Prédiction automatique du risque de non-remboursement des clients à l'aide d'un modèle de scoring

Ce document a pour objectif de détailler la construction du modèle ainsi que les métriques utilisées pour l'analyse de ses résultats.

Document réalisé dans le cadre du projet 7 du parcours Data Scientist
de la formation OpenClassrooms.

Table des matières

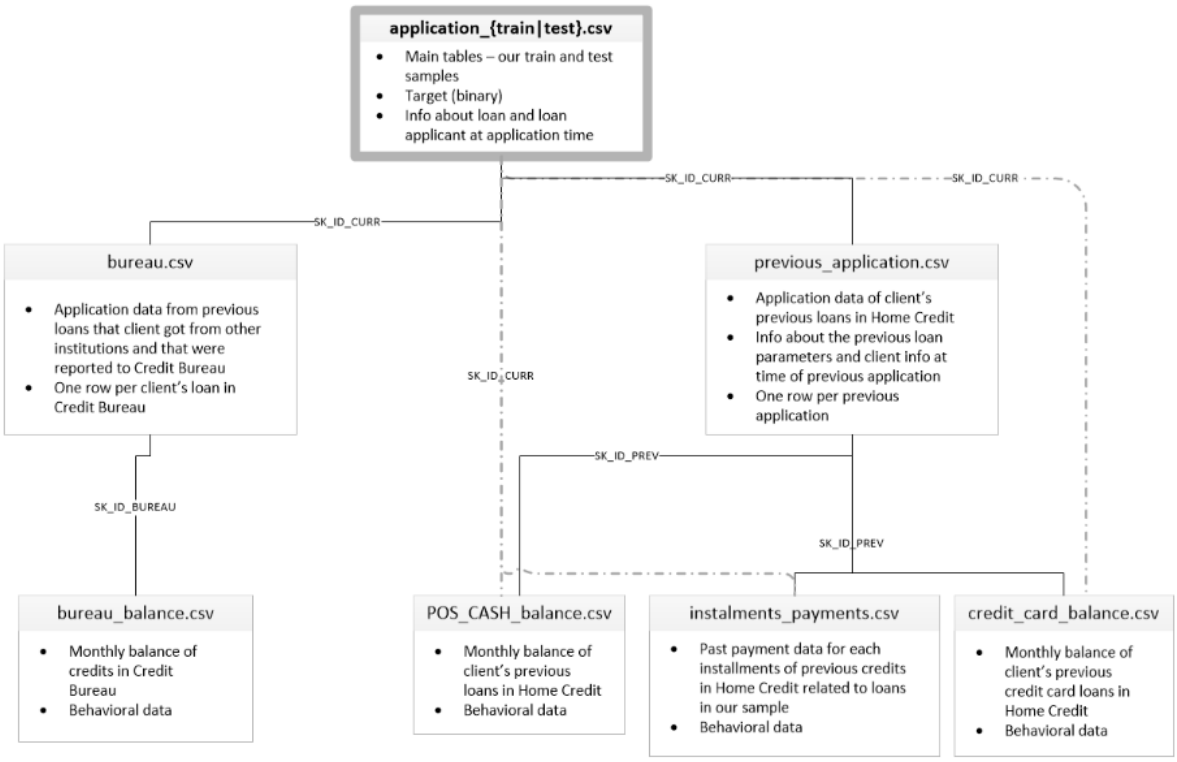
Données

Nous disposons de 7 fichiers CSV contenant des données relatives aux clients et à leurs prêts.

Une base de données « application_train.csv » nous a servi à entraîner notre modèle et contient une variable « TARGET » qui est la cible de notre prédiction.

La base de données « application_test.csv » n'a pas été utilisée pour entraîner le modèle, son utilisation est liée à l'élaboration d'une API de prédiction et d'un dashboard interactif et n'est pas détaillée dans ce document.

Structure des fichiers :



Feature engineering

Le feature engineering réalisé se base en grande partie sur le kernel kaggle suivant :

https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction

qui fournit une solide base pour l'élaboration de notre modèle.

Préparation des données

Nettoyage → Suppression des outliers, ajout d'un flag pour repérer ces outliers

Valeurs manquantes → Imputation de la médiane

Encodage des variables → Pour les variables à 2 catégories : Label Encoding
Pour les variables à plus de deux catégories : One-Hot Encoding

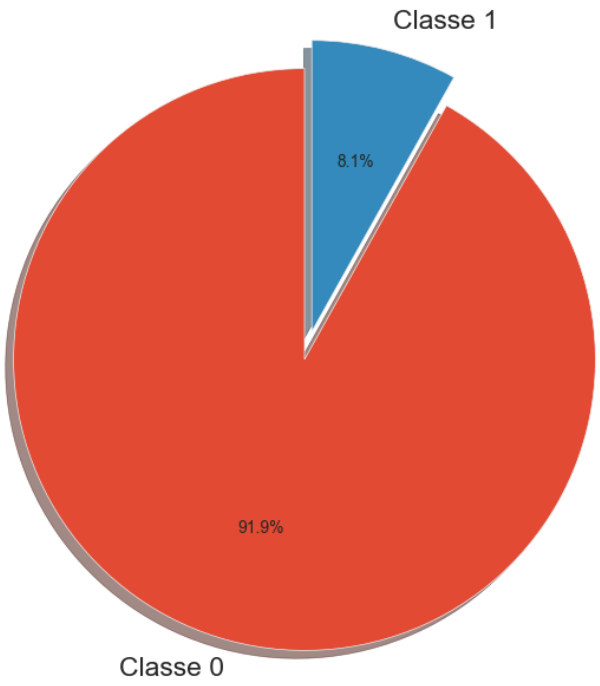
Création de variables → Ajout de 4 features liées au domaine bancaire :

CREDIT_INCOME_PERCENT : *Pourcentage du montant du crédit par rapport aux revenus du client*
ANNUITY_INCOME_PERCENT : *Pourcentage du crédit annuel par rapport aux revenus du client*
CREDIT_TERM : *Durée du paiement en mois*
DAYS_EMPLOYED_PERCENT : *Pourcentage des jours employés par rapport à l'âge du client*

Resampling

Le problème que nous traitons ici est celui du déséquilibre des cibles. En effet, dans les données que nous avons à disposition, la part des personnes ayant payé leur crédit (désigné par la classe 0) est bien supérieure à celle des personnes ne l'ayant pas remboursé (désigné par la classe 1).

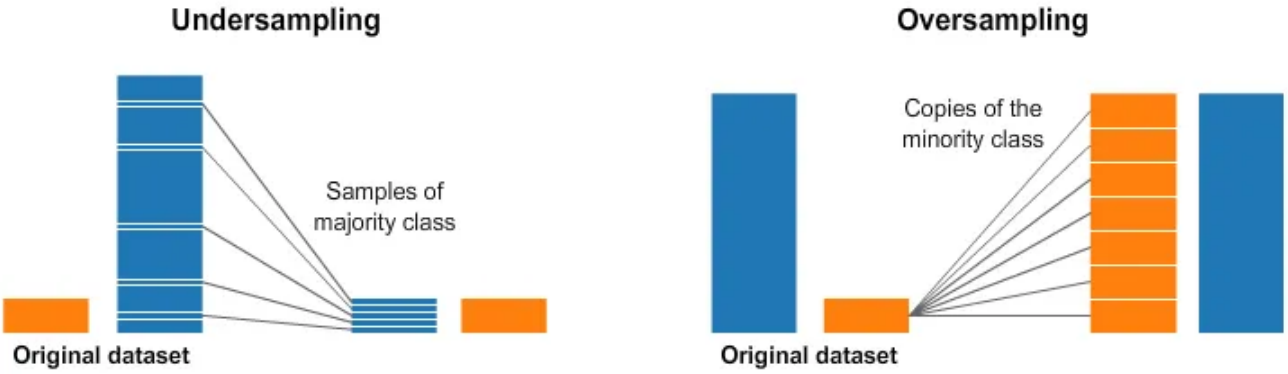
L'image ci-dessous représente le déséquilibre initial :



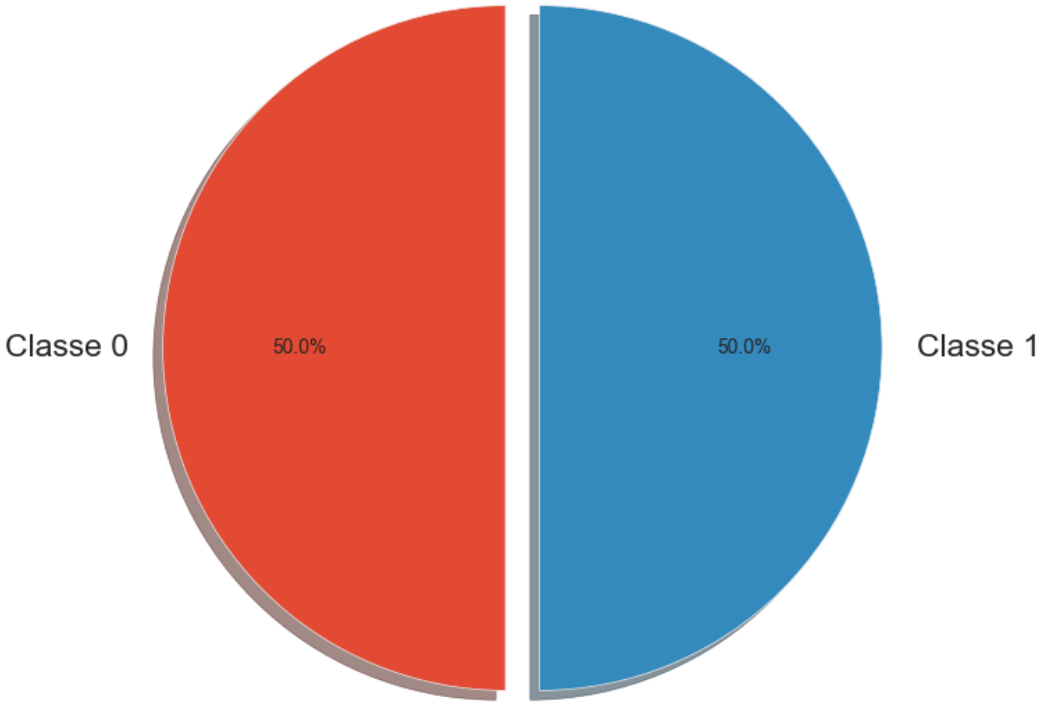
En l'état, entraîner le modèle sur de telles données représente un risque : le modèle sera très biaisé vers la classe majoritaire et ne prédira donc pas correctement les cibles.

Pour pallier à ce problème, il convient d'équilibrer notre jeu de données.
Pour ce faire, nous faisons appel à une librairie Python : **Imblearn**, et en particulier une technique nommée **Random Under Sampler**.

L'undersampling et l'oversampling sont deux techniques possibles pour le rééquilibrage de classes :



L'image ci-dessous représente les classes après rééquilibrage :



Construction du modèle

Accuracy, Precision, Recall, F1 Score

Nous avons testé plusieurs modèles de classification et retenu le **XGBoost** .
Voici les résultats de notre première tentative :

Les métriques d'erreur qui nous intéressent sont surtout :

- Précision : Un coefficient qui détermine que, lorsque le modèle prédit un 1, il a raison à X %.

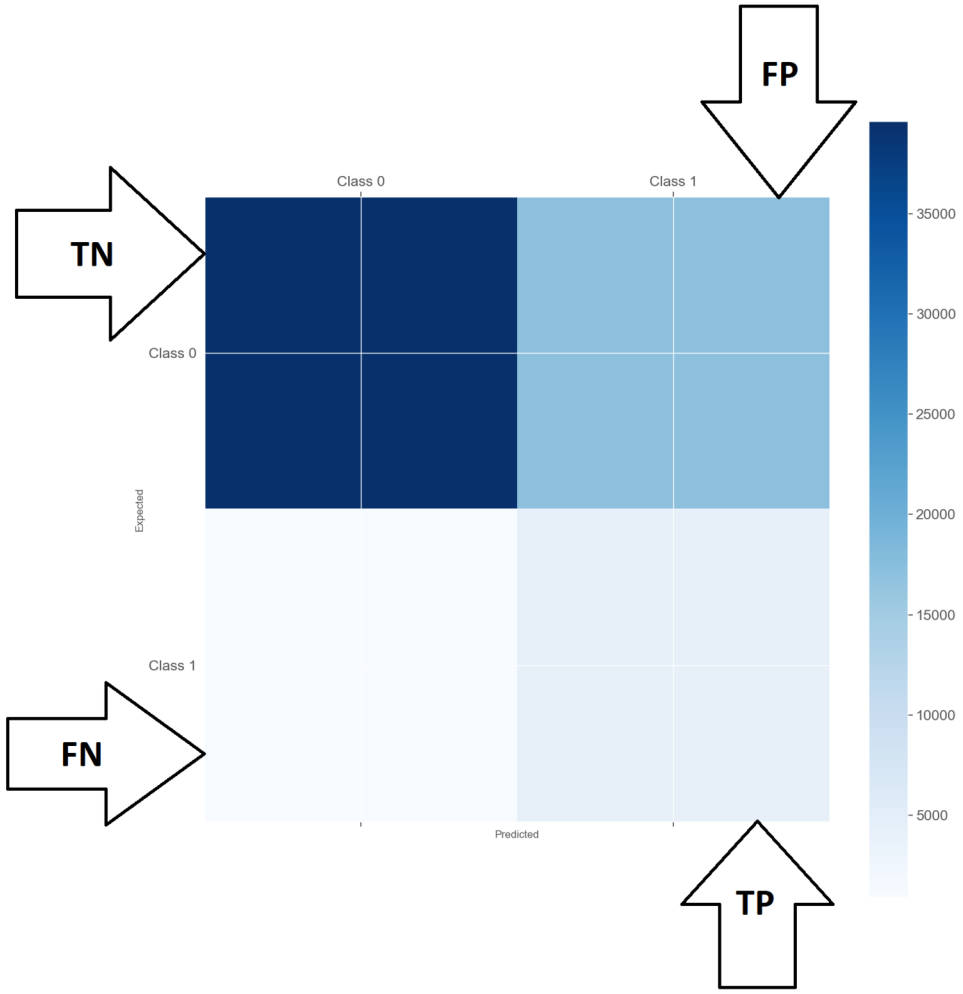
- Recall : Un coefficient qui détermine le pourcentage de détection des 1 du modèle.

	Accuracy	Precision	Recall	F1_score
Logistic regression	0.68	0.16	0.68	0.25
Random Forest	0.69	0.17	0.75	0.27
XGBoost	0.71	0.19	0.81	0.3

Les trois modèles donnent donc des résultats similaires, avec le XGBoost légèrement plus performant. Malheureusement ces résultats sont encore peu convaincants, l'interprétation que nous pouvons en faire est la suivante : le modèle parvient à détecter 81 % des classes 1, mais lorsqu'il prédit une classe 1, il n'a raison que dans 19 % des cas.

Matrice de confusion

La matrice de confusion permet de visualiser à quel point les classifications de notre modèle sont exactes.



Les 4 catégories possibles sont les suivantes :

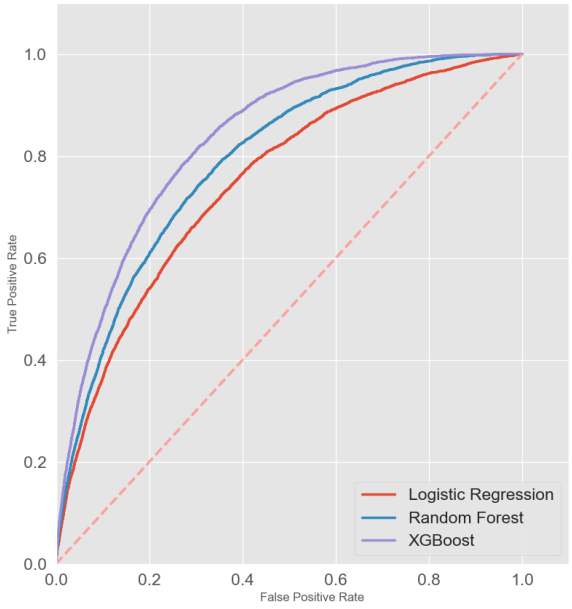
- True Negative (TN) : Le modèle prédit une classe 0 et cela correspond à la réalité.
- False Negative (FN) : Le modèle prédit une classe 0 alors que ces clients sont en réalité en classe 1 (défaut de paiement).
- False Positive (FP) : Le modèle prédit une classe 1 alors que ces clients sont en réalité en classe 0 (pas de défaut de paiement).
- True Positive (TP) : Le modèle prédit une classe 1 et cela correspond à la réalité.

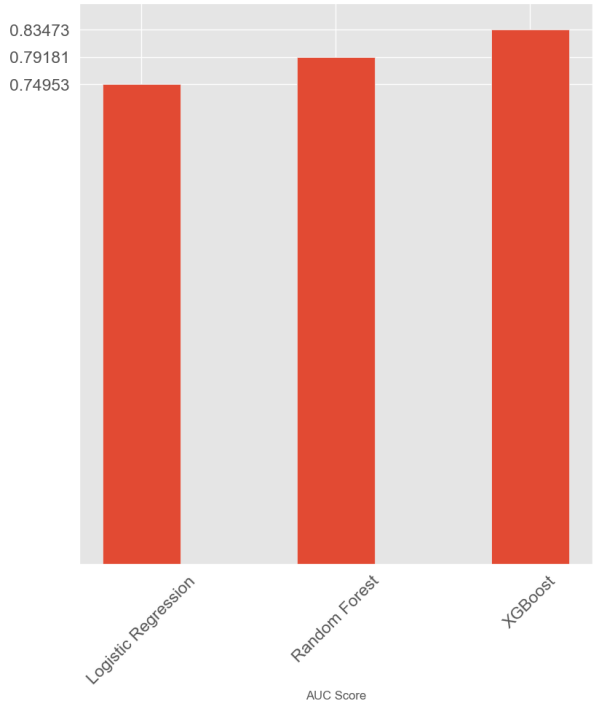
Courbe ROC / Score AUC

La courbe ROC (Receiver Operating Characteristic) est un outil souvent utilisé pour les classificateurs binaires, elle croise le taux de TP avec le taux de FP.

La figure suivante représente la courbe ROC des modèles testés, ainsi que celle d'un classificateur purement aléatoire (en pointillé). Un classificateur performant devrait s'éloigner de cette ligne, et se rapprocher le plus possible du coin supérieur gauche.

Une deuxième mesure consiste à évaluer l'aire sous la courbe (Area Under the Curve ou AUC). Dans ce cas, un modèle parfait obtiendrait un score AUC de 1, alors qu'un modèle purement aléatoire aurait un score de 0,5.

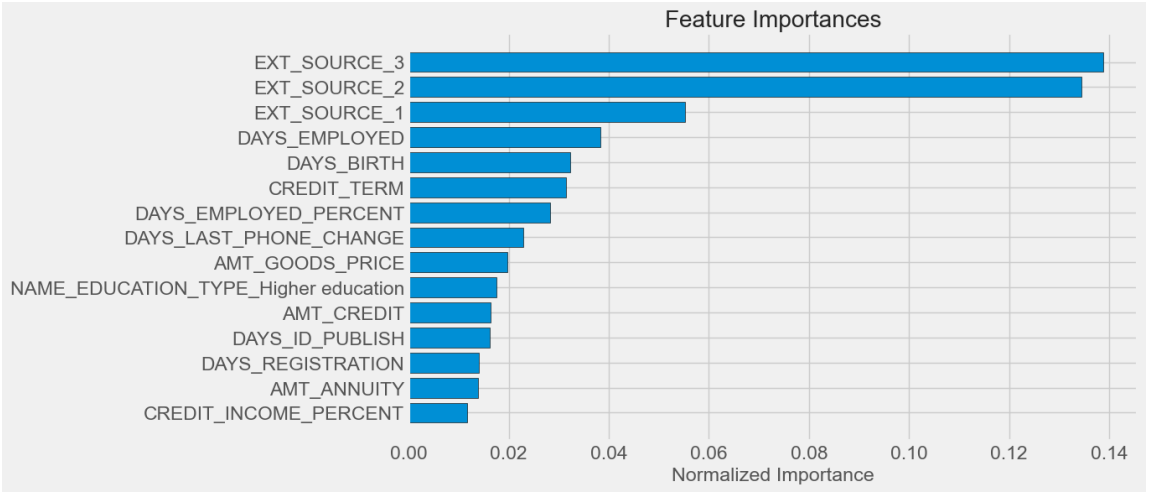




Les performances de nos modèles ne sont pas excellentes et sont assez proches l'une de l'autre. Nous aimerions que les courbes se rapprochent du coin supérieur gauche.

Feature Importance

Nous pouvons également visualiser sur quelles variables le modèle s'appuie pour effectuer ses prédictions :



Optimisation

Afin d'améliorer les performances de notre modèle, nous pouvons jouer sur ses *hyperparamètres*.

En effet, le modèle XGBoost possède trois types d'hyperparamètres :

- General Parameters : Réglages des préférences du modèle
- Booster Parameters : Réglages du modèle lors des étapes de prédictions
- Learning Task Parameters : Réglages du type de prédiction souhaité

Voici les hyperparamètres utilisés pour optimiser notre modèle :

- ***N_estimators***
- ***Max_depth :*** Profondeur maximale d'un arbre, ce paramètre est utilisé pour contrôler le sur-ajustement : une valeur plus élevée permettra au modèle d'apprendre des relations plus spécifiques à l'échantillon donné.
- ***Learning_rate***
- ***Subsample :*** Précise la part des observations qui doivent être des échantillons aléatoires pour chaque arbre. Des valeurs faibles rendent l'algorithme plus conservateur et empêche le sur-ajustement, mais des valeurs trop petites peuvent mener à un sous-ajustement.
- ***Colsample_bytree***
- ***Min_child_weight :*** La somme minimale des poids des observations requises chez un enfant. Ce paramètre contrôle le sur-ajustement : les valeurs plus élevées empêchent le modèle d'apprendre des relations très spécifiques à l'échantillon en question, tandis que les valeurs moins élevées peuvent mener à un sous-ajustement.

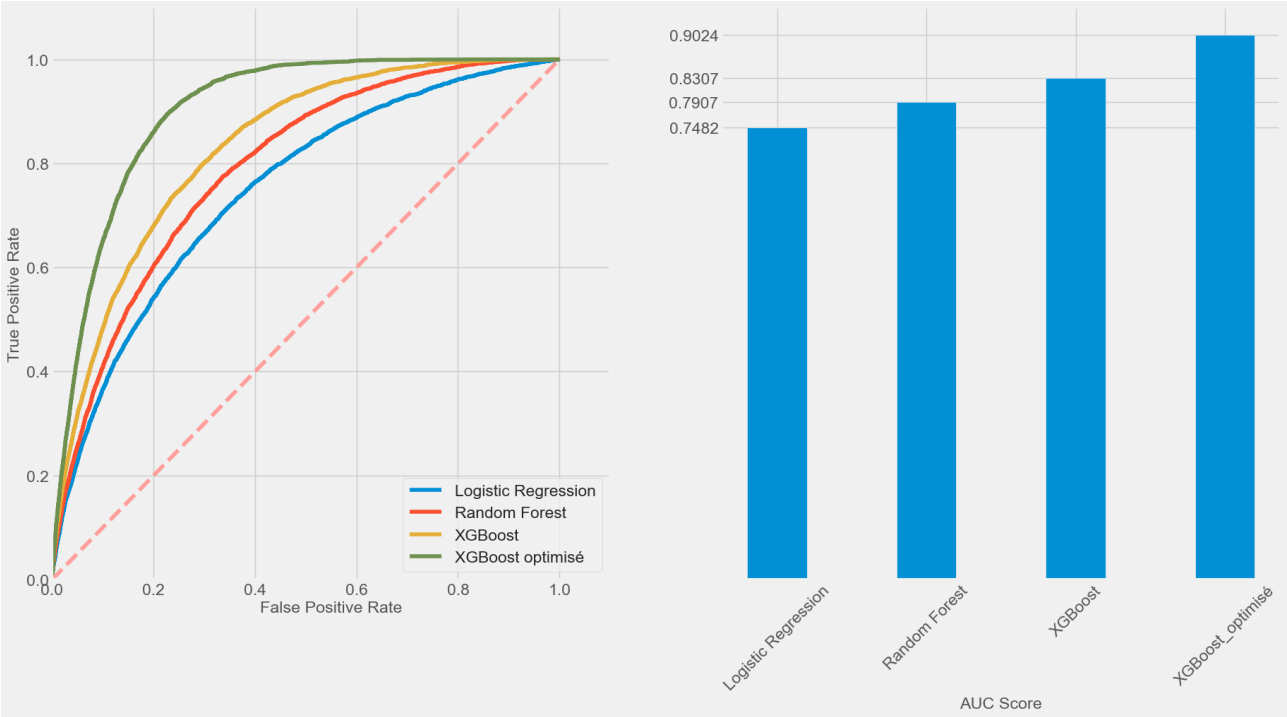
Paramètres optimisés :

```
xgb.XGBClassifier(booster='gbtree',
                  colsample_bytree=0.6784538670198459,
                  eval_metric='auc',
                  learning_rate=0.10310087264740633,
                  max_depth=6,
                  min_child_weight=2,
                  n_estimators=1144,
                  objective='binary:logistic',
                  random_state=0,
                  subsample=0.4915549740714592,
                  n_jobs=-1
                  )
```

Résultats

	Accuracy	Precision	Recall	F1_score
Logistic regression	0.68	0.16	0.68	0.25
Random Forest	0.69	0.17	0.75	0.27
XGBoost	0.71	0.19	0.81	0.3
XGBoost_optimisé	0.72	0.21	0.95	0.35

On peut observer que l'optimisation des paramètres a légèrement amélioré les performances de notre modèle.



Conclusion sur le modèle :

Les performances de notre modèle sont encore nettement améliorables, ce qui peut probablement s'expliquer par le feature engineering qui pourrait être plus poussé. En effet, nous nous sommes basés sur un point de départ relativement simple qui aurait pu être amélioré davantage, par exemple en créant plus de variables utiles aux modèles de classification comme des moyennes, médianes, etc...