**Note méthodologique**



*Implémentez un modèle de scoring*

Charlotte MILLET

Table des matières

[Contexte du projet 4](#_Toc110875062)

[Prétraitement des données 4](#_Toc110875063)

[Méthodologie d’entraînement 5](#_Toc110875064)

[GridSearchCV 6](#_Toc110875065)

[GridSearchCV + SMOTE 6](#_Toc110875066)

[GridSearchCV + SMOTE Sous-Echantillonné 6](#_Toc110875067)

[Métrique d’évaluation 6](#_Toc110875068)

[Fonction de coût métier (1 page max) 6](#_Toc110875069)

[API 7](#_Toc110875070)

[Limites et améliorations possibles (1 page max) 7](#_Toc110875071)

# Contexte du projet

La société financière *Prêt à dépenser* propose des crédits à la consommation pour des personnes ayant peu ou pas du tout d'historique de prêt.

L’entreprise souhaite mettre en œuvre un outil de *scoring crédit* pour calculer la probabilité qu’un client rembourse son crédit. Cet outil classifiera ensuite 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.).

De plus, les chargés de relation client ont fait remonter le fait que les clients sont de plus en plus demandeurs de transparence vis-à-vis des décisions d’octroi de crédit. Cette demande de transparence des clients va tout à fait dans le sens des valeurs que l’entreprise veut incarner.

Prêt à dépenser décide donc de développer un dashboard interactif pour que les chargés de relation client puissent à la fois expliquer de façon la plus transparente possible les décisions d’octroi de crédit, mais également permettre à leurs clients de disposer de leurs informations personnelles et de les explorer facilement.

# Prétraitement des données

Les données à disposition pour réaliser cet outil sont disponibles sur le site de [Kaggle](https://www.kaggle.com/c/home-credit-default-risk/data). Ces données présentent beaucoup d’informations sur chacun des clients : des informations générales (âge, sexe), des informations bancaires sur les prêts précédents (nombre de prêts précédents, durée), etc.

Avant d’entraîner des modèles de *Machine Learning* sur ces données, il faut réaliser des étapes de prétraitement.

* ***Feature engineering***

Afin d’avoir des variables interprétables plus facilement par les conseillers bancaires, nous créons des nouvelles variables à partir de celles disponibles et nous encodons les variables catégorielles.

* **Suppression des variables comprenant trop de valeurs manquantes**

Les données présentent de nombreuses valeurs manquantes. Nous supprimons dans un premier temps les variables qui présentent un taux de complétion inférieur à 50%.

* **Séparation train/test**

Pour mener à bien l’entraînement de nos modèles, nous séparons notre jeu de données en un jeu d’entraînement (70%) et un jeu de test (30%).

* **Imputation des valeurs manquantes**

Nous imputons ensuite les données manquantes restantes à l’aide de la médiane (ou valeur la plus fréquente) de la variable dans le jeu de données d’entraînement pour les données numériques (pour les variables catégorielles).

* **Normalisation**

Les variables finales sont enfin normalisées (normalisation calculée sur le jeu d’entraînement).

# Méthodologie d’entraînement

L’objectif ici est d’entraîner des modèles de *Machine Learning* pour prédire si la demande de crédit d’un client doit être acceptée ou non. Pour cela, on s’aide de la variable ‘TARGET’ qui est notre variable cible. La variable ‘TARGET’ vaut 1 si le crédit doit être accepté, 0 sinon.

Il est à noter que les données sont déséquilibrées puisque nous avons 91% de clients à qui il faudrait accorder un prêt.

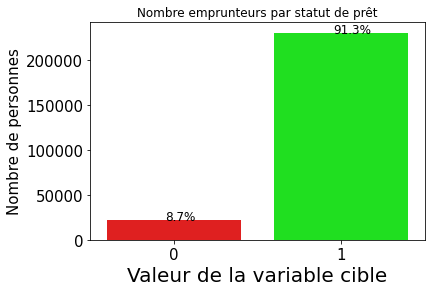
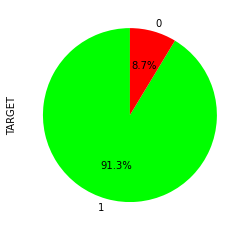


Figure 1 - Répartition des clients

Figure 2 - Répartition des clients

Nous entraînement tout d’abord un classifieur basique qui va nous servir de baseline : un *DummyClassifier* avec la stratégie *most\_frequent*. Ce classifieur va cherche la valeur de la variable cible la plus fréquente dans le jeu d’entraînement (1 dans notre cas) et va prédire cette valeur pour tous les individus du jeu de de test. Nous nous servons de ce modèle afin de mettre en perspective les résultats obtenus avec les modèles suivants.

Nous allons ensuite entraîner plusieurs modèles :

* Une Forêt aléatoire (*Random Forest*) dans laquelle nous ferons varier les hyperparamètres *n\_estimators*, *max\_depth*, *bootstrap* et *min\_samples\_leaf*.
* Des Algorithmes de *Gradient Boosting*
  + XGBoost (*eXtreme Gradient Boosting*) dans lequel nous ferons varier *max\_depth*, *learning\_rate* et *n\_estimators*.
  + LightGBM (*Light Gradient Boosting Machine*) dans lequel nous ferons varier *max\_depth*, *num\_leaves* et *n\_estimators*.
* Une Régression logistique (*Logistic Regression*) dans laquelle nous ferons varier *penalty* et *C* (inverse de la force de régularisation).

Ces modèles vont être entraînés sur le jeu de données d’entraînement créé dans la partie précédente et qui contient 176 493 lignes et 515 colonnes.

Nous avons vu précédemment que le jeu de données était déséquilibré (91% de clients solvables). Ce déséquilibre peut rendre difficile l’apprentissage de nos modèles et nous allons donc essayer de réduire son impact. Il existe pour cela plusieurs solutions :

* Sur-échantillonner la classe minoritaire. Pour cela, nous pouvons par exemple utiliser SMOTE (Synthetic Minority Oversampling Technique). L’algorithme SMOTE fonctionne de la manière suivante :
  + Il choisit aléatoirement un individu de la classe minoritaire (**individu 1**).
  + Il cherche ses *k* plus proches voisins (*k* est un paramètre de SMOTE).
  + Il choisit aléatoirement un individu parmi les plus proches voisins (**individu 2**).
  + Il crée un nouvel individu entre les **individus 1 et** **2**. Le positionnement de ce nouvel individu par rapport aux deux précédents dépend du paramètre *alpha* de SMOTE.
  + Ces étapes sont répétées jusqu’à avoir le nombre d’individus désiré.
* Sous-échantillonner la classe majoritaire. Nous choisissons pour cette étape la fonction *RandomUnderSampler*. Cette fonction effectue *N* (*N* est le nombre d’individus de la classe minoritaire) tirages au sort d’individus parmi ceux de la classe majoritaire. À la fin des tirages, elle ne conserve que les individus tirés au sort. Les tirages peuvent se faire avec ou sans remise.

## GridSearchCV

Dans un premier temps, nous entraînons chacun de ces modèles à l’aide de *GridSearchCV*, qui va nous permettre de trouver les hyperparamètres qui donnent les meilleurs résultats. De plus, *GridSearchCV* réalise une *cross validation*. C’est-à-dire que pour chaque combinaison d’hyperparamètres à tester, le jeu de données va être coupé en N parties où N-1 parties serviront à l’entraînement du modèle et la dernière partie servira à évaluer le modèle ainsi testé. L’opération est répété N fois (pour chaque combinaison d’hyperparamètres) et le résultat final est la moyenne des résultats obtenus sur chacune des *cross validation*.

## GridSearchCV + SMOTE

Nous avons ensuite réalisé un sur-échantillonnage de notre classe minoritaire jusqu’à n’avoir que 2 fois plus d’individus majoritaires. Nous avons ensuite entrainés chacun des modèles présentés précédemment à l’aide d’une grille de recherche d’hyperparamètres (*GridSearchCV*) sur notre jeu de données sur-échantillonné.

## GridSearchCV + SMOTE Sous-Echantillonné

Enfin, nous avons sur-échantillonné notre classe minoritaire puis sous-échantillonné notre classe majoritaire puis nous avons encore réentraîné nos modèles avec *GridSearchCV*.

# Métrique d’évaluation

Pour choisir le meilleur modèle, nous devons choisir une métrique d’évaluation. Nous avons tout d’abord calculé plusieurs métriques pour chacun des modèles testés afin d’avoir la comparaison la plus pertinente possible.

La métrique choisi pour l’optimisation des hyperparamètres et le choix du meilleur modèle est l’aire sous la courbe ROC (AUC). Le meilleur modèle est alors *XGBoost* entraîné simplement avec *GridSearchCV*. Ce modèle est d’ailleurs le meilleur pour la plupart des métriques calculées.  
Cependant, nous pouvons remarquer que le modèle *LGBM* entraîné avec une grille de recherche simple (*GridSearchCV*) a des scores très proches du modèle précédent. De plus, *LGBM* présente un temps d’entraînement bien plus faible que *XGBoost*. Nous choisirons donc finalement ce modèle comme notre meilleur modèle.

Il a pour hyperparamètres :

Figure 3 - Tableau des performances des modèles testés

* *Max\_depth* : 5
* *N\_estimators* : 300
* *Num\_leaves* : 20

# Fonction de coût métier

Le modèle précédent a été entrainé afin d’optimiser le métrique choisie « *roc\_auc* ». Cependant, notre modèle a pour vocation de servir à une analyse métier : Doit-on accepter la demande de crédit d’un client ? Nous pouvons alors définir une fonction de coût métier en rapport avec cette analyse. En effet, le but de la banque est de perdre le moins d’argent possible, elle veut donc limiter le nombre de crédits accordés à des clients non solvables.

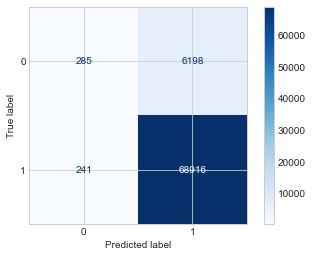
Nous pouvons définir 4 classifications de clients :

* Les **vrais positifs** sont les clients solvables dont la demande de crédit est accordée.
* Les **faux positifs** sont les clients non solvables dont la demande de crédit est accordée.
* Les **vrais négatifs** sont les clients non solvables dont la demande de crédit est refusée.
* Les **faux négatifs** sont les clients solvables dont la demande de crédit est refusée.

Dans le cas des vrais positifs et des vrais négatifs, la classification est correcte, la banque ne va pas perdre d’argent.  
Dans le cas des faux positifs et des faux négatifs, cela peut poser problème. Un client faux positif risque de ne pas rembourser son prêt et cela constitue un risque de perte d’argent pour la banque. Un client faux négatif, quant à lui, constitue un manque à gagner car ce client ne rapportera rien à la banque alors qu’il aurait à priori remboursé son prêt.

Un client étiqueté comme **faux positif** (fp) coûte 10 fois plus cher qu’un **faux négatif** (fn). La fonction de coût métier doit pénaliser 10 fois plus les **faux positifs**.

Nous avons donc la fonction de coup suivante :



Notre meilleur modèle pour le moment (LGBM) présente la matrice de confusion ci-contre (qui représente un coût métier de 62 221).

Nous cherchons ensuite à minimiser le coût métier de notre modèle. Pour cela, nous utiliserons deux approches différentes.

Figure - Matrice de confusion du modèle LGBM

Dans la première approche, nous gardons notre meilleur modèle tel quel. Précédemment, nous avions établi qu’un client était solvable si son score était supérieur ou égal à 0.5. Nous allons maintenant chercher le seuil de probabilité (si le score d’un client est supérieur ou égal au seuil alors sa demande de crédit est accordée) qui minimise le coût métier. Nous trouvons alors que le seuil optimal est 0,91.

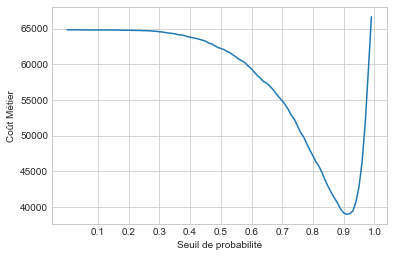
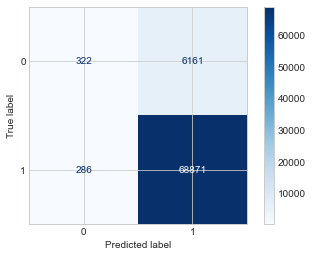
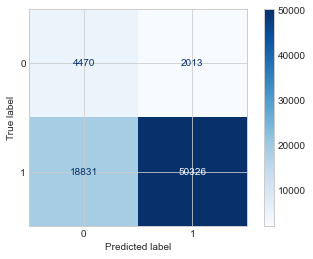
   
La seconde approche se base sur la fonction *fbeta\_score* (moyenne harmonique pondérée du rappel et de la précision). Cette fonction attend un paramètre *beta* qui donnera plus de poids à la précision si . Nous allons tester plusieurs valeurs de beta puis réentraîner notre meilleur modèle (*LGBM*) sur une grille de recherche d’hyperparamètres (avec comme métrique d’évaluation *fbeta\_score*). Pour chacun des beta testés, nous calculons le coût métier. Nous choisirons comme beta optimal celui qui minimisera ce coût. Nous trouvons finalement un coût métier de 61 896 pour tous les beta testés.

Figure 8 - Matrice de confusion avec fbeta\_score

Figure 6 - Coût métier en fonction du seuil de probabilité choisi

Figure 7 - Matrice de confusion pour le meilleur seuil

# Interprétabilité locale et globale du modèle

Pour interpréter notre modèle, nous pouvons utiliser les bibliothèques *Shap* ou *Lime*. J’ai choisi d’utiliser *Shap*.

## Interprétabilité locale

L’interprétation locale permet de savoir quelles variables ont influencé la prédiction d’un client en particulier. Par exemple, dans le cas d’un client choisi aléatoirement, nous pouvons voir que les variables *CODE\_GENDER*, *EXT\_SOURCE\_2* et *DAYS\_LAST\_PHONE\_CHANGE* (variables en rose) ont influencé positivement sa demande de crédit alors que *DAYS\_EMPLOYED* et *REGION\_RATING\_CLIENT\_W\_CITY* (variables en bleu) ont plutôt joué en sa défaveur.

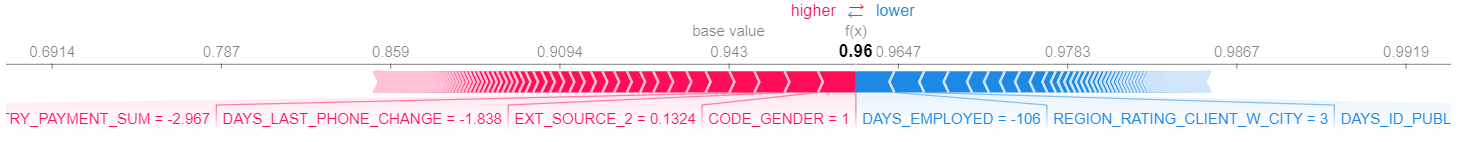


Figure 9 - Force plot

## Interprétabilité Globale

Figure 10 - Summary plot

L’interprétation Globale permet de son côté de savoir quelles variables ont le plus d’importance sur les prédictions pour l’ensemble des clients.

Nous pouvons voir ici que les variables *EXT\_SOURCE\_2*, *EXT\_SOURCE\_3*, *CODE\_GENDER* et *PAYMENT\_RATE* sont les quatre variables les plus importantes dans la prédiction.

# API

Pour ce projet, une api de prédiction a été déployée sur Heroku. Lorsque nous choisissons un identifiant de client, l’API va chercher dans un fichier de données toutes les valeurs des variables qui lui sont associées. L’API va ensuite prédire et affiché le score associé au client à l’aide du meilleur modèle que nous lui avons également donné.

# Limites et améliorations possibles

Plusieurs améliorations peuvent être envisageables :

* Dans un premier temps nous avons gardé toutes les variables. Il serait intéressant de se demander si des critères comme le sexe du client doivent vraiment entrer en compte dans la décision.  
  De plus, au-delà de l’éthique, il y a peut-être des variables qui ne sont pas pertinentes pour entraîner un modèle. Seule une personne du secteur bancaire pourrait nous éclairer sur ce point afin de faire une présélection des variables plus pertinente avant l’entrainement.
* Ensuite, nous avons les données concernant un client mais pas celle du reste de sa famille (conjoint/conjointe). Avoir le revenu du reste du foyer pourrait améliorer la décision.
* Les poids attribués aux faux positifs et faux négatifs ont été choisi de manière arbitraire. Il faudrait faire une étude plus poussée avec la banque afin d’identifier le coût réel de chacun des cas.
* Le jeu de données comprend beaucoup de valeurs manquantes. Le premier point pourrait nous permettre de sélectionner des variables plus pertinentes et donc peut-être d’éliminer des variables qui comprennent beaucoup de valeurs manquantes et peu pertinentes, que nous avons gardées dans notre étude. Nous pourrions aussi essayer de trouver d’autres sources de données afin de compléter celles déjà obtenues.