Projet P7 OC DS :
Implémentez un modèle de scoring

Candidat : David Capelle
Evaluateur : Ibrahima Diakite
Mentor : Nicolas Michel

Date de soutenance : 25/01/2022

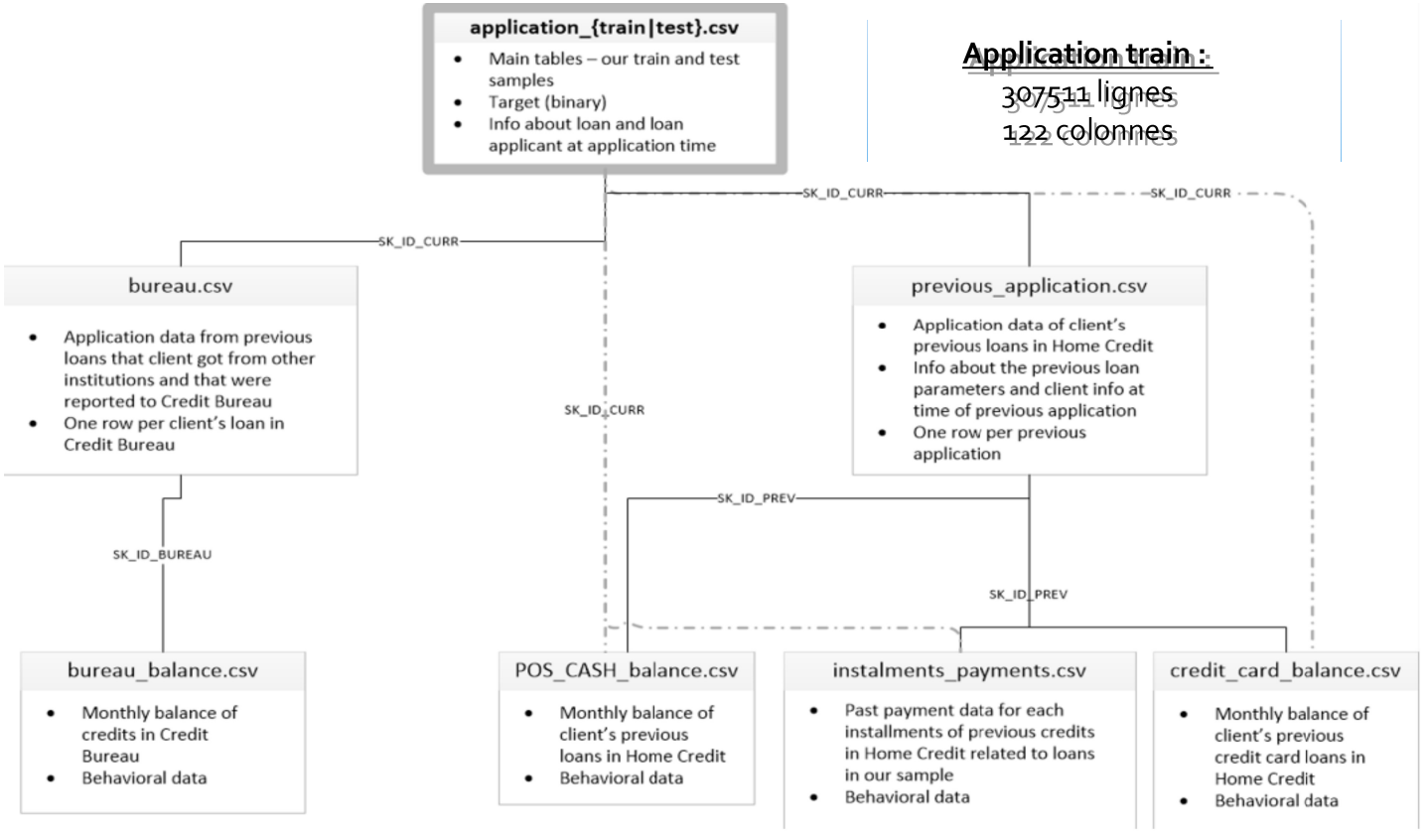
Plan de la soutenance

- Présentation du projet
- Présentation des données et du kernel Kaggle
- Modélisation : entraînement, optimisation et analyse de résultats
- Présentation du dashboard interactif
- Conclusion

Présentation du projet

- Souhait de développer un modèle de scoring de la probabilité de défaut de paiement du client
- Etude des données à partir d'un kernel kaggle pour faciliter le préparation et le feature enginneering
- Développement d'un dashboard intéractif pour visualiser les informations du client et clients similaires
- Déploiement sur un serveur distant du dashboard avec l'API de prédiction

Présentation des données



Application train :
307511 lignes
122 colonnes

Application test :
48744 lignes
121 colonnes

Pas de targ

Traitement des données : kernel kaggle

- Simplification du traitement des données pour les tâches de feature engineering avec le kernel kaggle :
  - Merge des différents datasets
  - Création de nouvelles variables
  - Agrégation de données
  - Encodage OneHot des variables catégorielles

=> jeu de données de 307507 lignes et 798 variables

Autres traitements des données

- Traitement des valeurs manquantes :
  - suppression des colonnes avec moins de 80 % de données disponibles
  - imputation des valeurs manquantes par la médiane

  => réduction du nombre de variables (798 à 566)
- Feature selection sur les variables importantes :
  - Utilisation du modèle LightGBM pour sélectionner jeu de données avec AUC > 0,7

  => réduction du nombre de variables (556 à 334)

Modélisation - Principes

- Définition d'une baseline (Dummy Classifier)
- Détermination d'une fonction de revenu personnalisée
- Détermination d'une méthode d'échantillonnage des données
  - selon métrique AUC et revenue function avec le modèle LogisticRegression
- Optimisation des hyper-paramètres par validation croisée avec GridSearcCV
- Analyse des résultats suivant métriques AUC et revenue function
- Choix du modèle final
- Ajustement du seuil de probabilité pour classe « Client défaillant »
- Interprétation de l'importance des variables

Fonction de revenu net personnalisée

- **Composante revenu** = (TN x TNRW) + (TP x TPRW)
- **Composante coût** = (FP x FPCW) + (FN x FNCW)
- **Revenu net** = (TN x TNRW) + (TP x TPRW) + (FP x FPCW) + (FN x FNCW)
- **Revenu net optimum** = (TN+FP) x TNRW + (TP+FN) x TPRW
- **Revenu net normalisé** = Revenu net / Revenu net optimum
- La fonction de revenu net accorde des poids au coût pour chaque faux positif/faux négatif et au revenu pour chaque vrai négatif/vrai positif
- Accord d’un poids plus élevé aux faux négatifs car le coût d'un faux négatif est plus important pour la banque entraînant une plus grande perte qu'un faux positif

Poids de la fonction de revenu net

- **TNRW (valeur = 10)** : poids accordé au revenu des intérêts annuels d'un client qui rembourse un prêt
- **TPRW (valeur = 1)** : poids accordé à la perte évitée si on accorde un crédit à un client défaillant
- **FPCW (valeur = -1)** : poids accordé au coût représenté par les revenus d'intérêts perdus en prédisant un client en tant que défaillant, alors qu'il est non défaillant
- **FNCW (valeur = -100)** : poids accordé au coût représenté par le montant du capital perdu en accordant un prêt à un client défaillant

Déséquilibrage des classes

Choix de la méthode d'échantillonnage des données

- Méthodes testées : RandomUnderSampler, SMOTE, Class weight
- Résultats :

Modèle	Méthode	AUC	Fonction Rev. net norm.
LogisticRegression	-	0.5101	0.1386
LogisticRegression	RandomUnderSampler	0.6933	0.3965
LogisticRegression	SMOTE	0.6920	0.3915
LogisticRegression	Class Weight	**0.6937**	**0.3983**

=> **Class weight plus efficace, mais choix de RandomUnderSampler pour minimiser la consommation des ressources système.**

Modélisation

Optimisation des hyper-paramètres

- **LogisticRegression** : 'max_iter' : [**500**], 'solver' : ['**sag**', 'saga'], 'C' : [**10**, 1.0, 0.1, 0.01]
- **SVC** : 'max_iter' : [**500**], 'C': [50, 10, 1.0, 0.1, **0.01**], 'kernel': ['poly', 'rbf', '**sigmoid**'], 'gamma': ['**scale**']
- **Random Forest** : 'max_features' : ['log2', '**sqrt**'], 'n_estimators' : [10, 100, **500**]
- **GradientBoosting** : 'learning_rate': [**0.1**, 0.05, 0.01], 'n_estimators' : [**100**]
- **MLP Perceptron** : 'max_iter' : [**100**], 'hidden_layer_sizes': [**(200, 100)**], 'activation': ['**tanh**'], 'solver': ['**sgd**', 'adam'],'alpha': [**0.0001**, 0.05], 'learning_rate': ['**adaptive**']

=> Le réglage des hyper-paramètres a été contraint par la limitation des ressources système

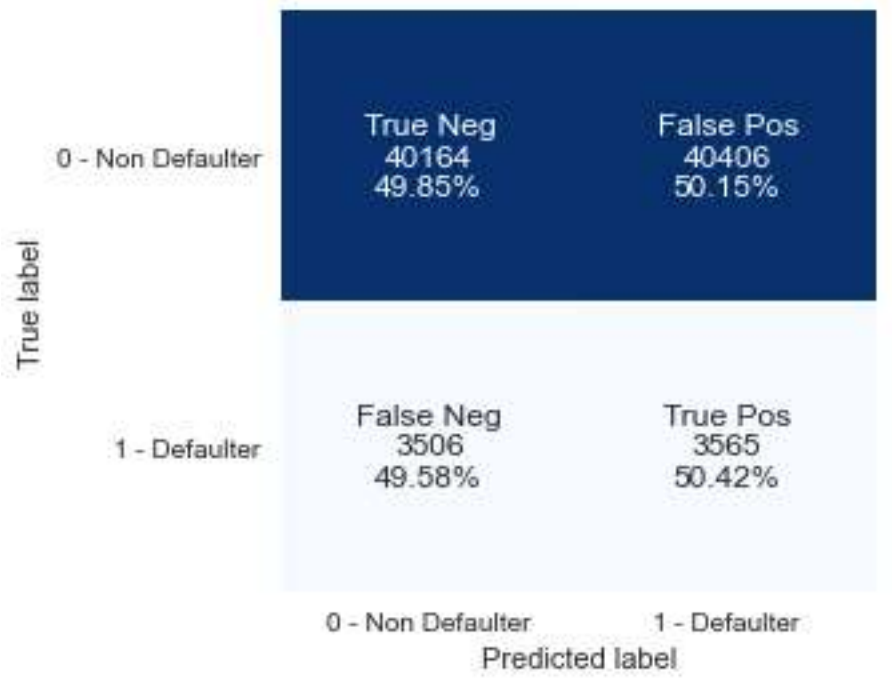
Modélisation

Analyse des résultats - Dummy classifier (baseline)

- Modèle simple, sans optimisation d'hyper-paramètres
- Résultats et métrique :
  - AUC = 0,5
  - **Ce modèle fictif n'est pas un bon modèle de prédiction, il y autant de bonnes et mauvaises prédictions par classe**

Modélisation – Matrice de confusion

Analyse des résultats - Dummy classifier (baseline)



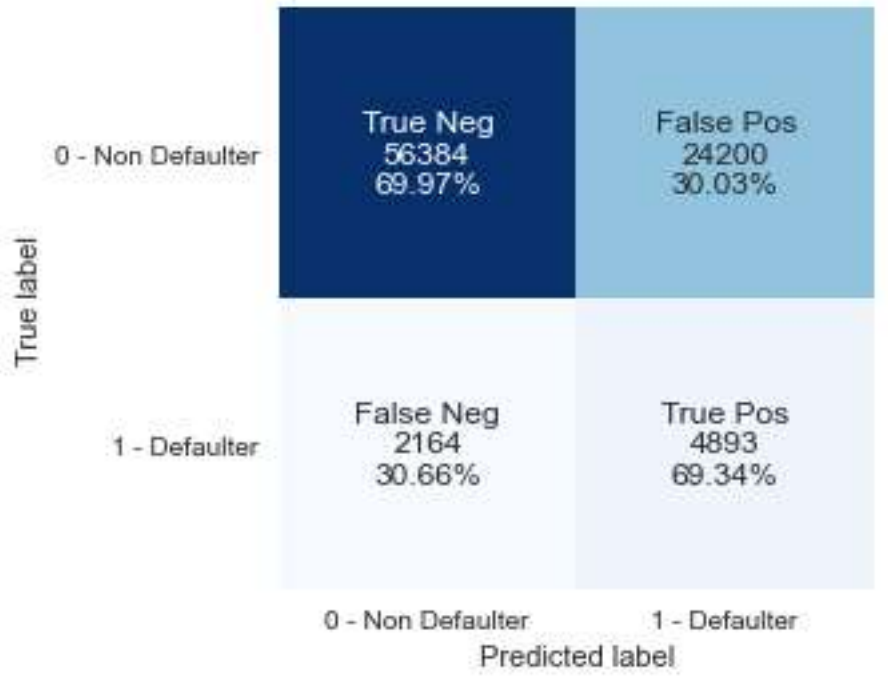
Modélisation

Analyse des résultats - LogisticRegression

- Modèle avec optimisation d'hyper-paramètres et sous-échantillonnage des données.
- Résultats et métriques :
  - AUC = 0,6965
  - Fonction revenue net normalisée = 0,4037
  - **Amélioration des résultats avec ce modèle de prédiction, il y moins de faux négatif et faux positif (environ 30%)**

Modélisation – Matrice de confusion

Analyse des résultats - LogisticRegression



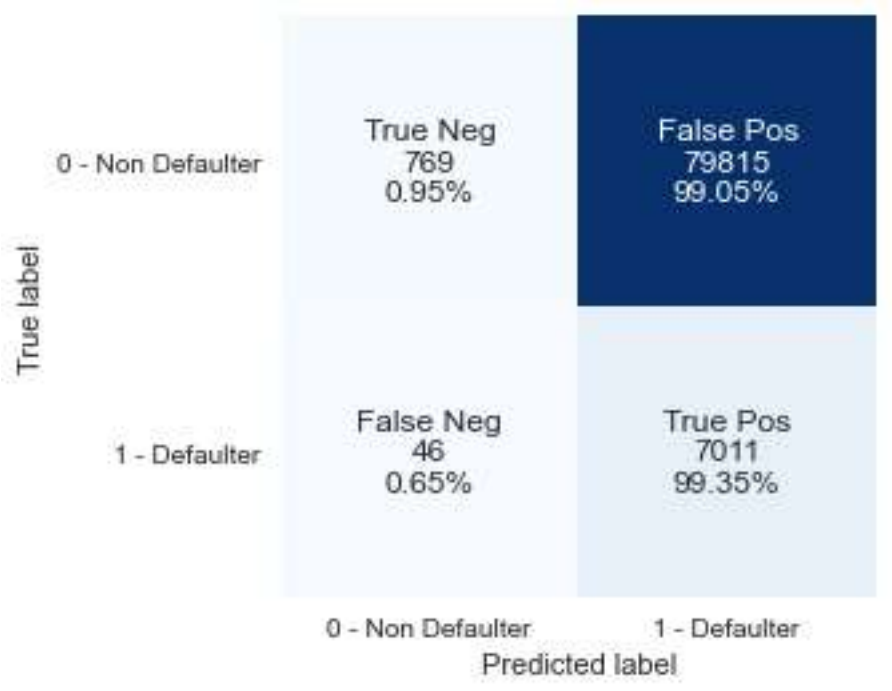
Modélisation

Analyse des résultats - SVC

- Modèle avec optimisation d'hyper-paramètres et sous-échantillonnage des données.
- Résultats et métriques :
  - AUC = 0,5015
  - Fonction revenue net normalisée = **-** 0,0858
  - **Ce modèle est éliminé car il ne prédit que des faux positifs et des vrais positifs (99 %, classe « client défaillant »)**

Modélisation – Matrice de confusion

Analyse des résultats - SVC



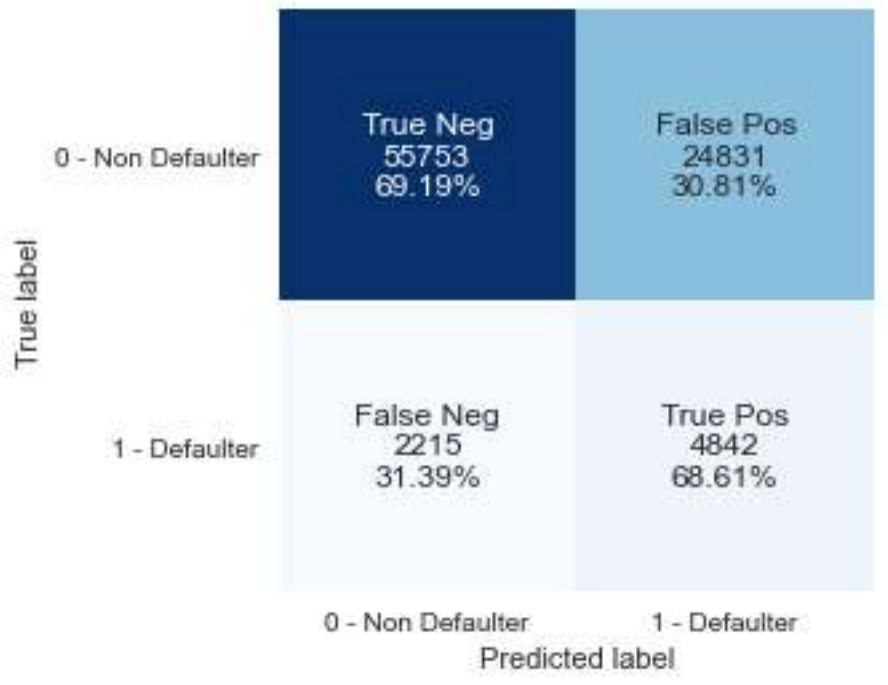
Modélisation

Analyse des résultats - Random Forest

- Modèle avec optimisation d'hyper-paramètres et sous-échantillonnage des données.
- Résultats et métriques :
  - AUC = 0,6890
  - Fonction revenue net normalisée = 0,3888
  - **Ce modèle propose des prédictions moins justes que la modèle LogisticRegression**

Modélisation – Matrice de confusion

Analyse des résultats - Random Forest



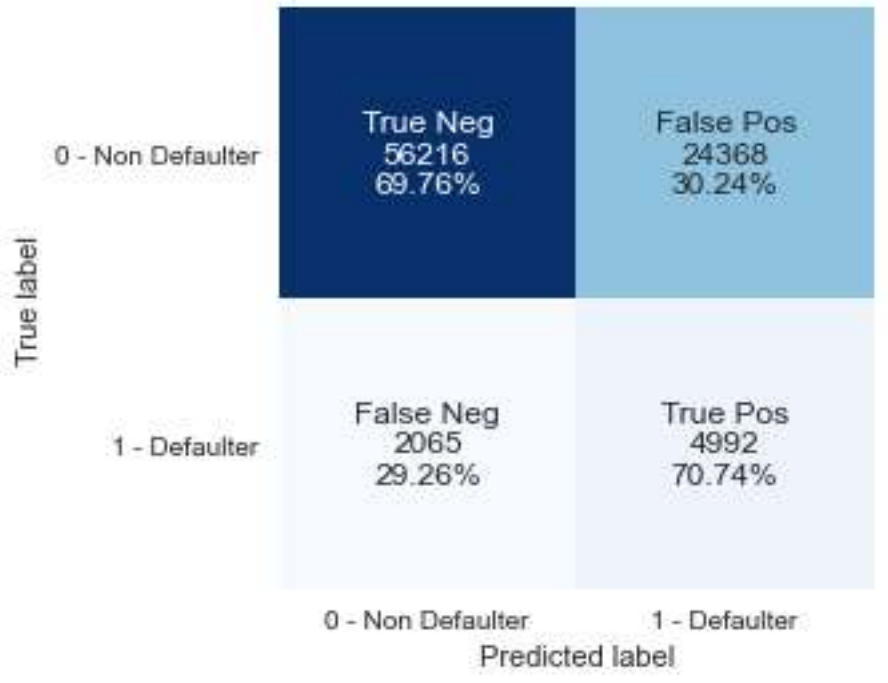
Modélisation

Analyse des résultats - Gradient Boosting

- Modèle avec optimisation d'hyper-paramètres et sous-échantillonnage des données.
- Résultats et métriques :
  - AUC = 0,7025
  - Fonction revenue net normalisée = 0,4137
  - **Ce modèle permet de minimiser le nombre de faux négatifs (moins de 30%) tout en limitant les faux positifs**

Modélisation – Matrice de confusion

Analyse des résultats - Gradient Boosting



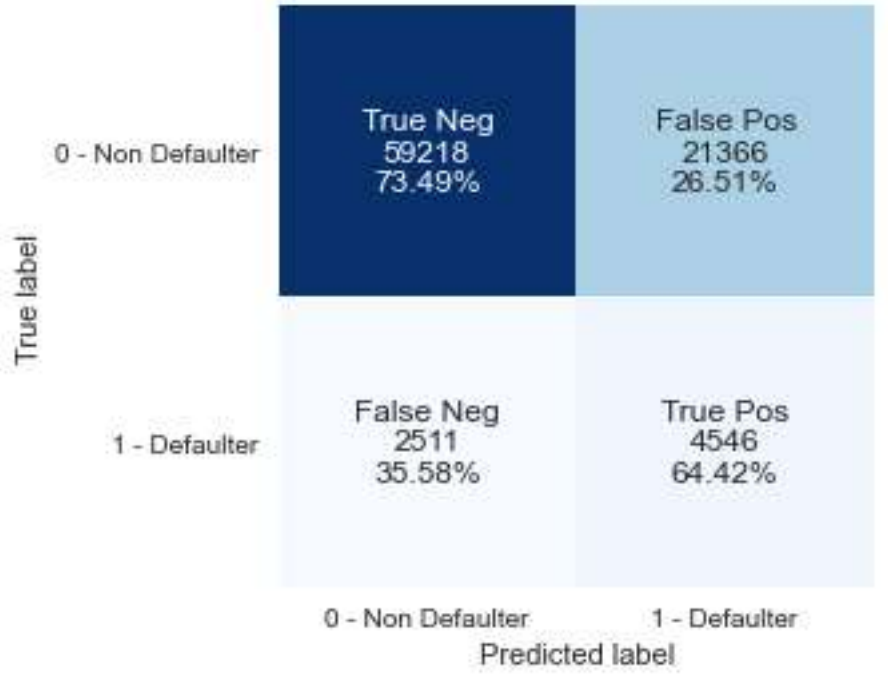
Modélisation

Analyse des résultats - MLP Perceptron

- Modèle avec optimisation d'hyper-paramètres et sous-échantillonnage des données.
- Résultats et métriques :
  - AUC = 0,6895
  - Fonction revenue net normalisée = 0,3989
  - **Ce modèle permet de minimiser le nombre de faux positifs, mais est moins performant que le modèle Gradient Boosting sur les faux négatifs (35%)**

Modélisation – Matrice de confusion

Analyse des résultats - MLP Perceptron



Modélisation

Synthèse des métriques sur les prédictions

Modèle	Méthode échantillonnage	AUC	Fonction Rev. net norm.
LogisticRegression	RandomUnderSampler	0.6965	0.4037
SVC	RandomUnderSampler	0.5015	- 0.0858
Random Forest	RandomUnderSampler	0.6890	0.3888
Gradient Boosting	**RandomUnderSampler**	**0.7025**	**0.4137**
MLP Perceptron	RandomUnderSampler	0.6895	0.3989

=> **Choix du modèle final sur les métriques = Gradient Boosting**

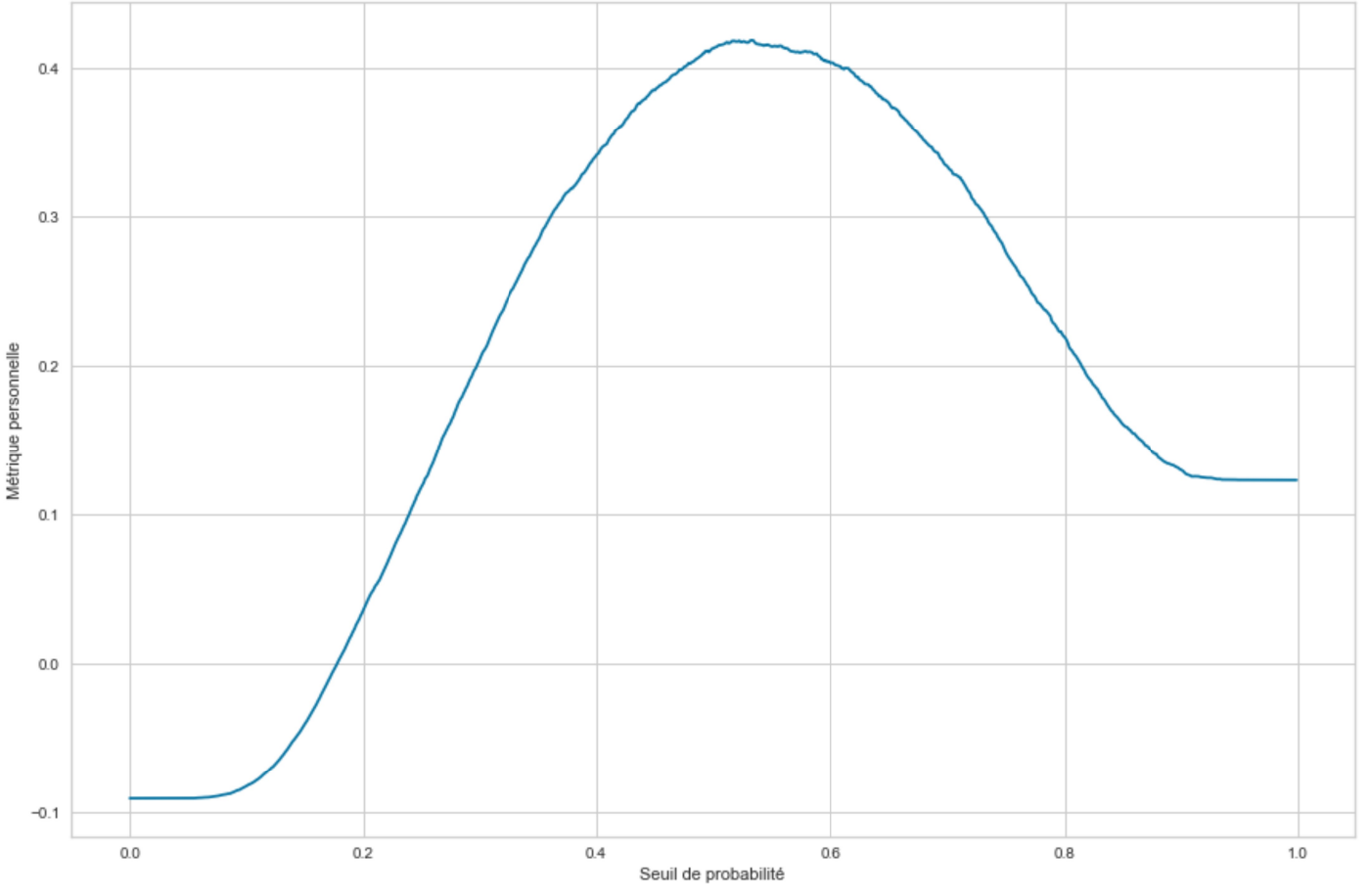
Modélisation

Ajustement du seuil de probabilité pour classe « Client défaillant »

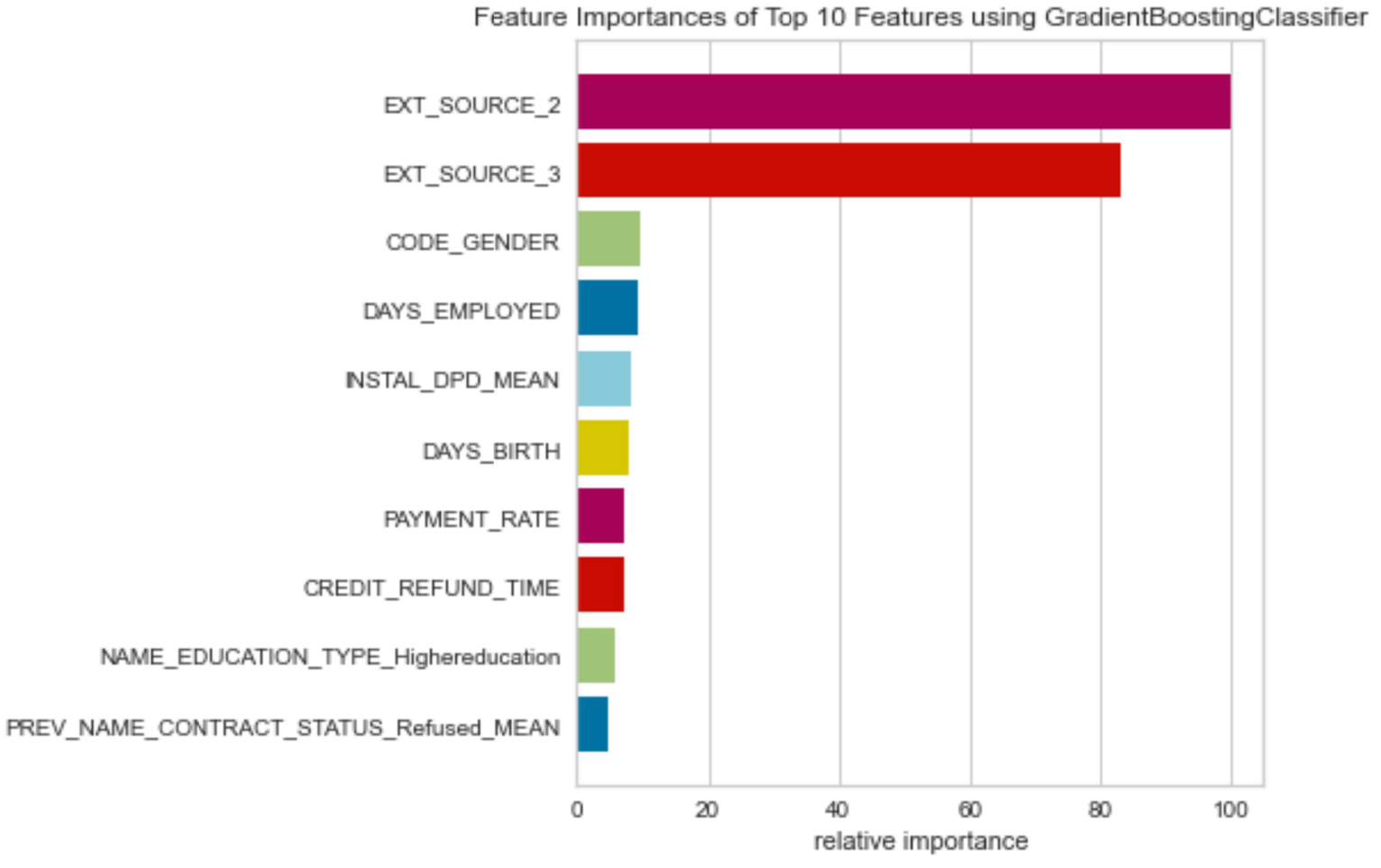
- Par défaut, seuil de probabilité = 0,5
- Détermination du seuil (threshold) qui optimise la fonction de revenu net sur le modèle Gradient Boosting => **Seuil = 0,52**
- Métriques associées :
  - AUC = 0,7014
  - Fonction revenu net normalisée = 0,4182

Modélisation

Ajustement du seuil de probabilité pour classe « Client défaillant »

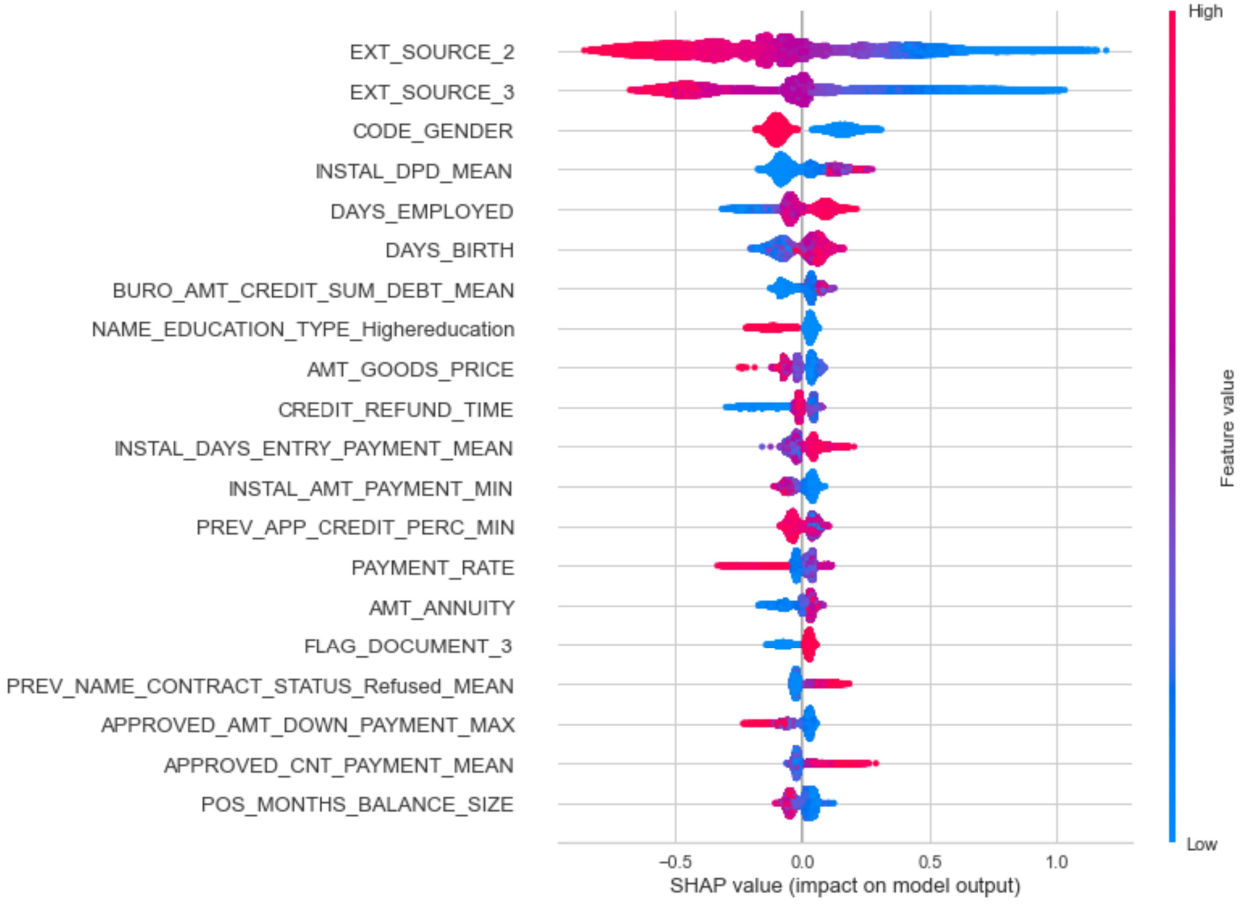


Importance des variables du modèle



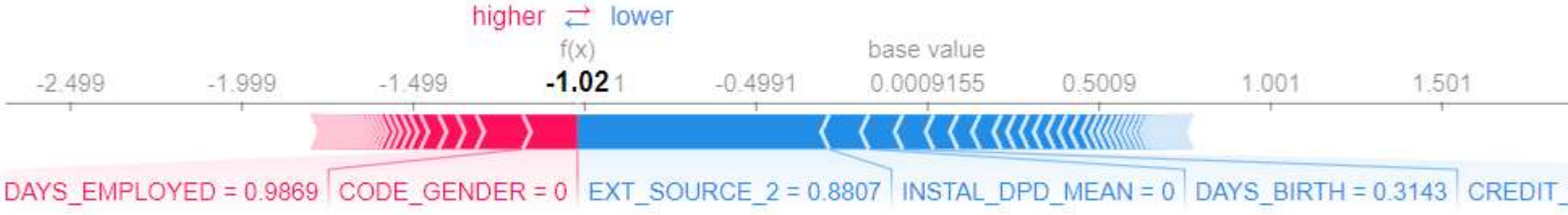
Analyse SHAP

Interprétabilité globale du modèle



Analyse SHAP

Interprétabilité locale (cas client non défaillant bien prédit)



Présentation applications web

API de prédiction et dashboard

- API de prédiction
  - Technologies : développement Flask, déploiement sur serveur Heroku
  - Url : https://davidp7apiflask.herokuapp.com/
- Dashboard interactif
  - Technologie : développement Streamlit, bibliothèque streamlit-aggrid, déploiement sur serveur Heroku
  - Url : https://davidp7dashboard.herokuapp.com

Conclusion

Points forts et axes d'amélioration

- Points forts
  - Le mise en œuvre d'une fonction de revenu net permet de proposer une métrique adaptée et paramétrable à la problématique du projet
  - Le ré-échantillonnage des données permet de corriger le déséquilibrage des classes.
- Axes d'amélioration
  - Utilisation d'une méthode d'échantillonnage des données plus performante (Class Weight ou SMOTE)
  - Optimisation plus fine des hyper-paramètres