Note méthodologique

Projet 7 :

Implémentez un modèle de scoring

Table des matières :

Introduction :

Les problèmes de machine learning peuvent se différencier selon deux critères :

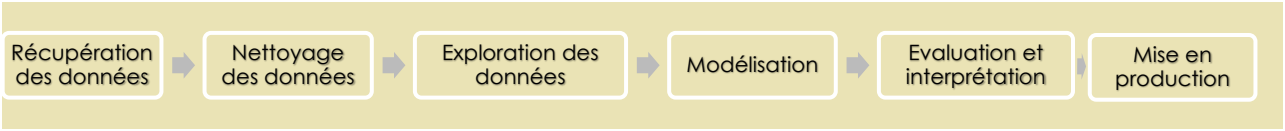
- Les données dont on dispose sont annotées ou non, si c'est le cas, on a affaire à un problème d'apprentissage **supervisé**. Sinon on utilise un algorithme d'apprentissage **non supervisé.**

  Dans le cadre de ce projet on est en face à un problème d'apprentissage **supervisé**.

- Le type de résultat que nous souhaitons prédire, s'il s'agit d'un nombre (par exemple le prix d'un appartement), c'est un problème de **régression**. S'il s'agit plutôt d'une valeur discrète, d'une catégorie (par exemple le type d'animal présent sur une photo), alors c'est un problème de **classification**.

  Dans le cadre de ce projet on est en face à un problème de **classification**

Tout projet de machine learning passe par ses étapes :



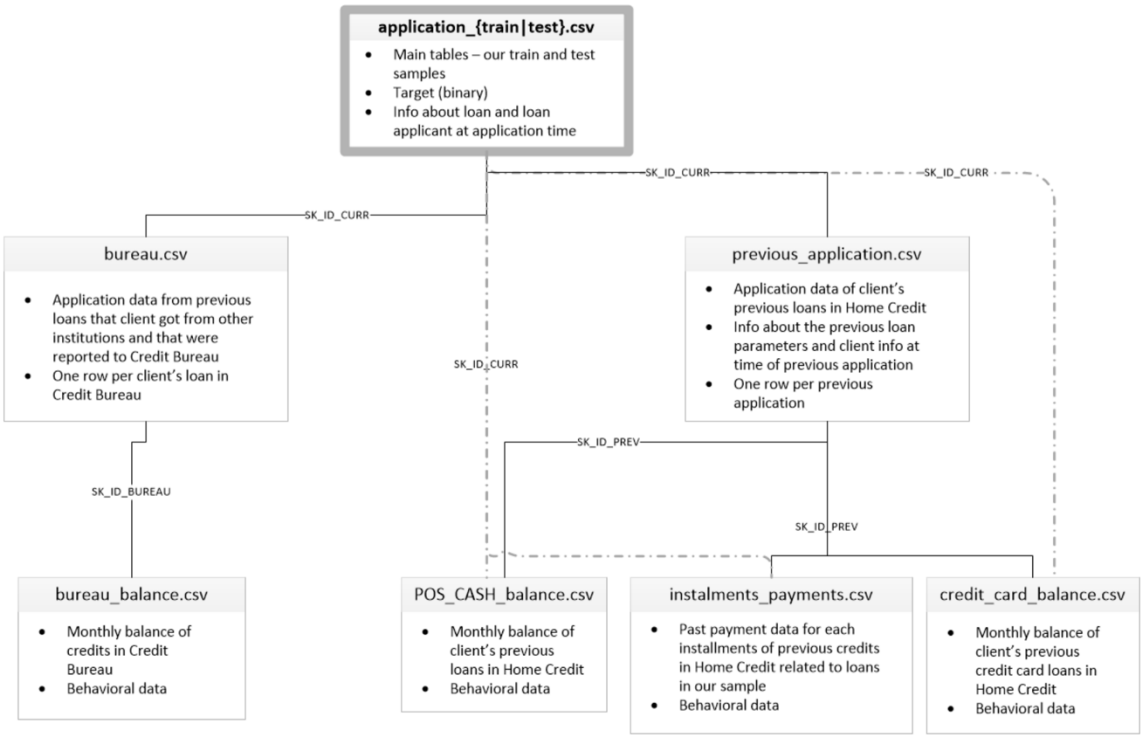
- Pour de ce projet l'entreprise souhaite mettre en œuvre un outil de "scoring crédit" pour calculer la **probabilité** qu'un client rembourse son crédit, puis **classifie** la demande en crédit **accordé** ou **refusé**. Elle souhaite donc développer un algorithme de classification en s'appuyant sur des sources de données variées (données comportementales, données provenant d'autres institutions financières, etc.).

1- La méthodologie d'entraînement du modèle :

1-1 Feature engineering :

La première étape consiste à récupérer des données cohérentes afin d'exécuter l'algorithme de Machine Learning :

Présentation des données :



Nous disposons de **7** fichiers de **données** en plus d'un fichier de test que nous n'utiliserons que pour la prédiction finale sur le Dashboard.

Le fichier principal d'entrainement : train contient des données relatives au client avec la Target (**0 : client sans de difficulté de paiement ,1 : client avec difficulté de paiement**), sera mergé avec les autres fichiers afin d'obtenir un fichier riche en données.

Nous utiliserons **un kernel Kaggle existant** pour faciliter la préparation des données nécessaires. Ce kernel sera analysé et adapté afin de s'assurer qu'il répond aux besoins de notre mission.

Le kernel fourni permet la jointure des fichiers, la création de nouvelles variables synthétique ainsi que l'encodage des variables qualitatives par la méthode get_dummies.

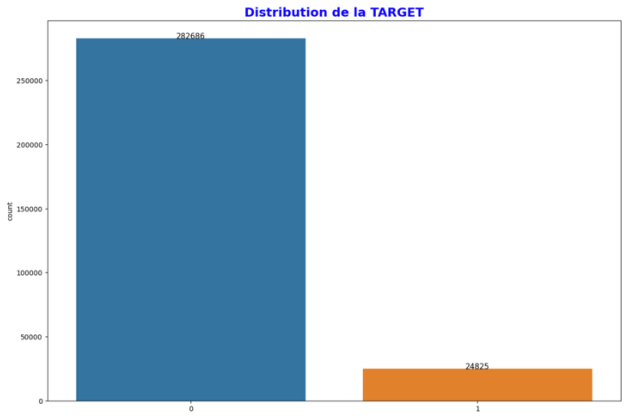
L'imputation des données manquantes a été faite selon deux stratégies : une imputation par médiane pour les variables quantitatives et une imputation par la variable la plus fréquente pour les variables qualitatives.

1- 2 Algorithmes utilisés pour la construction du modèle de scoring :

Préparation des données :

Avant la modélisation il est nécessaire de normaliser les données, il y a plusieurs façons, pour ce projet c'est le MinMaxScaler qui sera utilisé.

L'analyse exploratoire des données a montré un déséquilibre entre les deux classes, en effet nous avons **92%** de données dans la classe **0** et **8%** des données pour la classe **1**.



Dans de tel cas, on obtient une précision assez élevée simplement en prédisant la classe majoritaire, mais on ne parvient pas à capturer la classe minoritaire.

Face à ce problème il existe plusieurs **techniques d'échantillonnage** des données : Random Under-Sampling, Random Over-Sampling, SMOTE, NearMiss, …)

Pour le projet on teste différentes techniques (SMOTE, SMOTE borderline et class-weights).

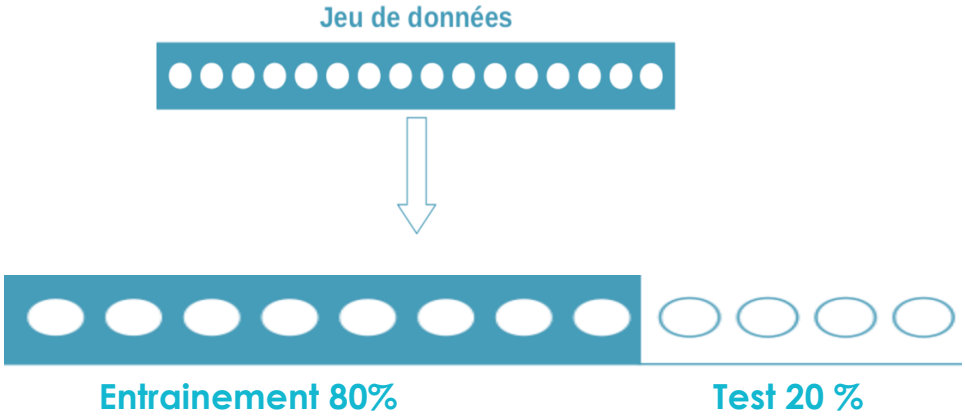
Les techniques SMOTE n'ont pas donné de bons résultats (le modèle n'arrive pas à détecter la classe minoritaire).

La méthode class-weights a mieux fonctionnée. Cette méthode attribue des poids différents aux classes majoritaires et minoritaires. La différence de poids influencera le classement des classes lors de la phase d'entraînement. Le but est de fixer un poids de classe plus élevé et en même temps en réduisant le poids de la classe majoritaire.

1-3- Implémentation des algorithmes de machine learning :

Il existe plusieurs algorithmes de machine learning, pour ce projet trois algorithmes de gradient boosting ont été utilisés : **GradientboostClassifier**, **CatboostClassifier** et le **XgboostClassifier** en raison de leur précision, rapidité, en particulier pour les données complexes et volumineuses.

La première étape dans le cadre de la modélisation est de couper notre jeu de données en deux parties : un jeu **d'entraînement**, et un jeu de **test**. Le jeu de test n'est pas utilisé pour entraîner le modèle, mais uniquement pour l'évaluer. Le découpage se fait grâce à la fonction **train_test_split** () de sckit-learn.



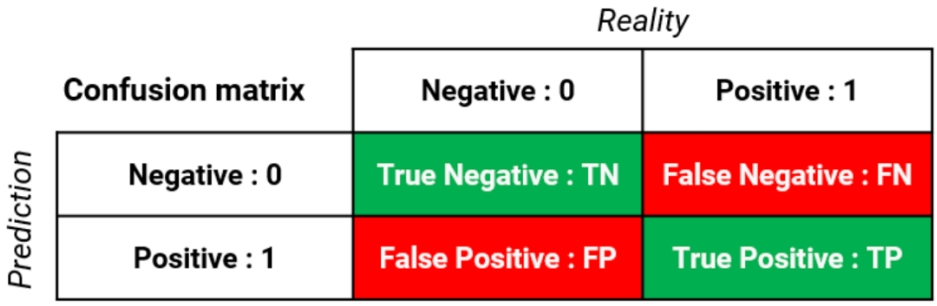
La construction des modèles ci-dessus a été faite en passant par une pipeline qui intègre : la Normalisation et le ré échantillonnage.

Pour être en mesure de vraiment comprendre puis d'améliorer les performances de notre modèle, nous devons d'abord établir une base de référence pour les données dont nous disposons, le **DummyClassifier** sera notre **Baseline**.

Avant de présenter les résultats de la baseline des modèles testés, il faudra présenter les **métriques** à prendre en considération dans le cadre de notre projet.

Les métriques qui nous intéressent sont la **précision** et le **recall** qui se basent sur la **matrice de confusion** qui est indispensable pour définir les différentes métriques de classification

- **La matrice de confusion** : C'est un tableau à 4 valeurs représentant les différentes combinaisons de valeurs réelles et valeurs prédites comme dans la figure ci-dessous :



Voici une explication de ce que représente ces éléments dans notre cas :

TP : Le client a des difficultés de paiement et le modèle a prédit qu'il a des difficultés de paiement.

TN : Le client n'a pas de difficultés de paiement et le modèle a prédit qu'il n'a pas de difficultés de paiement.

FP : Le client n'a pas de difficultés de paiement et le modèle a prédit qu'il a des difficultés de paiement. (**Erreur de type1**)

FN : Le client a des difficultés de paiement et le modèle a prédit qu'il n'a pas de difficultés de paiement. (**Erreur de type 2**)

La **précision** indique quelle fraction des positifs prédits est réellement positive.

$$Precision = \frac{TP}{TP + FP}$$

De tous les cas que nous avons qualifiés de classe 1 (dans notre exemple, de tous les cas prédits comme client avec défaut de paiement) combien nous avons eu raison (c'est-à-dire combien ils étaient vraiment des clients qui ne remboursent pas).

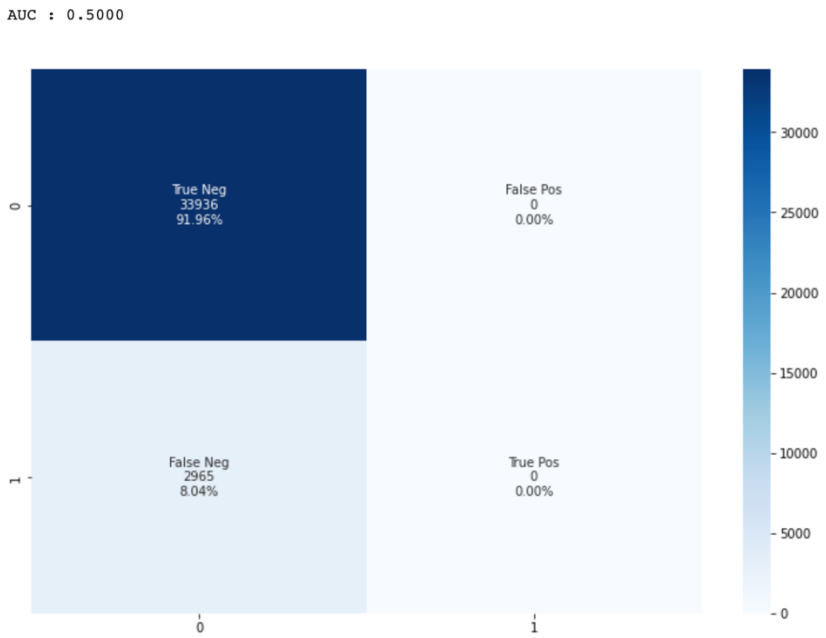
Le **recall** mesure la capacité du modèle à prédire les vais positifs.

$$Recall = \frac{TP}{TP + FN}$$

De tous les cas possibles où la cible était 1, combien nous pourrions capturer, dans notre exemple serait, de tous les cas possibles de client avec problème de paiement, combien notre modèle pourrait identifier.

On s'intéressera aussi à la **courbe ROC** et score **AUC** : Il s'agit d'un graphique du taux de faux positifs par rapport au taux de vrais pour un certain nombre de valeurs de seuil candidates différentes comprises entre 0,0 et 1,0. En d'autres termes, il trace le taux de fausses alarmes par rapport au taux de succès.

<u>DummyClassifier :</u>



Ici le modèle a correctement prédit la classe 0 (TN, quadrant supérieur gauche) 92 % du temps, mais les prédictions pour la classe 1 (TP, quadrant inférieur droit) sont à 0 %. C'est un modèle de classificateur qui fait des prédictions sans essayer de trouver des modèles dans les données. Le modèle par défaut examine essentiellement quelle classe est la plus fréquente dans

l'ensemble de données d'apprentissage et effectue des prédictions basées sur cette étiquette.

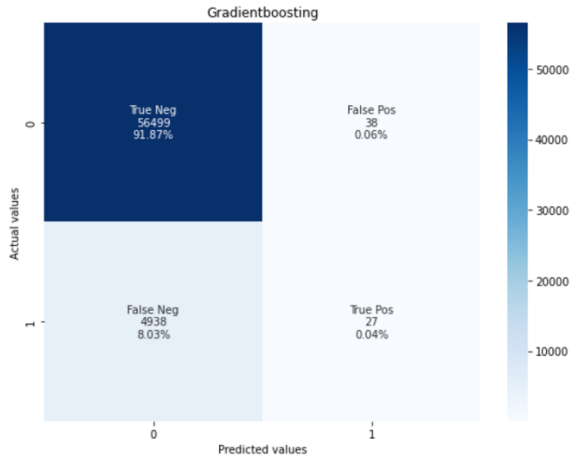
Les modèles ont été testé avec les paramètres de base et puis optimisés avec des GridSearchs afin d'obtenir les meilleures performances.

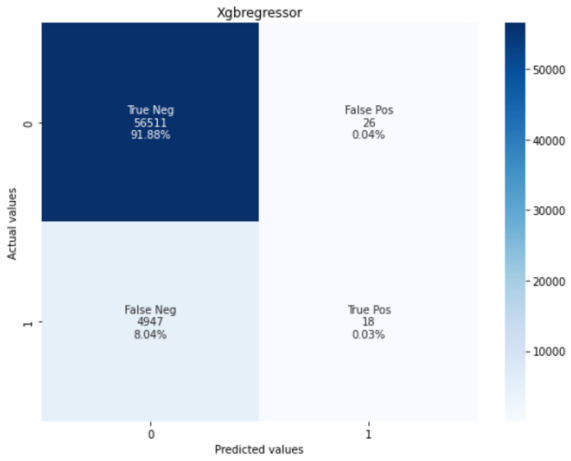
Ici les premiers résultats :

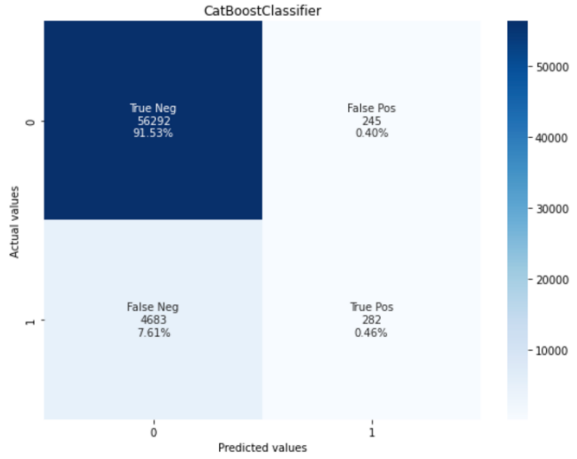
	Models	Auc	Accuracy	Precision	Recall	F1
0	Gradientboosting	0.7413	0.9191	0.4154	0.0054	0.0107
1	Xgbregressor	0.7436	0.9191	0.4091	0.0036	0.0072
2	CatBoostClassifier	0.7846	0.9199	0.5351	0.0568	0.1027

Les modèles présentent des résultats presque similaires. Le catboostclassifier est mieux en AUC et recall.

Pour le dernier modèle on peut traduire le résultat en disant que le modèle prédit 5% de cas positifs, mais que, quand il prédit une classe 1, il a raison dans 53% des cas.

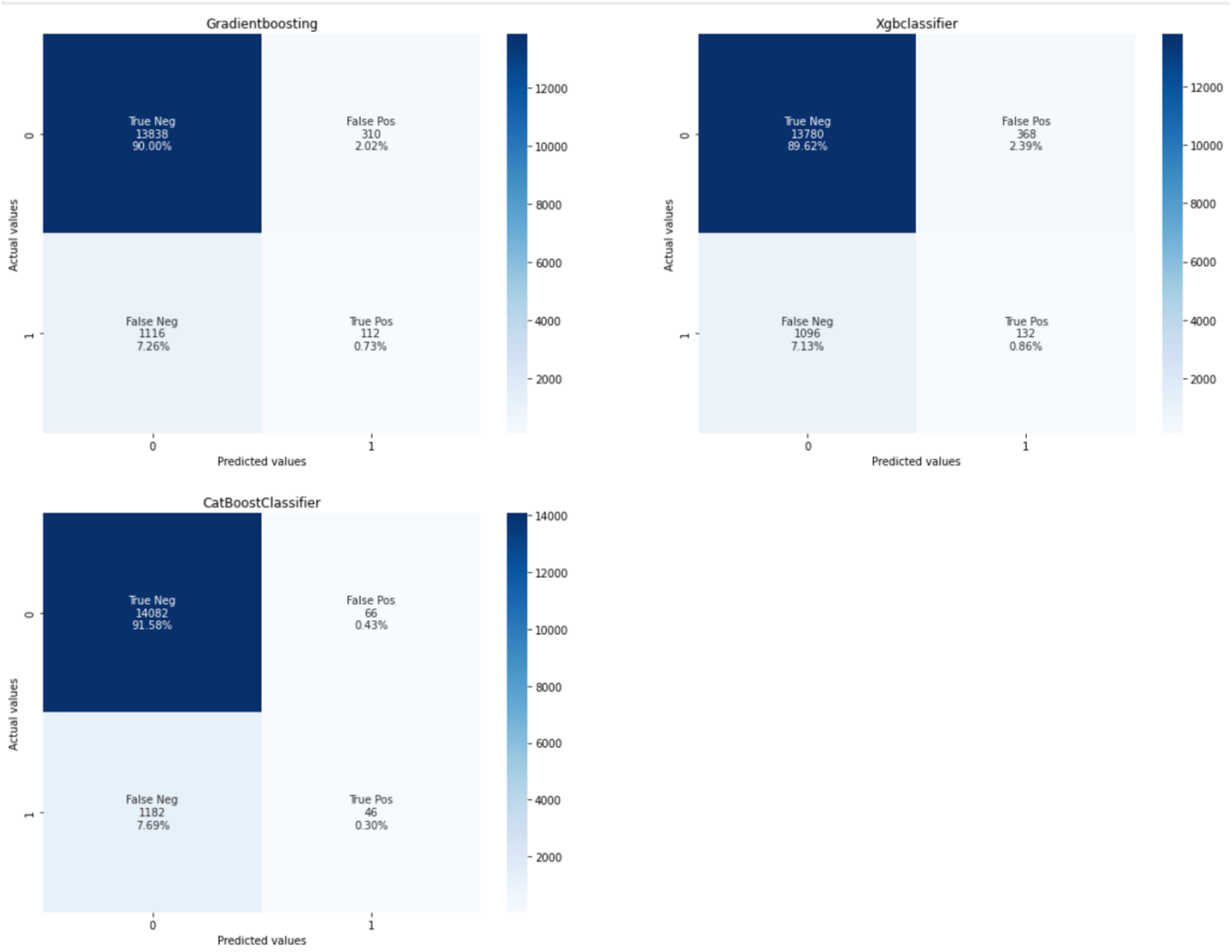






- Optimisation des modèles ave des GridSearchs en optimisant les hyper paramètres :

	Models	Auc	Accuracy	Precision	Recall	F1
0	Gradientboosting	0.7071	0.9073	0.2654	0.0912	0.1358
1	Xgbclassifier	0.7079	0.9048	0.2640	0.1075	0.1528
2	CatBoostClassifier	0.7474	0.9188	0.4107	0.0375	0.0687



En vue des résultats obtenus, et des temps de calcul le modèle choisi sera le **catboostclassifier** sur lequel on se focalisera pour détailler les hyper paramètres ainsi l'analyse des résultats avec et sans la fonction cout.

GridSearchCV est le processus de réglage des hyper paramètres afin de déterminer les valeurs optimales pour un modèle donné. Les performances d'un modèle dépendent de manière significative de la valeur des hyper paramètres.

Dans le cas d'un problème de classification, on fait attention à **stratifier** la validation croisée pour éviter d'introduire des biais.

```python
# définition des Hyperparamètres
classifier = CatBoostClassifier
param_cat = {
            "depth": [2,4, 6],
            "learning_rate": [1,2],
            "n_estimators": [50,100],

             "l2_leaf_reg":[0.5,1]

}
```

Depth : Profondeur de l'arbre

Learning_rate : La fréquence à laquelle les pondérations du modèle sont mises à jour après avoir travaillé sur chaque lot d'exemples d'entrainement.

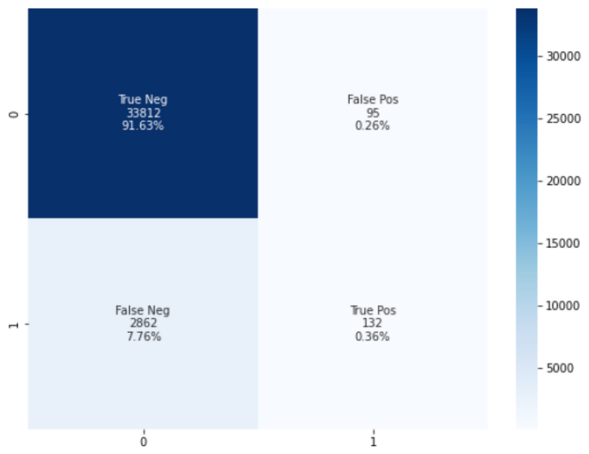
N_estimators : Le nombre maximum d'arbres

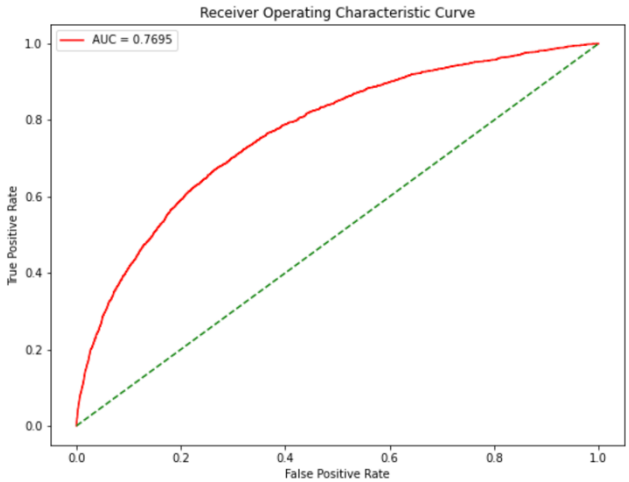
L2_leaf_reg : Coefficient du terme de régularisation L2 de la fonction de coût.

	Models	Auc	Accuracy	Precision	Recall	F1
0	CatboostClassifier	0.769521	0.919867	0.581498	0.044088	0.081962

	precision	recall	f1-score	support
0.0	0.92	1.00	0.96	33907
1.0	0.58	0.04	0.08	2994
accuracy			0.92	36901
macro avg	0.75	0.52	0.52	36901
weighted avg	0.89	0.92	0.89	36901

AUC : 0.7695





2- La fonction coût métier :

L'entreprise Prêt à dépenser est une société **financière** qui propose des crédits et donc essayera de lutter contre les défauts de paiement et **diminuer les pertes financières**.

Le modèle ne permettra pas d'éviter complètement ce risque mais on peut **minimiser** les pertes en définissant une fonction coût afin de **pénaliser** les **erreurs** de prédiction qui peuvent **coûter** cher à l'entreprise.

L'idée est d'éviter les clients avec un fort risque de défaut de paiement. Il est donc nécessaire de pénaliser les FN.

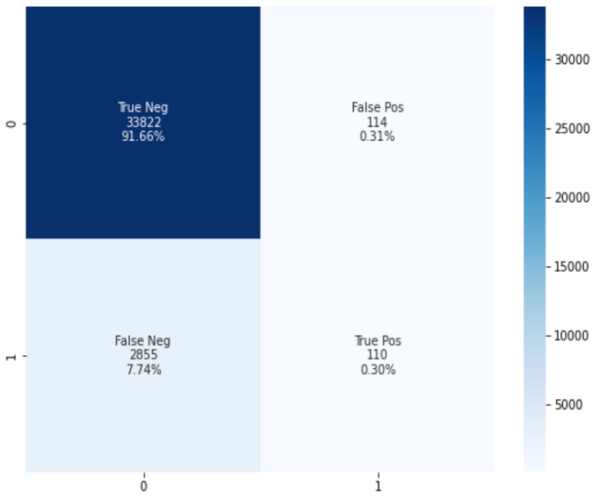
Pour ceci nous créons une matrice de confusion, mais au lieu de mettre les quantités prévues, nous allons mettre des pondérations pour chaque élément de la matrice. Ci-dessous un exemple :

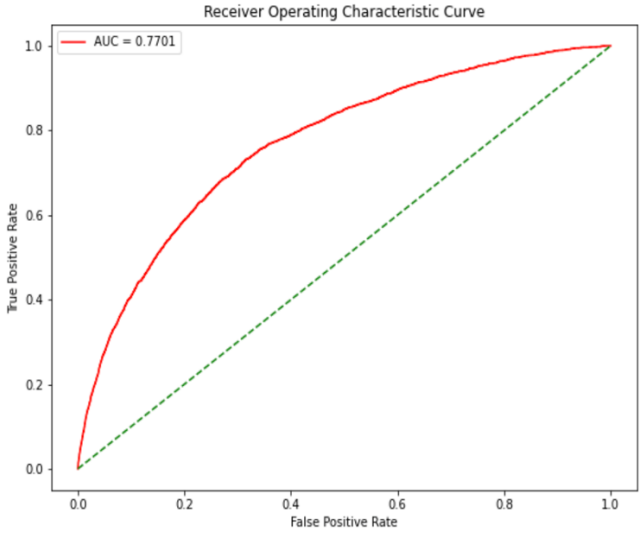
Client qui rembourse et bien identifié (gain pour l'entreprise)	**True Négatif : + 1**	**False positif : 0**	**Client rembourse et mal identifié**
Client ne rembourse pas et mal identité	**False Négatif : -10**	**True positif : 0**	**Client ne rembourse pas et bien identifié**

(Perte pour l'entreprise)

Résultat après implémentation du modèle avec la fonction coût personnalisée :

```
AUC : 0.7701
```





Avec l'implémentation de la métrique métier les FN sont de 7.74 % au lieu de 7.76% et l'AUC de 77%.

Threshold :

Le threshold est le seuil de décision que nous fixons pour qu'un client appartient à la classe 0 ou 1.

La matrice de confusion dépend du seuil de classification qui est par défaut de 50%.

Selon les besoins, le choix du seuil diffère. On peut privilégier la détection des clients ayant des difficultés de paiement en abaissant le seuil.

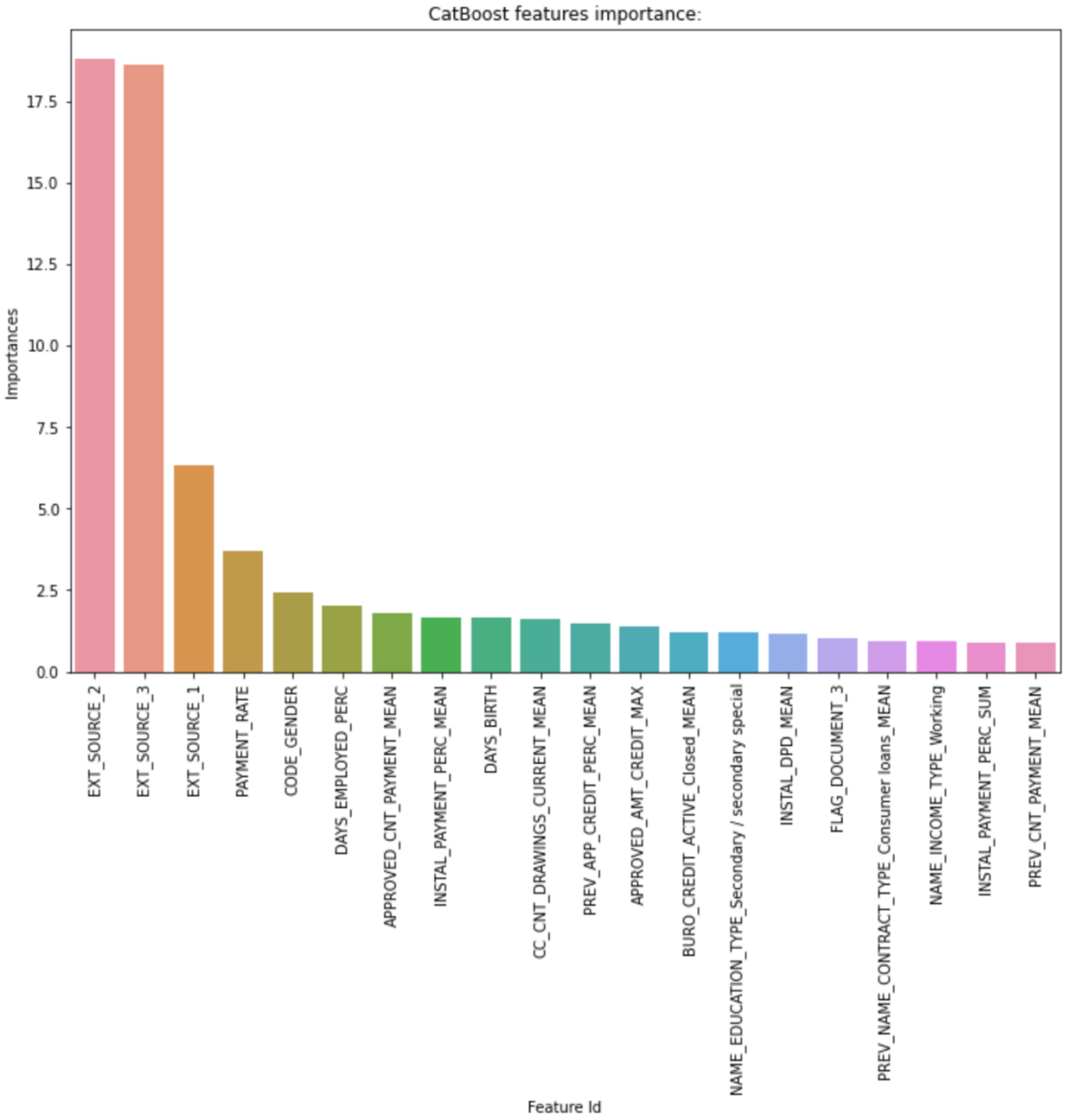
Pour un même modèle, on obtient plusieurs matrices de confusion en fonction du seuil choisi.

3- L'interprétabilité globale et locale du modèle :

L'importance des features permet de **comprendre** la relation entre les caractéristiques et la variable cible. Elle fait référence à des techniques qui calculent un score pour toutes les variables d'entrée pour un modèle donné - les scores représentent simplement « **l'importance** » de chaque fonctionnalité. Un score plus élevé signifie que la variable spécifique aura un effet plus important sur le modèle utilisé pour prédire une certaine variable.

L'importance des fonctionnalités est également utile pour **interpréter** et **communiquer** le modèle à d'autres parties prenantes.

Ci-dessous l'importance des features du modèle catboostclassifier.



4- Les limites et les améliorations possibles :

Les résultats obtenus peuvent être améliorés en essayant d'autres hyper paramètres au niveau de l'optimisation du modèle qui peuvent augmenter ses performances. La taille du dataset et les temps de calcul n'ont pas permis de tester tous les hyperparamètres.

L'hypothèse de la fonction métier personnalisée est à discuter avec l'équipe métier et donc peut être modifiée.

L'ajustement du seuil de probabilité pour l'attribution du crédit doit-être aussi discuté aussi avec les équipes métier ce qui permettra de répondre aux besoins de l'entreprise.