

**P7 - Implémentez un modèle de scoring**

*Note méthodologique*

[**Introduction**](#_4d6ol4j5whgc) **3**

[**Méthodologie d'entraînement du modèle**](#_i11aosn613ar) **4**

[Pré-processing](#_hmnmgdaeif4m) 4

[EDA](#_o0ety9jahwnt) 4

[Nettoyage](#_ypluz4o38vj3) 4

[Modélisation](#_kv83p2yz3so0) 4

[Train/test split](#_lif1d49t2g8b) 4

[Encodage, imputation et Standardisation](#_xok01mh01xmt) 4

[Entraînement de plusieurs modèles](#_zbqaubeh3hra) 5

[Description des métriques d’évaluation](#_4vgka59k96y9) 5

[Gestion du déséquilibre des classes](#_kc8qgdmjmhmi) 5

[Première approche : Re-sampling](#_nygmhqs8xbaa) 5

[Seconde approche : hyperparamètre class\_weights](#_oc5ntpcb2yc9) 6

[Feature Engineering](#_m0eebe7umjza) 6

[Feature Creation](#_21bo7dednfay) 6

[Feature selection](#_7sd3gntc7heb) 6

[**Evaluation de la performance du modèle**](#_9r3h7rdsmf9q) **6**

[Fonction coût métier](#_ambvqepo6hvd) 6

[Algorithme d’optimisation et métrique d’évaluation](#_blnjq2lj03ze) 7

[Variation du seuil de décision](#_2e5bl11noq8q) 7

[**Interprétabilité de la prédiction**](#_98vb6jbqtow5) **8**

[Interprétation globale : analyse des coefficients du modèle](#_3w1izwrv77pz) 8

[Interprétation locale : Shapley Values via le module SHAP](#_z0vp2qwjw0ox) 8

[**Limites et améliorations possibles**](#_otm54gn395tw) **9**

[Explorer l’ensemble de la base de donnée](#_ps015arwbe9m) 9

[Faire de l’ensembling (fusion de plusieurs modèles)](#_fnfcoquv1lk5) 10

[Réduction de dimension (PCA) pour éviter la “curse of dimensionality”](#_pn99xi05rz) 10

*Guilhem Berthou - Pierre-Antoine Ganaye (mentor)*

[*https://github.com/guilhembr/p7\_scoring*](https://github.com/guilhembr/p7_scoring)

## **Introduction**

Pour réaliser le nettoyage et l’exploration de la donnée, nous avons utilisé le [Notebook de la compétition](https://colab.research.google.com/drive/1uorVxsO816YOQMbkizlakGC21wS-xVHh#scrollTo=uN03kboJEVSi). Ce *notebook* a été développé dans le cadre de la compétition de *machine learning* Kaggle proposée par [*Home Credit default risk*](https://www.kaggle.com/c/home-credit-default-risk)*.* L’objectif de cette compétition est d’utiliser un jeu de données d’historique de demande de prêt afin de prédire si un prospect sera ou non capable de rembourser un crédit.

Il s’agit donc d’un problème de classification supervisée :

* Supervisée : Les données sont étiquetées dans le jeu de données d'entraînement (*application\_train*) qui servira de base à l'entraînement d’un modèle afin de prédire la capacité de remboursement (le *label*) en fonction des *features* client.
* Classification : Le *label* est une variable binaire, 0 (le client rembourse son crédit), 1 (le client est insolvable).

Le jeu de données contient 7 sources de données. Dans ce *notebook* nous nous concentrons sur les 2 jeu principaux :

* *application\_train/application\_test* : contiennent les informations de chaque demande de prêt. Chaque ligne des datasets correspond à une demande identifiée par SK\_ID\_CURR. Le jeu d'entraînement contient les TARGET. A l’inverse le jeu de test n’est pas étiqueté.

## **Méthodologie d'entraînement du modèle**

### Pré-processing

#### EDA

Nous constatons tout d’abord qu’il s’agit d’un dataset dont les classes de la variable à prédire (TARGET) sont déséquilibrées. Il y a ainsi beaucoup plus de lignes correspondant à des remboursements qu’à des défauts. Nous détaillerons dans la partie *c)* les réponses possibles à ce problème.

Nous constatons également une tendance claire : il est plus probable que les prospects les plus jeunes soient en défaut.

Les 3 variables EXT\_SOURCE sont négativement corrélées avec la TARGET. Plus leur valeur augmente, plus les clients sont solvables. Nous n’avons pas d’information supplémentaire quant à ce à quoi elles correspondent.

#### Nettoyage

Les variables correspondant à des durées mesurées par rapport à la date d’analyse du dataset sont négatives (DAYS\_BIRTH). Nous les passons en valeur absolue pour plus de lisibilité. Les valeurs aberrantes de la variable DAYS\_EMPLOYED sont remplacées par un champ vide.

### Modélisation

#### Train/test split

Du fait du contexte de compétition Kaggle, nous devons séparer le dataset *application\_train* - qui est le seul à être étiqueté - en un jeu d'entraînement et un jeu de test afin de mesurer la performance de nos modèles.

Le jeu étant relativement volumineux (*307.511, 243*) nous utilisons l'hyper paramètre stratify de la fonction train\_test\_split de *scikit learn* pour obtenir un échantillon représentatif :

* Premier split : séparation *d’application\_train* en un jeu test (*X\_train* = 80%) et un jeu de test (20%)
* Second split : séparation de *X\_train* en un échantillon (*X\_train\_sample* = 10%.*X\_train*)

#### Encodage, imputation et Standardisation

Dans la version finale de notre *notebook* de pre-processing (Voir [Summary Notebook](https://github.com/guilhembr/P7_Scoring/blob/main/model_training.ipynb)) nous avons pris les arbitrages suivants :

* Variables catégorielles : *One Hot Encoding* (OHE), imputation des valeurs manquantes par la valeur la plus fréquente (most frequent)
* Variables numériques : Imputation des valeurs manquantes par la médiane (median), Standardisation via StandardScaling.

#### Entraînement de plusieurs modèles

Nous avons entraîné plusieurs modèles afin d’identifier les plus performants pour répondre à notre problématique : DummyClassifier (baseline), LogisticRegression, Decision Tree, Random Forest, LGBM. Les hyper paramètres optimisés lors de l'entraînement sont détaillés dans le [notebook](https://github.com/guilhembr/P7_Scoring/blob/main/exploratory_notebook.ipynb) et sont recherchés via *GridSearchCV* afin de se prémunir du surapprentissage.

#### Description des métriques d’évaluation

Le choix des métriques dans un problème de classes déséquilibrées est important. En effet, utiliser une métrique classique d’un problème de classification comme la précision (accuracy\_score) serait peu pertinent. Le classifier pourrait prédire la classe la plus commune (clients solvables) et avoir un score de précision élevé sans être capable d’identifier les clients en défaut.

Nous avons donc utilisé l’AUROC (Area Under the ROC Curve) comme paramètre de scoring à optimiser lors de notre recherche d'hyper paramètres via *GridSearchCV*.

* + *AUROC* : on cherche à maximiser l’aire sous la courbe ROC qui est la représentation graphique des taux de vrais positifs en fonction des taux de faux positifs obtenus en faisant varier les seuils de décisions du classifieur : **%TP = f(%FP)**

En complément, pour apprécier la performance des modèles nous avons étudié les résultats des métriques suivantes :

* + *Recall* : Aussi appelé sensibilité, elle correspond au taux de vrais positifs (**TP / TP + FN**)
  + *F1-score* : Moyenne harmonique de la *précision* (**TP / TP + FP**) et du *recall* (on cherchera donc à le maximiser)
  + *Matrice de Confusion* : Représentation sous forme de matrice de la qualité d’un modèle. Chaque ligne correspond à une classe réelle, chaque colonne correspond à une classe estimée.

### Gestion du déséquilibre des classes

#### Première approche : Re-sampling

La première approche possible pour répondre à un problème de classes déséquilibrées est de rééquilibrer le nombre d’items pour chaque classe dans le dataset : le *re-sampling*. Les représentations graphiques de ces approches sont disponibles dans le [notebook](https://github.com/guilhembr/P7_Scoring/blob/main/exploratory_notebook.ipynb).

* + *Undersampling / Oversampling* : consistent respectivement à réduire le nombre d’item de la classe majoritaire pour être en ligne avec la classe minoritaire, et dupliquer des items de la classe minoritaire pour atteindre la population de la classe majoritaire. Le risque de provoquer du surapprentissage.
  + *Tomek links* : est une méthode *d’undersampling* consistant à privilégier la suppression des items proches (dans un plan en 2 composantes principales) des 2 classes afin de faciliter la distinction.
  + *SMOTE :*  est une méthode *d’oversampling* qui crée des plus proches voisins de la classe minoritaire (selon l’approche KNN)

#### Seconde approche : hyperparamètre class\_weights

La seconde approche possible pour répondre à ce problème est d’indiquer au modèle que l’on se situe dans une situation de déséquilibre de classes via un hyper-paramètre class\_weights. En indiquant le mode balanced à ce paramètre, le modèle ajuste automatiquement le nombre d'items de chaque classe en les pondérant de manière inversement proportionnelle à la fréquence de leur classe.

### Feature Engineering

#### Feature Creation

Nous avons créé 5 variables en fonction de notre compréhension des enjeux métiers du problème (ex : Ratio d’endettement façon *DTI - Debt Income Ratio* / montant de l’apport etc.).

#### Feature selection

Afin d’améliorer la performance de notre modèle nous avons également sélectionné les *features* les plus importantes en éliminant les variables non nécessaires via RFECV *(Recursive Feature Elimination with Cross-Validation)*. Cet algorithme proposé via *scikit learn* permet de sélectionner les features par récursivité (*Cross-Validation* en maximisant *l’AUROC*) en considérant un nombre de plus en plus faible de variables selon leur feature\_importance.

## **Evaluation de la performance du modèle**

### Fonction coût métier

L’objectif de ce projet est d’être capable de prédire la solvabilité d’un client. Autrement dit de prémunir la banque face au risque de défaut. Si nous traduisons cette problématique métier dans notre processus de modélisation, nous devons adapter la métrique de mesure de performance des modèles.

La matrice de confusion peut ainsi être lue du point de vue de la banque comme suit :

* **TP** (*True Positives*) : Clients insolvables correctement identifiés
  + Perte = 0 ; Gain = 0
* **FP** (*False Positives*) : Clients identifiés à tort comme insolvables
  + Perte = Intérêts du prêt (perte d’opportunité car prêt non alloué à tort)
* **TN** *(True Negatives)* : Clients solvables correctement reconnus
  + Gain = Intérêt du prêt
* **FN** (*False Negative)* : Clients insolvables non identifiés
  + Perte = Principal du prêt + intérêts non perçus

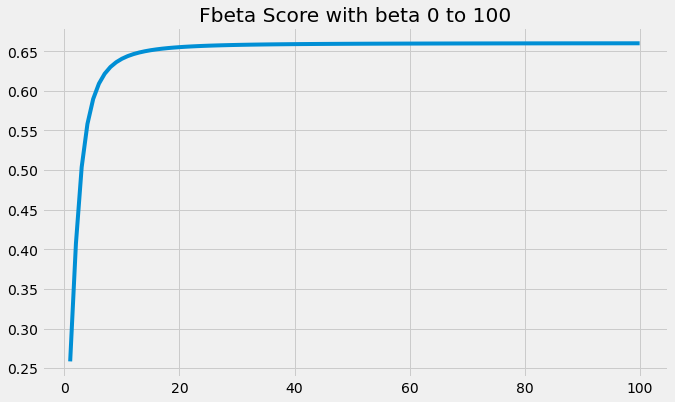
Les faux négatifs (FN) correspondent clairement au pire scénario que nous devons éviter. Il conviendrait donc de préférer le *recall* (% de clients insolvables identifiés) à la *précision* (% de clients insolvables identifiés comme tels).

### Algorithme d’optimisation et métrique d’évaluation

Afin de tenir compte de cette contrainte opérationnelle nous avons utilisé le fbeta\_score. Il s’agit d’un hyper-paramètre similaire au f1\_score qui ajoute une dimension beta permettant de déterminer le point du *recall* dans le score combiné :

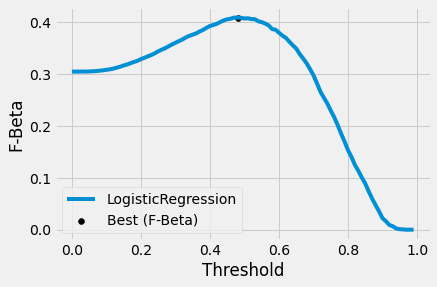
* beta < 1 favorise la *précision*
* beta > 1 favorise le *recall*

Nous avons ainsi défini un score à optimiser basé sur ce fbeta\_score lors de notre entraînement via la fonction make\_scorer de *scikit learn.* Nous avons fait varier la valeur de beta en fonction du fbeta\_score afin d’identifier la valeur à retenir.



### Variation du seuil de décision

Une fois le beta optimal identifié, nous avons fait varier le seuil de décision (à partir duquel une prédiction doit être considérée comme égale à 1) afin de maximiser le fbeta\_score.

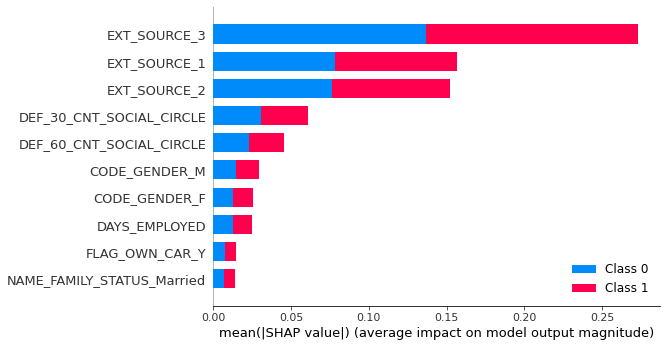


## **Interprétabilité de la prédiction**

Le modèle optimal que nous avons retenu à ce stade est la Régression Logistique, car il présentait un bon équilibre entre haute performance et temps d'entraînement réduit.

### Interprétation globale : analyse des coefficients du modèle

L’objectif de l’interprétation globale de notre modèle est d’obtenir une compréhension générale de l’importance des features utilisées par notre modèle pour calculer sa prédiction. Les coefficients d’une régression linéaire sont des associations conditionnelles : il permettent de quantifier la variation de *l'output* du modèle quand une feature donnée varie, en conservant les autres constantes.



Conformément à nos attente, nous constatons que les variables EXT\_SOURCES sont les plus contributrices, suivies par DEF\_30\_CNT\_SOCIAL\_CIRCLE et DEF\_60\_CNT\_SOCIAL\_CIRCLE qui correspondent aux nombres de personnes en défaut - respectivement depuis 30 et 60 jours - dans l’entourage des prospects.

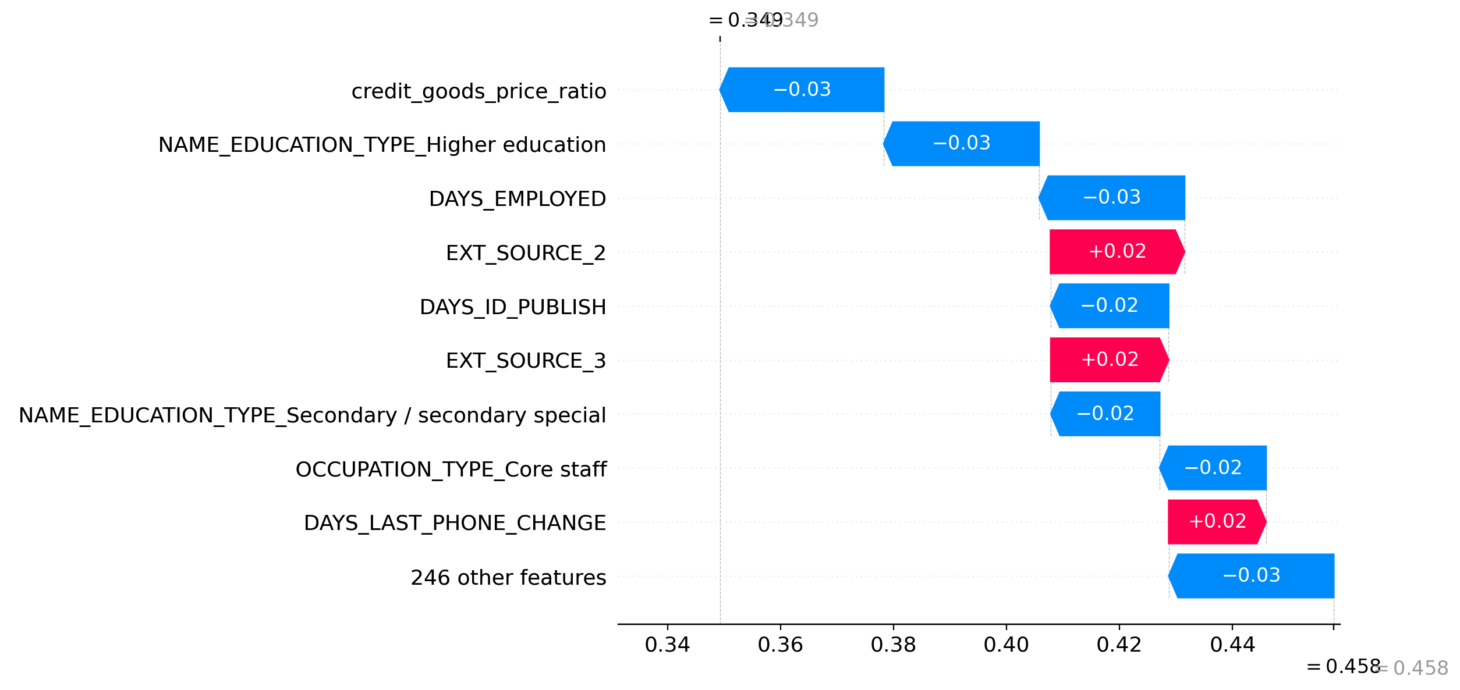
### Interprétation locale : Shapley Values via le module SHAP

Afin d’expliquer chaque sortie du modèle, nous avons calculé les valeurs de Shapley. Les valeurs de Shapley proviennent de la théorie des jeux et permettent de moyenner l’impact qu’une variable a pour toutes les combinaisons de variables possibles.

La méthode SHAP (*SHapley Additive exPlanation*) permet ainsi de décrire une prédiction comme la sommes des différents effets des variables (appréciés par les valeurs de shapley) que l’on ajoute à la valeur de base du modèle (valeur moyenne prédite sur le dataset d'entraînement).

On peut ainsi visualiser pour un prospect donné, la décomposition de son score :

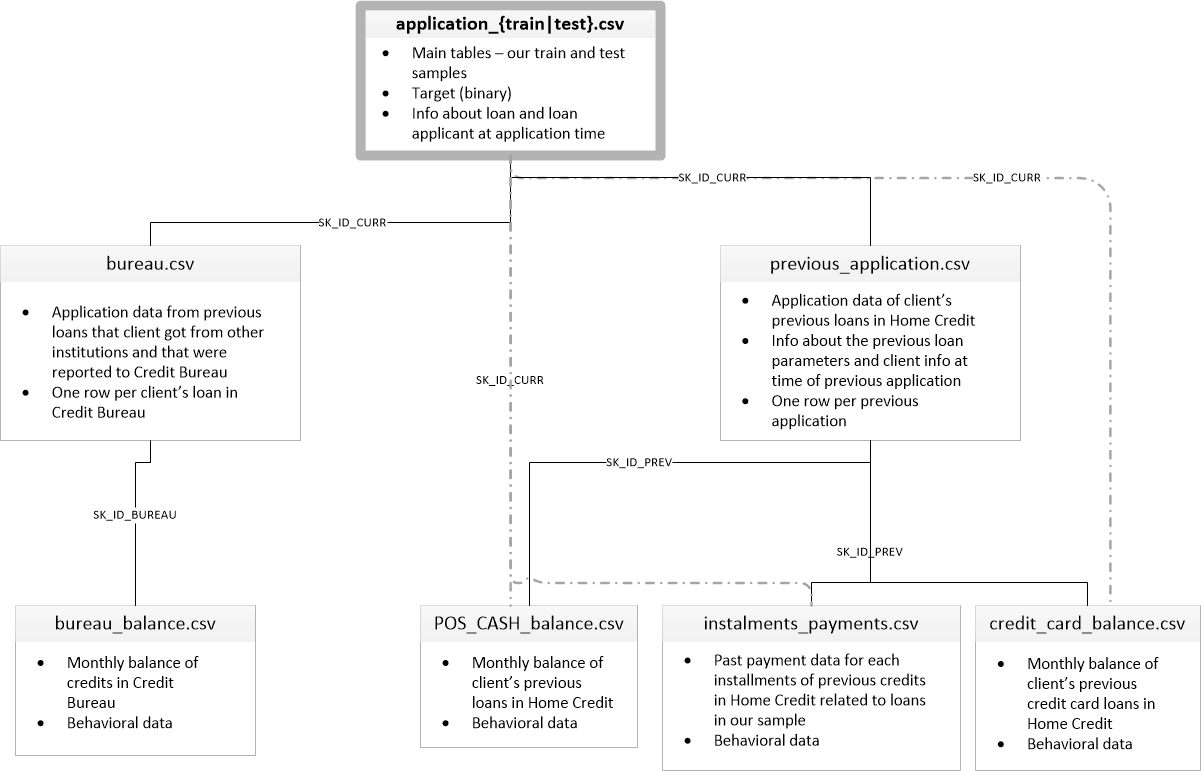
* Valeur de base : 0.458
* Prédiction : 0.349
* Interprétation des variables les plus contributrices :
  + Credit\_goods\_price\_ratio : % du prêt servant à financer l’achat du bien. Ici le prêt correspond uniquement à une partie du financement du bien. Cela est signe de bonne santé financière, réduisant ainsi le score du prospect.
  + NAME\_EDUCATION\_TYPE : Le prospect a reçu une Higher education réduisant ainsi sa probabilité de défaut



## **Limites et améliorations possibles**

### Explorer l’ensemble de la base de donnée

Nous avons exclusivement utilisé les données provenant de la table application\_train. Afin d’améliorer les performances de notre modèle, il conviendrait d’étendre l’analyse à l’ensemble des données présente dans la base de données. En réalisant des jointures avec les tables décrites ci-dessous nous pourrions ajouter des variables relatives à l’historique client ainsi qu’au solde de leur compte.



### Faire de l’ensembling (fusion de plusieurs modèles)

Les [gagnants](https://www.kaggle.com/competitions/home-credit-default-risk/discussion/64821) des compétitions [Kaggle](https://www.kaggle.com/code/amrmahmoud123/1-guide-to-ensembling-methods/notebook) réalisent généralement de [*l’ensembling*](https://www.toptal.com/machine-learning/ensemble-methods-kaggle-machine-learn)afin de bénéficier du [*wisdom of the crowd*](https://medium.com/nerd-for-tech/introduction-to-ensemble-methods-e81fd6f8180b). En entraînant 3 à 5 modèles sur le dataset, on obtient des prédictions qui sont ensuite utilisées comme input par un classifier qui réalise la prédiction finale : par exemple [StackingClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html).

### Réduction de dimension (PCA) pour éviter la “curse of dimensionality”

En complément de la sélection de features réalisée via RFECV, il pourrait être intéressant de réaliser une réduction de dimension par analyse en composante principale afin de réduire le nombre de variables du dataset post-encodage. En conservant le dataset actuel (dataset\_1) dans lequel les features ont été sélectionnées, on pourrait réaliser une ACP sur un dataset auxiliaire (dataset\_2) et inclure les variables résultant de ce traitement au dataset\_1.