |



La méthodologie d'entraînement

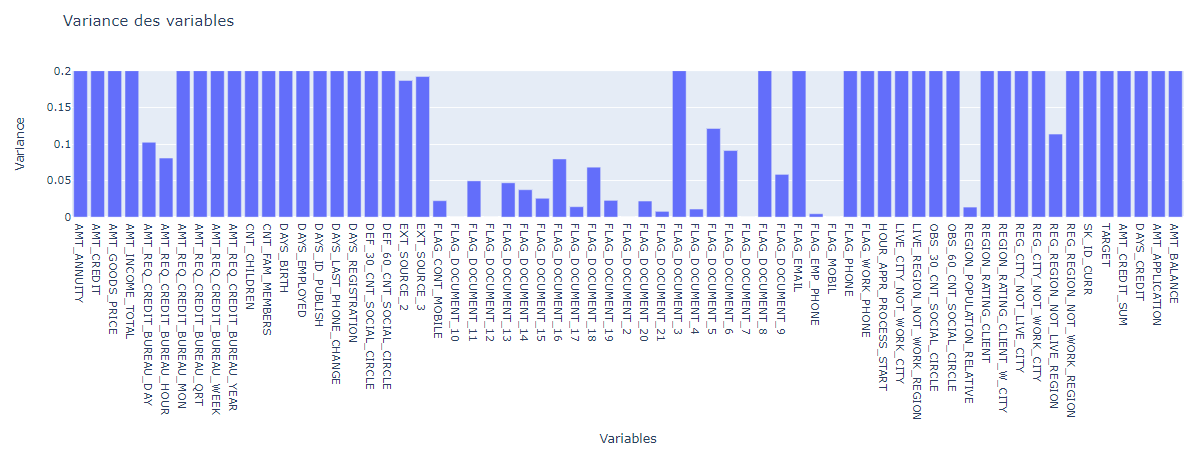
du modèle

# Choix des variables explicatives

Les variables explicatives sont les variables qui influencent le plus notre variable dite « cible », à savoir ici la variable « TARGET » qui nous renseigne sur le remboursement ou non du prêt par le client.

Après élimination des variables dont le taux de remplissage est inférieur à 68%, on se retrouve avec **80 variables**.

On utilise alors la variance, qui est la métrique permettant de rendre compte de la dispersion des données pour chaque variable. On observe les variances de chaque variable



Et on décide alors d’élminer les variables dont la variance est inférieure à 0.1. On se retrouve alors avec **58 variables**.

Ensuite, après élimination manuelle des variables redondantes, on étudie la corrélation entre nos variables restantes, et on sélectionne celles dont la corrélation mutuelle est d’au moins 0.3. On se retrouve avec **11 variables**.

Pour obtenir un **top-10**, on utilise le module SelectKBest de Scikit-Learn, qui pour chaque variable nous calcule un score et nous renvoie un nombre prédéfini de top variables.

# Transformation des variables

Pour que nos variables suivent un comportement le plus proche d’une loi normale, nous appliquons une transformation numérique sur celles dont la représentation graphique est la plus éloignée d’une **courbe de Gauss**.

Ici, nous appliquons la fonction **logarithme népérien** à 3 variables.

# Séparation du jeu de données en test et entraînement

Maintenant que nous avons nos 10 variables explicatives et notre variable cible, nous allons pouvoir séparer notre jeu de données en jeu d’entraînement et jeu de test.

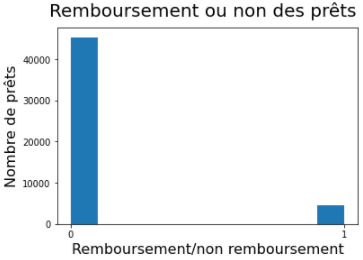
Le jeu d’entraînement représente les données sur lesquelles le modèle va s’entraîner : il va itérer un nombre de fois défini sur les données pour « apprendre » et faire les meilleures prédictions.

Une fois l’apprentissage effectué, on teste le modèle sur un jeu vierge (le jeu de test) pour évaluer l’apprentissage du modèle.

Le plus souvent, on sépare le jeu de données du jeu d’entraînement avec la proportion suivante : 2**/3 pour le jeu d’entraînement, 1/3 pour le jeu de test**, ce qui est le cas ici.

# Resampling

Particulièrement lorsqu’il s’agit de scoring (classification binaire), il est important de veiller à une égale répartition des données selon les 2 classes. Ici, il y a une **sur-représentation** des clients ayant remboursé leur prêt.



Pour pallier à ce problème et permettre au modèle d’apprendre le plus justement possible, on a utilisé la technique **SMOTE** : elle permet de reconstituer des échantillons fictifs pour la classe sous-représentée.

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

# Fonction coût métier

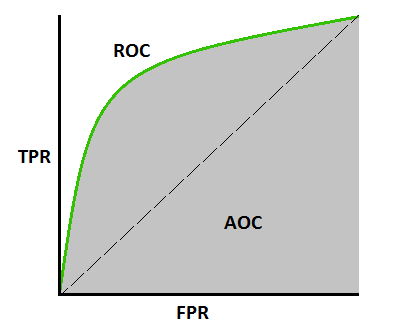
Chaque méthode de modélisation a sa fonction coût. Ici, puisque notre meilleur modèle est le Random Forest, nous allons nous intéresser à la fonction coût du Random Forest pour un problème de classification binaire.

La fonction coût pour Random Forest en classification est la minimisation du coefficient de GINI pour chaque nœud de chaque arbre.

# GridSearchCV

Maintenant que nos données sont prêtes pour la modélisation, il s’agit de choisir le meilleur modèle de scoring en en testant 4. Nous mesurons leur performance après les avoir fine-tuné grâce au Grid SearchCV, c’est-à-dire après avoir trouvé les **meilleurs hyperparamètres** optimisant les performances du modèle pour nos données.

# Métrique d’évaluation : ROC\_AUC

Puisqu’il s’agit de remboursement de prêt, la situation très dommageable pour Prêt à dépenser serait qu’un client soit catégorisé comme bon payeur, alors qu’il est un mauvais payeur (faux négatif).

On va donc chercher à maximiser le recall ou TPR (True Positive Rate).

En même temps, on veut pouvoir identifier les potentiels nouveaux clients bons payeurs, on va donc chercher à limiter le nombre de faux positifs (client catégorisé comme mauvais payeur alors qu’il est bon payeur).

On va donc chercher à minimiser le FPR (False Positive Rate).

On utilise ainsi la courbe ROC\_AUC, qui prend en compte le recall en ordonnée, et le FPR en abscisse.

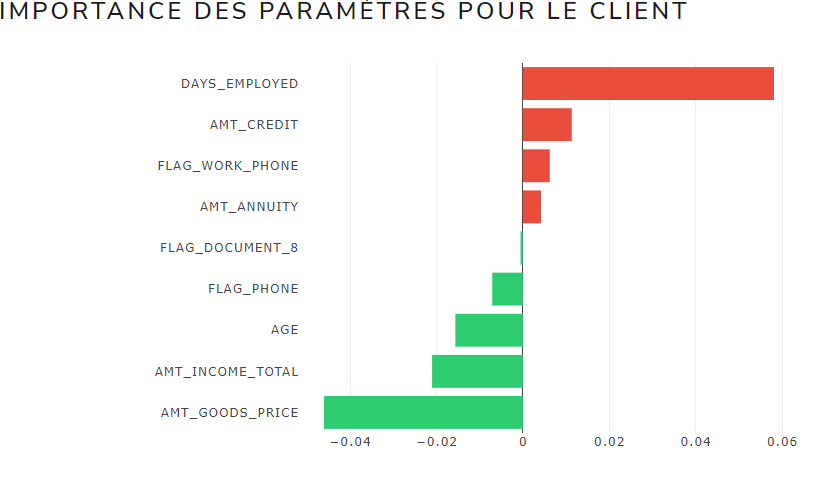
L’interprétabilité globale et locale du modèle

Notre modèle est à destination d’un public néophyte (les conseillers de clientèle) qui devra présenter les résultats du modèle à ses clients, et être capable d’expliquer ces résultats. C’est pourquoi nous avons créé un module pour expliquer les résultats du scoring.

Nous avons utilisé deux approches : **Shap** et **Lime** (Local Interpretable Model-agnostic Explanation). L’avantage de ces deux librairies est qu’elles s’appliquent à n’importe quel modèle de Machine Learning, et en particulier les modèles « black box » comme les réseaux de neurones ou les modèles basés sur la descente de gradient. A noter que les modèles « white box » peuvent déjà s’interpréter seuls via les scores de feature importance.

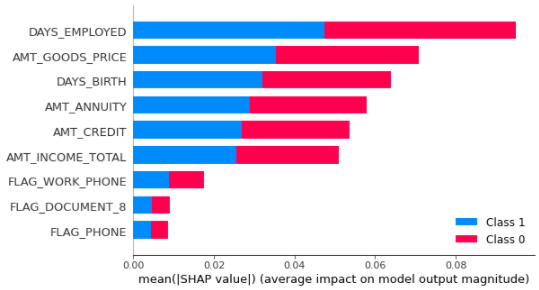
La différence entre une interprétation globale et une interprétation locale est le scope : en interprétation globale, on s’intéresse à l’ensemble du jeu de données, alors qu’en interprétation locale on zoom sur un client (dans notre cas) en particulier.

# Nous avons utilisé Lime pour expliquer localement le modèle

Pour un point donné, Lime perturbe aléatoirement les features, et récupère les prédictions pour ces points additionnels. Cela permet de dresser la situation pour un point donné.

Ici, nous utilisons le TabularExplainer puisque nos données sont sous forme de dataframe.

# Nous avons utilisé Shap pour expliquer globalement le modèle



On ajoute chaque feature une par une au modèle et on évalue sa contribution à la prédiction. Pour « enlever » les autres features, on les remplace par leur moyenne ou leur médiane.

Ici, nous avons utilisé un TreeExplainer puisque notre meilleure modèle est un algorithme basé sur des arbres de décision.

Les limites et les améliorations possibles

En termes d’**explicabilité du modèle**, nous constatons les limites des deux approches.

Par exemple, Lime ne gère pas correctement les données encodées, alors que Shap a besoin de l’entièreté du dataframe sinon il peut connaître des problèmes d’imprécision. Il faudrait trouver une manière optimale de combiner les deux approches.

En termes de choix de **métrique d’évaluation**, nous nous sommes basées sur notre intuition : peut-être faudrait-il ainsi plutôt choisir une métrique d’évaluation en fonction des besoins des équipes métiers.

En termes de **qualité des données**, il serait souhaitable d’avoir moins de données manquantes, d’avoir des données mises à jour à l’instant T. Il serait aussi intéressant de créer de nouvelles variables à partir des variables existantes en coordination avec les équipes métier.